Package ‘keras’

April 5, 2019

Type    Package
Title   R Interface to ‘Keras’
Version 2.2.4.1
Description Interface to ‘Keras’ <https://keras.io>, a high-level neural networks ‘API’. ‘Keras’ was developed with a focus on enabling fast experimentation, supports both convolution based networks and recurrent networks (as well as combinations of the two), and runs seamlessly on both ‘CPU’ and ‘GPU’ devices.

Encoding UTF-8
License MIT + file LICENSE
URL https://keras.rstudio.com

BugReports https://github.com/rstudio/keras/issues
Depends R (>= 3.2)
Imports generics (>= 0.0.1), reticulate (>= 1.10), tensorflow (>= 1.13.1), tfruns (>= 1.0), magrittr, zeallot, methods, R6

Suggests ggplot2, testthat, knitr, rmarkdown
SystemRequirements Keras >= 2.0 (https://keras.io)

RoxygenNote 6.1.1
VignetteBuilder knitr
NeedsCompilation no

Author Daniel Falbel [ctb, cph, cre],
JJ Allaire [aut, cph],
François Chollet [aut, cph],
RStudio [ctb, cph, fnd],
Google [ctb, cph, fnd],
Yuan Tang [ctb, cph] (<https://orcid.org/0000-0001-5243-233X>),
Wouter Van Der Bijl [ctb, cph],
Martin Studer [ctb, cph],
Sigrid Keydana [ctb]

Maintainer Daniel Falbel <daniel@rstudio.com>
Repository CRAN
Date/Publication 2019-04-05 21:30:08 UTC
R topics documented:

keras-package ................................................................. 9
activation_relu ................................................................. 10
application_densenet .......................................................... 11
application_inception_resnet_v2 ............................................. 12
application_inception_v3 ..................................................... 13
application_mobilenet .......................................................... 15
application_mobilenet_v2 ..................................................... 16
application_nasnet ............................................................. 18
application_resnet50 ........................................................... 19
application_vgg ................................................................. 21
application_xception .......................................................... 22
backend ......................................................................... 23
bidirectional ................................................................. 24
callback_csv_logger ............................................................. 25
callback_early_stopping ...................................................... 25
callback_lambda ................................................................. 26
callback_learning_rate_scheduler .......................................... 27
callback_model_checkpoint .................................................. 28
callback_progbar_logger ...................................................... 29
callback_reduce_lr_on_plateau .............................................. 29
callback_remote_monitor ..................................................... 30
callback_tensorboard ........................................................... 31
callback_terminate_on_naan ................................................ 32
clone_model ................................................................. 33
compile.keras.engine.training.Model ....................................... 33
constraints ................................................................. 34
count_params ................................................................. 36
create_layer ................................................................. 36
create_wrapper ............................................................... 37
dataset_boston_housing ..................................................... 37
dataset_cifar10 ................................................................. 38
dataset_cifar100 ................................................................. 39
dataset_fashion_mnist ....................................................... 39
dataset_imdb ................................................................. 40
dataset_mnist ................................................................. 41
dataset_reuters ................................................................. 41
evaluate.keras.engine.training.Model ..................................... 42
evaluate_generator ............................................................. 43
export_savedmodel.keras.engine.training.Model .................................. 45
fit.keras.engine.training.Model ........................................... 45
fit_generator ................................................................. 47
fit_image_data_generator ................................................... 49
fit_text_tokenizer ............................................................. 50
flow_images_from_data ...................................................... 51
flow_images_from_dataframe ................................................ 52
flow_images_from_directory ................................................ 54
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>freeze_weights</td>
<td>56</td>
</tr>
<tr>
<td>generator_next</td>
<td>57</td>
</tr>
<tr>
<td>get_config</td>
<td>57</td>
</tr>
<tr>
<td>get_file</td>
<td>58</td>
</tr>
<tr>
<td>get_input_at</td>
<td>59</td>
</tr>
<tr>
<td>get_layer</td>
<td>60</td>
</tr>
<tr>
<td>get_weights</td>
<td>61</td>
</tr>
<tr>
<td>hdf5_matrix</td>
<td>61</td>
</tr>
<tr>
<td>imagenet_decode_predictions</td>
<td>62</td>
</tr>
<tr>
<td>imagenet_preprocess_input</td>
<td>62</td>
</tr>
<tr>
<td>image_data_generator</td>
<td>63</td>
</tr>
<tr>
<td>image_load</td>
<td>65</td>
</tr>
<tr>
<td>image_to_array</td>
<td>65</td>
</tr>
<tr>
<td>implementation</td>
<td>66</td>
</tr>
<tr>
<td>initializer_constant</td>
<td>67</td>
</tr>
<tr>
<td>initializer_glorot_normal</td>
<td>67</td>
</tr>
<tr>
<td>initializer_glorot_uniform</td>
<td>68</td>
</tr>
<tr>
<td>initializer_he_normal</td>
<td>69</td>
</tr>
<tr>
<td>initializer_he_uniform</td>
<td>69</td>
</tr>
<tr>
<td>initializer_identity</td>
<td>70</td>
</tr>
<tr>
<td>initializer_lecun_normal</td>
<td>70</td>
</tr>
<tr>
<td>initializer_lecun_uniform</td>
<td>71</td>
</tr>
<tr>
<td>initializer_ones</td>
<td>72</td>
</tr>
<tr>
<td>initializer_orthogonal</td>
<td>72</td>
</tr>
<tr>
<td>initializer_random_normal</td>
<td>73</td>
</tr>
<tr>
<td>initializer_random_uniform</td>
<td>73</td>
</tr>
<tr>
<td>initializer_truncated_normal</td>
<td>74</td>
</tr>
<tr>
<td>initializer_variance_scaling</td>
<td>74</td>
</tr>
<tr>
<td>initializer_zeros</td>
<td>75</td>
</tr>
<tr>
<td>install_keras</td>
<td>76</td>
</tr>
<tr>
<td>is_keras_available</td>
<td>78</td>
</tr>
<tr>
<td>KerasCallback</td>
<td>78</td>
</tr>
<tr>
<td>KerasConstraint</td>
<td>80</td>
</tr>
<tr>
<td>KerasLayer</td>
<td>81</td>
</tr>
<tr>
<td>KerasWrapper</td>
<td>81</td>
</tr>
<tr>
<td>keras_array</td>
<td>82</td>
</tr>
<tr>
<td>keras_model</td>
<td>82</td>
</tr>
<tr>
<td>keras_model_custom</td>
<td>83</td>
</tr>
<tr>
<td>keras_model_sequential</td>
<td>84</td>
</tr>
<tr>
<td>k_abs</td>
<td>85</td>
</tr>
<tr>
<td>k_all</td>
<td>86</td>
</tr>
<tr>
<td>k_any</td>
<td>87</td>
</tr>
<tr>
<td>k_arange</td>
<td>87</td>
</tr>
<tr>
<td>k_argmax</td>
<td>88</td>
</tr>
<tr>
<td>k_argmin</td>
<td>89</td>
</tr>
<tr>
<td>k_backend</td>
<td>89</td>
</tr>
<tr>
<td>k_batch_dot</td>
<td>90</td>
</tr>
<tr>
<td>k_batch_flatten</td>
<td>91</td>
</tr>
<tr>
<td>Topics Documented</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>k_batch_get_value</td>
<td>91</td>
</tr>
<tr>
<td>k_batch_normalization</td>
<td>92</td>
</tr>
<tr>
<td>k_batch_set_value</td>
<td>93</td>
</tr>
<tr>
<td>k_bias_add</td>
<td>93</td>
</tr>
<tr>
<td>k_binary_crossentropy</td>
<td>94</td>
</tr>
<tr>
<td>k_cast</td>
<td>95</td>
</tr>
<tr>
<td>k_cast_to_floatx</td>
<td>95</td>
</tr>
<tr>
<td>k_categorical_crossentropy</td>
<td>96</td>
</tr>
<tr>
<td>k_clear_session</td>
<td>97</td>
</tr>
<tr>
<td>k_clip</td>
<td>97</td>
</tr>
<tr>
<td>k_concatenate</td>
<td>98</td>
</tr>
<tr>
<td>k_constant</td>
<td>98</td>
</tr>
<tr>
<td>k_conv1d</td>
<td>99</td>
</tr>
<tr>
<td>k_conv2d</td>
<td>100</td>
</tr>
<tr>
<td>k_conv2d_transpose</td>
<td>100</td>
</tr>
<tr>
<td>k_conv3d</td>
<td>101</td>
</tr>
<tr>
<td>k_conv3d_transpose</td>
<td>102</td>
</tr>
<tr>
<td>k_cos</td>
<td>103</td>
</tr>
<tr>
<td>k_count_params</td>
<td>103</td>
</tr>
<tr>
<td>k_ctc_batch_cost</td>
<td>104</td>
</tr>
<tr>
<td>k_ctc_decode</td>
<td>105</td>
</tr>
<tr>
<td>k_ctc_label_dense_to_sparse</td>
<td>106</td>
</tr>
<tr>
<td>k_cumprod</td>
<td>106</td>
</tr>
<tr>
<td>k_cumsum</td>
<td>107</td>
</tr>
<tr>
<td>k_depthwise_conv2d</td>
<td>108</td>
</tr>
<tr>
<td>k_dot</td>
<td>108</td>
</tr>
<tr>
<td>k_dropout</td>
<td>109</td>
</tr>
<tr>
<td>k_dtype</td>
<td>110</td>
</tr>
<tr>
<td>k_elu</td>
<td>110</td>
</tr>
<tr>
<td>k_epsilon</td>
<td>111</td>
</tr>
<tr>
<td>k_equal</td>
<td>111</td>
</tr>
<tr>
<td>k_eval</td>
<td>112</td>
</tr>
<tr>
<td>k_exp</td>
<td>113</td>
</tr>
<tr>
<td>k_expand_dims</td>
<td>113</td>
</tr>
<tr>
<td>k_eye</td>
<td>114</td>
</tr>
<tr>
<td>k_flatten</td>
<td>115</td>
</tr>
<tr>
<td>k_floatx</td>
<td>115</td>
</tr>
<tr>
<td>k_foldl</td>
<td>116</td>
</tr>
<tr>
<td>k_foldr</td>
<td>117</td>
</tr>
<tr>
<td>k_function</td>
<td>117</td>
</tr>
<tr>
<td>k_get_session</td>
<td>118</td>
</tr>
<tr>
<td>k_get_value</td>
<td>119</td>
</tr>
<tr>
<td>k_get_variable_shape</td>
<td>119</td>
</tr>
<tr>
<td>k_gradients</td>
<td>120</td>
</tr>
<tr>
<td>k_greater</td>
<td>121</td>
</tr>
<tr>
<td>k_greater_equal</td>
<td>122</td>
</tr>
</tbody>
</table>
topics documented:

- k_hard_sigmoid .................................................. 123
- k_identity ..................................................... 124
- k_image_data_format .......................................... 124
- k_int_shape .................................................... 125
- k_in_test_phase .............................................. 125
- k_in_top_k .................................................... 126
- k_in_train_phase ............................................. 127
- k_is_keras_tensor ............................................ 127
- k_is_placeholder ........................................... 128
- k_is_sparse ................................................... 128
- k_is_tensor ................................................... 129
- k_l2_normalize .............................................. 130
- k_learning_phase ........................................... 130
- k_less ........................................................ 131
- k_less_equal ............................................... 131
- k_local_conv1d .............................................. 132
- k_local_conv2d ............................................. 133
- k_log .......................................................... 134
- k_logsumexp .................................................. 134
- k_manual_variable_initialization .......................... 135
- k_map_fn ....................................................... 136
- k_max .......................................................... 136
- k_maximum .................................................... 137
- k_mean ........................................................ 138
- k_min .......................................................... 138
- k_minimum ..................................................... 139
- k_moving_average_update ................................... 140
- k_ndim ........................................................ 140
- k_normalize_batch_in_training ............................... 141
- k_not_equal .................................................. 142
- k_ones ........................................................ 142
- k_ones_like ................................................ 143
- k_one_hot .................................................... 144
- k_permute_dimensions ...................................... 144
- k_placeholder ............................................... 145
- k_pool2d ........................................................ 146
- k_pool3d ...................................................... 146
- k_pow .......................................................... 147
- k_print_tensor ............................................... 147
- k_prod ........................................................ 148
- k_random_binomial ........................................... 148
- k_random_normal ............................................. 149
- k_random_normal_variable .................................. 150
- k_random_uniform ........................................... 151
- k_random_uniform_variable ................................ 152
- k_relu ........................................................ 153
- k_repeat ...................................................... 153
- k_repeat_elements .......................................... 154
<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>k_reset_uids</td>
<td>155</td>
</tr>
<tr>
<td>k_reshape</td>
<td>155</td>
</tr>
<tr>
<td>k_resize_images</td>
<td>156</td>
</tr>
<tr>
<td>k_resize_volumes</td>
<td>156</td>
</tr>
<tr>
<td>k_reverse</td>
<td>157</td>
</tr>
<tr>
<td>k_rnn</td>
<td>158</td>
</tr>
<tr>
<td>k_round</td>
<td>159</td>
</tr>
<tr>
<td>k_separable_conv2d</td>
<td>159</td>
</tr>
<tr>
<td>k_set_learning_phase</td>
<td>160</td>
</tr>
<tr>
<td>k_set_value</td>
<td>161</td>
</tr>
<tr>
<td>k_shape</td>
<td>161</td>
</tr>
<tr>
<td>k_sigmoid</td>
<td>162</td>
</tr>
<tr>
<td>k_sign</td>
<td>162</td>
</tr>
<tr>
<td>k_sin</td>
<td>163</td>
</tr>
<tr>
<td>k_softmax</td>
<td>164</td>
</tr>
<tr>
<td>k_softplus</td>
<td>164</td>
</tr>
<tr>
<td>k_softsign</td>
<td>165</td>
</tr>
<tr>
<td>k_sparse_categorical_crossentropy</td>
<td>165</td>
</tr>
<tr>
<td>k_spatial_2d_padding</td>
<td>166</td>
</tr>
<tr>
<td>k_spatial_3d_padding</td>
<td>167</td>
</tr>
<tr>
<td>k_sqrt</td>
<td>167</td>
</tr>
<tr>
<td>k_square</td>
<td>168</td>
</tr>
<tr>
<td>k_squeeze</td>
<td>169</td>
</tr>
<tr>
<td>k_stack</td>
<td>169</td>
</tr>
<tr>
<td>k_std</td>
<td>170</td>
</tr>
<tr>
<td>k_stop_gradient</td>
<td>170</td>
</tr>
<tr>
<td>k_sum</td>
<td>171</td>
</tr>
<tr>
<td>k_switch</td>
<td>171</td>
</tr>
<tr>
<td>k_tanh</td>
<td>172</td>
</tr>
<tr>
<td>k_tanh</td>
<td>172</td>
</tr>
<tr>
<td>k_temporal_padding</td>
<td>173</td>
</tr>
<tr>
<td>k_tile</td>
<td>174</td>
</tr>
<tr>
<td>k_to_dense</td>
<td>175</td>
</tr>
<tr>
<td>k_transpose</td>
<td>175</td>
</tr>
<tr>
<td>k_truncated_normal</td>
<td>176</td>
</tr>
<tr>
<td>k_update</td>
<td>177</td>
</tr>
<tr>
<td>k_update_add</td>
<td>177</td>
</tr>
<tr>
<td>k_update_sub</td>
<td>178</td>
</tr>
<tr>
<td>k_var</td>
<td>178</td>
</tr>
<tr>
<td>k_variable</td>
<td>179</td>
</tr>
<tr>
<td>k_zeros</td>
<td>180</td>
</tr>
<tr>
<td>k_zeros_like</td>
<td>180</td>
</tr>
<tr>
<td>layer_activation</td>
<td>181</td>
</tr>
<tr>
<td>layer_activation_elu</td>
<td>182</td>
</tr>
<tr>
<td>layer_activation_leaky_relu</td>
<td>183</td>
</tr>
<tr>
<td>layer_activation_parametric_relu</td>
<td>184</td>
</tr>
<tr>
<td>layer_activation_relu</td>
<td>185</td>
</tr>
<tr>
<td>layer_activation_selu</td>
<td>186</td>
</tr>
<tr>
<td>layer_activation_softmax</td>
<td>187</td>
</tr>
</tbody>
</table>
layer_activation_thresholded_relu .......................... 188
layer_activity_regularization .................................. 189
layer_add .......................................................... 190
layer_alpha_dropout ............................................. 190
layer_average ...................................................... 192
layer_average_pooling_1d ...................................... 192
layer_average_pooling_2d ...................................... 193
layer_average_pooling_3d ...................................... 195
layer_batch_normalization ..................................... 196
layer_concatenate ............................................... 198
layer_conv_1d .................................................... 198
layer_conv_2d .................................................... 200
layer_conv_2d_transpose ....................................... 203
layer_conv_3d .................................................... 205
layer_conv_3d_transpose ....................................... 207
layer_conv_lstm_2d ............................................. 209
layer_cropping_1d ............................................... 212
layer_cropping_2d ............................................... 213
layer_cropping_3d ............................................... 214
layer_cudnn_gru ................................................ 215
layer_cudnn_lstm .............................................. 217
layer_dense ....................................................... 218
layer_depthwise_conv_2d ...................................... 220
layer_dot .......................................................... 222
layer_dropout ..................................................... 223
layer_embedding .................................................. 224
layer_flatten ...................................................... 225
layer_gaussian_dropout ........................................ 226
layer_gaussian_noise ............................................ 227
layer_global_average_pooling_1d ............................ 228
layer_global_average_pooling_2d ............................ 229
layer_global_average_pooling_3d ............................ 230
layer_global_max_pooling_1d .................................. 231
layer_global_max_pooling_2d .................................. 232
layer_global_max_pooling_3d .................................. 233
layer_gru .......................................................... 234
layer_input ........................................................ 237
layer_lambda ....................................................... 238
layer_locally_connected_1d .................................... 239
layer_locally_connected_2d .................................... 240
layer_lstm ........................................................ 242
layer_masking ..................................................... 245
layer_maximum .................................................... 246
layer_max_pooling_1d .......................................... 247
layer_max_pooling_2d .......................................... 248
layer_max_pooling_3d .......................................... 249
layer_minimum ..................................................... 250
layer_multiply ..................................................... 251
layer_permute .......................................................... 251
layer_repeat_vector .................................................. 252
layer_reshape .......................................................... 253
layer_separable_conv_1d .............................................. 254
layer_separable_conv_2d .............................................. 256
layer_simple_rnn ...................................................... 259
layer_spatial_dropout_1d .............................................. 262
layer_spatial_dropout_2d .............................................. 263
layer_spatial_dropout_3d .............................................. 264
layer_subtract .......................................................... 265
layer_upsampling_1d ................................................... 265
layer_upsampling_2d ................................................... 266
layer_upsampling_3d ................................................... 267
layer_zero_padding_1d ................................................ 268
layer_zero_padding_2d ................................................ 269
layer_zero_padding_3d ................................................ 271
loss_mean_squared_error ............................................. 272
make_sampling_table .................................................. 273
metric_binary_accuracy .............................................. 274
model_to_json .......................................................... 277
model_to_yaml .......................................................... 277
multi_gpu_model ........................................................ 278
normalize ............................................................... 280
optimizer_adadelta .................................................... 281
optimizer_adagrad ..................................................... 281
optimizer_adam ........................................................ 282
optimizer_adamax ...................................................... 283
optimizer_nadam ........................................................ 284
optimizer_rmsprop ..................................................... 285
optimizer_sgd .......................................................... 285
pad_sequences .......................................................... 286
plot.keras_training_history ......................................... 287
pop_layer ............................................................... 288
predict.keras.engine.training.Model ................................. 288
predict_generator ...................................................... 289
predict_on_batch ....................................................... 290
predict_proba ........................................................... 291
regularizer_l1 .......................................................... 292
reset_states ............................................................ 292
save_model_hdf5 ....................................................... 293
save_model_weights_hdf5 ............................................. 294
save_text_tokenizer .................................................... 295
sequences_to_matrix .................................................. 296
serialize_model ........................................................ 296
skipgrams ............................................................... 297
summary.keras.engine.training.Model ............................... 298
texts_to_matrix ........................................................ 299
texts_to_sequences ..................................................... 300
Description

Keras is a high-level neural networks API, developed with a focus on enabling fast experimentation. Keras has the following key features:

Details

- Allows the same code to run on CPU or on GPU, seamlessly.
- User-friendly API which makes it easy to quickly prototype deep learning models.
- Built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.
- Supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, etc. This means that Keras is appropriate for building essentially any deep learning model, from a memory network to a neural Turing machine.
- Is capable of running on top of multiple back-ends including TensorFlow, CNTK, or Theano.

See the package website at https://keras.rstudio.com for complete documentation.

Author(s)

Maintainer: Daniel Falbel <daniel@rstudio.com> [contributor, copyright holder]

Authors:

- JJ Allaire [copyright holder]
- François Chollet [copyright holder]

Other contributors:

- RStudio [contributor, copyright holder, funder]
- Google [contributor, copyright holder, funder]
See Also

Useful links:

- [https://keras.rstudio.com](https://keras.rstudio.com)
- Report bugs at [https://github.com/rstudio/keras/issues](https://github.com/rstudio/keras/issues)

---

activation_relu  

Description

Activations functions can either be used through `layer_activation()`, or through the activation argument supported by all forward layers.

Usage

```r
activation_relu(x, alpha = 0, max_value = NULL, threshold = 0)
activation_elu(x, alpha = 1)
activation_selu(x)
activation_hard_sigmoid(x)
activation_linear(x)
activation_sigmoid(x)
activation_softmax(x, axis = -1)
activation_softplus(x)
activation_softsign(x)
activation_tanh(x)
activation_exponential(x)
```
application_densenet

Arguments

- x: Tensor
- alpha: Alpha value
- max_value: Max value
- threshold: Threshold value for thresholded activation.
- axis: Integer, axis along which the softmax normalization is applied

Details

- `activation_selu()` to be used together with the initialization "lecun_normal".
- `activation_selu()` to be used together with the dropout variant "AlphaDropout".

Value

Tensor with the same shape and dtype as x.

References

- `activation_selu()`: Self-Normalizing Neural Networks

application_densenet

Instantiates the DenseNet architecture.

Description

Instantiates the DenseNet architecture.

Usage

```python
application_densenet(blocks, include_top = TRUE, weights = "imagenet",
input_tensor = NULL, input_shape = NULL, pooling = NULL,
classes = 1000)

application_densenet121(include_top = TRUE, weights = "imagenet",
input_tensor = NULL, input_shape = NULL, pooling = NULL,
classes = 1000)

application_densenet169(include_top = TRUE, weights = "imagenet",
input_tensor = NULL, input_shape = NULL, pooling = NULL,
classes = 1000)

application_densenet201(include_top = TRUE, weights = "imagenet",
input_tensor = NULL, input_shape = NULL, pooling = NULL,
classes = 1000)

densenet_preprocess_input(x, data_format = NULL)
```
Arguments

blocks
numbers of building blocks for the four dense layers.
include_top
whether to include the fully-connected layer at the top of the network.
weights
one of NULL (random initialization), 'imagenet' (pre-training on ImageNet), or the path to the weights file to be loaded.
input_tensor
optional Keras tensor (i.e. output of layer_input()) to use as image input for the model.
input_shape
optional shape list, only to be specified if include_top is FALSE (otherwise the input shape has to be (224, 224, 3) (with channels_last data format) or (3, 224, 224) (with channels_first data format). It should have exactly 3 inputs channels.
pooling
optional pooling mode for feature extraction when include_top is FALSE. - NULL means that the output of the model will be the 4D tensor output of the last convolutional layer. - avg means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor. - max means that global max pooling will be applied.
classes
optional number of classes to classify images into, only to be specified if include_top is TRUE, and if no weights argument is specified.
x
a 3D or 4D array consists of RGB values within [0, 255].
data_format
data format of the image tensor.

Details

Optionally loads weights pre-trained on ImageNet. Note that when using TensorFlow, for best performance you should set image_data_format='channels_last' in your Keras config at ~/.keras/keras.json.
The model and the weights are compatible with TensorFlow, Theano, and CNTK. The data format convention used by the model is the one specified in your Keras config file.

Inception-ResNet v2 model, with weights trained on ImageNet

Description

Inception-ResNet v2 model, with weights trained on ImageNet

Usage

application_inception_resnet_v2(include_top = TRUE,
weights = "imagenet", input_tensor = NULL, input_shape = NULL,
pooling = NULL, classes = 1000)
inception_resnet_v2_preprocess_input(x)
## Arguments

**include_top** whether to include the fully-connected layer at the top of the network.

**weights** NULL (random initialization), imagenet (ImageNet weights), or the path to the weights file to be loaded.

**input_tensor** optional Keras tensor to use as image input for the model.

**input_shape** optional shape list, only to be specified if `include_top` is FALSE (otherwise the input shape has to be (299, 299, 3). It should have exactly 3 inputs channels, and width and height should be no smaller than 75. E.g. (150, 150, 3) would be one valid value.

**pooling** Optional pooling mode for feature extraction when include_top is FALSE.

- NULL means that the output of the model will be the 4D tensor output of the last convolutional layer.
- avg means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor.
- max means that global max pooling will be applied.

**classes** optional number of classes to classify images into, only to be specified if `include_top` is TRUE, and if no `weights` argument is specified.

**x** Input tensor for preprocessing

## Details

Do note that the input image format for this model is different than for the VGG16 and ResNet models (299x299 instead of 224x224).

The `inception_resnet_v2_preprocess_input()` function should be used for image preprocessing.

## Value

A Keras model instance.

## Reference


---

**Inception V3 model, with weights pre-trained on ImageNet.**

**Description**

Inception V3 model, with weights pre-trained on ImageNet.
Usage

```python
application_inception_v3(include_top = TRUE, weights = "imagenet",
input_tensor = NULL, input_shape = NULL, pooling = NULL,
classes = 1000)

inception_v3_preprocess_input(x)
```

Arguments

- `include_top`: whether to include the fully-connected layer at the top of the network.
- `weights`: NULL (random initialization), imagenet (ImageNet weights), or the path to the weights file to be loaded.
- `input_tensor`: optional Keras tensor to use as image input for the model.
- `input_shape`: optional shape list, only to be specified if `include_top` is FALSE (otherwise the input shape has to be HR99L R99L SI. It should have exactly 3 input channels, and width and height should be no smaller than 75. E.g. (150, 150, 3) would be one valid value.
- `pooling`: Optional pooling mode for feature extraction when `include_top` is FALSE.
  - NULL means that the output of the model will be the 4D tensor output of the last convolutional layer.
  - avg means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor.
  - max means that global max pooling will be applied.
- `classes`: optional number of classes to classify images into, only to be specified if `include_top` is TRUE, and if no `weights` argument is specified.
- `x`: Input tensor for preprocessing

Details

Do note that the input image format for this model is different than for the VGG16 and ResNet models (299x299 instead of 224x224).

The `inception_v3_preprocess_input()` function should be used for image preprocessing.

Value

A Keras model instance.

Reference

- Rethinking the Inception Architecture for Computer Vision
application_mobilenet  

---

**Application_mobilenet**  

**Description**  

MobileNet model architecture.

**Usage**

```ruby
application_mobilenet(input_shape = NULL, alpha = 1,
depth_multiplier = 1, dropout = 0.001, include_top = TRUE,
weights = "imagenet", input_tensor = NULL, pooling = NULL,
classes = 1000)

mobilenet_preprocess_input(x)

mobilenet_decode_predictions(preds, top = 5)

mobilenet_load_model_hdf5(filepath)
```

**Arguments**

- **input_shape**: optional shape list, only to be specified if include_top is FALSE (otherwise the input shape has to be (224, 224, 3) (with channels_last data format) or (3, 224, 224) (with channels_first data format). It should have exactly 3 inputs channels, and width and height should be no smaller than 32. E.g. (200, 200, 3) would be one valid value.

- **alpha**: controls the width of the network.
  - If alpha < 1.0, proportionally decreases the number of filters in each layer.
  - If alpha > 1.0, proportionally increases the number of filters in each layer.
  - If alpha = 1, default number of filters from the paper are used at each layer.

- **depth_multiplier**: depth multiplier for depthwise convolution (also called the resolution multiplier)

- **dropout**: dropout rate

- **include_top**: whether to include the fully-connected layer at the top of the network.

- **weights**: NULL (random initialization), imagenet (ImageNet weights), or the path to the weights file to be loaded.

- **input_tensor**: optional Keras tensor (i.e. output of layer_input()) to use as image input for the model.

- **pooling**: Optional pooling mode for feature extraction when include_top is FALSE. - NULL means that the output of the model will be the 4D tensor output of the last convolutional layer. - avg means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor. - max means that global max pooling will be applied.
application_mobilenet_v2

```
classes  optional number of classes to classify images into, only to be specified if include_top is TRUE, and if no weights argument is specified.
x        input tensor, 4D
preds    Tensor encoding a batch of predictions.
top      integer, how many top-guesses to return.
filepath File path
```

**Details**

The `mobilenet_preprocess_input()` function should be used for image preprocessing. To load a saved instance of a MobileNet model use the `mobilenet_load_model_hdf5()` function. To prepare image input for MobileNet use `mobilenet_preprocess_input()`. To decode predictions use `mobilenet_decode_predictions()`.

**Value**

`application_mobilenet()` and `mobilenet_load_model_hdf5()` return a Keras model instance. `mobilenet_preprocess_input()` returns image input suitable for feeding into a `mobilenet` model. `mobilenet_decode_predictions()` returns a list of data frames with variables `class_name`, `class_description`, and `score` (one data frame per sample in batch input).

**Reference**

- MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications.

---

```
application_mobilenet_v2

MobileNetV2 model architecture
```

**Description**

MobileNetV2 model architecture

**Usage**

```
application_mobilenet_v2(input_shape = NULL, alpha = 1, include_top = TRUE, weights = "imagenet", input_tensor = NULL, pooling = NULL, classes = 1000)

mobilenet_v2_preprocess_input(x)

mobilenet_v2_decode_predictions(preds, top = 5)

mobilenet_v2_load_model_hdf5(filepath)
```
Arguments

input_shape: optional shape list, only to be specified if include_top is FALSE (otherwise the input shape has to be (224, 224, 3) (with channels_last data format) or (3, 224, 224) (with channels_first data format). It should have exactly 3 inputs channels, and width and height should be no smaller than 32. E.g. (200, 200, 3) would be one valid value.

alpha: controls the width of the network.
  - If alpha < 1.0, proportionally decreases the number of filters in each layer.
  - If alpha > 1.0, proportionally increases the number of filters in each layer.
  - If alpha = 1, default number of filters from the paper are used at each layer.

include_top: whether to include the fully-connected layer at the top of the network.

weights: NULL (random initialization), imagenet (ImageNet weights), or the path to the weights file to be loaded.

input_tensor: optional Keras tensor (i.e. output of layer_input()) to use as image input for the model.

pooling: Optional pooling mode for feature extraction when include_top is FALSE. - NULL means that the output of the model will be the 4D tensor output of the last convolutional layer. - avg means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor. - max means that global max pooling will be applied.

classes: optional number of classes to classify images into, only to be specified if include_top is TRUE, and if no weights argument is specified.

x: input tensor, 4D

preds: Tensor encoding a batch of predictions.

top: integer, how many top-guesses to return.

filepath: File path

Value

application_mobilenet_v2() and mobilenet_v2_load_model_hdf5() return a Keras model instance. mobilenet_v2_preprocess_input() returns image input suitable for feeding into a mobilenet v2 model. mobilenet_v2_decode_predictions() returns a list of data frames with variables class_name, class_description, and score (one data frame per sample in batch input).

Reference

- MobileNetV2: Inverted Residuals and Linear Bottlenecks

See Also

application_mobilenet
**application_nasnet**

*Instantiates a NASNet model.*

**Description**

Note that only TensorFlow is supported for now, therefore it only works with the data format `image_data_format='channels_last'` in your Keras config at `~/.keras/keras.json`.

**Usage**

```python
application_nasnet(input_shape = NULL, penultimate_filters = 4032L,
num_blocks = 6L, stem_block_filters = 96L, skip_reduction = TRUE,
filter_multiplier = 2L, include_top = TRUE, weights = NULL,
input_tensor = NULL, pooling = NULL, classes = 1000,
default_size = NULL)
```

```python
application_nasnetlarge(input_shape = NULL, include_top = TRUE,
weights = NULL, input_tensor = NULL, pooling = NULL,
classes = 1000)
```

```python
application_nasnetmobile(input_shape = NULL, include_top = TRUE,
weights = NULL, input_tensor = NULL, pooling = NULL,
classes = 1000)
```

```python
nasnet_preprocess_input(x)
```

**Arguments**

- **input_shape**
  Optional shape list, the input shape is by default `(331, 331, 3)` for NASNet-Large and `(224, 224, 3)` for NASNetMobile. It should have exactly 3 inputs channels, and width and height should be no smaller than 32. E.g. `(224, 224, 3)` would be one valid value.

- **penultimate_filters**
  Number of filters in the penultimate layer. NASNet models use the notation `NASNet (N @ P)`, where: - N is the number of blocks - P is the number of penultimate filters

- **num_blocks**
  Number of repeated blocks of the NASNet model. NASNet models use the notation `NASNet (N @ P)`, where: - N is the number of blocks - P is the number of penultimate filters

- **stem_block_filters**
  Number of filters in the initial stem block

- **skip_reduction**
  Whether to skip the reduction step at the tail end of the network. Set to `FALSE` for CIFAR models.

- **filter_multiplier**
  Controls the width of the network.
• If \texttt{filter\_multiplier} < 1.0, proportionally decreases the number of filters in each layer.
• If \texttt{filter\_multiplier} > 1.0, proportionally increases the number of filters in each layer. - If \texttt{filter\_multiplier} = 1, default number of filters from the paper are used at each layer.

\textbf{include\_top} \quad \text{Whether to include the fully-connected layer at the top of the network.}

\textbf{weights} \quad \text{NULL (random initialization) or imagenet (ImageNet weights)}

\textbf{input\_tensor} \quad \text{Optional Keras tensor (i.e. output of \texttt{layer\_input()}) to use as image input for the model.}

\textbf{pooling} \quad \text{Optional pooling mode for feature extraction when \texttt{include\_top} is FALSE. - null means that the output of the model will be the 4D tensor output of the last convolutional layer. - avg means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor. - max means that global max pooling will be applied.}

\textbf{classes} \quad \text{Optional number of classes to classify images into, only to be specified if \texttt{include\_top} is TRUE, and if no \texttt{weights} argument is specified.}

\textbf{default\_size} \quad \text{Specifies the default image size of the model}

\textbf{x} \quad \text{a 4D array consists of RGB values within [0, 255].}

---

\textbf{Description}

ResNet50 model for Keras.

\textbf{Usage}

\texttt{application\_resnet50(include\_top = TRUE, weights = "imagenet", input\_tensor = NULL, input\_shape = NULL, pooling = NULL, classes = 1000)}

\textbf{Arguments}

\textbf{include\_top} \quad \text{whether to include the fully-connected layer at the top of the network.}

\textbf{weights} \quad \text{NULL (random initialization), imagenet (ImageNet weights), or the path to the weights file to be loaded.}

\textbf{input\_tensor} \quad \text{optional Keras tensor to use as image input for the model.}

\textbf{input\_shape} \quad \text{optional shape list, only to be specified if \texttt{include\_top} is FALSE (otherwise the input shape has to be (224, 224, 3). It should have exactly 3 inputs channels, and width and height should be no smaller than 32. E.g. (200, 200, 3) would be one valid value.}

\textbf{pooling} \quad \text{Optional pooling mode for feature extraction when \texttt{include\_top} is FALSE.
• NULL means that the output of the model will be the 4D tensor output of the last convolutional layer.
• avg means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor.
• max means that global max pooling will be applied.

classes
optional number of classes to classify images into, only to be specified if `include_top` is TRUE, and if no `weights` argument is specified.

Details
Optionally loads weights pre-trained on ImageNet.
The `imagenet_preprocess_input()` function should be used for image preprocessing.

Value
A Keras model instance.

Reference
- Deep Residual Learning for Image Recognition

Examples

```r
## Not run:
library(keras)

# instantiate the model
model <- application_resnet50(weights = 'imagenet')

# load the image
ing_path <- "elephant.jpg"
ing <- image_load(img_path, target_size = c(224,224))
x <- image_to_array(img)

# ensure we have a 4d tensor with single element in the batch dimension,
# the preprocess the input for prediction using resnet50
x <- array_reshape(x, c(1, dim(x)))
x <- imagenet_preprocess_input(x)

# make predictions then decode and print them
preds <- model %>% predict(x)
imagenet_decode_predictions(preds, top = 3)[[1]]

## End(Not run)
```
Description

VGG16 and VGG19 models for Keras.

Usage

application_vgg16(include_top = TRUE, weights = "imagenet",
input_tensor = NULL, input_shape = NULL, pooling = NULL,
classes = 1000)

application_vgg19(include_top = TRUE, weights = "imagenet",
input_tensor = NULL, input_shape = NULL, pooling = NULL,
classes = 1000)

Arguments

include_top whether to include the 3 fully-connected layers at the top of the network.
weights NULL (random initialization), imagenet (ImageNet weights), or the path to the
input_tensor optional Keras tensor to use as image input for the model.
input_shape optional shape list, only to be specified if include_top is FALSE (otherwise the
input_shape has to be (224, 224, 3) It should have exactly 3 inputs channels,
and width and height should be no smaller than 32. E.g. (200, 200, 3) would
be one valid value.
pooling Optional pooling mode for feature extraction when include_top is FALSE.
• NULL means that the output of the model will be the 4D tensor output of the
last convolutional layer.
• avg means that global average pooling will be applied to the output of the
last convolutional layer, and thus the output of the model will be a 2D ten-
sor.
• max means that global max pooling will be applied.
classes optional number of classes to classify images into, only to be specified if include_top is TRUE, and if no weights argument is specified.

Details

Optionally loads weights pre-trained on ImageNet.
The imagenet_preprocess_input() function should be used for image preprocessing.

Value

Keras model instance.
Reference

- Very Deep Convolutional Networks for Large-Scale Image Recognition

Examples

```r
# Not run:
library(keras)

model <- application_xception(weights = 'imagenet', include_top = FALSE)

img_path <- "elephant.jpg"
img <- image_load(img_path, target_size = c(224,224))
x <- image_to_array(img)
x <- array_reshape(x, c(1, dim(x)))
x <- imagenet_preprocess_input(x)

features <- model %>% predict(x)

# End(Not run)
```

---

table

|          | Description
|----------|-------------
|          | Xception V1 model for Keras.

Description

Xception V1 model for Keras.

Usage

```r
application_xception(include_top = TRUE, weights = "imagenet",
                     input_tensor = NULL, input_shape = NULL, pooling = NULL,
                     classes = 1000)

xception_preprocess_input(x)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>include_top</td>
<td>whether to include the fully-connected layer at the top of the network.</td>
</tr>
<tr>
<td>weights</td>
<td>NULL (random initialization), imagenet (ImageNet weights), or the path to the weights file to be loaded.</td>
</tr>
<tr>
<td>input_tensor</td>
<td>optional Keras tensor to use as image input for the model.</td>
</tr>
<tr>
<td>input_shape</td>
<td>optional shape list, only to be specified if include_top is FALSE (otherwise the input shape has to be (299, 299, 3). It should have exactly 3 inputs channels, and width and height should be no smaller than 75. E.g. (150, 150, 3) would be one valid value.</td>
</tr>
<tr>
<td>pooling</td>
<td>Optional pooling mode for feature extraction when include_top is FALSE.</td>
</tr>
</tbody>
</table>
• NULL means that the output of the model will be the 4D tensor output of the last convolutional layer.
• avg means that global average pooling will be applied to the output of the last convolutional layer, and thus the output of the model will be a 2D tensor.
• max means that global max pooling will be applied.

classes

optional number of classes to classify images into, only to be specified if include_top is TRUE, and if no weights argument is specified.

x

Input tensor for preprocessing

Details

On ImageNet, this model gets to a top-1 validation accuracy of 0.790 and a top-5 validation accuracy of 0.945.

Do note that the input image format for this model is different than for the VGG16 and ResNet models (299x299 instead of 224x224).

The xception_preprocess_input() function should be used for image preprocessing.

This application is only available when using the TensorFlow back-end.

Value

A Keras model instance.

Reference

• Xception: Deep Learning with Depthwise Separable Convolutions

backend

Keras backend tensor engine

Description

Obtain a reference to the keras.backend Python module used to implement tensor operations.

Usage

backend(convert = TRUE)

Arguments

convert

TRUE to automatically convert Python objects to their R equivalent. If you pass FALSE you can do manual conversion using the py_to_r() function.

Value

Reference to Keras backend python module.
Note
See the documentation here https://keras.io/backend/ for additional details on the available functions.

---

**bidirectional**  
**Bidirectional wrapper for RNNs.**

**Description**
Bidirectional wrapper for RNNs.

**Usage**

```
bidirectional(object, layer, merge_mode = "concat", input_shape = NULL,  
batch_input_shape = NULL, batch_size = NULL, dtype = NULL,  
name = NULL, trainable = NULL, weights = NULL)
```

**Arguments**

- **object**: Model or layer object.
- **layer**: Recurrent instance.
- **merge_mode**: Mode by which outputs of the forward and backward RNNs will be combined. One of 'sum', 'mul', 'concat', 'ave', NULL. If NULL, the outputs will not be combined, they will be returned as a list.
- **input_shape**: Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.
- **batch_input_shape**: Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors.
- **batch_size**: Fixed batch size for layer.
- **dtype**: The data type expected by the input, as a string (float32, float64, int32...)
- **name**: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- **trainable**: Whether the layer weights will be updated during training.
- **weights**: Initial weights for layer.

**See Also**

Other layer wrappers: **time_distributed**
**callback_csv_logger**

Callback that streams epoch results to a csv file

**Description**

Supports all values that can be represented as a string

**Usage**

```python
callback_csv_logger(filename, separator = ""," append = FALSE)
```

**Arguments**

- `filename` : filename of the csv file, e.g. 'run/log.csv'.
- `separator` : string used to separate elements in the csv file.

**See Also**

Other callbacks: `callback_early_stopping`, `callback_lambda`, `callback_learning_rate_scheduler`, `callback_model_checkpoint`, `callback_progbar_logger`, `callback_reduce_lr_on_plateau`, `callback_remote_monitor`, `callback_tensorboard`, `callback_terminate_on_naan`

**callback_early_stopping**

Stop training when a monitored quantity has stopped improving.

**Description**

Stop training when a monitored quantity has stopped improving.

**Usage**

```python
callback_early_stopping(monitor = "val_loss", min_delta = 0,
patience = 0, verbose = 0, mode = c("auto", "min", "max"),
baseline = NULL, restore_best_weights = FALSE)
```
callback_lambda

Arguments

- **monitor**: quantity to be monitored.
- **min_delta**: minimum change in the monitored quantity to qualify as an improvement, i.e. an absolute change of less than min_delta, will count as no improvement.
- **patience**: number of epochs with no improvement after which training will be stopped.
- **verbose**: verbosity mode, 0 or 1.
- **mode**: one of "auto", "min", "max". In min mode, training will stop when the quantity monitored has stopped decreasing; in max mode it will stop when the quantity monitored has stopped increasing; in auto mode, the direction is automatically inferred from the name of the monitored quantity.
- **baseline**: Baseline value for the monitored quantity to reach. Training will stop if the model doesn’t show improvement over the baseline.
- **restore_best_weights**: Whether to restore model weights from the epoch with the best value of the monitored quantity. If FALSE, the model weights obtained at the last step of training are used.

See Also

Other callbacks: callback_csv_logger, callback_lambda, callback_learning_rate_scheduler, callback_model_checkpoint, callback_progbar_logger, callback_reduce_lr_on_plateau, callback_remote_monitor, callback_tensorboard, callback_terminate_on_naan

---

callback_lambda Create a custom callback

Description

This callback is constructed with anonymous functions that will be called at the appropriate time. Note that the callbacks expects positional arguments, as:

- **on_epoch_begin** and **on_epoch_end** expect two positional arguments: epoch, logs
- **on_batch_begin** and **on_batch_end** expect two positional arguments: batch, logs
- **on_train_begin** and **on_train_end** expect one positional argument: logs

Usage

```python
callback_lambda(on_epoch_begin = NULL, on_epoch_end = NULL,
on_batch_begin = NULL, on_batch_end = NULL, on_train_begin = NULL,
on_train_end = NULL)
```
callback_learning_rate_scheduler

Arguments

- on_epoch_begin called at the beginning of every epoch.
- on_epoch_end called at the end of every epoch.
- on_batch_begin called at the beginning of every batch.
- on_batch_end called at the end of every batch.
- on_train_begin called at the beginning of model training.
- on_train_end called at the end of model training.

See Also

Other callbacks: callback_csv_logger, callback_early_stopping, callback_learning_rate_scheduler, callback_model_checkpoint, callback_progbar_logger, callback_reduce_lr_on_plateau, callback_remote_monitor, callback_tensorboard, callback_terminate_on_naan

callback_learning_rate_scheduler

Learning rate scheduler.

Description

Learning rate scheduler.

Usage

callback_learning_rate_scheduler(schedule)

Arguments

- schedule a function that takes an epoch index as input (integer, indexed from 0) and current learning rate and returns a new learning rate as output (float).

See Also

Other callbacks: callback_csv_logger, callback_early_stopping, callback_lambda, callback_model_checkpoint, callback_progbar_logger, callback_reduce_lr_on_plateau, callback_remote_monitor, callback_tensorboard, callback_terminate_on_naan
callback_model_checkpoint

Save the model after every epoch.

Description

filepath can contain named formatting options, which will be filled the value of epoch and keys in logs (passed in on_epoch_end). For example: if filepath is weights.{epoch:02d}-{val_loss:.2f}.hdf5, then the model checkpoints will be saved with the epoch number and the validation loss in the filename.

Usage

```
callback_model_checkpoint(filepath, monitor = "val_loss", verbose = 0,
    save_best_only = false, save_weights_only = false, mode = c("auto",
    "min", "max"), period = 1)
```

Arguments

- `filepath` string, path to save the model file.
- `monitor` quantity to monitor.
- `verbose` verbosity mode, 0 or 1.
- `save_best_only` if `save_best_only`=TRUE, the latest best model according to the quantity monitored will not be overwritten.
- `save_weights_only` if TRUE, then only the model's weights will be saved (save_model_weights_hdf5(filepath)), else the full model is saved (save_model_hdf5(filepath)).
- `mode` one of "auto", "min", "max". If `save_best_only`=TRUE, the decision to overwrite the current save file is made based on either the maximization or the minimization of the monitored quantity. For val_acc, this should be max, for val_loss this should be min, etc. In auto mode, the direction is automatically inferred from the name of the monitored quantity.
- `period` Interval (number of epochs) between checkpoints.

For example

```
if filepath is weights.{epoch:02d}-{val_loss:.2f}.hdf5., then the model checkpoints will be saved with the epoch number and the validation loss in the filename.
```

See Also

Other callbacks: callback_csv_logger, callback_early_stopping, callback_lambda, callback_learning_rate_scheduler, callback_progbar_logger, callback_reduce_lr_on_plateau, callback_remote_monitor, callback_tensorboard, callback_terminate_on_naan
callback_progbar_logger

Callback that prints metrics to stdout.

Description

Callback that prints metrics to stdout.

Usage

```
callback_progbar_logger(count_mode = "samples",
                        stateful_metrics = NULL)
```

Arguments

- `count_mode` One of "steps" or "samples". Whether the progress bar should count samples seen or steps (batches) seen.
- `stateful_metrics` List of metric names that should not be averaged over an epoch. Metrics in this list will be logged as-is in `on_epoch_end`. All others will be averaged in `on_epoch_end`.

See Also

Other callbacks: callback_csv_logger, callback_early_stopping, callback_lambda, callback_learning_rate_scheduler, callback_model_checkpoint, callback_reduce_lr_on_plateau, callback_remote_monitor, callback_tensorboard, callback_terminate_on_naan

callback_reduce_lr_on_plateau

Reduce learning rate when a metric has stopped improving.

Description

Models often benefit from reducing the learning rate by a factor of 2-10 once learning stagnates. This callback monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

Usage

```
callback_reduce_lr_on_plateau(monitor = "val_loss", factor = 0.1,
                              patience = 10, verbose = 0, mode = c("auto", "min", "max"),
                              min_delta = 1e-04, cooldown = 0, min_lr = 0)
```
**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>monitor</td>
<td>quantity to be monitored.</td>
</tr>
<tr>
<td>factor</td>
<td>factor by which the learning rate will be reduced. new_lr = lr • factor</td>
</tr>
<tr>
<td>patience</td>
<td>number of epochs with no improvement after which learning rate will be reduced.</td>
</tr>
<tr>
<td>verbose</td>
<td>int. 0: quiet, 1: update messages.</td>
</tr>
<tr>
<td>mode</td>
<td>one of &quot;auto&quot;, &quot;min&quot;, &quot;max&quot;. In min mode, lr will be reduced when the quantity monitored has stopped decreasing; in max mode it will be reduced when the quantity monitored has stopped increasing; in auto mode, the direction is automatically inferred from the name of the monitored quantity.</td>
</tr>
<tr>
<td>min_delta</td>
<td>threshold for measuring the new optimum, to only focus on significant changes.</td>
</tr>
<tr>
<td>cooldown</td>
<td>number of epochs to wait before resuming normal operation after lr has been reduced.</td>
</tr>
<tr>
<td>min_lr</td>
<td>lower bound on the learning rate.</td>
</tr>
</tbody>
</table>

**See Also**

Other callbacks: callback_csv_logger, callback_early_stopping, callback_lambda, callback_learning_rate_scheduler, callback_model_checkpoint, callback_progbar_logger, callback_remote_monitor, callback_tensorboard, callback_terminate_on_naan

---

**Callback used to stream events to a server.**

**Description**

Callback used to stream events to a server.

**Usage**

```python
callback_remote_monitor(root = "http://localhost:9000",
                        path = "/publish/epoch/end/", field = "data", headers = NULL,
                        send_as_json = FALSE)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>root</td>
<td>root url of the target server.</td>
</tr>
<tr>
<td>path</td>
<td>path relative to root to which the events will be sent.</td>
</tr>
<tr>
<td>field</td>
<td>JSON field under which the data will be stored.</td>
</tr>
<tr>
<td>headers</td>
<td>Optional named list of custom HTTP headers. Defaults to: list(Accept = &quot;application/json&quot;, Content-Type = &quot;application/json&quot;)</td>
</tr>
<tr>
<td>send_as_json</td>
<td>Whether the request should be sent as application/json.</td>
</tr>
</tbody>
</table>
**callback_tensorboard**

**Details**

Events are sent to root + '/publish/epoch/end/' by default. Calls are HTTP POST, with a data argument which is a JSON-encoded dictionary of event data. If send_as_json is set to True, the content type of the request will be application/json. Otherwise the serialized JSON will be send within a form.

**See Also**

Other callbacks: callback_csv_logger, callback_early_stopping, callback_lambda, callback_learning_rate_scheduler, callback_model_checkpoint, callback_progbar_logger, callback_reduce_lr_on_plateau, callback_tensorboard, callback_terminate_on_naan

---

**callback_tensorboard**  
*TensorBoard basic visualizations*

**Description**

This callback writes a log for TensorBoard, which allows you to visualize dynamic graphs of your training and test metrics, as well as activation histograms for the different layers in your model.

**Usage**

```python
callback_tensorboard(log_dir = NULL, histogram_freq = 0,
batch_size = 32, write_graph = TRUE, write_grads = FALSE,
write_images = FALSE, embeddings_freq = 0,
embeddings_layer_names = NULL, embeddings_metadata = NULL,
embeddings_data = NULL, update_freq = "epoch")
```

**Arguments**

- `log_dir`  
  The path of the directory where to save the log files to be parsed by Tensorboard. The default is NULL, which will use the active run directory (if available) and otherwise will use "logs".

- `histogram_freq`  
  Frequency (in epochs) at which to compute activation histograms for the layers of the model. If set to 0, histograms won’t be computed.

- `batch_size`  
  Size of batch of inputs to feed to the network for histograms computation.

- `write_graph`  
  Whether to visualize the graph in Tensorboard. The log file can become quite large when write_graph is set to TRUE.

- `write_grads`  
  Whether to visualize gradient histograms in TensorBoard. histogram_freq must be greater than 0.

- `write_images`  
  Whether to write model weights to visualize as image in Tensorboard.

- `embeddings_freq`  
  Frequency (in epochs) at which selected embedding layers will be saved.

- `embeddings_layer_names`  
  A list of names of layers to keep eye on. If NULL or empty list all the embedding layers will be watched.
callback_terminate_on_naan

**Description**

Callback that terminates training when a NaN loss is encountered.

**Usage**

```python
callback_terminate_on_naan()
```

**See Also**

Other callbacks: `callback_csv_logger`, `callback_early_stopping`, `callback_lambda`, `callback_learning_rate_scheduler`, `callback_model_checkpoint`, `callback_progbar_logger`, `callback_reduce_lr_on_plateau`, `callback_remote_monitor`, `callback_tensorboard`
**clone_model**

*Clone a model instance.*

**Description**

Model cloning is similar to calling a model on new inputs, except that it creates new layers (and thus new weights) instead of sharing the weights of the existing layers.

**Usage**

```python
clone_model(model, input_tensors = NULL)
```

**Arguments**

- **model**
  - Instance of Keras model (could be a functional model or a Sequential model).
- **input_tensors**
  - Optional list of input tensors to build the model upon. If not provided, placeholders will be created.

---

**compile.keras.engine.training.Model**

*Configure a Keras model for training*

**Description**

Configure a Keras model for training

**Usage**

```r
## S3 method for class 'keras.engine.training.Model'
compile(object, optimizer, loss,
metrics = NULL, loss_weights = NULL, sample_weight_mode = NULL,
weighted_metrics = NULL, target_tensors = NULL, ...)
```

**Arguments**

- **object**
  - Model object to compile.
- **optimizer**
  - Name of optimizer or optimizer instance.
- **loss**
  - Name of objective function or objective function. If the model has multiple outputs, you can use a different loss on each output by passing a dictionary or a list of objectives. The loss value that will be minimized by the model will then be the sum of all individual losses.
- **metrics**
  - List of metrics to be evaluated by the model during training and testing. Typically you will use metrics='accuracy'. To specify different metrics for different outputs of a multi-output model, you could also pass a named list such as metrics=list(output_a = 'accuracy').
loss_weights (Optional list specifying scalar coefficients to weight the loss contributions of different model outputs. The loss value that will be minimized by the model will then be the weighted sum of all individual losses, weighted by the loss_weights coefficients.

sample_weight_mode
If you need to do timestep-wise sample weighting (2D weights), set this to "temporal". NULL defaults to sample-wise weights (1D). If the model has multiple outputs, you can use a different sample_weight_mode on each output by passing a list of modes.

weighted_metrics
List of metrics to be evaluated and weighted by sample_weight or class_weight during training and testing.

target_tensors
By default, Keras will create a placeholder for the model’s target, which will be fed with the target data during training. If instead you would like to use your own target tensor (in turn, Keras will not expect external data for these targets at training time), you can specify them via the target_tensors argument. It should be a single tensor (for a single-output sequential model).

See Also

---

### constraints

<table>
<thead>
<tr>
<th>Weight constraints</th>
</tr>
</thead>
</table>

**Description**

Functions that impose constraints on weight values.

**Usage**

```python
constraint_maxnorm(max_value = 2, axis = 0)

constraint_nonneg()

constraint_unitnorm(axis = 0)

constraint_minmaxnorm(min_value = 0, max_value = 1, rate = 1, axis = 0)
```
Arguments

max_value  The maximum norm for the incoming weights.

axis   The axis along which to calculate weight norms. For instance, in a dense layer the weight matrix has shape input_dim, output_dim, set axis to 0 to constrain each weight vector of length input_dim. In a convolution 2D layer with dim_ordering="tf", the weight tensor has shape rows, cols, input_depth, output_depth, set axis to c(0, 1, 2) to constrain the weights of each filter tensor of size rows, cols, input_depth.

min_value  The minimum norm for the incoming weights.

rate   The rate for enforcing the constraint: weights will be rescaled to yield (1 - rate) * norm + rate * norm.clip(low, high). Effectively, this means that rate=1.0 stands for strict enforcement of the constraint, while rate<1.0 means that weights will be rescaled at each step to slowly move towards a value inside the desired interval.

Details

- constraint_maxnorm() constrains the weights incident to each hidden unit to have a norm less than or equal to a desired value.
- constraint_nonneg() constrains the weights to be non-negative
- constraint_unitnorm() constrains the weights incident to each hidden unit to have unit norm.
- constraint_minmaxnorm() constrains the weights incident to each hidden unit to have the norm between a lower bound and an upper bound.

Custom constraints

You can implement your own constraint functions in R. A custom constraint is an R function that takes weights (w) as input and returns modified weights. Note that keras backend() tensor functions (e.g. k_greater_equal()) should be used in the implementation of custom constraints. For example:

```r
nonneg_constraint <- function(w) {
  w * k_cast(k_greater_equal(w, 0), k_floatx())
}
```

```r
layer_dense(units = 32, input_shape = c(784),
             kernel_constraint = nonneg_constraint)
```

Note that models which use custom constraints cannot be serialized using save_model_hdf5(). Rather, the weights of the model should be saved and restored using save_model_weights_hdf5().

See Also

KerasConstraint
count_params

Count the total number of scalars composing the weights.

**Description**

Count the total number of scalars composing the weights.

**Usage**

```
count_params(object)
```

**Arguments**

- **object**: Layer or model object

**Value**

An integer count

**See Also**

Other layer methods: `get_config, get_input_at, get_weights, reset_states`

---

create_layer

Create a Keras Layer

**Description**

Create a Keras Layer

**Usage**

```
create_layer(layer_class, object, args = list())
```

**Arguments**

- **layer_class**: Python layer class or R6 class of type KerasLayer
- **object**: Object to compose layer with. This is either a `keras_model_sequential()` to add the layer to, or another Layer which this layer will call.
- **args**: List of arguments to layer constructor function

**Value**

A Keras layer
**create_wrapper**

Create a Keras Wrapper

**Description**

Create a Keras Wrapper

**Usage**

```r
create_wrapper(wrapper_class, object, args = list())
```

**Arguments**

- `wrapper_class` : R6 class of type KerasWrapper
- `object` : Object to compose layer with. This is either a `keras_model_sequential()` to add the layer to, or another Layer which this layer will call.
- `args` : List of arguments to layer constructor function

**Value**

A Keras wrapper

**Note**

The `object` parameter can be missing, in which case the layer is created without a connection to an existing graph.

---

**dataset_boston_housing**

*Boston housing price regression dataset*

**Description**

Dataset taken from the StatLib library which is maintained at Carnegie Mellon University.

**Usage**

```r
dataset_boston_housing(path = "boston_housing.npz", test_split = 0.2, seed = 113L)
```
Arguments

- **path**: Path where to cache the dataset locally (relative to ~/.keras/datasets).
- **test_split**: Fraction of the data to reserve as test set.
- **seed**: Random seed for shuffling the data before computing the test split.

Value

Lists of training and test data: train$x, train$y, test$x, test$y.

Samples contain 13 attributes of houses at different locations around the Boston suburbs in the late 1970s. Targets are the median values of the houses at a location (in k$).

See Also

Other datasets: dataset_cifar100, dataset_cifar10, dataset_fashion_mnist, dataset_imdb, dataset_mnist, dataset_reuters

---

**dataset_cifar10**  
*CIFAR10 small image classification*

Description

Dataset of 50,000 32x32 color training images, labeled over 10 categories, and 10,000 test images.

Usage

```r
dataset_cifar10()
```

Value

Lists of training and test data: train$x, train$y, test$x, test$y.

The x data is an array of RGB image data with shape (num_samples, 3, 32, 32).

The y data is an array of category labels (integers in range 0-9) with shape (num_samples).

See Also

Other datasets: dataset_boston_housing, dataset_cifar100, dataset_fashion_mnist, dataset_imdb, dataset_mnist, dataset_reuters
**dataset_cifar100**  
*CIFAR100 small image classification*

**Description**

Dataset of 50,000 32x32 color training images, labeled over 100 categories, and 10,000 test images.

**Usage**

```r
dataset_cifar100(label_mode = c("fine", "coarse"))
```

**Arguments**

- `label_mode` one of "fine", "coarse".

**Value**

Lists of training and test data: `train$x`, `train$y`, `test$x`, `test$y`.

The `x` data is an array of RGB image data with shape `(num_samples, 3, 32, 32)`.

The `y` data is an array of category labels with shape `(num_samples)`.

**See Also**

Other datasets: `dataset_boston_housing`, `dataset_cifar10`, `dataset_fashion_mnist`, `dataset_imdb`, `dataset_mnist`, `dataset_reuters`

---

**dataset_fashion_mnist**  
*Fashion-MNIST database of fashion articles*

**Description**

Dataset of 60,000 28x28 grayscale images of the 10 fashion article classes, along with a test set of 10,000 images. This dataset can be used as a drop-in replacement for MNIST. The class labels are encoded as integers from 0-9 which correspond to T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt,

**Usage**

```r
dataset_fashion_mnist()
```
Details

Dataset of 60,000 28x28 grayscale images of 10 fashion categories, along with a test set of 10,000 images. This dataset can be used as a drop-in replacement for MNIST. The class labels are:

- 0 - T-shirt/top
- 1 - Trouser
- 2 - Pullover
- 3 - Dress
- 4 - Coat
- 5 - Sandal
- 6 - Shirt
- 7 - Sneaker
- 8 - Bag
- 9 - Ankle boot

Value

Lists of training and test data: \texttt{train\_x, train\_y, test\_x, test\_y}, where \texttt{x} is an array of grayscale image data with shape \texttt{(num\_samples, 28, 28)} and \texttt{y} is an array of article labels (integers in range 0-9) with shape \texttt{(num\_samples)}.

See Also

Other datasets: \texttt{dataset\_boston\_housing, dataset\_cifar100, dataset\_cifar10, dataset\_imdb, dataset\_mnist, dataset\_reuters}

dataset\_imdb \quad \textit{IMDB Movie reviews sentiment classification}

Description

Dataset of 25,000 movies reviews from IMDB, labeled by sentiment (positive/negative). Reviews have been preprocessed, and each review is encoded as a sequence of word indexes (integers). For convenience, words are indexed by overall frequency in the dataset, so that for instance the integer "3" encodes the 3rd most frequent word in the data. This allows for quick filtering operations such as: "only consider the top 10,000 most common words, but eliminate the top 20 most common words".

Usage

dataset\_imdb(path = "imdb.npz", num\_words = NULL, skip\_top = 0L, maxlen = NULL, seed = 113L, start\_char = 1L, oov\_char = 2L, index\_from = 3L)

dataset\_imdb\_word\_index(path = "imdb\_word\_index.json")
**Arguments**

- **path**
  Where to cache the data (relative to `~/.keras/dataset`).

- **num_words**
  Max number of words to include. Words are ranked by how often they occur (in the training set) and only the most frequent words are kept.

- **skip_top**
  Skip the top N most frequently occurring words (which may not be informative).

- **maxlen**
  Sequences longer than this will be filtered out.

- **seed**
  Random seed for sample shuffling.

- **start_char**
  The start of a sequence will be marked with this character. Set to 1 because 0 is usually the padding character.

- **oov_char**
  Words that were cut out because of the num_words or skip_top limit will be replaced with this character.

- **index_from**
  Index actual words with this index and higher.

**Details**

As a convention, "0" does not stand for a specific word, but instead is used to encode any unknown word.

**Value**

Lists of training and test data: `train$x`, `train$y`, `test$x`, `test$y`.

The `x` data includes integer sequences. If the `num_words` argument was specific, the maximum possible index value is `num_words-1`. If the `maxlen` argument was specified, the largest possible sequence length is `maxlen`.

The `y` data includes a set of integer labels (0 or 1).

The `dataset_imdb_word_index()` function returns a list where the names are words and the values are integer.

**See Also**

Other datasets: `dataset_boston_housing`, `dataset_cifar100`, `dataset_cifar10`, `dataset_fashion_mnist`, `dataset_mnist`, `dataset_reuters`

---

**dataset_mnist**

*MNIST database of handwritten digits*

**Description**

Dataset of 60,000 28x28 grayscale images of the 10 digits, along with a test set of 10,000 images.

**Usage**

```
dataset_mnist(path = "mnist.npz")
```
Arguments

path  Path where to cache the dataset locally (relative to ~/.keras/datasets).

Value

Lists of training and test data: train$x, train$y, test$x, test$y, where x is an array of grayscale image data with shape (num_samples, 28, 28) and y is an array of digit labels (integers in range 0-9) with shape (num_samples).

See Also

Other datasets: dataset_boston_housing, dataset_cifar10, dataset_cifar100, dataset_fashion_mnist, dataset_imdb, dataset_reuters

dataset_reuters  Reuters newswire topics classification

Description

Dataset of 11,228 newswires from Reuters, labeled over 46 topics. As with dataset_imdb, each wire is encoded as a sequence of word indexes (same conventions).

Usage

dataset_reuters(path = "reuters.npz", num_words = NULL,
skip_top = 0L, maxlen = NULL, test_split = 0.2, seed = 113L,
start_char = 1L, oov_char = 2L, index_from = 3L)

dataset_reuters_word_index(path = "reuters_word_index.pkl")

Arguments

path  Where to cache the data (relative to ~/.keras/dataset).
num_words  Max number of words to include. Words are ranked by how often they occur (in the training set) and only the most frequent words are kept
skip_top  Skip the top N most frequently occurring words (which may not be informative).
maxlen  Truncate sequences after this length.
test_split  Fraction of the dataset to be used as test data.
seed  Random seed for sample shuffling.
start_char  The start of a sequence will be marked with this character. Set to 1 because 0 is usually the padding character.
oov_char  words that were cut out because of the num_words or skip_top limit will be replaced with this character.
index_from  index actual words with this index and higher.
Value

Lists of training and test data: `train$x`, `train$y`, `test$x`, `test$y` with same format as `dataset_imdb()`. The `dataset_reuters_word_index()` function returns a list where the names are words and the values are integer. E.g. `word_index["giraffe"]` might return 1234.

See Also

Other datasets: `dataset_boston_housing`, `dataset_cifar100`, `dataset_cifar10`, `dataset_fashion_mnist`, `dataset_imdb`, `dataset_mnist`
evaluate_generator

Value

Named list of model test loss (or losses for models with multiple outputs) and model metrics.

See Also


evaluate_generator  Evaluates the model on a data generator.

Description

The generator should return the same kind of data as accepted by test_on_batch().

Usage

evaluate_generator(object, generator, steps, max_queue_size = 10, workers = 1)

Arguments

object  Model object to evaluate
generator  Generator yielding lists (inputs, targets) or (inputs, targets, sample_weights)
steps  Total number of steps (batches of samples) to yield from generator before stopping.
max_queue_size  Maximum size for the generator queue. If unspecified, max_queue_size will default to 10.
workers  Maximum number of threads to use for parallel processing. Note that parallel processing will only be performed for native Keras generators (e.g. flow_images_from_directory()) as R based generators must run on the main thread.

Value

Named list of model test loss (or losses for models with multiple outputs) and model metrics.

See Also

export_savedmodel.keras.engine.training.Model

Export a Saved Model

Description

Serialize a model to disk.

Usage

```r
## S3 method for class 'keras.engine.training.Model'
export_savedmodel(object,
    export_dir_base, overwrite = TRUE, versioned = !overwrite,
    remove_learning_phase = TRUE, as_text = FALSE, ...)
```

Arguments

- `object`  
  An R object.
- `export_dir_base`  
  A string containing a directory in which to export the SavedModel.
- `overwrite`  
  Should the `export_dir_base` directory be overwritten?
- `versioned`  
  Should the model be exported under a versioned subdirectory?
- `remove_learning_phase`  
  Should the learning phase be removed by saving and reloading the model? Defaults to TRUE.
- `as_text`  
  Whether to write the SavedModel in text format.
- `...`  
  Unused

Value

The path to the exported directory, as a string.

fit.keras.engine.training.Model

Train a Keras model

Description

Trains the model for a fixed number of epochs (iterations on a dataset).
Usage

```r
## S3 method for class 'keras.engine.training.Model'
fit(object, x = NULL, y = NULL,
    batch_size = NULL, epochs = 10,
    verbose = getOption("keras.fit_verbose", default = 1),
    callbacks = NULL, view_metrics = getOption("keras.view_metrics",
        default = "auto"), validation_split = 0, validation_data = NULL,
    shuffle = TRUE, class_weight = NULL, sample_weight = NULL,
    initial_epoch = 0, steps_per_epoch = NULL, validation_steps = NULL,
    ...)
```

Arguments

object

Model to train.

x

Vector, matrix, or array of training data (or list if the model has multiple inputs).
If all inputs in the model are named, you can also pass a list mapping input
names to data. `x` can be `NULL` (default) if feeding from framework-native tensors
(e.g. TensorFlow data tensors).

y

Vector, matrix, or array of target (label) data (or list if the model has multiple
outputs). If all outputs in the model are named, you can also pass a list mapping
output names to data. `y` can be `NULL` (default) if feeding from framework-native
tensors (e.g. TensorFlow data tensors).

batch_size

Integer or `NULL`. Number of samples per gradient update. If unspecified, `batch_size`
will default to 32.

epochs

Number of epochs to train the model. Note that in conjunction with `initial_epoch`,
epochs is to be understood as "final epoch". The model is not trained for a num-
ber of iterations given by `epochs`, but merely until the epoch of index `epochs` is
reached.

verbose

Verbosity mode (0 = silent, 1 = progress bar, 2 = one line per epoch).

callbacks

List of callbacks to be called during training.

view_metrics

View realtime plot of training metrics (by epoch). The default ("auto") will
display the plot when running within RStudio, metrics were specified during
model `compile()`, epochs > 1 and verbose > 0. Use the global `keras.view_metrics`
option to establish a different default.

validation_split

Float between 0 and 1. Fraction of the training data to be used as validation
data. The model will set apart this fraction of the training data, will not train on
it, and will evaluate the loss and any model metrics on this data at the end of
each epoch. The validation data is selected from the last samples in the `x` and `y`
data provided, before shuffling.

validation_data

Data on which to evaluate the loss and any model metrics at the end of each
epoch. The model will not be trained on this data. This could be a list (`x_val, y_val`) or a list (`x_val, y_val, val_sample_weights`). `validation_data` will override `validation_split`.
*fit_generator*

**shuffle**

shuffle: Logical (whether to shuffle the training data before each epoch) or string (for "batch"). "batch" is a special option for dealing with the limitations of HDF5 data; it shuffles in batch-sized chunks. Has no effect when steps_per_epoch is not NULL.

**class_weight**

Optional named list mapping indices (integers) to a weight (float) value, used for weighting the loss function (during training only). This can be useful to tell the model to "pay more attention" to samples from an under-represented class.

**sample_weight**

Optional array of the same length as x, containing weights to apply to the model's loss for each sample. In the case of temporal data, you can pass a 2D array with shape (samples, sequence_length), to apply a different weight to every timestep of every sample. In this case you should make sure to specify sample_weight_mode="temporal" in compile().

**initial_epoch**

Integer, Epoch at which to start training (useful for resuming a previous training run).

**steps_per_epoch**

Total number of steps (batches of samples) before declaring one epoch finished and starting the next epoch. When training with input tensors such as TensorFlow data tensors, the default NULL is equal to the number of samples in your dataset divided by the batch size, or 1 if that cannot be determined.

**validation_steps**

Only relevant if steps_per_epoch is specified. Total number of steps (batches of samples) to validate before stopping.

... Unused

**Value**

A history object that contains all information collected during training.

**See Also**


---

**fit_generator**

Fits the model on data yielded batch-by-batch by a generator.

**Description**

The generator is run in parallel to the model, for efficiency. For instance, this allows you to do real-time data augmentation on images on CPU in parallel to training your model on GPU.
Usage

```r
fit_generator(object, generator, steps_per_epoch, epochs = 1,
              verbose = getOption("keras.fit_verbose", default = 1),
              callbacks = NULL, view_metrics = getOption("keras.view_metrics",
                                                          default = "auto"), validation_data = NULL, validation_steps = NULL,
              class_weight = NULL, max_queue_size = 10, workers = 1,
              initial_epoch = 0)
```

Arguments

- **object**: Keras model object
- **generator**: A generator (e.g. like the one provided by `flow_images_from_directory()` or a custom R generator function).
  
  The output of the generator must be a list of one of these forms:
  
  - (inputs, targets)
  - (inputs, targets, sample_weights)

  This list (a single output of the generator) makes a single batch. Therefore, all arrays in this list must have the same length (equal to the size of this batch). Different batches may have different sizes. For example, the last batch of the epoch is commonly smaller than the others, if the size of the dataset is not divisible by the batch size. The generator is expected to loop over its data indefinitely. An epoch finishes when `steps_per_epoch` batches have been seen by the model.

- **steps_per_epoch**: Total number of steps (batches of samples) to yield from generator before declaring one epoch finished and starting the next epoch. It should typically be equal to the number of samples if your dataset divided by the batch size.

- **epochs**: Integer. Number of epochs to train the model. An epoch is an iteration over the entire data provided, as defined by `steps_per_epoch`. Note that in conjunction with `initial_epoch`, `epochs` is to be understood as "final epoch". The model is not trained for a number of iterations given by `epochs`, but merely until the epoch of index `epochs` is reached.

- **verbose**: Verbosity mode (0 = silent, 1 = progress bar, 2 = one line per epoch).

- **callbacks**: List of callbacks to apply during training.

- **view_metrics**: View realtime plot of training metrics (by epoch). The default ("auto") will display the plot when running within RStudio. `metrics` were specified during model `compile()`, `epochs > 1` and `verbose > 0`. Use the global `keras.view_metrics` option to establish a different default.

- **validation_data**: this can be either:
  
  - a generator for the validation data
  - a list (inputs, targets)
  - a list (inputs, targets, sample_weights). on which to evaluate the loss and any model metrics at the end of each epoch. The model will not be trained on this data.
Validation Steps

- **validation_steps**: Only relevant if `validation_data` is a generator. Total number of steps (batches of samples) to yield from generator before stopping at the end of every epoch. It should typically be equal to the number of samples of your validation dataset divided by the batch size.

Class Weight

- **class_weight**: Optional named list mapping class indices (integer) to a weight (float) value, used for weighting the loss function (during training only). This can be useful to tell the model to "pay more attention" to samples from an under-represented class.

Max Queue Size

- **max_queue_size**: Maximum size for the generator queue. If unspecified, `max_queue_size` will default to 10.

Workers

- **workers**: Maximum number of threads to use for parallel processing. Note that parallel processing will only be performed for native Keras generators (e.g. `flow_images_from_directory()`) as R based generators must run on the main thread.

Initial Epoch

- **initial_epoch**: Epoch at which to start training (useful for resuming a previous training run).

Value

- Training history object (invisibly)

See Also

fit_text_tokenizer

augment Whether to fit on randomly augmented samples
rounds If augment, how many augmentation passes to do over the data
seed random seed.

See Also

Other image preprocessing: flow_images_from_dataframe, flow_images_from_data, flow_images_from_directory, image_load, image_to_array

---

fit_text_tokenizer Update tokenizer internal vocabulary based on a list of texts or list of sequences.

Description

Update tokenizer internal vocabulary based on a list of texts or list of sequences.

Usage

fit_text_tokenizer(object, x)

Arguments

object Tokenizer returned by text_tokenizer()
x Vector/list of strings, or a generator of strings (for memory-efficiency); Alternatively a list of “sequence” (a sequence is a list of integer word indices).

Note

Required before using texts_to_sequences(), texts_to_matrix(), or sequences_to_matrix().

See Also

Other text tokenization: save_text_tokenizer, sequences_to_matrix, text_tokenizer, texts_to_matrix, texts_to_sequences_generator, texts_to_sequences
flow_images_from_data  Generates batches of augmented/normalized data from image data and labels

Description

Generates batches of augmented/normalized data from image data and labels

Usage

flow_images_from_data(x, y = NULL, generator = image_data_generator(),
        batch_size = 32, shuffle = TRUE, sample_weight = NULL,
        seed = NULL, save_to_dir = NULL, save_prefix = "",
        save_format = "png", subset = NULL)

Arguments

x  data. Should have rank 4. In case of grayscale data, the channels axis should have value 1, and in case of RGB data, it should have value 3.
y  labels (can be NULL if no labels are required)
generator  Image data generator to use for augmenting/normalizing image data.
batch_size  int (default: 32).
shuffle  boolean (default: TRUE).
sample_weight  Sample weights.
seed  int (default: NULL).
save_to_dir  NULL or str (default: NULL). This allows you to optionally specify a directory to which to save the augmented pictures being generated (useful for visualizing what you are doing).
save_prefix  str (default: "). Prefix to use for filenames of saved pictures (only relevant if save_to_dir is set).
save_format  one of "png", "jpeg" (only relevant if save_to_dir is set). Default: "png".
subset  Subset of data ("training" or "validation") if validation_split is set in image_data_generator().

Details

Yields batches indefinitely, in an infinite loop.

Yields

(x, y) where x is an array of image data and y is an array of corresponding labels. The generator loops indefinitely.
flow_images_from_dataframe

*Takes the dataframe and the path to a directory and generates batches of augmented/normalized data.*

### Description

Takes the dataframe and the path to a directory and generates batches of augmented/normalized data.

### Usage

```r
flow_images_from_dataframe(dataframe, directory = NULL,
    x_col = "filename", y_col = "class",
    generator = image_data_generator(), target_size = c(256, 256),
    color_mode = "rgb", classes = NULL, class_mode = "categorical",
    batch_size = 32, shuffle = TRUE, seed = NULL, save_to_dir = NULL,
    save_prefix = "", save_format = "png", subset = NULL,
    interpolation = "nearest", drop_duplicates = TRUE)
```

### Arguments

- **dataframe**: `data.frame` containing the filepaths relative to directory (or absolute paths if directory is `NULL`) of the images in a character column. It should include other columns depending on the `class_mode`:
  - if `class_mode` is "categorical" (default value) it must include the `y_col` column with the class/es of each image. Values in column can be character/list if a single class or list if multiple classes.
  - if `class_mode` is "binary" or "sparse" it must include the given `y_col` column with class values as strings.
  - if `class_mode` is "other" it should contain the columns specified in `y_col`.
  - if `class_mode` is "input" or `NULL` no extra column is needed.

- **directory**: character, path to the directory to read images from. If `NULL`, data in `x_col` column should be absolute paths.

- **x_col**: character, column in `dataframe` that contains the filenames (or absolute paths if `directory` is `NULL`).

- **y_col**: string or list, column/s in `dataframe` that has the target data.

- **generator**: `Image data generator` to use for augmenting/normalizing image data.

- **target_size**: Either `NULL` (default to original size) or integer vector (`img_height`, `img_width`).
flow_images_from_dataframe

- **color_mode**: one of "grayscale", "rgb". Default: "rgb". Whether the images will be converted to have 1 or 3 color channels.

- **classes**: optional list of classes (e.g. `c('dogs', 'cats')`). Default: NULL. If not provided, the list of classes will be automatically inferred from the `y_col`, which will map to the label indices, will be alphanumeric). The dictionary containing the mapping from class names to class indices can be obtained via the attribute `class_indices`.

- **class_mode**: one of "categorical", "binary", "sparse", "input", "other" or None. Default: "categorical". Mode for yielding the targets:
  - "binary": 1D array of binary labels,
  - "categorical": 2D array of one-hot encoded labels. Supports multi-label output.
  - "sparse": 1D array of integer labels,
  - "input": images identical to input images (mainly used to work with autoencoders),
  - "other": array of `y_col` data, NULL, no targets are returned (the generator will only yield batches of image data, which is useful to use in `predict_generator`).

- **batch_size**: int (default: 32).

- **shuffle**: boolean (default: TRUE).

- **seed**: int (default: NULL).

- **save_to_dir**: NULL or str (default: NULL). This allows you to optionally specify a directory to which to save the augmented pictures being generated (useful for visualizing what you are doing).

- **save_prefix**: str (default: "). Prefix to use for filenames of saved pictures (only relevant if `save_to_dir` is set).

- **save_format**: one of "png", "jpeg" (only relevant if `save_to_dir` is set). Default: "png".

- **subset**: Subset of data ("training" or "validation") if validation_split is set in `image_data_generator()`.

- **interpolation**: Interpolation method used to resample the image if the target size is different from that of the loaded image. Supported methods are "nearest", "bilinear", and "bicubic". If PIL version 1.1.3 or newer is installed, "lanczos" is also supported. If PIL version 3.4.0 or newer is installed, "box" and "hamming" are also supported. By default, "nearest" is used.

- **drop_duplicates**: Boolean, whether to drop duplicate rows based on filename.

**Details**

Yields batches indefinitely, in an infinite loop.

**Yields**

`(x, y)` where `x` is an array of image data and `y` is a array of corresponding labels. The generator loops indefinitely.
flow_images_from_directory

Generates batches of data from images in a directory (with optional augmented/normalized data)

Description
Generates batches of data from images in a directory (with optional augmented/normalized data)

Usage
flow_images_from_directory(directory, generator = image_data_generator(), target_size = c(256, 256), color_mode = "rgb", classes = NULL, class_mode = "categorical", batch_size = 32, shuffle = TRUE, seed = NULL, save_to_dir = NULL, save_prefix = "", save_format = "png", follow_links = FALSE, subset = NULL, interpolation = "nearest")

Arguments
- directory: path to the target directory. It should contain one subdirectory per class. Any PNG, JPG, BMP, PPM, or TIF images inside each of the subdirectories directory tree will be included in the generator. See this script for more details.
- generator: Image data generator (default generator does no data augmentation/normalization transformations)
- target_size: integer vector, default: c(256, 256). The dimensions to which all images found will be resized.
- color_mode: one of "grayscale", "rgb". Default: "rgb". Whether the images will be converted to have 1 or 3 color channels.
- classes: optional list of class subdirectories (e.g. c(‘dogs’, ‘cats’)). Default: NULL. If not provided, the list of classes will be automatically inferred (and the order of the classes, which will map to the label indices, will be alphanumeric).
class_mode one of "categorical", "binary", "sparse" or NULL. Default: "categorical". Determines the type of label arrays that are returned: "categorical" will be 2D one-hot encoded labels, "binary" will be 1D binary labels, "sparse" will be 1D integer labels. If NULL, no labels are returned (the generator will only yield batches of image data, which is useful to use predict_generator(), evaluate_generator(), etc.).

batch_size int (default: 32).

shuffle boolean (default: TRUE).

seed int (default: NULL).

save_to_dir NULL or str (default: NULL). This allows you to optionally specify a directory to which to save the augmented pictures being generated (useful for visualizing what you are doing).

save_prefix str (default: "). Prefix to use for filenames of saved pictures (only relevant if save_to_dir is set).

save_format one of "png", "jpeg" (only relevant if save_to_dir is set). Default: "png".

follow_links whether to follow symlinks inside class subdirectories (default: FALSE).

subset Subset of data ("training" or "validation") if validation_split is set in image_data_generator().

interpolation Interpolation method used to resample the image if the target size is different from that of the loaded image. Supported methods are "nearest", "bilinear", and "bicubic". If PIL version 1.1.3 or newer is installed, "lanczos" is also supported. If PIL version 3.4.0 or newer is installed, "box" and "hamming" are also supported. By default, "nearest" is used.

Details

Yields batches indefinitely, in an infinite loop.

Yields

(x, y) where x is an array of image data and y is a array of corresponding labels. The generator loops indefinitely.

See Also

Other image preprocessing: fit_image_data_generator, flow_images_from_dataframe, flow_images_from_data, image_load, image_to_array
freeze_weights

Freeze and unfreeze weights

Description

Freeze weights in a model or layer so that they are no longer trainable.

Usage

freeze_weights(object, from = NULL, to = NULL)

unfreeze_weights(object, from = NULL, to = NULL)

Arguments

object  
Keras model or layer object

from  
Layer instance, layer name, or layer index within model

to  
Layer instance, layer name, or layer index within model

Note

The from and to layer arguments are both inclusive.

When applied to a model, the freeze or unfreeze is a global operation over all layers in the model (i.e. layers not within the specified range will be set to the opposite value, e.g. unfrozen for a call to freeze).

Models must be compiled again after weights are frozen or unfrozen.

Examples

```r
## Not run:
# instantiate a VGG16 model
conv_base <- application_vgg16(
  weights = "imagenet",
  include_top = FALSE,
  input_shape = c(150, 150, 3)
)

# freeze it's weights
freeze_weights(conv_base)

# create a composite model that includes the base + more layers
model <- keras_model_sequential()
#> conv_base
#> layer_flatten()
#> layer_dense(units = 256, activation = "relu")
#> layer_dense(units = 1, activation = "sigmoid")

# compile
```
```r
model %>% compile(
  loss = "binary_crossentropy",
  optimizer = optimizer_rmsprop(lr = 2e-5),
  metrics = c("accuracy")
)

# unfreeze weights from "block5_conv1" on
unfreeze_weights(conv_base, from = "block5_conv1")

# compile again since we froze or unfroze weights
model %>% compile(
  loss = "binary_crossentropy",
  optimizer = optimizer_rmsprop(lr = 2e-5),
  metrics = c("accuracy")
)

## End(Not run)
```

---

**generator_next**

Retrieve the next item from a generator

**Description**

Use to retrieve items from generators (e.g. `image_data_generator()`). Will return either the next item or NULL if there are no more items.

**Usage**

```r
generator_next(generator, completed = NULL)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>generator</td>
<td>Generator</td>
</tr>
<tr>
<td>completed</td>
<td>Sentinel value to return from <code>generator_next()</code> if the iteration completes (defaults to NULL but can be any R value you specify).</td>
</tr>
</tbody>
</table>

---

**get_config**

Layer/Model configuration

**Description**

A layer config is an object returned from `get_config()` that contains the configuration of a layer or model. The same layer or model can be reinstatiated later (without its trained weights) from this configuration using `from_config()`. The config does not include connectivity information, nor the class name (those are handled externally).
Usage

get_config(object)
from_config(config)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Layer or model object</td>
</tr>
<tr>
<td>config</td>
<td>Object with layer or model configuration</td>
</tr>
</tbody>
</table>

Value

get_config() returns an object with the configuration, from_config() returns a re-instantation of the object.

Note

Objects returned from get_config() are not serializable. Therefore, if you want to save and restore a model across sessions, you can use the model_to_json() or model_to_yaml() functions (for model configuration only, not weights) or the save_model_hdf5() function to save the model configuration and weights to a file.

See Also


Other layer methods: count_params, get_input_at, get_weights, reset_states

---

get_file  Downloads a file from a URL if it not already in the cache.

Description

Passing the MD5 hash will verify the file after download as well as if it is already present in the cache.

Usage

get_file(fname, origin, file_hash = NULL, cache_subdir = "datasets", hash_algorithm = "auto", extract = FALSE, archive_format = "auto", cache_dir = NULL)
get_input_at

Arguments

fname Name of the file. If an absolute path /path/to/file.txt is specified the file will be saved at that location.
origin Original URL of the file.
file_hash The expected hash string of the file after download. The sha256 and md5 hash algorithms are both supported.
cache_subdir Subdirectory under the Keras cache dir where the file is saved. If an absolute path /path/to/folder is specified the file will be saved at that location.
hash_algorithm Select the hash algorithm to verify the file. options are 'md5', 'sha256', and 'auto'. The default 'auto' detects the hash algorithm in use.
extra True tries extracting the file as an Archive, like tar or zip.
arhive_format Archive format to try for extracting the file. Options are 'auto', 'tar', 'zip', and None. 'tar' includes tar, tar.gz, and tar.bz files. The default 'auto' is ('tar', 'zip'). None or an empty list will return no matches found.
cache_dir Location to store cached files, when NULL it defaults to the Keras configuration directory.

Value
Path to the downloaded file

get_input_at(object, node_index)
get_output_at(object, node_index)
get_input_shape_at(object, node_index)
get_output_shape_at(object, node_index)
get_input_mask_at(object, node_index)
get_output_mask_at(object, node_index)

Description
Whenever you are calling a layer on some input, you are creating a new tensor (the output of the layer), and you are adding a "node" to the layer, linking the input tensor to the output tensor. When you are calling the same layer multiple times, that layer owns multiple nodes indexed as 1, 2, 3. These functions enable you to retrieve various tensor properties of layers with multiple nodes.
get_layer

**Arguments**

- **object**
  - Layer or model object
- **node_index**
  - Integer, index of the node from which to retrieve the attribute. E.g. node_index = 1 will correspond to the first time the layer was called.

**Value**

- A tensor (or list of tensors if the layer has multiple inputs/outputs).

**See Also**

Other layer methods: `count_params`, `get_config`, `get_weights`, `reset_states`

---

**get_layer**

Retrieves a layer based on either its name (unique) or index.

**Description**

Indices are based on order of horizontal graph traversal (bottom-up) and are 1-based. If name and index are both provided, index will take precedence.

**Usage**

```r
get_layer(object, name = NULL, index = NULL)
```

**Arguments**

- **object**
  - Keras model object
- **name**
  - String, name of layer.
- **index**
  - Integer, index of layer (0-based)

**Value**

- A layer instance.

**See Also**

get_weights

Layer/Model weights as R arrays

Description
Layer/Model weights as R arrays

Usage
get_weights(object)
set_weights(object, weights)

Arguments
object       Layer or model object
weights      Weights as R array

See Also
Other model persistence: model_to_json, model_to_yaml, save_model_hdf5, save_model_weights_hdf5, serialize_model
Other layer methods: count_params, get_config, get_input_at, reset_states

hdf5_matrix

Representation of HDF5 dataset to be used instead of an R array

Description
Representation of HDF5 dataset to be used instead of an R array

Usage
hdf5_matrix(datapath, dataset, start = 0, end = NULL, normalizer = NULL)

Arguments
datapath       string, path to a HDF5 file
dataset        string, name of the HDF5 dataset in the file specified in datapath
start          int, start of desired slice of the specified dataset
end            int, end of desired slice of the specified dataset
normalizer     function to be called on data when retrieved
Details
Providing start and end allows use of a slice of the dataset.
Optionally, a normalizer function (or lambda) can be given. This will be called on every slice of data retrieved.

Value
An array-like HDF5 dataset.

imagenet_decode_predictions
Decodes the prediction of an ImageNet model.

Description
Decodes the prediction of an ImageNet model.

Usage
imagenet_decode_predictions(preds, top = 5)

Arguments
preds Tensor encoding a batch of predictions.
top integer, how many top-guesses to return.

Value
List of data frames with variables class_name, class_description, and score (one data frame per sample in batch input).

imagenet_preprocess_input
Preprocesses a tensor or array encoding a batch of images.

Description
Preprocesses a tensor or array encoding a batch of images.

Usage
imagenet_preprocess_input(x, data_format = NULL, mode = "caffe")
**Arguments**

- **x**
  Input Numpy or symbolic tensor, 3D or 4D.

- **data_format**
  Data format of the image tensor/array.

- **mode**
  One of "caffe", "tf", or "torch"
  - **caffe**: will convert the images from RGB to BGR, then will zero-center each color channel with respect to the ImageNet dataset, without scaling.
  - **tf**: will scale pixels between -1 and 1, sample-wise.
  - **torch**: will scale pixels between 0 and 1 and then will normalize each channel with respect to the ImageNet dataset.

**Value**

Preprocessed tensor or array.

---

**image_data_generator**

Generate batches of image data with real-time data augmentation. The data will be looped over (in batches).

**Description**

Generate batches of image data with real-time data augmentation. The data will be looped over (in batches).

**Usage**

```python
image_data_generator(featurewise_center = FALSE,
samplewise_center = FALSE, featurewise_std_normalization = FALSE,
samplewise_std_normalization = FALSE, zca_whitening = FALSE,
zca_epsilon = 1e-06, rotation_range = 0, width_shift_range = 0,
height_shift_range = 0, brightness_range = NULL, shear_range = 0,
zoom_range = 0, channel_shift_range = 0, fill_mode = "nearest",
cval = 0, horizontal_flip = FALSE, vertical_flip = FALSE,
rescale = NULL, preprocessing_function = NULL, data_format = NULL,
validation_split = 0)
```

**Arguments**

- **featurewise_center**
  Set input mean to 0 over the dataset, feature-wise.

- **samplewise_center**
  Boolean. Set each sample mean to 0.

- **featurewise_std_normalization**
  Divide inputs by std of the dataset, feature-wise.

- **samplewise_std_normalization**
  Divide each input by its std.
zca_whitening  apply ZCA whitening.
zca_epsilon  Epsilon for ZCA whitening. Default is 1e-6.
rotation_range  degrees (0 to 180).
width_shift_range  fraction of total width.
height_shift_range  fraction of total height.
brightness_range  the range of brightness to apply
shear_range  shear intensity (shear angle in radians).
zoom_range  amount of zoom. If scalar z, zoom will be randomly picked in the range [1-z, 1+z].
A sequence of two can be passed instead to select this range.
channel_shift_range  shift range for each channels.
fill_mode  One of "constant", "nearest", "reflect" or "wrap". Points outside the boundaries of the input are filled according to the given mode:
    • "constant": kkkkkkkk|abcd|kkkkkkkk (cval=k)
    • "nearest": aaaaaaaaa|abcd|dddddddd
    • "reflect": abcddcba|abcd|dcbaabcd
    • "wrap": abcdabcd|abcd|abcdabcd
cval  value used for points outside the boundaries when fill_mode is 'constant'. Default is 0.
horizontal_flip  whether to randomly flip images horizontally.
vertical_flip  whether to randomly flip images vertically.
rescale  rescaling factor. If NULL or 0, no rescaling is applied, otherwise we multiply the data by the value provided (before applying any other transformation).
preprocessing_function  function that will be implied on each input. The function will run before any other modification on it. The function should take one argument: one image (tensor with rank 3), and should output a tensor with the same shape.
data_format  'channels_first' or 'channels_last'. In 'channels_first' mode, the channels dimension (the depth) is at index 1, in 'channels_last' mode it is at index 3. It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".
validation_split  fraction of images reserved for validation (strictly between 0 and 1).
image_load  \hspace{1cm} Loads an image into PIL format.

**Description**

Loads an image into PIL format.

**Usage**

```python
image_load(path, grayscale = FALSE, target_size = NULL,
            interpolation = "nearest")
```

**Arguments**

- `path`  
  Path to image file

- `grayscale`  
  Boolean, whether to load the image as grayscale.

- `target_size`  
  Either NULL (default to original size) or integer vector (img_height, img_width).

- `interpolation`  
  Interpolation method used to resample the image if the target size is different from that of the loaded image. Supported methods are "nearest", "bilinear", and "bicubic". If PIL version 1.1.3 or newer is installed, "lanczos" is also supported. If PIL version 3.4.0 or newer is installed, "box" and "hamming" are also supported. By default, "nearest" is used.

**Value**

A PIL Image instance.

**See Also**

Other image preprocessing: `fit_image_data_generator`, `flow_images_from_dataframe`, `flow_images_from_data`, `flow_images_from_directory`, `image_to_array`

---

image_to_array  \hspace{1cm} 3D array representation of images

**Description**

3D array that represents an image with dimensions (height, width, channels) or (channels, height, width) depending on the data_format.
Usage

image_to_array(img, data_format = c("channels_last", "channels_first"))

image_array_resize(img, height, width, data_format = c("channels_last", "channels_first"))

image_array_save(img, path, data_format = NULL, file_format = NULL, scale = TRUE)

Arguments

- **img**: Image
- **data_format**: Image data format ("channels_last" or "channels_first")
- **height**: Height to resize to
- **width**: Width to resize to
- **path**: Path to save image to
- **file_format**: Optional file format override. If omitted, the format to use is determined from the filename extension. If a file object was used instead of a filename, this parameter should always be used.
- **scale**: Whether to rescale image values to be within 0,255

See Also

Other image preprocessing: fit_image_data_generator, flow_images_from_dataframe, flow_images_from_data, flow_images_from_directory, image_load

---

implementation | Keras implementation

### Description

Obtain a reference to the Python module used for the implementation of Keras.

### Usage

implementation()

### Details

There are currently two Python modules which implement Keras:

- keras ("keras")
- tensorflow.keras ("tensorflow")

This function returns a reference to the implementation being currently used by the keras package. The default implementation is "keras". You can override this by setting the KERAS_IMPLEMENTATION environment variable to "tensorflow".
**Value**

Reference to the Python module used for the implementation of Keras.

---

**initializer_constant**  
*Initializer that generates tensors initialized to a constant value.*

---

**Description**

Initializer that generates tensors initialized to a constant value.

**Usage**

```python
sicaller_constant(value = 0)
```

**Arguments**

- `value`  
  float; the value of the generator tensors.

**See Also**

Other initializers: `initializer_glorot_normal`, `initializer_glorot_uniform`, `initializer_he_normal`, `initializer_he_uniform`, `initializer_identity`, `initializer_lecun_normal`, `initializer_lecun_uniform`, `initializer_ones`, `initializer_orthogonal`, `initializer_random_normal`, `initializer_random_uniform`, `initializer_truncated_normal`, `initializer_variance_scaling`, `initializer_zeros`

---

**initializer_glorot_normal**  
*Glorot normal initializer, also called Xavier normal initializer.*

---

**Description**

It draws samples from a truncated normal distribution centered on 0 with 

\[ \text{stddev} = \sqrt{\frac{2}{\text{fan}_\text{in} + \text{fan}_\text{out}}} \]

where \( \text{fan}_\text{in} \) is the number of input units in the weight tensor and \( \text{fan}_\text{out} \) is the number of output units in the weight tensor.

**Usage**

```python
sicaller_glorot_normal(seed = NULL)
```

**Arguments**

- `seed`  
  Integer used to seed the random generator.
**initializer_glorot_uniform**

Glorot uniform initializer, also called Xavier uniform initializer.

**Description**

It draws samples from a uniform distribution within \(-\text{limit, limit}\) where \(\text{limit is } \sqrt{6 / (\text{fan\_in + fan\_out})}\) where \text{fan\_in}\ is the number of input units in the weight tensor and \text{fan\_out}\ is the number of output units in the weight tensor.

**Usage**

```python
initializer_glorot_uniform(seed = NULL)
```

**Arguments**

- **seed**: Integer used to seed the random generator.

**References**


**See Also**

Other initializers: `initializer_constant`, `initializer_glorot_normal`, `initializer_he_normal`, `initializer_he_uniform`, `initializer_identity`, `initializer_lecun_normal`, `initializer_lecun_uniform`, `initializer_ones`, `initializer_orthogonal`, `initializer_random_normal`, `initializer_random_uniform`, `initializer_truncated_normal`, `initializer_variance_scaling`, `initializer_zeros`
**initializer_he_normal**  
*He normal initializer.*

**Description**
It draws samples from a truncated normal distribution centered on 0 with \[ \text{stddev} = \sqrt{2 / \text{fan\	extunderscore in}} \]
where \( \text{fan\	extunderscore in} \) is the number of input units in the weight tensor.

**Usage**
`initializer_he_normal(seed = NULL)`

**Arguments**
- `seed` Integer used to seed the random generator.

**References**
He et al., http://arxiv.org/abs/1502.01852

**See Also**
Other initializers: `initializer_constant`, `initializer_glorot_normal`, `initializer_glorot_uniform`, `initializer_he_uniform`, `initializer_identity`, `initializer_lecun_normal`, `initializer_lecun_uniform`, `initializer_ones`, `initializer_orthogonal`, `initializer_random_normal`, `initializer_random_uniform`, `initializer_truncated_normal`, `initializer_variance_scaling`, `initializer_zeros`

**initializer_he_uniform**  
*He uniform variance scaling initializer.*

**Description**
It draws samples from a uniform distribution within \([-\text{limit}, \text{limit}]\) where \( \text{limit} = \sqrt{6 / \text{fan\	extunderscore in}} \), where \( \text{fan\	extunderscore in} \) is the number of input units in the weight tensor.

**Usage**
`initializer_he_uniform(seed = NULL)`

**Arguments**
- `seed` Integer used to seed the random generator.

**References**
He et al., http://arxiv.org/abs/1502.01852
**initializer_identity**  
Initializer that generates the identity matrix.

**Description**

Only use for square 2D matrices.

**Usage**

initializer_identity(gain = 1)

**Arguments**

- **gain**
  Multiplicative factor to apply to the identity matrix

**See Also**

Other initializers:  
initializer_constant, initializer_glorot_normal, initializer_glorot_uniform,  
initializer_he_normal, initializer_identity, initializer_lecun_normal, initializer_lecun_uniform,  
initializer_ones, initializer_orthogonal, initializer_random_normal, initializer_random_uniform,  
initializer_truncated_normal, initializer_variance_scaling, initializer_zeros

**initializer_lecun_normal**

LeCun normal initializer.

**Description**

It draws samples from a truncated normal distribution centered on 0 with \( \text{stddev} \leq \sqrt{\frac{1}{\text{fan}_\text{in}}} \)
where \( \text{fan}_\text{in} \) is the number of input units in the weight tensor.

**Usage**

initializer_lecun_normal(seed = NULL)

**Arguments**

- **seed**
  A Python integer. Used to seed the random generator.
**initializer_lecun_uniform**

*LeCun uniform initializer.*

**Description**

It draws samples from a uniform distribution within $-\text{limit, limit}$ where $\text{limit}$ is $\sqrt{3 / \text{fan}_\text{in}}$ where $\text{fan}_\text{in}$ is the number of input units in the weight tensor.

**Usage**

```python
initializer_lecun_uniform(seed = NULL)
```

**Arguments**

- **seed** Integer used to seed the random generator.

**References**


**See Also**

Other initializers: `initializer_constant, initializer_glorot_normal, initializer_glorot_uniform, initializer_he_normal, initializer_he_uniform, initializer_identity, initializer_lecun_uniform, initializer_ones, initializer_orthogonal, initializer_random_normal, initializer_random_uniform, initializer_truncated_normal, initializer_variance_scaling, initializer_zeros`
**initializer_ones**

*Initializer that generates tensors initialized to 1.*

**Description**

Initializer that generates tensors initialized to 1.

**Usage**

```python
initializer_ones()
```

**See Also**

Other initializers: `initializer_constant`, `initializer_glorot_normal`, `initializer_glorot_uniform`, `initializer_he_normal`, `initializer_he_uniform`, `initializer_identity`, `initializer_lecun_normal`, `initializer_lecun_uniform`, `initializer_orthogonal`, `initializer_random_normal`, `initializer_random_uniform`, `initializer_truncated_normal`, `initializer_variance_scaling`, `initializer_zeros`

**initializer_orthogonal**

*Initializer that generates a random orthogonal matrix.*

**Description**

Initializer that generates a random orthogonal matrix.

**Usage**

```python
initializer_orthogonal(gain = 1, seed = NULL)
```

**Arguments**

- `gain`  
  Multiplicative factor to apply to the orthogonal matrix.
- `seed`  
  Integer used to seed the random generator.

**References**


**See Also**

Other initializers: `initializer_constant`, `initializer_glorot_normal`, `initializer_glorot_uniform`, `initializer_he_normal`, `initializer_he_uniform`, `initializer_identity`, `initializer_lecun_normal`, `initializer_lecun_uniform`, `initializer_orthogonal`, `initializer_random_normal`, `initializer_random_uniform`, `initializer_truncated_normal`, `initializer_variance_scaling`, `initializer_zeros`
initializer_random_normal

Initializer that generates tensors with a normal distribution.

Description

Initializer that generates tensors with a normal distribution.

Usage

initializer_random_normal(mean = 0, stddev = 0.05, seed = NULL)

Arguments

- mean: Mean of the random values to generate.
- stddev: Standard deviation of the random values to generate.
- seed: Integer used to seed the random generator.

See Also

Other initializers: initializer_constant, initializer_glorot_normal, initializer_glorot_uniform, initializer_he_normal, initializer_he_uniform, initializer_identity, initializer_lecun_normal, initializer_lecun_uniform, initializer_ones, initializer_orthogonal, initializer_random_uniform, initializer_truncated_normal, initializer_variance_scaling, initializer_zeros

initializer_random_uniform

Initializer that generates tensors with a uniform distribution.

Description

Initializer that generates tensors with a uniform distribution.

Usage

initializer_random_uniform(minval = -0.05, maxval = 0.05, seed = NULL)

Arguments

- minval: Lower bound of the range of random values to generate.
- maxval: Upper bound of the range of random values to generate. Defaults to 1 for float types.
- seed: seed
See Also

Other initializers: initializer_constant, initializer_glorot_normal, initializer_glorot_uniform, initializer_he_normal, initializer_he_uniform, initializer_identity, initializer_lecun_normal, initializer_lecun_uniform, initializer_ones, initializer_orthogonal, initializer_random_normal, initializer_truncated_normal, initializer_variance_scaling, initializer_zeros

initializer_truncated_normal

*Initializer that generates a truncated normal distribution.*

Description

These values are similar to values from an initializer_random_normal() except that values more than two standard deviations from the mean are discarded and re-drawn. This is the recommended initializer for neural network weights and filters.

Usage

initializer_truncated_normal(mean = 0, stddev = 0.05, seed = NULL)

Arguments

- **mean**: Mean of the random values to generate.
- **stddev**: Standard deviation of the random values to generate.
- **seed**: Integer used to seed the random generator.

See Also

Other initializers: initializer_constant, initializer_glorot_normal, initializer_glorot_uniform, initializer_he_normal, initializer_he_uniform, initializer_identity, initializer_lecun_normal, initializer_lecun_uniform, initializer_ones, initializer_orthogonal, initializer_random_normal, initializer_truncated_normal, initializer_variance_scaling, initializer_zeros

initializer_variance_scaling

*Initializer capable of adapting its scale to the shape of weights.*

Description

With distribution="normal", samples are drawn from a truncated normal distribution centered on zero, with stddev = sqrt(scale / n) where n is:

- number of input units in the weight tensor, if mode = "fan_in"
- number of output units, if mode = "fan_out"
- average of the numbers of input and output units, if mode = "fan_avg"
initializer_zeros

Usage

initializer_variance_scaling(scale = 1, mode = c("fan_in", "fan_out", "fan_avg"), distribution = c("normal", "uniform"), seed = NULL)

Arguments

scale Scaling factor (positive float).
mode One of "fan_in", "fan_out", "fan_avg".
distribution One of "normal", "uniform"
seed Integer used to seed the random generator.

details

With distribution="uniform", samples are drawn from a uniform distribution within -limit, limit, with limit = sqrt(3 * scale / n).

See Also

Other initializers: initializer_constant, initializer_glorot_normal, initializer_glorot_uniform, initializer_he_normal, initializer_he_uniform, initializer_identity, initializer_lecun_normal, initializer_lecun_uniform, initializer_ones, initializer_orthogonal, initializer_random_normal, initializer_random_uniform, initializer_truncated_normal, initializer_zeros

initializer_zeros  

Initializer that generates tensors initialized to 0.

Description

Initializer that generates tensors initialized to 0.

Usage

initializer_zeros()

See Also

Other initializers: initializer_constant, initializer_glorot_normal, initializer_glorot_uniform, initializer_he_normal, initializer_he_uniform, initializer_identity, initializer_lecun_normal, initializer_lecun_uniform, initializer_ones, initializer_orthogonal, initializer_random_normal, initializer_random_uniform, initializer_truncated_normal, initializer_variance_scaling
Install Keras and the TensorFlow backend

Description

Keras and TensorFlow will be installed into an "r-tensorflow" virtual or conda environment. Note that "virtualenv" is not available on Windows (as this isn’t supported by TensorFlow).

Usage

install_keras(method = c("auto", "virtualenv", "conda"),
               conda = "auto", version = "default", tensorflow = "default",
               extra_packages = c("tensorflow-hub"))

Arguments

- method: Installation method ("virtualenv" or "conda")
- conda: Path to conda executable (or "auto" to find conda using the PATH and other conventional install locations).
- version: Version of Keras to install. Specify "default" to install the latest release. Otherwise specify an alternate version (e.g. "2.2.2").
- tensorflow: TensorFlow version to install. Specify "default" to install the CPU version of the latest release. Specify "gpu" to install the GPU version of the latest release. You can also provide a full major.minor.patch specification (e.g. "1.1.0"), appending "-gpu" if you want the GPU version (e.g. "1.1.0-gpu"). Alternatively, you can provide the full URL to an installer binary (e.g. for a nightly binary).
- extra_packages: Additional PyPI packages to install along with Keras and TensorFlow.

GPU Installation

Keras and TensorFlow can be configured to run on either CPUs or GPUs. The CPU version is much easier to install and configure so is the best starting place especially when you are first learning how to use Keras. Here's the guidance on CPU vs. GPU versions from the TensorFlow website:

- **TensorFlow with CPU support only.** If your system does not have a NVIDIA® GPU, you must install this version. Note that this version of TensorFlow is typically much easier to install, so even if you have an NVIDIA GPU, we recommend installing this version first.
- **TensorFlow with GPU support.** TensorFlow programs typically run significantly faster on a GPU than on a CPU. Therefore, if your system has a NVIDIA® GPU meeting all prerequisites and you need to run performance-critical applications, you should ultimately install this version.

To install the GPU version:

1. Ensure that you have met all installation prerequisites including installation of the CUDA and cuDNN libraries as described in TensorFlow GPU Prerequisites.
install_keras

2. Pass `tensorflow = "gpu"` to `install_keras()`. For example:

   ```r
   install_keras(tensorflow = "gpu")
   ```

**Windows Installation**

The only supported installation method on Windows is "conda". This means that you should install Anaconda 3.x for Windows prior to installing Keras.

**Custom Installation**

Installing Keras and TensorFlow using `install_keras()` isn’t required to use the Keras R package. You can do a custom installation of Keras (and desired backend) as described on the Keras website and the Keras R package will find and use that version.

See the documentation on custom installations for additional information on how version of Keras and TensorFlow are located by the Keras package.

**Additional Packages**

If you wish to add additional PyPI packages to your Keras / TensorFlow environment you can either specify the packages in the `extra_packages` argument of `install_keras()`, or alternatively install them into an existing environment using the `reticulate::py_install()` function.

**Examples**

```r
## Not run:

# default installation
library(keras)
install_keras()

# install using a conda environment (default is virtualenv)
install_keras(method = "conda")

# install with GPU version of TensorFlow
# (NOTE: only do this if you have an NVIDIA GPU + CUDA!)
install_keras(tensorflow = "gpu")

# install a specific version of TensorFlow
install_keras(tensorflow = "1.2.1")
install_keras(tensorflow = "1.2.1-gpu")

## End(Not run)
```
is_keras_available  

**Description**
Probe to see whether the Keras python package is available in the current system environment.

**Usage**
```r
is_keras_available(version = NULL)
```

**Arguments**
- **version**: Minimum required version of Keras (defaults to NULL, no required version).

**Value**
Logical indicating whether Keras (or the specified minimum version of Keras) is available.

**Examples**
```r
## Not run:
# testthat utility for skipping tests when Keras isn't available
skip_if_no_keras <- function(version = NULL) {
  if (!is_keras_available(version))
    skip("Required keras version not available for testing")
}

# use the function within a test
test_that("keras function works correctly", {
  skip_if_no_keras()
  # test code here
})

## End(Not run)
```

KerasCallback  

**Description**
Base R6 class for Keras callbacks

**Usage**
```r
KerasCallback
```
Format

An R6Class generator object

Details

The logs named list that callback methods take as argument will contain keys for quantities relevant to the current batch or epoch.

Currently, the fit.keras.engine.training.Model() method for sequential models will include the following quantities in the logs that it passes to its callbacks:

- **on_epoch_end**: logs include acc and loss, and optionally include val_loss (if validation is enabled in fit), and val_acc (if validation and accuracy monitoring are enabled).
- **on_batch_begin**: logs include size, the number of samples in the current batch.
- **on_batch_end**: logs include loss, and optionally acc (if accuracy monitoring is enabled).

Value

KerasCallback.

Fields

- params Named list with training parameters (eg. verbosity, batch size, number of epochs...).
- model Reference to the Keras model being trained.

Methods

- on_epoch_begin(epoch, 1ogs) Called at the beginning of each epoch.
- on_epoch_end(epoch, 1ogs) Called at the end of each epoch.
- on_batch_begin(batch, 1ogs) Called at the beginning of each batch.
- on_batch_end(batch, 1ogs) Called at the end of each batch.
- on_train_begin(logs) Called at the beginning of training.
- on_train_end(logs) Called at the end of training.

Examples

```r
## Not run:
library(keras)

LossHistory <- R6::R6Class("LossHistory",
  inherit = KerasCallback,
  public = list(
    losses = NULL,
    on_batch_end = function(batch, logs = list()) {
      self$losses <- c(self$losses, logs[["loss"]])
    }
  )
```
KerasConstraint

Description
Base R6 class for Keras constraints

Usage
KerasConstraint

Format
An R6Class generator object

Details
You can implement a custom constraint either by creating an R function that accepts a weights \( w \) parameter, or by creating an R6 class that derives from KerasConstraint and implements a `call` method.

Methods

`call(w)` Constrain the specified weights.

Note
Models which use custom constraints cannot be serialized using `save_model_hdf5()`. Rather, the weights of the model should be saved and restored using `save_model_weights_hdf5()`.

See Also

`constraints`

Examples

```r
## Not run:
CustomNonNegConstraint <- R6::R6Class(
  "CustomNonNegConstraint",
  inherit = KerasConstraint,
  public = list(
    call = function(x) {
      w * k_cast(k_greater_equal(w, 0), k_floatx())
    }
  )
)```
KerasLayer

) )

layer_dense(units = 32, input_shape = c(784),
    kernel_constraint = CustomNonNegConstraint$new())

## End(Not run)

---

**KerasLayer**  
*Base R6 class for Keras layers*

**Description**

Base R6 class for Keras layers

**Usage**

KerasLayer

**Format**

An R6Class generator object #'

**Value**

KerasLayer.

**Methods**

build(input_shape) Creates the layer weights (must be implemented by all layers that have weights)  
call(inputs, mask) Call the layer on an input tensor.  
compute_output_shape(input_shape) Compute the output shape for the layer.  
add_loss(losses, inputs) Add losses to the layer.  
add_weight(name, shape, dtype, initializer, regularizer, trainable, constraint) Adds a weight variable to the layer.
KerasWrapper  

**Base R6 class for Keras wrappers**

### Description
Base R6 class for Keras wrappers

### Usage
KerasWrapper

### Format
An R6Class generator object

### Value
KerasWrapper.

### Methods
- **build(input_shape)** Builds the wrapped layer. Subclasses can extend this to perform custom operations on that layer.
- **call(inputs, mask)** Calls the wrapped layer on an input tensor.
- **compute_output_shape(input_shape)** Computes the output shape for the wrapped layer.
- **add_loss(losses, inputs)** Subclasses can use this to add losses to the wrapped layer.
- **add_weight(name, shape, dtype, initializer, regularizer, trainable, constraint)** Subclasses can use this to add weights to the wrapped layer.

**keras_array**

**Keras array object**

### Description
Convert an R vector, matrix, or array object to an array that has the optimal in-memory layout and floating point data type for the current Keras backend.

### Usage
keras_array(x, dtype = NULL)

### Arguments
- **x** Object or list of objects to convert
- **dtype** NumPy data type (e.g. float32, float64). If this is unspecified then R doubles will be converted to the default floating point type for the current Keras backend.
Details

Keras does frequent row-oriented access to arrays (for shuffling and drawing batches) so the order of arrays created by this function is always row-oriented ("C" as opposed to "Fortran" ordering, which is the default for R arrays).

If the passed array is already a NumPy array with the desired dtype and "C" order then it is returned unmodified (no additional copies are made).

Value

NumPy array with the specified dtype (or list of NumPy arrays if a list was passed for \(x\)).

<table>
<thead>
<tr>
<th>keras_model</th>
<th>Keras Model</th>
</tr>
</thead>
</table>

Description

A model is a directed acyclic graph of layers.

Usage

\[
\text{keras}_\text{model}(\text{inputs}, \text{outputs} = \text{NULL})
\]

Arguments

<table>
<thead>
<tr>
<th>inputs</th>
<th>Input layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>outputs</td>
<td>Output layer</td>
</tr>
</tbody>
</table>

See Also


Examples

```r
## Not run:
library(keras)

# input layer
inputs <- layer_input(shape = c(784))

# outputs compose input + dense layers
predictions <- inputs %>%
  layer_dense(units = 64, activation = 'relu') %>%
  layer_dense(units = 64, activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax')
```
# create and compile model
model <- keras_model(inputs = inputs, outputs = predictions)
model %>% compile(
  optimizer = 'rmsprop',
  loss = 'categorical_crossentropy',
  metrics = c('accuracy')
)

## End(Not run)

---

**keras_model_custom**  
Create a Keras custom model

**Description**
Create a Keras custom model

**Usage**
keras_model_custom(model_fn, name = NULL)

**Arguments**
- **model_fn**: Function that returns an R custom model
- **name**: Optional name for model

**Details**
For documentation on using custom models, see [https://keras.rstudio.com/articles/custom_models.html](https://keras.rstudio.com/articles/custom_models.html).

**Value**
A Keras model

---

**keras_model_sequential**  
Keras Model composed of a linear stack of layers

**Description**
Keras Model composed of a linear stack of layers

**Usage**
keras_model_sequential(layers = NULL, name = NULL)
Arguments

- **layers**: List of layers to add to the model
- **name**: Name of model

Note

The first layer passed to a Sequential model should have a defined input shape. What that means is that it should have received an `input_shape` or `batch_input_shape` argument, or for some type of layers (recurrent, Dense...) an `input_dim` argument.

See Also

Other model functions: `compile`, `evaluate`, `fit`, `get_config`, `get_layer`, `keras_model`, `multi_gpu_model`, `pop_layer`, `predict`, `predict_generator`, `predict_on_batch`, `predict_proba`, `summary`, `train_on_batch`

Examples

```r
## Not run:
library(keras)

model <- keras_model_sequential()
model %>%
  layer_dense(units = 32, input_shape = c(784)) %>%
  layer_activation('relu') %>%
  layer_dense(units = 10) %>%
  layer_activation('softmax')

model %>% compile(
  optimizer = 'rmsprop',
  loss = 'categorical_crossentropy',
  metrics = c('accuracy')
)

## End(Not run)
```

---

**k_abs**

*Element-wise absolute value.*

Description

Element-wise absolute value.

Usage

`k_abs(x)`
Arguments

x Tensor or variable.

Value

A tensor.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

Description

Bitwise reduction (logical AND).

Usage

k_all(x, axis = NULL, keepdims = FALSE)

Arguments

x Tensor or variable.

axis Axis along which to perform the reduction (axis indexes are 1-based).

keepdims whether the drop or broadcast the reduction axes.

Value

A uint8 tensor (0s and 1s).

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_any**

*Bitwise reduction (logical OR).*

---

**Description**

Bitwise reduction (logical OR).

**Usage**

```python
k_any(x, axis = NULL, keepdims = FALSE)
```

**Arguments**

- **x**: Tensor or variable.
- **axis**: Axis along which to perform the reduction (axis indexes are 1-based).
- **keepdims**: whether the drop or broadcast the reduction axes.

**Value**

A uint8 tensor (0s and 1s).

---

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/ backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_arange**

*Creates a 1D tensor containing a sequence of integers.*

---

**Description**

The function arguments use the same convention as Theano’s arange: if only one argument is provided, it is in fact the "stop" argument. The default type of the returned tensor is 'int32' to match TensorFlow's default.

**Usage**

```python
k_arange(start, stop = NULL, step = 1, dtype = "int32")
```
**Arguments**

- **start**: Start value.
- **stop**: Stop value.
- **step**: Difference between two successive values.
- **dtype**: Integer dtype to use.

**Value**

An integer tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_argmax**  
*Returns the index of the maximum value along an axis.*

---

**Description**

Returns the index of the maximum value along an axis.

**Usage**

```
k_argmax(x, axis = -1)
```

**Arguments**

- **x**: Tensor or variable.
- **axis**: Axis along which to perform the reduction (axis indexes are 1-based). Pass -1 (the default) to select the last axis.

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).
**k_argmin**

| k_argmin | Returns the index of the minimum value along an axis. |

**Description**

Returns the index of the minimum value along an axis.

**Usage**

```python
ek_argmin(x, axis = -1)
```

**Arguments**

- `x`: Tensor or variable.
- `axis`: Axis along which to perform the reduction (axis indexes are 1-based). Pass -1 (the default) to select the last axis.

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_backend**

| k_backend | Active Keras backend |

**Description**

Active Keras backend

**Usage**

```python
k_backend()
```

**Value**

The name of the backend Keras is currently using.
**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_batch_dot**

**Batchwise dot product.**

**Description**

`batch_dot` is used to compute dot product of `x` and `y` when `x` and `y` are data in batch, i.e. in a shape of `(batch_size)`. `batch_dot` results in a tensor or variable with less dimensions than the input. If the number of dimensions is reduced to 1, we use `expand_dims` to make sure that `ndim` is at least 2.

**Usage**

`k_batch_dot(x, y, axes)`

**Arguments**

- `x` Keras tensor or variable with 2 more more axes.
- `y` Keras tensor or variable with 2 or more axes
- `axes` List of (or single) integer with target dimensions (axis indexes are 1-based). The lengths of `axes[[1]]` and `axes[[2]]` should be the same.

**Value**

A tensor with shape equal to the concatenation of `x`’s shape (less the dimension that was summed over) and `y`’s shape (less the batch dimension and the dimension that was summed over). If the final rank is 1, we reshape it to `(batch_size, 1)`.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).
**k_batch_flatten**

* Turn a nD tensor into a 2D tensor with same 1st dimension.

**Description**

In other words, it flattens each data samples of a batch.

**Usage**

```
k_batch_flatten(x)
```

**Arguments**

- `x` A tensor or variable.

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_batch_get_value**

* Returns the value of more than one tensor variable.

**Description**

Returns the value of more than one tensor variable.

**Usage**

```
k_batch_get_value(ops)
```

**Arguments**

- `ops` List of ops to evaluate.

**Value**

A list of arrays.
**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

**See Also**

- `k_batch_set_value()`

---

**k_batch_normalization**  
Applies batch normalization on x given mean, var, beta and gamma.

**Description**

i.e. returns output <- (x - mean) / (sqrt(var) + epsilon) * gamma + beta

**Usage**

```python  
k_batch_normalization(x, mean, var, beta, gamma, axis = -1, epsilon = 0.001)  
```

**Arguments**

- **x**: Input tensor or variable.
- **mean**: Mean of batch.
- **var**: Variance of batch.
- **beta**: Tensor with which to center the input.
- **gamma**: Tensor by which to scale the input.
- **axis**: Axis (axis indexes are 1-based). Pass -1 (the default) to select the last axis.
- **epsilon**: Fuzz factor.

**Value**

A tensor.

---

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).
**k_batch_set_value**  
*Sets the values of many tensor variables at once.*

**Description**

Sets the values of many tensor variables at once.

**Usage**

```python
k_batch_set_value(lists)
```

**Arguments**

- `lists` a list of lists (tensor, value). value should be an R array.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

**See Also**

- `k_batch_get_value()`

---

**k_bias_add**  
*Adds a bias vector to a tensor.*

**Description**

Adds a bias vector to a tensor.

**Usage**

```python
k_bias_add(x, bias, data_format = NULL)
```

**Arguments**

- `x` Tensor or variable.
- `bias` Bias tensor to add.
- `data_format` string, "channels_last" or "channels_first".

**Value**

Output tensor.
Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

$k_{\text{binary\_crossentropy}}$  
**Binary crossentropy between an output tensor and a target tensor.**

---

**Description**

Binary crossentropy between an output tensor and a target tensor.

**Usage**

$k_{\text{binary\_crossentropy}}(\text{target, output, from\_logits = FALSE})$

**Arguments**

- **target**: A tensor with the same shape as output.
- **output**: A tensor.
- **from\_logits**: Whether output is expected to be a logits tensor. By default, we consider that output encodes a probability distribution.

**Value**

A tensor.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_cast**

Casts a tensor to a different dtype and returns it.

**Description**

You can cast a Keras variable but it still returns a Keras tensor.

**Usage**

```python
k_cast(x, dtype)
```

**Arguments**

- `x` : Keras tensor (or variable).
- `dtype` : String, either ('float16', 'float32', or 'float64').

**Value**

Keras tensor with dtype `dtype`.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_cast_to_floatx**

Cast an array to the default Keras float type.

**Description**

Cast an array to the default Keras float type.

**Usage**

```python
ek_cast_to_floatx(x)
```

**Arguments**

- `x` : Array.

**Value**

The same array, cast to its new type.
k_categorical_crossentropy

Categorical crossentropy between an output tensor and a target tensor.

Description

Categorical crossentropy between an output tensor and a target tensor.

Usage

k_categorical_crossentropy(target, output, from_logits = FALSE,
axis = -1)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>target</td>
<td>A tensor of the same shape as output.</td>
</tr>
<tr>
<td>output</td>
<td>A tensor resulting from a softmax (unless from_logits is TRUE, in which case output is expected to be the logits).</td>
</tr>
<tr>
<td>from_logits</td>
<td>Logical, whether output is the result of a softmax, or is a tensor of logits.</td>
</tr>
<tr>
<td>axis</td>
<td>Axis (axis indexes are 1-based). Pass -1 (the default) to select the last axis.</td>
</tr>
</tbody>
</table>

Value

Output tensor.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_clear_session**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destroys the current TF graph and creates a new one.</td>
</tr>
</tbody>
</table>

**Usage**

```
k_clear_session()
```

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_clip**

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element-wise value clipping.</td>
</tr>
</tbody>
</table>

**Usage**

```
k_clip(x, min_value, max_value)
```

**Arguments**

- `x` : Tensor or variable.
- `min_value` : Float or integer.
- `max_value` : Float or integer.

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).
**k_concatenate**

**Concatenates a list of tensors alongside the specified axis.**

**Description**

Concatenates a list of tensors alongside the specified axis.

**Usage**

```python
k_concatenate(tensors, axis = -1)
```

**Arguments**

- **tensors**: list of tensors to concatenate.
- **axis**: concatenation axis (axis indexes are 1-based). Pass -1 (the default) to select the last axis.

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

**k_constant**

**Creates a constant tensor.**

**Description**

Creates a constant tensor.

**Usage**

```python
k_constant(value, dtype = NULL, shape = NULL, name = NULL)
```

**Arguments**

- **value**: A constant value
- **dtype**: The type of the elements of the resulting tensor.
- **shape**: Optional dimensions of resulting tensor.
- **name**: Optional name for the tensor.
**k_conv1d**

**Value**

A Constant Tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_conv1d**

1D convolution.

**Description**

1D convolution.

**Usage**

```python
k_conv1d(x, kernel, strides = 1, padding = "valid",
data_format = NULL, dilation_rate = 1)
```

**Arguments**

- `x`: Tensor or variable.
- `kernel`: kernel tensor.
- `strides`: stride integer.
- `padding`: string, "same", "causal" or "valid".
- `data_format`: string, "channels_last" or "channels_first".
- `dilation_rate`: integer dilate rate.

**Value**

A tensor, result of 1D convolution.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).
**k_conv2d**  
2D convolution.

**Description**  
2D convolution.

**Usage**  
```python  
k_conv2d(x, kernel, strides = c(1, 1), padding = "valid",  
data_format = NULL, dilation_rate = c(1, 1))  ```

**Arguments**  
- `x`: Tensor or variable.  
- `kernel`: kernel tensor.  
- `strides`: strides  
- `padding`: string, "same" or "valid".  
- `data_format`: string, "channels_last" or "channels_first". Whether to use Theano or TensorFlow/CNTK data format for inputs/kernels/outputs.  
- `dilation_rate`: vector of 2 integers.

**Value**  
A tensor, result of 2D convolution.

**Keras Backend**  
This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).  
You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_conv2d_transpose**  
2D deconvolution (i.e. transposed convolution).

**Description**  
2D deconvolution (i.e. transposed convolution).

**Usage**  
```python  
k_conv2d_transpose(x, kernel, output_shape, strides = c(1, 1),  
padding = "valid", data_format = NULL)  ```
**Arguments**

- **x**: Tensor or variable.
- **kernel**: Kernel tensor.
- **output_shape**: 1D int tensor for the output shape.
- **strides**: Strides list.
- **padding**: String, "same" or "valid".
- **data_format**: String, "channels_last" or "channels_first". Whether to use Theano or TensorFlow/CNTK data format for inputs/kernels/outputs.

**Value**

A tensor, result of transposed 2D convolution.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_conv3d**

3D convolution.

**Description**

3D convolution.

**Usage**

```python
k_conv3d(x, kernel, strides = c(1, 1, 1), padding = "valid",
         data_format = NULL, dilation_rate = c(1, 1, 1))
```

**Arguments**

- **x**: Tensor or variable.
- **kernel**: Kernel tensor.
- **strides**: Strides.
- **padding**: String, "same" or "valid".
- **data_format**: String, "channels_last" or "channels_first". Whether to use Theano or TensorFlow/CNTK data format for inputs/kernels/outputs.
- **dilation_rate**: List of 3 integers.

**Value**

A tensor, result of 3D convolution.
Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

k_conv3dTranspose

3D deconvolution (i.e. transposed convolution).

Description

3D deconvolution (i.e. transposed convolution).

Usage

k_conv3dTranspose(x, kernel, output_shape, strides = c(1, 1, 1),
padding = "valid", data_format = NULL)

Arguments

x input tensor.

kernel kernel tensor.

output_shape 1D int tensor for the output shape.

strides strides

padding string, "same" or "valid".

data_format string, "channels_last" or "channels_first". Whether to use Theano or TensorFlow/CNTK data format for inputs/kernels/outputs.

Value

A tensor, result of transposed 3D convolution.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_cos**

*Computes cos of x element-wise.*

**Description**

Computes cos of x element-wise.

**Usage**

```
k_cos(x)
```

**Arguments**

- `x` Tensor or variable.

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_count_params**

*Returns the static number of elements in a Keras variable or tensor.*

**Description**

Returns the static number of elements in a Keras variable or tensor.

**Usage**

```
k_count_params(x)
```

**Arguments**

- `x` Keras variable or tensor.

**Value**

Integer, the number of elements in `x`, i.e., the product of the array's static dimensions.
Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_ctc_batch_cost**

*Runs CTC loss algorithm on each batch element.*

**Description**

Runs CTC loss algorithm on each batch element.

**Usage**

`k_ctc_batch_cost(y_true, y_pred, input_length, label_length)`

**Arguments**

- `y_true` tensor (samples, max_string_length) containing the truth labels.
- `y_pred` tensor (samples, time_steps, num_categories) containing the prediction, or output of the softmax.
- `input_length` tensor (samples, 1) containing the sequence length for each batch item in `y_pred`.
- `label_length` tensor (samples, 1) containing the sequence length for each batch item in `y_true`.

**Value**

Tensor with shape (samples,1) containing the CTC loss of each element.

---

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_ctc_decode**

Decodes the output of a softmax.

**Description**

Can use either greedy search (also known as best path) or a constrained dictionary search.

**Usage**

```python
k_ctc_decode(y_pred, input_length, greedy = TRUE, beam_width = 100L, top_paths = 1)
```

**Arguments**

- `y_pred` tensor (samples, time_steps, num_categories) containing the prediction, or output of the softmax.
- `input_length` tensor (samples, ) containing the sequence length for each batch item in `y_pred`.
- `greedy` perform much faster best-path search if TRUE. This does not use a dictionary.
- `beam_width` if `greedy` is FALSE: a beam search decoder will be used with a beam of this width.
- `top_paths` if `greedy` is FALSE, how many of the most probable paths will be returned.

**Value**

If `greedy` is TRUE, returns a list of one element that contains the decoded sequence. If FALSE, returns the `top_paths` most probable decoded sequences. Important: blank labels are returned as ~1. Tensor (top_paths) that contains the log probability of each decoded sequence.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).
k_ctc_label_dense_to_sparse

Converts CTC labels from dense to sparse.

Description

Converts CTC labels from dense to sparse.

Usage

k_ctc_label_dense_to_sparse(labels, label_lengths)

Arguments

labels dense CTC labels.
label_lengths length of the labels.

Value

A sparse tensor representation of the labels.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

k_cumprod

Cumulative product of the values in a tensor, alongside the specified axis.

Description

Cumulative product of the values in a tensor, alongside the specified axis.

Usage

k_cumprod(x, axis = 1)

Arguments

x A tensor or variable.
axis An integer, the axis to compute the product (axis indexes are 1-based).
**k_cumsum**

**Value**

A tensor of the cumulative product of values of x along axis.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

### k_cumsum

*Cumulative sum of the values in a tensor, alongside the specified axis.*

---

**Description**

Cumulative sum of the values in a tensor, alongside the specified axis.

**Usage**

`k_cumsum(x, axis = 1)`

**Arguments**

- **x**: A tensor or variable.
- **axis**: An integer, the axis to compute the sum (axis indexes are 1-based).

**Value**

A tensor of the cumulative sum of values of x along axis.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).
k_depthwise_conv2d

Description
Depthwise 2D convolution with separable filters.

Usage
```python
k_depthwise_conv2d(x, depthwise_kernel, strides = c(1, 1),
                   padding = "valid", data_format = NULL, dilation_rate = c(1, 1))
```

Arguments
- `x`: input tensor
- `depthwise_kernel`: convolution kernel for the depthwise convolution.
- `strides`: strides (length 2).
- `padding`: string, "same" or "valid".
- `data_format`: string, "channels_last" or "channels_first".
- `dilation_rate`: vector of integers, dilation rates for the separable convolution.

Value
Output tensor.

Keras Backend
This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).
You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

k_dot

Description
When attempting to multiply a nD tensor with a nD tensor, it reproduces the Theano behavior. (e.g. 
```python
(2, 3) * (4, 3, 5) -> (2, 4, 5)
```

Usage
```python
k_dot(x, y)
```


**k_dropout**

**Arguments**

- **x** Tensor or variable.
- **y** Tensor or variable.

**Value**

A tensor, dot product of `x` and `y`.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_dropout**

Sets entries in `x` to zero at random, while scaling the entire tensor.

**Description**

Sets entries in `x` to zero at random, while scaling the entire tensor.

**Usage**

```python
k_dropout(x, level, noise_shape = NULL, seed = NULL)
```

**Arguments**

- **x** tensor
- **level** fraction of the entries in the tensor that will be set to 0.
- **noise_shape** shape for randomly generated keep/drop flags, must be broadcastable to the shape of `x`
- **seed** random seed to ensure determinism.

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_dtype**

*Returns the dtype of a Keras tensor or variable, as a string.*

**Description**

Returns the dtype of a Keras tensor or variable, as a string.

**Usage**

```python
k_dtype(x)
```

**Arguments**

- `x`  
  Tensor or variable.

**Value**

String, dtype of `x`.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

**k_elu**

*Exponential linear unit.*

**Description**

Exponential linear unit.

**Usage**

```python
k_elu(x, alpha = 1)
```

**Arguments**

- `x`  
  A tensor or variable to compute the activation function for.
- `alpha`  
  A scalar, slope of negative section.

**Value**

A tensor.
**k_epsilon**

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

| k_epsilon | Fuzz factor used in numeric expressions. |

---

### Description

Fuzz factor used in numeric expressions.

### Usage

- `k_epsilon()`
- `k_set_epsilon(e)`

### Arguments

- `e` float. New value of epsilon.

---

**k_equal**

**Element-wise equality between two tensors.**

### Description

Element-wise equality between two tensors.

### Usage

- `k_equal(x, y)`

### Arguments

- `x` Tensor or variable.
- `y` Tensor or variable.
**Value**

A bool tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

`k_eval` 
*Evaluates the value of a variable.*

---

**Description**

Evaluates the value of a variable.

**Usage**

`k_eval(x)`

**Arguments**

- **x** A variable.

**Value**

An R array.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_exp**  
*Element-wise exponential.*

**Description**  
Element-wise exponential.

**Usage**  
k_exp(x)

**Arguments**  
x  
Tensor or variable.

**Value**  
A tensor.

**Keras Backend**  
This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.). You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions)

---

**k_expand_dims**  
*Adds a 1-sized dimension at index axis.*

**Description**  
Adds a 1-sized dimension at index axis.

**Usage**  
k_expand_dims(x, axis = -1)

**Arguments**  
x  
A tensor or variable.

axis  
Position where to add a new axis (axis indexes are 1-based). Pass -1 (the default) to select the last axis.

**Value**  
A tensor with expanded dimensions.
k_eye

**Description**

Instantiate an identity matrix and returns it.

**Usage**

k_eye(size, dtype = NULL, name = NULL)

**Arguments**

- size: Integer, number of rows/columns.
- dtype: String, data type of returned Keras variable.
- name: String, name of returned Keras variable.

**Value**

A Keras variable, an identity matrix.

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_flatten**

*Flatten a tensor.*

**Description**

Flatten a tensor.

**Usage**

```python
k_flatten(x)
```

**Arguments**

- `x`: A tensor or variable.

**Value**

A tensor, reshaped into 1-D

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_floatx**

*Default float type*

**Description**

Default float type

**Usage**

```python
k_floatx()
```

**k_set_floatx(floatx)**

**Arguments**

- `floatx`: String, 'float16', 'float32', or 'float64'.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_foldl**

*Reduce elems using fn to combine them from left to right.*

---

**Description**

Reduce elems using fn to combine them from left to right.

**Usage**

```
k_foldl(fn, elems, initializer = NULL, name = NULL)
```

**Arguments**

- `fn` Function that will be called upon each element in elems and an accumulator
- `elems` tensor
- `initializer` The first value used (first element of `elems` in case of `NULL`)
- `name` A string name for the foldl node in the graph

**Value**

Tensor with same type and shape as `initializer`.

---

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_foldr**

Reduce `elems` using `fn` to combine them from right to left.

---

**Description**

Reduce `elems` using `fn` to combine them from right to left.

**Usage**

```plaintext
k_foldr(fn, elems, initializer = NULL, name = NULL)
```

**Arguments**

- **fn**: Function that will be called upon each element in `elems` and an accumulator
- **elems**: tensor
- **initializer**: The first value used (last element of `elems` in case of `NULL`)
- **name**: A string name for the foldr node in the graph

**Value**

Tensor with same type and shape as `initializer`.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_function**

Instantiates a Keras function

---

**Description**

Instantiates a Keras function

**Usage**

```plaintext
k_function(inputs, outputs, updates = NULL, ...)
```

**Arguments**

- **inputs**: List of placeholder tensors.
- **outputs**: List of output tensors.
- **updates**: List of update ops.
- **...**: Named arguments passed to `tf$Session$run`. 
Value

Output values as R arrays.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

\textbf{k\_gather} \texttt{(reference, indices)}

\begin{itemize}
  \item \texttt{reference} A tensor.
  \item \texttt{indices} Indices. Dimension indices are 1-based. Note however that if you pass a tensor for \texttt{indices} they will be passed as-is, in which case indices will be 0 based because no normalizing of R 1-based axes to Python 0-based axes is performed.
\end{itemize}

Value

A tensor of same type as \texttt{reference}.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_get_session**  
*TF session to be used by the backend.*

**Description**

If a default TensorFlow session is available, we will return it. Else, we will return the global Keras session. If no global Keras session exists at this point: we will create a new global session. Note that you can manually set the global session via `k_set_session()`.

**Usage**

```python
k_get_session()

k_set_session(session)
```

**Arguments**

- **session**  
  A TensorFlow Session.

**Value**

A TensorFlow session

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_get_uid**  
*Get the uid for the default graph.*

**Description**

Get the uid for the default graph.

**Usage**

```python
k_get_uid(prefix = ")
```

**Arguments**

- **prefix**  
  An optional prefix of the graph.
Value

A unique identifier for the graph.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_get_value**

Returns the value of a variable.

Description

Returns the value of a variable.

Usage

k_get_value(x)

Arguments

x input variable.

Value

An R array.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_get_variable_shape**

* Returns the shape of a variable.

**Description**

Returns the shape of a variable.

**Usage**

```python
k_get_variable_shape(x)
```

**Arguments**

- `x`: A variable.

**Value**

A vector of integers.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_gradients**

* Returns the gradients of variables w.r.t. loss.

**Description**

Returns the gradients of variables w.r.t. loss.

**Usage**

```python
k_gradients(loss, variables)
```

**Arguments**

- `loss`: Scalar tensor to minimize.
- `variables`: List of variables.

**Value**

A gradients tensor.
Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_greater**

*Element-wise truth value of (x > y).*

**Description**

Element-wise truth value of (x > y).

**Usage**

```
k_greater(x, y)
```

**Arguments**

- `x`: Tensor or variable.
- `y`: Tensor or variable.

**Value**

A bool tensor.

---

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_greater_equal**

*Element-wise truth value of (x >= y).*

**Description**

Element-wise truth value of (x >= y).

**Usage**

```
k_greater_equal(x, y)
```
**k_hard_sigmoid**

**Arguments**
- **x**: Tensor or variable.
- **y**: Tensor or variable.

**Value**
A bool tensor.

**Keras Backend**
This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_hard_sigmoid**

*Segment-wise linear approximation of sigmoid.*

**Description**
Faster than sigmoid. Returns 0. if \( x < -2.5 \), 1. if \( x > 2.5 \). In \(-2.5 \leq x \leq 2.5\), returns \( 0.2 * x + 0.5 \).

**Usage**

```
k_hard_sigmoid(x)
```

**Arguments**
- **x**: A tensor or variable.

**Value**
A tensor.

**Keras Backend**
This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).
k_identity

*Returns a tensor with the same content as the input tensor.*

**Description**

Returns a tensor with the same content as the input tensor.

**Usage**

```python
k_identity(x, name = NULL)
```

**Arguments**

- **x**: The input tensor.
- **name**: String, name for the variable to create.

**Value**

A tensor of the same shape, type and content.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.). You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

k_image_data_format

*Default image data format convention ('channels_first' or 'channels_last').*

**Description**

Default image data format convention ('channels_first' or 'channels_last').

**Usage**

```python
k_image_data_format()
```

```python
k_set_image_data_format(data_format)
```

**Arguments**

- **data_format**: string, 'channels_first' or 'channels_last'. 
**k_int_shape**

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_int_shape**  
*Returns the shape of tensor or variable as a list of int or NULL entries.*

---

**Description**

Returns the shape of tensor or variable as a list of int or NULL entries.

**Usage**

k_int_shape(x)

**Arguments**

- **x**  
  Tensor or variable.

**Value**

A list of integers (or NULL entries).

---

**k_in_test_phase**  
*Selects x in test phase, and alt otherwise.*

---

**Description**

Note that alt should have the *same shape* as x.

**Usage**

k_in_test_phase(x, alt, training = NULL)
**k_in_top_k**

**Arguments**

- **x**: What to return in test phase (tensor or function that returns a tensor).
- **alt**: What to return otherwise (tensor or function that returns a tensor).
- **training**: Optional scalar tensor (or R logical or integer) specifying the learning phase.

**Value**

Either `x` or `alt` based on `k_learning_phase()`.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**Description**

Returns whether the targets are in the top k predictions.

**Usage**

```r
k_in_top_k(predictions, targets, k)
```

**Arguments**

- **predictions**: A tensor of shape `(batch_size, classes)` and type `float32`.
- **targets**: A 1D tensor of length `batch_size` and type `int32` or `int64`.
- **k**: An int, number of top elements to consider.

**Value**

A 1D tensor of length `batch_size` and type `bool`. `output[i]` is `TRUE` if `predictions[i, targets[i]] is within top-k values of predictions[[i]]`.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
k_in_train_phase

Selects x in train phase, and alt otherwise.

**Description**

Note that alt should have the same shape as x.

**Usage**

`k_in_train_phase(x, alt, training = NULL)`

**Arguments**

- **x**: What to return in train phase (tensor or function that returns a tensor).
- **alt**: What to return otherwise (tensor or function that returns a tensor).
- **training**: Optional scalar tensor (or R logical or integer) specifying the learning phase.

**Value**

Either x or alt based on the training flag. the training flag defaults to `k_learning_phase()`.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

k_is_keras_tensor

Returns whether x is a Keras tensor.

**Description**

A "Keras tensor" is a tensor that was returned by a Keras layer

**Usage**

`k_is_keras_tensor(x)`

**Arguments**

- **x**: A candidate tensor.

**Value**

A logical: Whether the argument is a Keras tensor.
**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

### k_is_placeholder

**Description**

Returns whether \( x \) is a placeholder.

**Usage**

\[
k_{is\_placeholder}(x)
\]

**Arguments**

\( x \)

A candidate placeholder.

**Value**

A logical

---

### k_is_sparse

**Description**

Returns whether a tensor is a sparse tensor.

**Usage**

\[
k_{is\_sparse}(\text{tensor})
\]

**Arguments**

\( \text{tensor} \)

A tensor instance.
**k_is_tensor**

**Value**

A logical

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_is_tensor**  
Returns whether \( x \) is a symbolic tensor.

**Description**

Returns whether \( x \) is a symbolic tensor.

**Usage**

\[
\text{k_is_tensor}(x)
\]

**Arguments**

\( x \)  
A candidate tensor.

**Value**

A logical: Whether the argument is a symbolic tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
### k_l2_normalize

Normalizes a tensor wrt the L2 norm alongside the specified axis.

**Description**

Normalizes a tensor wrt the L2 norm alongside the specified axis.

**Usage**

```python
k_l2_normalize(x, axis = NULL)
```

**Arguments**

- **x**: Tensor or variable.
- **axis**: Axis along which to perform normalization (axis indexes are 1-based).

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

### k_learning_phase

Returns the learning phase flag.

**Description**

The learning phase flag is a bool tensor (0 = test, 1 = train) to be passed as input to any Keras function that uses a different behavior at train time and test time.

**Usage**

```python
k_learning_phase()
```

**Value**

Learning phase (scalar integer tensor or R integer).
k_less

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

| k_less | Element-wise truth value of \((x < y)\). |

Description

Element-wise truth value of \((x < y)\).

Usage

\(\text{k_less}(x, y)\)

Arguments

<table>
<thead>
<tr>
<th>x</th>
<th>Tensor or variable.</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>Tensor or variable.</td>
</tr>
</tbody>
</table>

Value

A bool tensor.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

| k_less_equal | Element-wise truth value of \((x <= y)\). |

Description

Element-wise truth value of \((x <= y)\).

Usage

\(\text{k_less_equal}(x, y)\)
Arguments

\textit{x} \quad \text{Tensor or variable.}
\textit{y} \quad \text{Tensor or variable.}

Value

A bool tensor.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

\begin{verbatim}

k_local_conv1d(inputs, kernel, kernel_size, strides, data_format = NULL)

\end{verbatim}

Description

Apply 1D conv with un-shared weights.

Usage

\texttt{k_local_conv1d(inputs, kernel, kernel_size, strides, data_format = NULL)}

Arguments

\begin{itemize}
  \item \texttt{inputs} \quad 3D tensor with shape: (batch_size, steps, input_dim)
  \item \texttt{kernel} \quad the unshared weight for convolution, with shape (output_length, feature_dim, filters)
  \item \texttt{kernel_size} \quad a list of a single integer, specifying the length of the 1D convolution window
  \item \texttt{strides} \quad a list of a single integer, specifying the stride length of the convolution
  \item \texttt{data_format} \quad the data format, channels_first or channels_last
\end{itemize}

Value

the tensor after 1d conv with un-shared weights, with shape (batch_size, output_length, filters)

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_local_conv2d**

Describe 2D conv with un-shared weights.

**Usage**

```python
k_local_conv2d(inputs, kernel, kernel_size, strides, output_shape,
    data_format = NULL)
```

**Arguments**

- **inputs**: 4D tensor with shape: (batch_size, filters, new_rows, new_cols) if data_format='channels_first' or 4D tensor with shape: (batch_size, new_rows, new_cols, filters) if data_format='channels_last'.
- **kernel**: the unshared weight for convolution, with shape (output_items, feature dim, filters)
- **kernel_size**: a list of 2 integers, specifying the width and height of the 2D convolution window.
- **strides**: a list of 2 integers, specifying the strides of the convolution along the width and height.
- **output_shape**: a list with (output_row, output_col)
- **data_format**: the data format, channels_first or channels_last

**Value**

A 4d tensor with shape: (batch_size, filters, new_rows, new_cols) if data_format='channels_first' or 4D tensor with shape: (batch_size, new_rows, new_cols, filters) if data_format='channels_last'.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_log**

*Element-wise log.*

**Description**
Element-wise log.

**Usage**
`k_log(x)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x</code></td>
<td>Tensor or variable.</td>
</tr>
</tbody>
</table>

**Value**
A tensor.

**Keras Backend**
This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).
You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_logsumexp**

*Computes log(sum(exp(elements across dimensions of a tensor))).*

**Description**
This function is more numerically stable than `log(sum(exp(x)))`. It avoids overflows caused by taking the exp of large inputs and underflows caused by taking the log of small inputs.

**Usage**
`k_logsumexp(x, axis = NULL, keepdims = FALSE)`

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>x</code></td>
<td>A tensor or variable.</td>
</tr>
<tr>
<td><code>axis</code></td>
<td>An integer, the axis to reduce over (axis indexes are 1-based).</td>
</tr>
<tr>
<td><code>keepdims</code></td>
<td>A boolean, whether to keep the dimensions or not. If keepdims is FALSE, the rank of the tensor is reduced by 1. If keepdims is TRUE, the reduced dimension is retained with length 1.</td>
</tr>
</tbody>
</table>
Value

The reduced tensor.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_manual_variable_initialization**

Sets the manual variable initialization flag.

Description

This boolean flag determines whether variables should be initialized as they are instantiated (default), or if the user should handle the initialization (e.g. via `tf$initialize_all_variables()`).

Usage

```r
k_manual_variable_initialization(value)
```

Arguments

- `value` Logical

---

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
### k_map_fn

**Map the function fn over the elements elems and return the outputs.**

**Description**

Map the function fn over the elements elems and return the outputs.

**Usage**

```python
k_map_fn(fn, elems, name = NULL, dtype = NULL)
```

**Arguments**

- **fn**: Function that will be called upon each element in elems
- **elems**: tensor
- **name**: A string name for the map node in the graph
- **dtype**: Output data type.

**Value**

Tensor with dtype dtype.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

### k_max

**Maximum value in a tensor.**

**Description**

Maximum value in a tensor.

**Usage**

```python
k_max(x, axis = NULL, keepdims = FALSE)
```
**k_maximum**

**Arguments**

- `x`: A tensor or variable.
- `axis`: An integer, the axis to find maximum values (axis indexes are 1-based).
- `keepdims`: A boolean, whether to keep the dimensions or not. If `keepdims` is FALSE, the rank of the tensor is reduced by 1. If `keepdims` is TRUE, the reduced dimension is retained with length 1.

**Value**

A tensor with maximum values of `x`.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_maximum**

Element-wise maximum of two tensors.

**Description**

Element-wise maximum of two tensors.

**Usage**

```python
k_maximum(x, y)
```

**Arguments**

- `x`: Tensor or variable.
- `y`: Tensor or variable.

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).
**k_mean**

*Mean of a tensor, alongside the specified axis.*

**Description**

Mean of a tensor, alongside the specified axis.

**Usage**

```python
k_mean(x, axis = NULL, keepdims = FALSE)
```

**Arguments**

- `x`: A tensor or variable.
- `axis`: A list of axes to compute the mean over (axis indexes are 1-based).
- `keepdims`: A boolean, whether to keep the dimensions or not. If `keepdims` is FALSE, the rank of the tensor is reduced by 1 for each entry in `axis`. If `keep_dims` is TRUE, the reduced dimensions are retained with length 1.

**Value**

A tensor with the mean of elements of `x`.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

**k_min**

*Minimum value in a tensor.*

**Description**

Minimum value in a tensor.

**Usage**

```python
k_min(x, axis = NULL, keepdims = FALSE)
```
**Arguments**

- **x**
  A tensor or variable.

- **axis**
  An integer, axis to find minimum values (axis indexes are 1-based).

- **keepdims**
  A boolean, whether to keep the dimensions or not. If `keepdims` is FALSE, the rank of the tensor is reduced by 1. If `keepdims` is TRUE, the reduced dimension is retained with length 1.

**Value**

A tensor with minimum values of `x`.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**Description**

Element-wise minimum of two tensors.

**Usage**

```
k_minimum(x, y)
```

**Arguments**

- **x**
  Tensor or variable.

- **y**
  Tensor or variable.

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).
**k_moving_average_update**

*Compute the moving average of a variable.*

**Description**

Compute the moving average of a variable.

**Usage**

```python
k_moving_average_update(x, value, momentum)
```

**Arguments**

- `x`: A Variable.
- `value`: A tensor with the same shape as `x`.
- `momentum`: The moving average momentum.

**Value**

An operation to update the variable.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_ndim**

*Returs the number of axes in a tensor, as an integer.*

**Description**

Returns the number of axes in a tensor, as an integer.

**Usage**

```python
k_ndim(x)
```

**Arguments**

- `x`: Tensor or variable.
Value

Integer (scalar), number of axes.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_normalize_batch_in_training**

*Computes mean and std for batch then apply batch_normalization on batch.*

---

**Description**

Computes mean and std for batch then apply batch_normalization on batch.

**Usage**

```python
k_normalize_batch_in_training(x, gamma, beta, reduction_axes, epsilon = 0.001)
```

**Arguments**

- **x**: Input tensor or variable.
- **gamma**: Tensor by which to scale the input.
- **beta**: Tensor with which to center the input.
- **reduction_axes**: iterable of integers, axes over which to normalize.
- **epsilon**: Fuzz factor.

**Value**

A list length of 3, `(normalized_tensor, mean, variance)`. 

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_not_equal**

*Element-wise inequality between two tensors.*

**Description**

Element-wise inequality between two tensors.

**Usage**

```python
k_not_equal(x, y)
```

**Arguments**

- `x` Tensor or variable.
- `y` Tensor or variable.

**Value**

A bool tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_ones**

*Instantiates an all-ones tensor variable and returns it.*

**Description**

Instantiates an all-ones tensor variable and returns it.

**Usage**

```python
k_ones(shape, dtype = NULL, name = NULL)
```

**Arguments**

- `shape` Tuple of integers, shape of returned Keras variable.
- `dtype` String, data type of returned Keras variable.
- `name` String, name of returned Keras variable.
**k.ones_like**

**Value**

A Keras variable, filled with 1.0.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k.ones_like**

*Instantiates an all-ones variable of the same shape as another tensor.*

**Description**

Instantiates an all-ones variable of the same shape as another tensor.

**Usage**

```
k.ones_like(x, dtype = NULL, name = NULL)
```

**Arguments**

- **x**
  
  Keras variable or tensor.

- **dtype**
  
  String, dtype of returned Keras variable. NULL uses the dtype of x.

- **name**
  
  String, name for the variable to create.

**Value**

A Keras variable with the shape of x filled with ones.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_one_hot**

*Computes the one-hot representation of an integer tensor.*

**Description**

Computes the one-hot representation of an integer tensor.

**Usage**

```
k_one_hot(indices, num_classes)
```

**Arguments**

- **indices**
  - nD integer tensor of shape `(batch_size, dim1, dim2, ... dim(n-1))`
- **num_classes**
  - Integer, number of classes to consider.

**Value**

`(n + 1)D one hot representation of the input with shape (batch_size, dim1, dim2, ... dim(n-1), num_classes)`

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_permute_dimensions**

*Permutations axes in a tensor.*

**Description**

Permutations axes in a tensor.

**Usage**

```
k_permute_dimensions(x, pattern)
```

**Arguments**

- **x**
  - Tensor or variable.
- **pattern**
  - A list of dimension indices, e.g. `(1, 3, 2)`. Dimension indices are 1-based.

**Value**

A tensor.
**k_placeholder**

**Instantiates a placeholder tensor and returns it.**

**Description**

Instantiates a placeholder tensor and returns it.

**Usage**

```python
k_placeholder(shape = NULL, ndim = NULL, dtype = NULL,
               sparse = FALSE, name = NULL)
```

**Arguments**

- `shape`: Shape of the placeholder (integer list, may include NULL entries).
- `ndim`: Number of axes of the tensor. At least one of `shape`, `ndim` must be specified. If both are specified, `shape` is used.
- `dtype`: Placeholder type.
- `sparse`: Logical, whether the placeholder should have a sparse type.
- `name`: Optional name string for the placeholder.

**Value**

Tensor instance (with Keras metadata included).

---

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).
### k_pool2d  
2D Pooling.

#### Description
2D Pooling.

#### Usage
```python
k_pool2d(x, pool_size, strides = c(1, 1), padding = "valid",
         data_format = NULL, pool_mode = "max")
```

#### Arguments
- **x**: Tensor or variable.
- **pool_size**: list of 2 integers.
- **strides**: list of 2 integers.
- **padding**: string, "same" or "valid".
- **data_format**: string, "channels_last" or "channels_first".
- **pool_mode**: string, "max" or "avg".

#### Value
A tensor, result of 2D pooling.

#### Keras Backend
This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

### k_pool3d  
3D Pooling.

#### Description
3D Pooling.

#### Usage
```python
k_pool3d(x, pool_size, strides = c(1, 1, 1), padding = "valid",
         data_format = NULL, pool_mode = "max")
```
Arguments

- x: Tensor or variable.
- pool_size: list of 3 integers.
- strides: list of 3 integers.
- padding: string, "same" or "valid".
- data_format: string, "channels_last" or "channels_first".
- pool_mode: string, "max" or "avg".

Value

A tensor, result of 3D pooling.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

k_pow

Element-wise exponentiation.

Description

Element-wise exponentiation.

Usage

k_pow(x, a)

Arguments

- x: Tensor or variable.
- a: R integer.

Value

A tensor.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_print_tensor**

*Prints message and the tensor value when evaluated.*

**Description**

Note that `print_tensor` returns a new tensor identical to `x` which should be used in the following code. Otherwise the print operation is not taken into account during evaluation.

**Usage**

```python
k_print_tensor(x, message = '')
```

**Arguments**

- `x`: Tensor to print.
- `message`: Message to print jointly with the tensor.

**Value**

The same tensor `x`, unchanged.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_prod**

*Multiplies the values in a tensor, alongside the specified axis.*

**Description**

Multiplies the values in a tensor, alongside the specified axis.

**Usage**

```python
k_prod(x, axis = NULL, keepdims = FALSE)
```

**Arguments**

- `x`: A tensor or variable.
- `axis`: An integer, axis to compute the product over (axis indexes are 1-based).
- `keepdims`: A boolean, whether to keep the dimensions or not. If `keepdims` is `FALSE`, the rank of the tensor is reduced by 1. If `keepdims` is `TRUE`, the reduced dimension is retained with length 1.
**k_random_binomial**

**Value**

A tensor with the product of elements of \( x \).

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_random_binomial**  
*Returns a tensor with random binomial distribution of values.*

---

**Description**

Returns a tensor with random binomial distribution of values.

**Usage**

\[
\text{k_random_binomial}(\text{shape}, \ p = 0, \ \text{dtype} = \text{NULL}, \ \text{seed} = \text{NULL})
\]

**Arguments**

- **shape**  
  A list of integers, the shape of tensor to create.
- **p**  
  A float, \( 0. \leq \ p \leq 1 \), probability of binomial distribution.
- **dtype**  
  String, dtype of returned tensor.
- **seed**  
  Integer, random seed.

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).
**k_random_normal**

*Returns a tensor with normal distribution of values.*

**Description**

Returns a tensor with normal distribution of values.

**Usage**

```python
k_random_normal(shape, mean = 0, stddev = 1, dtype = NULL,
               seed = NULL)
```

**Arguments**

- **shape**: A list of integers, the shape of tensor to create.
- **mean**: A float, mean of the normal distribution to draw samples.
- **stddev**: A float, standard deviation of the normal distribution to draw samples.
- **dtype**: String, dtype of returned tensor.
- **seed**: Integer, random seed.

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_random_normal_variable**

*Instantiates a variable with values drawn from a normal distribution.*

**Description**

Instantiates a variable with values drawn from a normal distribution.

**Usage**

```python
k_random_normal_variable(shape, mean, scale, dtype = NULL, name = NULL,
                         seed = NULL)
```
**k_random_uniform**

**Arguments**

- **shape**: Tuple of integers, shape of returned Keras variable.
- **mean**: Float, mean of the normal distribution.
- **scale**: Float, standard deviation of the normal distribution.
- **dtype**: String, dtype of returned Keras variable.
- **name**: String, name of returned Keras variable.
- **seed**: Integer, random seed.

**Value**

A Keras variable, filled with drawn samples.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**Description**

Returns a tensor with uniform distribution of values.

**Usage**

```python
k_random_uniform(shape, minval = 0, maxval = 1, dtype = NULL, seed = NULL)
```

**Arguments**

- **shape**: A list of integers, the shape of tensor to create.
- **minval**: A float, lower boundary of the uniform distribution to draw samples.
- **maxval**: A float, upper boundary of the uniform distribution to draw samples.
- **dtype**: String, dtype of returned tensor.
- **seed**: Integer, random seed.

**Value**

A tensor.
**k_random_uniform_variable**

Inherits a variable with values drawn from a uniform distribution.

**Description**

Inherits a variable with values drawn from a uniform distribution.

**Usage**

```python
k_random_uniform_variable(shape, low, high, dtype = NULL, name = NULL, seed = NULL)
```

**Arguments**

- **shape**: Tuple of integers, shape of returned Keras variable.
- **low**: Float, lower boundary of the output interval.
- **high**: Float, upper boundary of the output interval.
- **dtype**: String, dtype of returned Keras variable.
- **name**: String, name of returned Keras variable.
- **seed**: Integer, random seed.

**Value**

A Keras variable, filled with drawn samples.

---

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).
**k_relu**

*Rectified linear unit.*

**Description**

With default values, it returns element-wise $\text{max}(x, 0)$.

**Usage**

$$k\_relu(x, \alpha = 0, \text{max\_value} = \text{NULL})$$

**Arguments**

- **x**: A tensor or variable.
- **alpha**: A scalar, slope of negative section (default=0).  
- **max\_value**: Saturation threshold.

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_repeat**

*Repeats a 2D tensor.*

**Description**

If $x$ has shape (samples, dim) and $n$ is 2, the output will have shape (samples, 2, dim).

**Usage**

$$k\_repeat(x, n)$$

**Arguments**

- **x**: Tensor or variable.
- **n**: Integer, number of times to repeat.
**k_repeat_elements**

**Value**

A tensor

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_repeat_elements**  
*Repeats the elements of a tensor along an axis.*

**Description**

If \( x \) has shape \((s_1, s_2, s_3)\) and \( \text{axis} \) is 2, the output will have shape \((s_1, s_2 \times \text{rep}, s_3)\).

**Usage**

\[
\text{k_repeat_elements}(x, \text{rep}, \text{axis})
\]

**Arguments**

- \( x \)  
  Tensor or variable.
- \( \text{rep} \)  
  Integer, number of times to repeat.
- \( \text{axis} \)  
  Axis along which to repeat (axis indexes are 1-based)

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_reset_uids**

Reset graph identifiers.

**Description**

Reset graph identifiers.

**Usage**

`k_reset_uids()`

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_reshape**

Reshapes a tensor to the specified shape.

**Description**

Reshapes a tensor to the specified shape.

**Usage**

`k_reshape(x, shape)`

**Arguments**

- `x` : Tensor or variable.
- `shape` : Target shape list.

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).
k_resize_images

Resizes the images contained in a 4D tensor.

Description

Resizes the images contained in a 4D tensor.

Usage

k_resize_images(x, height_factor, width_factor, data_format)

Arguments

x
Tensor or variable to resize.

height_factor
Positive integer.

width_factor
Positive integer.

data_format
string, "channels_last" or "channels_first".

Value

A tensor.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

k_resize_volumes

Resizes the volume contained in a 5D tensor.

Description

Resizes the volume contained in a 5D tensor.

Usage

k_resize_volumes(x, depth_factor, height_factor, width_factor, data_format)
**k_reverse**

**Arguments**

- **x**: Tensor or variable to resize.
- **depth_factor**: Positive integer.
- **height_factor**: Positive integer.
- **width_factor**: Positive integer.
- **data_format**: string, "channels_last" or "channels_first".

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

**Description**

Reverse a tensor along the specified axes.

**Usage**

\[ k_{\text{reverse}}(x, \text{axes}) \]

**Arguments**

- **x**: Tensor to reverse.
- **axes**: Integer or list of integers of axes to reverse (axis indexes are 1-based).

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).
Iterates over the time dimension of a tensor

Description
Iterates over the time dimension of a tensor

Usage

```
k_rnn(step_function, inputs, initial_states, go_backwards = FALSE,
       mask = NULL, constants = NULL, unroll = FALSE,
       input_length = NULL)
```

Arguments

- `step_function`: RNN step function.
- `inputs`: Tensor with shape (samples, ...) (no time dimension), representing input for the batch of samples at a certain time step.
- `initial_states`: Tensor with shape (samples, output_dim) (no time dimension), containing the initial values for the states used in the step function.
- `go_backwards`: Logical If TRUE, do the iteration over the time dimension in reverse order and return the reversed sequence.
- `mask`: Binary tensor with shape (samples, time, 1), with a zero for every element that is masked.
- `constants`: A list of constant values passed at each step.
- `unroll`: Whether to unroll the RNN or to use a symbolic loop (while_loop or scan depending on backend).
- `input_length`: Not relevant in the TensorFlow implementation. Must be specified if using unrolling with Theano.

Value
A list with:

- `last_output`: the latest output of the rnn, of shape (samples, ...)
- `outputs`: tensor with shape (samples, time, ...) where each entry `outputs[s, t]` is the output of the step function at time `t` for sample `s`.
- `new_states`: list of tensors, latest states returned by the step function, of shape (samples, ...).

Keras Backend
This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
### k_round

**Element-wise rounding to the closest integer.**

**Description**

In case of tie, the rounding mode used is "half to even".

**Usage**

```
k_round(x)
```

**Arguments**

- `x` Tensor or variable.

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

### k_separable_conv2d

**2D convolution with separable filters.**

**Description**

2D convolution with separable filters.

**Usage**

```
k_separable_conv2d(x, depthwise_kernel, pointwise_kernel, strides = c(1, 1), padding = "valid", data_format = NULL, dilation_rate = c(1, 1))
```
**Arguments**

- **x**  
  input tensor

- **depthwise_kernel**  
  convolution kernel for the depthwise convolution.

- **pointwise_kernel**  
  kernel for the 1x1 convolution.

- **strides**  
  strides list (length 2).

- **padding**  
  string, "same" or "valid".

- **data_format**  
  string, "channels_last" or "channels_first".

- **dilation_rate**  
  list of integers, dilation rates for the separable convolution.

**Value**

Output tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_set_learning_phase**  
*Sets the learning phase to a fixed value.*

---

**Description**

Sets the learning phase to a fixed value.

**Usage**

```
k_set_learning_phase(value)
```

**Arguments**

- **value**  
  Learning phase value, either 0 or 1 (integers).

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_set_value**

Sets the value of a variable, from an R array.

**Description**

Sets the value of a variable, from an R array.

**Usage**

```python
k_set_value(x, value)
```

**Arguments**

- `x`: Tensor to set to a new value.
- `value`: Value to set the tensor to, as an R array (of the same shape).

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

**k_shape**

Returns the symbolic shape of a tensor or variable.

**Description**

Returns the symbolic shape of a tensor or variable.

**Usage**

```python
k_shape(x)
```

**Arguments**

- `x`: A tensor or variable.

**Value**

A symbolic shape (which is itself a tensor).
Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_sigmoid**

*Element-wise sigmoid.*

**Description**

Element-wise sigmoid.

**Usage**

```python
k_sigmoid(x)
```

**Arguments**

- **x**
  - A tensor or variable.

**Value**

A tensor.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_sign**

*Element-wise sign.*

**Description**

Element-wise sign.

**Usage**

```python
k_sign(x)
```

**Arguments**

- **x**
  - Tensor or variable.
**k_sin**

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

```
.. code-block:: python

    k_sin(x)

Arguments

- x: Tensor or variable.

Value

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_softmax**  
*Softmax of a tensor.*

**Description**

Softmax of a tensor.

**Usage**

```python
k_softmax(x, axis = -1)
```

**Arguments**

- `x`: A tensor or variable.
- `axis`: The dimension softmax would be performed on. The default is -1 which indicates the last dimension.

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_softplus**  
*Softplus of a tensor.*

**Description**

Softplus of a tensor.

**Usage**

```python
k_softplus(x)
```

**Arguments**

- `x`: A tensor or variable.

**Value**

A tensor.
**k_softsign**

Softsign of a tensor.

**Description**

Softsign of a tensor.

**Usage**

```
k_softsign(x)
```

**Arguments**

- `x` A tensor or variable.

**Value**

A tensor.

---

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_sparse_categorical_crossentropy**

Categorical crossentropy with integer targets.

**Description**

Categorical crossentropy with integer targets.

**Usage**

```
k_sparse_categorical_crossentropy(target, output, from_logits = FALSE, axis = -1)
```
**k_spatial_2d_padding**

**Arguments**

- **target**: An integer tensor.
- **output**: A tensor resulting from a softmax (unless `from_logits` is TRUE, in which case output is expected to be the logits).
- **from_logits**: Boolean, whether output is the result of a softmax, or is a tensor of logits.
- **axis**: Axis (axis indexes are 1-based). Pass -1 (the default) to select the last axis.

**Value**

Output tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_spatial_2d_padding**  
**Pads the 2nd and 3rd dimensions of a 4D tensor.**

**Description**

Pads the 2nd and 3rd dimensions of a 4D tensor.

**Usage**

```
k_spatial_2d_padding(x, padding = list(list(1, 1), list(1, 1)),
                      data_format = NULL)
```

**Arguments**

- **x**: Tensor or variable.
- **padding**: Tuple of 2 lists, padding pattern.
- **data_format**: string, "channels_last" or "channels_first".

**Value**

A padded 4D tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).
**k_spatial_3d_padding**  
*Pads 5D tensor with zeros along the depth, height, width dimensions.*

**Description**

Pads these dimensions with respectively padding[1], padding[2], and padding[3] zeros left and right. For 'channels_last' data_format, the 2nd, 3rd and 4th dimension will be padded. For 'channels_first' data_format, the 3rd, 4th and 5th dimension will be padded.

**Usage**

```
k_spatial_3d_padding(x, padding = list(list(1, 1), list(1, 1), list(1, 1)), data_format = NULL)
```

**Arguments**

- `x`  
  Tensor or variable.

- `padding`  
  List of 3 lists, padding pattern.

- `data_format`  
  string, "channels_last" or "channels_first".

**Value**

A padded 5D tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_sqrt**  
*Element-wise square root.*

**Description**

Element-wise square root.

**Usage**

```
k_sqrt(x)
```

**Arguments**

- `x`  
  Tensor or variable.
**k_square**

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**k_square** Element-wise square.

---

**Description**

Element-wise square.

**Usage**

k_square(x)

**Arguments**

x Tensor or variable.

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_squeeze**

*Removes a 1-dimension from the tensor at index axis.*

**Description**

Removes a 1-dimension from the tensor at index axis.

**Usage**

`k_squeeze(x, axis)`

**Arguments**

- `x` A tensor or variable.
- `axis` Axis to drop (axis indexes are 1-based).

**Value**

A tensor with the same data as x but reduced dimensions.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_stack**

*Stacks a list of rank R tensors into a rank R+1 tensor.*

**Description**

Stacks a list of rank R tensors into a rank R+1 tensor.

**Usage**

`k_stack(x, axis = 1)`

**Arguments**

- `x` List of tensors.
- `axis` Axis along which to perform stacking (axis indexes are 1-based).

**Value**

A tensor.
**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_std**  
*Standard deviation of a tensor, alongside the specified axis.*

**Description**

Standard deviation of a tensor, alongside the specified axis.

**Usage**

```
k_std(x, axis = NULL, keepdims = FALSE)
```

**Arguments**

- `x`  
  A tensor or variable.

- `axis`  
  An integer, the axis to compute the standard deviation over (axis indexes are 1-based).

- `keepdims`  
  A boolean, whether to keep the dimensions or not. If `keepdims` is FALSE, the rank of the tensor is reduced by 1. If `keepdims` is TRUE, the reduced dimension is retained with length 1.

**Value**

A tensor with the standard deviation of elements of `x`.

---

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).
**k_stop_gradient**

`k_stop_gradient` returns variables but with zero gradient w.r.t. every other variable.

**Description**

Returns variables but with zero gradient w.r.t. every other variable.

**Usage**

\[ \text{k_stop_gradient}(\text{variables}) \]

**Arguments**

- **variables**: tensor or list of tensors to consider constant with respect to any other variable.

**Value**

A single tensor or a list of tensors (depending on the passed argument) that has constant gradient with respect to any other variable.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions)

---

**k_sum**

The sum of the values in a tensor, alongside the specified axis.

**Description**

Sum of the values in a tensor, alongside the specified axis.

**Usage**

\[ \text{k_sum}(x, \text{axis} = \text{NULL}, \text{keepdims} = \text{FALSE}) \]

**Arguments**

- **x**: A tensor or variable.
- **axis**: An integer, the axis to sum over (axis indexes are 1-based).
- **keepdims**: A boolean, whether to keep the dimensions or not. If keepdims is FALSE, the rank of the tensor is reduced by 1. If keepdims is TRUE, the reduced dimension is retained with length 1.
**k_switch**

**Value**
A tensor with sum of x.

**Keras Backend**
This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_switch**  
*Switches between two operations depending on a scalar value.*

---

**Description**
Note that both then_expression and else_expression should be symbolic tensors of the *same shape*.

**Usage**

`k_switch(condition, then_expression, else_expression)`

**Arguments**

- `condition`: tensor (int or bool).
- `then_expression`: either a tensor, or a function that returns a tensor.
- `else_expression`: either a tensor, or a function that returns a tensor.

**Value**
The selected tensor.

**Keras Backend**
This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).
**k_tanh**

*Element-wise tanh.*

**Description**

Element-wise tanh.

**Usage**

```
k_tanh(x)
```

**Arguments**

- `x` A tensor or variable.

**Value**

A tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_temporal_padding**

*Pads the middle dimension of a 3D tensor.*

**Description**

Pads the middle dimension of a 3D tensor.

**Usage**

```
k_temporal_padding(x, padding = c(1, 1))
```

**Arguments**

- `x` Tensor or variable.
- `padding` List of 2 integers, how many zeros to add at the start and end of dim 1.

**Value**

A padded 3D tensor.
k_tile

Creates a tensor by tiling x by n.

Description

Creates a tensor by tiling x by n.

Usage

k_tile(x, n)

Arguments

x A tensor or variable
n A list of integers. The length must be the same as the number of dimensions in x.

Value

A tiled tensor.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_to_dense**

Converting a sparse tensor into a dense tensor and returns it.

**Description**

Converts a sparse tensor into a dense tensor and returns it.

**Usage**

```python
k_to_dense(tensor)
```

**Arguments**

- `tensor`: A tensor instance (potentially sparse).

**Value**

A dense tensor.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g., TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

---

**k_transpose**

Transposes a tensor and returns it.

**Description**

Transposes a tensor and returns it.

**Usage**

```python
k_transpose(x)
```

**Arguments**

- `x`: Tensor or variable.

**Value**

A tensor.
Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

\texttt{k\_truncated\_normal} \hspace{1cm} \textit{Returns a tensor with truncated random normal distribution of values.}

---

**Description**

The generated values follow a normal distribution with specified mean and standard deviation, except that values whose magnitude is more than two standard deviations from the mean are dropped and re-picked.

**Usage**

\[
\texttt{k\_truncated\_normal(\text{shape, mean = 0, stddev = 1, dtype = NULL, seed = NULL})}
\]

**Arguments**

- **shape**: A list of integers, the shape of tensor to create.
- **mean**: Mean of the values.
- **stddev**: Standard deviation of the values.
- **dtype**: String, dtype of returned tensor.
- **seed**: Integer, random seed.

**Value**

A tensor.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
**k_update**

Update the value of \( x \) to \( \text{new}_x \).

**Description**
Update the value of \( x \) to \( \text{new}_x \).

**Usage**

\[ \text{k_update}(x, \text{new}_x) \]

**Arguments**

- \( x \)  
  A Variable.
- \( \text{new}_x \)  
  A tensor of same shape as \( x \).

**Value**

The variable \( x \) updated.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

**k_update_add**

Update the value of \( x \) by adding \( \text{increment} \).

**Description**
Update the value of \( x \) by adding \( \text{increment} \).

**Usage**

\[ \text{k_update_add}(x, \text{increment}) \]

**Arguments**

- \( x \)  
  A Variable.
- \( \text{increment} \)  
  A tensor of same shape as \( x \).

**Value**

The variable \( x \) updated.
Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

k_update_sub

Update the value of \( x \) by subtracting decrement.

Description

Update the value of \( x \) by subtracting decrement.

Usage

\[
k_{\text{update\_sub}}(x, \text{decrement})
\]

Arguments

- \( x \): A variable.
- \( \text{decrement} \): A tensor of same shape as \( x \).

Value

The variable \( x \) updated.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

k_var

Variance of a tensor, alongside the specified axis.

Description

Variance of a tensor, alongside the specified axis.

Usage

\[
k_{\text{var}}(x, \text{axis = NULL, keepdims = FALSE})
\]
k_variable

Arguments

args:
- x: A tensor or variable.
- axis: An integer, the axis to compute the variance over (axis indexes are 1-based).
- keepdims: A boolean, whether to keep the dimensions or not. If keepdims is FALSE, the rank of the tensor is reduced by 1. If keepdims is TRUE, the reduced dimension is retained with length 1.

Value

A tensor with the variance of elements of $x$.

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

k_variable

Instantiates a variable and returns it.

Description

Instantiates a variable and returns it.

Usage

k_variable(value, dtype = NULL, name = NULL, constraint = NULL)

Arguments

args:
- value: Numpy array, initial value of the tensor.
- dtype: Tensor type.
- name: Optional name string for the tensor.
- constraint: Optional projection function to be applied to the variable after an optimizer update.

Value

A variable instance (with Keras metadata included).

Keras Backend

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.
\begin{itemize}
\item \textbf{k_zeros}  \textit{Instantiates an all-zeros variable and returns it.}
\end{itemize}

**Description**

Instantiates an all-zeros variable and returns it.

**Usage**

\begin{verbatim}
k_zeros(shape, dtype = NULL, name = NULL)
\end{verbatim}

**Arguments**

\begin{itemize}
\item \textbf{shape}  Tuple of integers, shape of returned Keras variable
\item \textbf{dtype}  String, data type of returned Keras variable
\item \textbf{name}  String, name of returned Keras variable
\end{itemize}

**Value**

A variable (including Keras metadata), filled with 0.0.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

\begin{itemize}
\item \textbf{k_zeros_like}  \textit{Instantiates an all-zeros variable of the same shape as another tensor.}
\end{itemize}

**Description**

Instantiates an all-zeros variable of the same shape as another tensor.

**Usage**

\begin{verbatim}
k_zeros_like(x, dtype = NULL, name = NULL)
\end{verbatim}

**Arguments**

\begin{itemize}
\item \textbf{x}  Keras variable or Keras tensor.
\item \textbf{dtype}  String, dtype of returned Keras variable. NULL uses the dtype of x.
\item \textbf{name}  String, name for the variable to create.
\end{itemize}
**Value**

A Keras variable with the shape of x filled with zeros.

**Keras Backend**

This function is part of a set of Keras backend functions that enable lower level access to the core operations of the backend tensor engine (e.g. TensorFlow, CNTK, Theano, etc.).

You can see a list of all available backend functions here: https://keras.rstudio.com/articles/backend.html#backend-functions.

---

**layer_activation**  
*Apply an activation function to an output.*

---

**Description**

Apply an activation function to an output.

**Usage**

```python
layer_activation(object, activation, input_shape = NULL,
                 batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
                 name = NULL, trainable = NULL, weights = NULL)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Model or layer object</td>
</tr>
<tr>
<td>activation</td>
<td>Name of activation function to use. If you don’t specify anything, no activation is applied (ie. &quot;linear&quot; activation: ( a(x) = x )).</td>
</tr>
<tr>
<td>input_shape</td>
<td>Input shape (list of integers, does not include the samples axis) which is required when using this layer as the first layer in a model.</td>
</tr>
<tr>
<td>batch_input_shape</td>
<td>Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.</td>
</tr>
<tr>
<td>batch_size</td>
<td>Fixed batch size for layer</td>
</tr>
<tr>
<td>dtype</td>
<td>The data type expected by the input, as a string (float32, float64, int32...)</td>
</tr>
<tr>
<td>name</td>
<td>An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.</td>
</tr>
<tr>
<td>trainable</td>
<td>Whether the layer weights will be updated during training.</td>
</tr>
<tr>
<td>weights</td>
<td>Initial weights for layer</td>
</tr>
</tbody>
</table>
layer_activation_elu

See Also

Other core layers: layer_activity_regularization, layer_dense, layer_dropout, layer_flatten, layer_input, layer_lambda, layer_masking, layer_permute, layer_repeat_vector, layer_reshape

Other activation layers: layer_activation_elu, layer_activation_leaky_relu, layer_activation_parametric_relu, layer_activation_relu, layer_activation_selu, layer_activation_softmax, layer_activation_thresholded_relu

layer_activation_elu  Exponential Linear Unit.

Description

It follows: \( f(x) = \alpha \times (\exp(x) - 1.0) \) for \( x < 0 \), \( f(x) = x \) for \( x = 0 \).

Usage

layer_activation_elu(object, alpha = 1, input_shape = NULL, batch_input_shape = NULL, batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

object  Model or layer object
alpha  Scale for the negative factor.
input_shape  Input shape (list of integers, does not include the samples axis) which is required when using this layer as the first layer in a model.
batch_input_shape  Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.
batch_size  Fixed batch size for layer
dtype  The data type expected by the input, as a string (float32, float64, int32...)
name  An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
trainable  Whether the layer weights will be updated during training.
weights  Initial weights for layer.

See Also

Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs).

Other activation layers: layer_activation_leaky_relu, layer_activation_parametric_relu, layer_activation_relu, layer_activation_selu, layer_activation_softmax, layer_activation_thresholded_relu, layer_activation
**layer_activation_leaky_relu**

*Leaky version of a Rectified Linear Unit.*

**Description**

Allows a small gradient when the unit is not active: \( f(x) = \alpha \times x \) for \( x < 0 \), \( f(x) = x \) for \( x \geq 0 \).

**Usage**

```python
layer_activation_leaky_relu(object, alpha = 0.3, input_shape = NULL,
batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
name = NULL, trainable = NULL, weights = NULL)
```

**Arguments**

- **object**
  Model or layer object

- **alpha**
  float \( \geq 0 \). Negative slope coefficient.

- **input_shape**
  Input shape (list of integers, does not include the samples axis) which is required when using this layer as the first layer in a model.

- **batch_input_shape**
  Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors.

- **batch_size**
  Fixed batch size for layer

- **dtype**
  The data type expected by the input, as a string (`float32`, `float64`, `int32`...)

- **name**
  An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.

- **trainable**
  Whether the layer weights will be updated during training.

- **weights**
  Initial weights for layer.

**See Also**

Rectifier Nonlinearities Improve Neural Network Acoustic Models.

Other activation layers: `layer_activation_elu`, `layer_activation_parametric_relu`, `layer_activation_relu`, `layer_activation_selu`, `layer_activation_softmax`, `layer_activation_thresholded_relu`, `layer_activation`
layer_activation_parametric_relu

*Parametric Rectified Linear Unit.*

**Description**

It follows: \( f(x) = \alpha \cdot \max(x, 0) \) for \( x < 0 \), \( f(x) = x \) for \( x \geq 0 \), where \( \alpha \) is a learned array with the same shape as \( x \).

**Usage**

```python
layer_activation_parametric_relu(object, alpha_initializer = "zeros",
alpha_regularizer = NULL, alpha_constraint = NULL,
shared_axes = NULL, input_shape = NULL, batch_input_shape = NULL,
batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL,
weights = NULL)
```

**Arguments**

- **object**  
  Model or layer object

- **alpha_initializer**  
  Initializer function for the weights.

- **alpha_regularizer**  
  Regularizer for the weights.

- **alpha_constraint**  
  Constraint for the weights.

- **shared_axes**  
  The axes along which to share learnable parameters for the activation function. For example, if the incoming feature maps are from a 2D convolution with output shape (batch, height, width, channels), and you wish to share parameters across space so that each filter only has one set of parameters, set `shared_axes=c(1, 2)`.

- **input_shape**  
  Input shape (list of integers, does not include the samples axis) which is required when using this layer as the first layer in a model.

- **batch_input_shape**  
  Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors.

- **batch_size**  
  Fixed batch size for layer

- **dtype**  
  The data type expected by the input, as a string (float32, float64, int32...)

- **name**  
  An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.

- **trainable**  
  Whether the layer weights will be updated during training.

- **weights**  
  Initial weights for layer.
layer_activation_relu

Description
Rectified Linear Unit activation function

Usage
layer_activation_relu(object, max_value = NULL, negative_slope = 0,
threshold = 0, input_shape = NULL, batch_input_shape = NULL,
batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL,
weights = NULL)

Arguments
object
Model or layer object
max_value
loat, the maximum output value.
negative_slope
float >= 0 Negative slope coefficient.
threshold
float. Threshold value for thresholded activation.
input_shape
Input shape (list of integers, does not include the samples axis) which is required
when using this layer as the first layer in a model.
batch_input_shape
Shapes, including the batch size. For instance, batch_input_shape=c(10, 32)
indicates that the expected input will be batches of 10 32-dimensional vectors.
batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number
of 32-dimensional vectors.
batch_size
Fixed batch size for layer
dtype
The data type expected by the input, as a string (float32, float64, int32...)
name
An optional name string for the layer. Should be unique in a model (do not reuse
the same name twice). It will be autogenerated if it isn’t provided.
trainable
Whether the layer weights will be updated during training.
weights
Initial weights for layer.

See Also
Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.
Other activation layers: layer_activation_elu, layer_activation_leaky_relu, layer_activation_parametric_relu,
layer_activation_selu, layer_activation_softmax, layer_activation_thresholded_relu,
layer_activation_selu  *Scaled Exponential Linear Unit.*

**Description**

SELU is equal to: $\text{scale} \times \text{elu}(x, \text{alpha})$, where alpha and scale are pre-defined constants.

**Usage**

```python
layer_activation_selu(object, input_shape = NULL,
                       batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
                       name = NULL, trainable = NULL, weights = NULL)
```

**Arguments**

- **object**: Model or layer object
- **input_shape**: Input shape (list of integers, does not include the samples axis) which is required when using this layer as the first layer in a model.
- **batch_input_shape**: Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.
- **batch_size**: Fixed batch size for layer
- **dtype**: The data type expected by the input, as a string (float32, float64, int32...)
- **name**: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- **trainable**: Whether the layer weights will be updated during training.
- **weights**: Initial weights for layer.

**Details**

The values of alpha and scale are chosen so that the mean and variance of the inputs are preserved between two consecutive layers as long as the weights are initialized correctly (see initializer_lecun_normal) and the number of inputs is "large enough" (see article for more information).

**Note:**

- To be used together with the initialization "lecun_normal".
- To be used together with the dropout variant "AlphaDropout".

**See Also**

- Self-Normalizing Neural Networks, initializer_lecun_normal, layer_alpha_dropout
- Other activation layers: layer_activation_elu, layer_activation_leaky_relu, layer_activation_parametric_relu, layer_activation_relu, layer_activation_softmax, layer_activation_thresholded_relu
layer_activation_softmax

Softmax activation function.

Description

It follows: \( f(x) = \alpha \times (\exp(x) - 1.0) \) for \( x < 0 \), \( f(x) = x \) for \( x = 0 \).

Usage

layer_activation_softmax(object, axis = -1, input_shape = NULL, batch_input_shape = NULL, batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

- **object** : Model or layer object
- **axis** : Integer, axis along which the softmax normalization is applied.
- **input_shape** : Input shape (list of integers, does not include the samples axis) which is required when using this layer as the first layer in a model.
- **batch_input_shape** : Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.
- **batch_size** : Fixed batch size for layer
- **dtype** : The data type expected by the input, as a string (float32, float64, int32...)
- **name** : An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- **trainable** : Whether the layer weights will be updated during training.
- **weights** : Initial weights for layer.

See Also

Other activation layers: layer_activation_elu, layer_activation_leaky_relu, layer_activation_parametric_relu, layer_activation_relu, layer_activation_selu, layer_activation_thresholded_relu, layer_activation...
layer_activation_thresholded_relu

Thresholded Rectified Linear Unit.

Description

It follows: \( f(x) = x \) for \( x > \theta \), \( f(x) = 0 \) otherwise.

Usage

layer_activation_thresholded_relu(object, theta = 1,
input_shape = NULL, batch_input_shape = NULL, batch_size = NULL,
dtype = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

object  Model or layer object
theta   float >= 0. Threshold location of activation.
input_shape  Input shape (list of integers, does not include the samples axis) which is required
when using this layer as the first layer in a model.
batch_input_shape  Shapes, including the batch size. For instance, batch_input_shape =c(10, 32)
indicates that the expected input will be batches of 10 32-dimensional vectors.
batch_input_shape =list(NULL, 32) indicates batches of an arbitrary number
of 32-dimensional vectors.
batch_size  Fixed batch size for layer
dtype  The data type expected by the input, as a string (float32, float64, int32...)
name  An optional name string for the layer. Should be unique in a model (do not reuse
the same name twice). It will be autogenerated if it isn’t provided.
trainable  Whether the layer weights will be updated during training.
weights  Initial weights for layer.

See Also

Zero-bias autoencoders and the benefits of co-adapting features.

Other activation layers: layer_activation_elu, layer_activation_leaky_relu, layer_activation_parametric_relu,
layer_activation_relu, layer_activation_selu, layer_activation_softmax, layer_activation...
layer_activity_regularization

Layer that applies an update to the cost function based input activity.

Description

Layer that applies an update to the cost function based input activity.

Usage

layer_activity_regularization(object, l1 = 0, l2 = 0,
                          input_shape = NULL, batch_input_shape = NULL, batch_size = NULL,
                          dtype = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

object Model or layer object
l1 L1 regularization factor (positive float).
l2 L2 regularization factor (positive float).
input_shape Dimensionality of the input (integer) not including the samples axis. This argu-
               ment is required when using this layer as the first layer in a model.
batch_input_shape Shapes, including the batch size. For instance, batch_input_shape=c(10, 32)
                       indicates that the expected input will be batches of 10 32-dimensional vectors.
                       batch_input_shape=list(NULL, 32) indicates batches of an arbitrary num-
                       ber of 32-dimensional vectors.
batch_size Fixed batch size for layer
dtype The data type expected by the input, as a string (float32, float64, int32...)
name An optional name string for the layer. Should be unique in a model (do not reuse
       the same name twice). It will be autogenerated if it isn’t provided.
trainable Whether the layer weights will be updated during training.
weights Initial weights for layer.

Input shape

Arbitrary. Use the keyword argument input_shape (list of integers, does not include the samples
axis) when using this layer as the first layer in a model.

Output shape

Same shape as input.

See Also

Other core layers: layer_activation, layer_dense, layer_dropout, layer_flatten, layer_input,
layer_lambda, layer_masking, layer_permute, layer_repeat_vector, layer_reshape
**layer_add**

Layer that adds a list of inputs.

**Description**

It takes as input a list of tensors, all of the same shape, and returns a single tensor (also of the same shape).

**Usage**

```python
layer_add(inputs, batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL, weights = NULL)
```

**Arguments**

- **inputs**: A list of input tensors (at least 2).
- **batch_size**: Fixed batch size for layer.
- **dtype**: The data type expected by the input, as a string (float32, float64, int32...)
- **name**: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- **trainable**: Whether the layer weights will be updated during training.
- **weights**: Initial weights for layer.

**Value**

A tensor, the sum of the inputs.

**See Also**

Other merge layers: layer_average, layer_concatenate, layer_dot, layer_maximum, layer_minimum, layer_multiply, layer_subtract

---

**layer_alpha_dropout**

Applies Alpha Dropout to the input.

**Description**

Alpha Dropout is a dropout that keeps mean and variance of inputs to their original values, in order to ensure the self-normalizing property even after this dropout.

**Usage**

```python
layer_alpha_dropout(object, rate, noise_shape = NULL, seed = NULL, input_shape = NULL, batch_input_shape = NULL, batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL, weights = NULL)
```
layer_alpha_dropout

**Arguments**

- **object**  
  Model or layer object
- **rate**  
  float, drop probability (as with `layer_dropout()`). The multiplicative noise will have standard deviation `sqrt(rate / (1 - rate))`.  
- **noise_shape**  
  Noise shape
- **seed**  
  An integer to use as random seed.
- **input_shape**  
  Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.
- **batch_input_shape**  
  Shapes, including the batch size. For instance, `batch_input_shape=[10, 32]` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors.
- **batch_size**  
  Fixed batch size for layer
- **dtype**  
  The data type expected by the input, as a string (`float32`, `float64`, `int32`, ...)
- **name**  
  An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- **trainable**  
  Whether the layer weights will be updated during training.
- **weights**  
  Initial weights for layer.

**Details**

Alpha Dropout fits well to Scaled Exponential Linear Units by randomly setting activations to the negative saturation value.

**Input shape**

Arbitrary. Use the keyword argument `input_shape` (list of integers, does not include the samples axis) when using this layer as the first layer in a model.

**Output shape**

Same shape as input.

**References**

- [Self-Normalizing Neural Networks](#)

**See Also**

Other noise layers: `layer_gaussian_dropout`, `layer_gaussian_noise`
layer_average

Layer that averages a list of inputs.

Description

It takes as input a list of tensors, all of the same shape, and returns a single tensor (also of the same shape).

Usage

```
layer_average(inputs, batch_size = NULL, dtype = NULL, name = NULL,
              trainable = NULL, weights = NULL)
```

Arguments

- **inputs**: A list of input tensors (at least 2).
- **batch_size**: Fixed batch size for layer
- **dtype**: The data type expected by the input, as a string (float32, float64, int32...)
- **name**: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- **trainable**: Whether the layer weights will be updated during training.
- **weights**: Initial weights for layer.

Value

A tensor, the average of the inputs.

See Also

Other merge layers: layer_add, layer_concatenate, layer_dot, layer_maximum, layer_minimum, layer_multiply, layer_subtract

layer_average_pooling_1d

Average pooling for temporal data.

Description

Average pooling for temporal data.

Usage

```
layer_average_pooling_1d(object, pool_size = 2L, strides = NULL,
                         padding = "valid", data_format = "channels_last",
                         batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)
```
**layer_average_pooling_2d**

**Arguments**

- **object**: Model or layer object
- **pool_size**: Integer, size of the average pooling windows.
- **strides**: Integer, or NULL. Factor by which to downscale. E.g. 2 will halve the input. If NULL, it will default to pool_size.
- **padding**: One of "valid" or "same" (case-insensitive).
- **data_format**: One of channels_last (default) or channels_first. The ordering of the dimensions in the inputs.
- **batch_size**: Fixed batch size for layer
- **name**: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn't provided.
- **trainable**: Whether the layer weights will be updated during training.
- **weights**: Initial weights for layer.

**Input shape**

3D tensor with shape: (batch_size, steps, features).

**Output shape**

3D tensor with shape: (batch_size, downsampled_steps, features).

**See Also**

Other pooling layers: layer_average_pooling_2d, layer_average_pooling_3d, layer_global_average_pooling_1d, layer_global_average_pooling_3d, layer_global_max_pooling_1d, layer_global_max_pooling_2d, layer_global_max_pooling_3d, layer_max_pooling_1d, layer_max_pooling_2d, layer_max_pooling_3d, layer_average_pooling_Rd

---

layer_average_pooling_2d

*Average pooling operation for spatial data.*

**Description**

Average pooling operation for spatial data.

**Usage**

layer_average_pooling_2d(object, pool_size = c(2L, 2L), strides = NULL, padding = "valid", data_format = NULL, batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)
Arguments

object | Model or layer object
pool_size | integer or list of 2 integers, factors by which to downscale (vertical, horizontal). (2, 2) will halve the input in both spatial dimension. If only one integer is specified, the same window length will be used for both dimensions.
strides | Integer, list of 2 integers, or NULL. Strides values. If NULL, it will default to pool_size.
padding | One of "valid" or "same" (case-insensitive).
data_format | A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".
batch_size | Fixed batch size for layer
name | An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
trainable | Whether the layer weights will be updated during training.
weights | Initial weights for layer.

Input shape

- If data_format='channels_last': 4D tensor with shape: (batch_size, rows, cols, channels)
- If data_format='channels_first': 4D tensor with shape: (batch_size, channels, rows, cols)

Output shape

- If data_format='channels_last': 4D tensor with shape: (batch_size, pooled_rows, pooled_cols, channels)
- If data_format='channels_first': 4D tensor with shape: (batch_size, channels, pooled_rows, pooled_cols)

See Also

Other pooling layers: layer_average_pooling_1d, layer_average_pooling_3d, layer_global_average_pooling_1d, layer_global_average_pooling_2d, layer_global_average_pooling_3d, layer_global_max_pooling_1d, layer_global_max_pooling_2d, layer_global_max_pooling_3d, layer_max_pooling_1d, layer_max_pooling_2d, layer_max_pooling_3d
Average pooling operation for 3D data (spatial or spatio-temporal).

Usage

layer_average_pooling_3d(object, pool_size = c(2L, 2L, 2L),
                      strides = NULL, padding = "valid", data_format = NULL,
                      batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

- object: Model or layer object
- pool_size: list of 3 integers, factors by which to downscale (dim1, dim2, dim3). (2, 2, 2) will halve the size of the 3D input in each dimension.
- strides: list of 3 integers, or NULL. Strides values.
- padding: One of "valid" or "same" (case-insensitive).
- data_format: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, spatial_dim1, spatial_dim2, spatial_dim3, channels) while channels_first corresponds to inputs with shape (batch, channels, spatial_dim1, spatial_dim2, spatial_dim3).
  It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".
- batch_size: Fixed batch size for layer
- name: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- trainable: Whether the layer weights will be updated during training.
- weights: Initial weights for layer.

Input shape

- If data_format='channels_last': 5D tensor with shape: (batch_size, spatial_dim1, spatial_dim2, spatial_dim3, channels)
- If data_format='channels_first': 5D tensor with shape: (batch_size, channels, spatial_dim1, spatial_dim2, spatial_dim3)

Output shape

- If data_format='channels_last': 5D tensor with shape: (batch_size, pooled_dim1, pooled_dim2, pooled_dim3, channels)
- If data_format='channels_first': 5D tensor with shape: (batch_size, channels, pooled_dim1, pooled_dim2, pooled_dim3)
Batch normalization layer (Ioffe and Szegedy, 2014).

Description

Normalize the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1.

Usage

layer_batch_normalization(object, axis = -1L, momentum = 0.99, epsilon = 0.001, center = TRUE, scale = TRUE, beta_initializer = "zeros", gamma_initializer = "ones", moving_mean_initializer = "zeros", moving_variance_initializer = "ones", beta_regularizer = NULL, gamma_regularizer = NULL, beta_constraint = NULL, gamma_constraint = NULL, input_shape = NULL, batch_input_shape = NULL, batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

object  
Model or layer object

axis  
Integer, the axis that should be normalized (typically the features axis). For instance, after a Conv2D layer with data_format="channels_first", set axis=1 in BatchNormalization.

momentum  
Momentum for the moving mean and the moving variance.

epsilon  
Small float added to variance to avoid dividing by zero.

center  
If TRUE, add offset of beta to normalized tensor. If FALSE, beta is ignored.

scale  
If TRUE, multiply by gamma. If FALSE, gamma is not used. When the next layer is linear (also e.g. nn.relu), this can be disabled since the scaling will be done by the next layer.

beta_initializer  
Initializer for the beta weight.

gamma_initializer  
Initializer for the gamma weight.

moving_mean_initializer  
Initializer for the moving mean.
moving_variance_initializer
  Initializer for the moving variance.

beta_regularizer
  Optional regularizer for the beta weight.

gamma_regularizer
  Optional regularizer for the gamma weight.

beta_constraint
  Optional constraint for the beta weight.

gamma_constraint
  Optional constraint for the gamma weight.

input_shape
  Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

batch_input_shape
  Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.

batch_size
  Fixed batch size for layer

dtype
  The data type expected by the input, as a string (float32, float64, int32...)

name
  An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.

trainable
  Whether the layer weights will be updated during training.

weights
  Initial weights for layer.

Input shape

Arbitrary. Use the keyword argument input_shape (list of integers, does not include the samples axis) when using this layer as the first layer in a model.

Output shape

Same shape as input.

References

- Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift
layer_concatenate  

Layer that concatenates a list of inputs.

Description

It takes as input a list of tensors, all of the same shape except for the concatenation axis, and returns a single tensor, the concatenation of all inputs.

Usage

layer_concatenate(inputs, axis = -1, batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

- **inputs**: A list of input tensors (at least 2).
- **axis**: Concatenation axis.
- **batch_size**: Fixed batch size for layer.
- **dtype**: The data type expected by the input, as a string (float32, float64, int32...).
- **name**: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- **trainable**: Whether the layer weights will be updated during training.
- **weights**: Initial weights for layer.

Value

A tensor, the concatenation of the inputs alongside axis axis.

See Also

Other merge layers: layer_add, layer_average, layer_dot, layer_maximum, layer_minimum, layer_multiply, layer_subtract

layer_conv_1d  

1D convolution layer (e.g. temporal convolution).

Description

This layer creates a convolution kernel that is convolved with the layer input over a single spatial (or temporal) dimension to produce a tensor of outputs. If use_bias is TRUE, a bias vector is created and added to the outputs. Finally, if activation is not NULL, it is applied to the outputs as well. When using this layer as the first layer in a model, provide an input_shape argument (list of integers or NULL), e.g. (10, 128) for sequences of 10 vectors of 128-dimensional vectors, or (NULL, 128) for variable-length sequences of 128-dimensional vectors.
layer_conv_1d

Usage

layer_conv_1d(object, filters, kernel_size, strides = 1L,
               padding = "valid", data_format = "channels_last",
               dilation_rate = 1L, activation = NULL, use_bias = TRUE,
               kernel_initializer = "glorot_uniform", bias_initializer = "zeros",
               kernel_regularizer = NULL, bias_regularizer = NULL,
               activity_regularizer = NULL, kernel_constraint = NULL,
               bias_constraint = NULL, input_shape = NULL,
               batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
               name = NULL, trainable = NULL, weights = NULL)

Arguments

object     Model or layer object
filters    Integer, the dimensionality of the output space (i.e. the number of output filters
           in the convolution).
kernel_size An integer or list of a single integer, specifying the length of the 1D convolution
            window.
strides    An integer or list of a single integer, specifying the stride length of the con-
            volution. Specifying any stride value != 1 is incompatible with specifying any
dilation_rate an integer or list of a single integer, specifying the dilation rate to use for dilated
            convolution. Currently, specifying any dilation_rate value != 1 is incompat-
padding    One of "valid", "causal" or "same" (case-insensitive). "valid" means "no
            padding". "same" results in padding the input such that the output has the same
            length as the original input. "causal" results in causal (dilated) convolutions,
            e.g. output{t} does not depend on input{t+1}. Useful when modeling tem-
data_format A string, one of "channels_last" (default) or "channels_first". The or-
            dering of the dimensions in the inputs. "channels_last" corresponds to in-
            puts with shape (batch, length, channels) (default format for tempo-
dilation_rate an integer or list of a single integer, specifying the dilation rate to use for dilated
            convolution. Currently, specifying any dilation_rate value != 1 is incompat-
activation  Activation function to use. If you don’t specify anything, no activation is applied
            (i.e. "linear" activation: a(x) = x).
use_bias Boolean, whether the layer uses a bias vector.
kernki
bias_initializer Initializer for the bias vector.
kern
bias_regularizer Regularizer function applied to the bias vector.
activity_regularizer
   Regularizer function applied to the output of the layer (its "activation").
kernel_constraint
   Constraint function applied to the kernel matrix.
bias_constraint
   Constraint function applied to the bias vector.
input_shape
   Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.
batch_input_shape
   Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.
batch_size
   Fixed batch size for layer
dtype
   The data type expected by the input, as a string (float32, float64, int32...)
name
   An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
trainable
   Whether the layer weights will be updated during training.
weights
   Initial weights for layer.

Input shape
   3D tensor with shape: (batch_size, steps, input_dim)

Output shape
   3D tensor with shape: (batch_size, new_steps, filters) steps value might have changed due to padding or strides.

See Also
   Other convolutional layers: layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_depthwise_conv_2d, layer_separable_conv_1d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

layer_conv_2d
   2D convolution layer (e.g. spatial convolution over images).

Description
   This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If use_bias is TRUE, a bias vector is created and added to the outputs. Finally, if activation is not NULL, it is applied to the outputs as well. When using this layer as the first layer in a model, provide the keyword argument input_shape (list of integers, does not include the sample axis), e.g. input_shape=c(128, 128, 3) for 128x128 RGB pictures in data_format="channels_last".
**Usage**

```python
layer_conv_2d(object, filters, kernel_size, strides = c(1L, 1L),
padding = "valid", data_format = NULL, dilation_rate = c(1L, 1L),
activation = NULL, use_bias = TRUE,
kernel_initializer = "glorot_uniform", bias_initializer = "zeros",
kernel_regularizer = NULL, bias_regularizer = NULL,
activity_regularizer = NULL, kernel_constraint = NULL,
bias_constraint = NULL, input_shape = NULL,
batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
name = NULL, trainable = NULL, weights = NULL)
```

**Arguments**

- **object**: Model or layer object.
- **filters**: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel_size**: An integer or list of 2 integers, specifying the width and height of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides**: An integer or list of 2 integers, specifying the strides of the convolution along the width and height. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.
- **padding**: one of "valid" or "same" (case-insensitive). Note that "same" is slightly inconsistent across backends with strides != 1, as described [here](#).
- **data_format**: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels_last".
- **dilation_rate**: an integer or list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any `dilation_rate` value != 1 is incompatible with specifying any stride value != 1.
- **activation**: Activation function to use. If you don’t specify anything, no activation is applied (ie. "linear" activation: $a(x) = x$).
- **use_bias**: Boolean, whether the layer uses a bias vector.
- **kernel_initializer**: Initializer for the kernel weights matrix.
- **bias_initializer**: Initializer for the bias vector.
- **kernel_regularizer**: Regularizer function applied to the kernel weights matrix.
- **bias_regularizer**: Regularizer function applied to the bias vector.
activity_regularizer

Regularizer function applied to the output of the layer (its "activation").

kernel_constraint

Constraint function applied to the kernel matrix.

bias_constraint

Constraint function applied to the bias vector.

input_shape

Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

batch_input_shape

Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.

batch_size

Fixed batch size for layer

dtype

The data type expected by the input, as a string (float32, float64, int32...)

name

An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.

trainable

Whether the layer weights will be updated during training.

weights

Initial weights for layer.

Input shape

4D tensor with shape: (samples, channels, rows, cols) if data_format='channels_first' or 4D tensor with shape: (samples, rows, cols, channels) if data_format='channels_last'.

Output shape

4D tensor with shape: (samples, filters, new_rows, new_cols) if data_format='channels_first' or 4D tensor with shape: (samples, new_rows, new_cols, filters) if data_format='channels_last'. rows and cols values might have changed due to padding.

See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_depthwise_conv_2d, layer_separable_conv_1d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d
**layer_conv_2d_transpose**

Transposed 2D convolution layer (sometimes called Deconvolution).

**Description**

The need for transposed convolutions generally arises from the desire to use a transformation going in the opposite direction of a normal convolution, i.e., from something that has the shape of the output of some convolution to something that has the shape of its input while maintaining a connectivity pattern that is compatible with said convolution. When using this layer as the first layer in a model, provide the keyword argument `input_shape` (list of integers, does not include the sample axis), e.g. `input_shape=[128L, 128L, 3L]` for 128x128 RGB pictures in `data_format="channels_last"`.

**Usage**

```python
layer_conv_2d_transpose(object, filters, kernel_size, strides = [1, 1],
```
```python
apadding = "valid", output_padding = NULL, data_format = NULL,
```
```python
dilation_rate = [1, 1], activation = NULL, use_bias = TRUE,
```
```python
kernel_initializer = "glorot_uniform", bias_initializer = "zeros",
```
```python
kernel_regularizer = NULL, bias_regularizer = NULL,
```
```python
activity_regularizer = NULL, kernel_constraint = NULL,
```
```python
bias_constraint = NULL, input_shape = NULL,
```
```python
batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
```
```python
name = NULL, trainable = NULL, weights = NULL)
```

**Arguments**

- **object**
  - Model or layer object

- **filters**
  - Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).

- **kernel_size**
  - An integer or list of 2 integers, specifying the width and height of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.

- **strides**
  - An integer or list of 2 integers, specifying the strides of the convolution along the width and height. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any `dilation_rate` value != 1.

- **padding**
  - one of "valid" or "same" (case-insensitive).

- **output_padding**
  - An integer or list of 2 integers, specifying the amount of padding along the height and width of the output tensor. Can be a single integer to specify the same value for all spatial dimensions. The amount of output padding along a given dimension must be lower than the stride along that same dimension. If set to NULL (default), the output shape is inferred.
data_format A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

dilation_rate Dilation rate.
activation Activation function to use. If you don’t specify anything, no activation is applied (ie. "linear" activation: \( a(x) = x \)).
use_bias Boolean, whether the layer uses a bias vector.
kernel_initializer Initializer for the kernel weights matrix.
bias_initializer Initializer for the bias vector.
kernel_regularizer Regularizer function applied to the kernel weights matrix.
bias_regularizer Regularizer function applied to the bias vector.
activity_regularizer Regularizer function applied to the output of the layer (its "activation").
kernel_constraint Constraint function applied to the kernel matrix.
bias_constraint Constraint function applied to the bias vector.
input_shape Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.
batch_input_shape Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.
batch_size Fixed batch size for layer
dtype The data type expected by the input, as a string (float32, float64, int32...)
name An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
trainable Whether the layer weights will be updated during training.
weights Initial weights for layer.

Input shape

4D tensor with shape: (batch, channels, rows, cols) if data_format='channels_first' or 4D tensor with shape: (batch, rows, cols, channels) if data_format='channels_last'.
layer_conv_3d

Output shape

4D tensor with shape: (batch, filters, new_rows, new_cols) if data_format='channels_first' or 4D tensor with shape: (batch, new_rows, new_cols, filters) if data_format='channels_last'. rows and cols values might have changed due to padding.

References

- A guide to convolution arithmetic for deep learning
- Deconvolutional Networks

See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_depthwise_conv_2d, layer_separable_conv_1d, layer_separable_conv_2d, layer_separable_conv_3d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

layer_conv_3d 3D convolution layer (e.g. spatial convolution over volumes).

Description

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. If use_bias is TRUE, a bias vector is created and added to the outputs. Finally, if activation is not NULL, it is applied to the outputs as well. When using this layer as the first layer in a model, provide the keyword argument input_shape (list of integers, does not include the sample axis), e.g. input_shape=[128L, 128L, 128L, 3L] for 128x128x128 volumes with a single channel, in data_format="channels_last".

Usage

layer_conv_3d(object, filters, kernel_size, strides = c(1L, 1L, 1L), padding = "valid", data_format = NULL, dilation_rate = c(1L, 1L, 1L), activation = NULL, use_bias = TRUE, kernel_initializer = "glorot_uniform", bias_initializer = "zeros", kernel_regularizer = NULL, bias_regularizer = NULL, activity_regularizer = NULL, kernel_constraint = NULL, bias_constraint = NULL, input_shape = NULL, batch_input_shape = NULL, batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

object Model or layer object
filters Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
kernel_size  An integer or list of 3 integers, specifying the depth, height, and width of the 3D convolution window. Can be a single integer to specify the same value for all spatial dimensions.

strides An integer or list of 3 integers, specifying the strides of the convolution along each spatial dimension. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.

padding one of "valid" or "same" (case-insensitive).

data_format A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, spatial_dim1, spatial_dim2, spatial_dim3, channels) while channels_first corresponds to inputs with shape (batch, channels, spatial_dim1, spatial_dim2).

It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

dilation_rate an integer or list of 3 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any stride value != 1.

activation Activation function to use. If you don’t specify anything, no activation is applied (ie. "linear" activation: a(x) = x).

use_bias Boolean, whether the layer uses a bias vector.

kernel_initializer Initializer for the kernel weights matrix.

bias_initializer Initializer for the bias vector.

kernel_regularizer Regularizer function applied to the kernel weights matrix.

bias_regularizer Regularizer function applied to the bias vector.

activity_regularizer Regularizer function applied to the output of the layer (its "activation").

kernel_constraint Constraint function applied to the kernel matrix.

bias_constraint Constraint function applied to the bias vector.

input_shape Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

batch_input_shape Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.

batch_size Fixed batch size for layer

dtype The data type expected by the input, as a string (float32, float64, int32...)
layer_conv_3d_transpose

name An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.

trainable Whether the layer weights will be updated during training.

weights Initial weights for layer.

Input shape

5D tensor with shape: (samples, channels, conv_dim1, conv_dim2, conv_dim3) if data_format='channels_first' or 5D tensor with shape: (samples, conv_dim1, conv_dim2, conv_dim3, channels) if data_format='channels_last'.

Output shape

5D tensor with shape: (samples, filters, new_conv_dim1, new_conv_dim2, new_conv_dim3) if data_format='channels_first' or 5D tensor with shape: (samples, new_conv_dim1, new_conv_dim2, new_conv_dim3, filters) if data_format='channels_last'. new_conv_dim1, new_conv_dim2 and new_conv_dim3 values might have changed due to padding.

See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_depthwise_conv_2d, layer_separable_conv_1d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

layer_conv_3d_transpose

Transposed 3D convolution layer (sometimes called Deconvolution).

Description

The need for transposed convolutions generally arises from the desire to use a transformation going in the opposite direction of a normal convolution, i.e., from something that has the shape of the output of some convolution to something that has the shape of its input while maintaining a connectivity pattern that is compatible with said convolution.

Usage

layer_conv_3d_transpose(object, filters, kernel_size, strides = c(1, 1, 1), padding = "valid", output_padding = NULL, data_format = NULL, activation = NULL, use_bias = TRUE, kernel_initializer = "glorot_uniform", bias_initializer = "zeros", kernel_regularizer = NULL, bias_regularizer = NULL, activity_regularizer = NULL, kernel_constraint = NULL, bias_constraint = NULL, input_shape = NULL, batch_input_shape = NULL, batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL, weights = NULL)
Arguments

- **object**
  - Model or layer object

- **filters**
  - Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).

- **kernel_size**
  - An integer or list of 3 integers, specifying the depth, height, and width of the 3D convolution window. Can be a single integer to specify the same value for all spatial dimensions.

- **strides**
  - An integer or list of 3 integers, specifying the strides of the convolution along the depth, height and width. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.

- **padding**
  - one of "valid" or "same" (case-insensitive).

- **output_padding**
  - An integer or list of 3 integers, specifying the amount of padding along the depth, height, and width of the output tensor. Can be a single integer to specify the same value for all spatial dimensions. The amount of output padding along a given dimension must be lower than the stride along that same dimension. If set to NULL (default), the output shape is inferred.

- **data_format**
  - A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, depth, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, depth, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

- **activation**
  - Activation function to use. If you don’t specify anything, no activation is applied (i.e. "linear" activation: \( a(x) = x \)).

- **use_bias**
  - Boolean, whether the layer uses a bias vector.

- **kernel_initializer**
  - Initializer for the kernel weights matrix.

- **bias_initializer**
  - Initializer for the bias vector.

- **kernel_regularizer**
  - Regularizer function applied to the kernel weights matrix.

- **bias_regularizer**
  - Regularizer function applied to the bias vector.

- **activity_regularizer**
  - Regularizer function applied to the output of the layer (its "activation").

- **kernel_constraint**
  - Constraint function applied to the kernel matrix.

- **bias_constraint**
  - Constraint function applied to the bias vector.

- **input_shape**
  - Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.
layer_conv_lstm_2d

batch_input_shape
Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.

batch_size
Fixed batch size for layer

dtype
The data type expected by the input, as a string (float32, float64, int32...)

name
An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.

trainable
Whether the layer weights will be updated during training.

weights
Initial weights for layer.

Details
When using this layer as the first layer in a model, provide the keyword argument input_shape (list of integers, does not include the sample axis), e.g. input_shape = list(128, 128, 128, 3) for a 128x128x128 volume with 3 channels if data_format="channels_last".

References
- A guide to convolution arithmetic for deep learning
- Deconvolutional Networks

See Also
Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_depthwise_conv_2d, layer_separable_conv_1d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

layer_conv_lstm_2d

Convolutional LSTM.

Description
It is similar to an LSTM layer, but the input transformations and recurrent transformations are both convolutional.

Usage
layer_conv_lstm_2d(object, filters, kernel_size, strides = c(1L, 1L), padding = "valid", data_format = NULL, dilation_rate = c(1L, 1L), activation = "tanh", recurrent_activation = "hard_sigmoid", use_bias = TRUE, kernel_initializer = "glorot_uniform", recurrent_initializer = "orthogonal", bias_initializer = "zeros", unit_forget_bias = TRUE, kernel_regularizer = NULL,
recurrent_regularizer = NULL, bias_regularizer = NULL,
activity_regularizer = NULL, kernel_constraint = NULL,
recurrent_constraint = NULL, bias_constraint = NULL,
return_sequences = FALSE, go_backwards = FALSE, stateful = FALSE,
dropout = 0, recurrent_dropout = 0, batch_size = NULL,
name = NULL, trainable = NULL, weights = NULL,
input_shape = NULL)

Arguments

object
Model or layer object

filters
Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).

kernel_size
An integer or list of n integers, specifying the dimensions of the convolution window.

strides
An integer or list of n integers, specifying the strides of the convolution. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.

padding
One of "valid" or "same" (case-insensitive).

data_format
A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, time, ..., channels) while channels_first corresponds to inputs with shape (batch, time, channels, ...). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

dilation_rate
An integer or list of n integers, specifying the dilation rate to use for dilated convolution. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any strides value != 1.

activation
Activation function to use. If you don't specify anything, no activation is applied (ie. "linear" activation: $a(x) = x$).

recurrent_activation
Activation function to use for the recurrent step.

use_bias
Boolean, whether the layer uses a bias vector.

kernel_initializer
Initializer for the kernel weights matrix, used for the linear transformation of the inputs..

recurrent_initializer
Initializer for the recurrent_kernel weights matrix, used for the linear transformation of the recurrent state..

bias_initializer
Initializer for the bias vector.

unit_forget_bias
Boolean. If TRUE, add 1 to the bias of the forget gate at initialization. Use in combination with bias_initializer="zeros". This is recommended in Jozefowicz et al.
layer_conv_lstm_2d

kernel_regularizer
  Regularizer function applied to the kernel weights matrix.

recurrent_regularizer
  Regularizer function applied to the recurrent_kernel weights matrix.

bias_regularizer
  Regularizer function applied to the bias vector.

activity_regularizer
  Regularizer function applied to the output of the layer (its "activation").

kernel_constraint
  Constraint function applied to the kernel weights matrix.

recurrent_constraint
  Constraint function applied to the recurrent_kernel weights matrix.

bias_constraint
  Constraint function applied to the bias vector.

return_sequences
  Boolean. Whether to return the last output in the output sequence, or the full sequence.

go_backwards
  Boolean (default FALSE). If TRUE, process the input sequence backwards.

stateful
  Boolean (default FALSE). If TRUE, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.

dropout
  Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs.

recurrent_dropout
  Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state.

batch_size
  Fixed batch size for layer

name
  An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.

trainable
  Whether the layer weights will be updated during training.

weights
  Initial weights for layer.

input_shape
  Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

Input shape

- if data_format='channels_first' 5D tensor with shape: (samples, time, channels, rows, cols)
- if data_format='channels_last' 5D tensor with shape: (samples, time, rows, cols, channels)

References

- Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting
  The current implementation does not include the feedback loop on the cells output
layer_cropping_1d

See Also
Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_depthwise_conv_2d, layer_separable_conv_1d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

layer_cropping_1d Cropping layer for 1D input (e.g. temporal sequence).

Description
It crops along the time dimension (axis 1).

Usage
layer_cropping_1d(object, cropping = c(1L, 1L), batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments
object Model or layer object
cropping int or list of int (length 2) How many units should be trimmed off at the beginning and end of the cropping dimension (axis 1). If a single int is provided, the same value will be used for both.
batch_size Fixed batch size for layer
name An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
trainable Whether the layer weights will be updated during training.
weights Initial weights for layer.

Input shape
3D tensor with shape (batch, axis_to_crop, features)

Output shape
3D tensor with shape (batch, cropped_axis, features)

See Also
Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_2d, layer_cropping_3d, layer_depthwise_conv_2d, layer_separable_conv_1d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d
layer_cropping_2d  Cropping layer for 2D input (e.g. picture).

Description

It crops along spatial dimensions, i.e. width and height.

Usage

layer_cropping_2d(object, cropping = list(c(0L, 0L), c(0L, 0L)),
                   data_format = NULL, batch_size = NULL, name = NULL,
                   trainable = NULL, weights = NULL)

Arguments

- object: Model or layer object
- cropping: int, or list of 2 ints, or list of 2 lists of 2 ints.
  - If int: the same symmetric cropping is applied to width and height.
  - If list of 2 ints: interpreted as two different symmetric cropping values for
    height and width: (symmetric_height_crop, symmetric_width_crop).
  - If list of 2 lists of 2 ints: interpreted as
    ((top_crop, bottom_crop), (left_crop, right_crop))
- data_format: A string, one of channels_last (default) or channels_first. The ordering
  of the dimensions in the inputs. channels_last corresponds to inputs with shape
  (batch, height, width, channels) while channels_first corresponds to inputs with shape
  (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at
  ~/.keras/keras.json. If you never set it, then it will be "channels_last".
- batch_size: Fixed batch size for layer
- name: An optional name string for the layer. Should be unique in a model (do not reuse
  the same name twice). It will be autogenerated if it isn't provided.
- trainable: Whether the layer weights will be updated during training.
- weights: Initial weights for layer.

Input shape

4D tensor with shape:

- If data_format is "channels_last": (batch, rows, cols, channels)
- If data_format is "channels_first": (batch, channels, rows, cols)

Output shape

4D tensor with shape:

- If data_format is "channels_last": (batch, cropped_rows, cropped_cols, channels)
- If data_format is "channels_first": (batch, channels, cropped_rows, cropped_cols)
layer_cropping_3d

Cropping layer for 3D data (e.g. spatial or spatio-temporal).

Description
Cropping layer for 3D data (e.g. spatial or spatio-temporal).

Usage
layer_cropping_3d(object, cropping = list(c(1L, 1L), c(1L, 1L), c(1L, 1L)), data_format = NULL, batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

object Model or layer object
cropping int, or list of 3 ints, or list of 3 lists of 2 ints.

• If int: the same symmetric cropping is applied to depth, height, and width.
• If list of 3 ints: interpreted as two different symmetric cropping values for depth, height, and width: (symmetric_dim1_crop, symmetric_dim2_crop, symmetric_dim3_crop).
• If list of 3 list of 2 ints: interpreted as ((left_dim1_crop, right_dim1_crop), (left_dim2_crop, right_dim2_crop), (left_dim3_crop, right_dim3_crop)).
data_format A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, spatial_dim1, spatial_dim2, spatial_dim3, channels) while channels_first corresponds to inputs with shape (batch, channels, spatial_dim1, spatial_dim2, spatial_dim3). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".
batch_size Fixed batch size for layer
name An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
trainable Whether the layer weights will be updated during training.
weights Initial weights for layer.

Input shape
5D tensor with shape:

• If data_format is "channels_last": (batch, first_axis_to_crop, second_axis_to_crop, third_axis_to_crop, depth)
• If data_format is "channels_first": (batch, depth, first_axis_to_crop, second_axis_to_crop, third_axis_to_crop)
Output shape

5D tensor with shape:

- If `data_format"channels_last": (batch, first_cropped_axis, second_cropped_axis, third_cropped_axis, depth)
- If `data_format"channels_first": (batch, depth, first_cropped_axis, second_cropped_axis, third_cropped_axis)

See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_depthwise_conv_2d, layer_separable_conv_1d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

layer_cudnn_gru

Fast GRU implementation backed by


Description

Can only be run on GPU, with the TensorFlow backend.

Usage

```python
layer_cudnn_gru(object, units, kernel_initializer = "glorot_uniform",
                 recurrent_initializer = "orthogonal", bias_initializer = "zeros",
                 kernel_regularizer = NULL, recurrent_regularizer = NULL,
                 bias_regularizer = NULL, activity_regularizer = NULL,
                 kernel_constraint = NULL, recurrent_constraint = NULL,
                 bias_constraint = NULL, return_sequences = FALSE,
                 return_state = FALSE, stateful = FALSE, input_shape = NULL,
                 batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
                 name = NULL, trainable = NULL, weights = NULL)
```

Arguments

- **object**: Model or layer object
- **units**: Positive integer, dimensionality of the output space.
- **kernel_initializer**: Initializer for the kernel weights matrix, used for the linear transformation of the inputs.
- **recurrent_initializer**: Initializer for the recurrent_kernel weights matrix, used for the linear transformation of the recurrent state.
- **bias_initializer**: Initializer for the bias vector.
Regularizer function applied to the kernel weights matrix.

recurrent_regularizer
Regularizer function applied to the recurrent_kernel weights matrix.

bias_regularizer
Regularizer function applied to the bias vector.

activity_regularizer
Regularizer function applied to the output of the layer (its "activation").

kernel_constraint
Constraint function applied to the kernel weights matrix.

recurrent_constraint
Constraint function applied to the recurrent_kernel weights matrix.

bias_constraint
Constraint function applied to the bias vector.

return_sequences
Boolean. Whether to return the last output in the output sequence, or the full sequence.

return_state
Boolean (default FALSE). Whether to return the last state in addition to the output.

stateful
Boolean (default FALSE). If TRUE, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.

input_shape
Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

batch_input_shape
Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.

batch_size
Fixed batch size for layer

dtype
The data type expected by the input, as a string (float32, float64, int32...)

name
An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.

trainable
Whether the layer weights will be updated during training.

weights
Initial weights for layer.

References

- On the Properties of Neural Machine Translation: Encoder-Decoder Approaches
- Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling
- A Theoretically Grounded Application of Dropout in Recurrent Neural Networks

See Also

Other recurrent layers: layer_cudnn_lstm, layer_gru, layer_lstm, layer_simple_rnn
**layer_cudnn_lstm**

Fast LSTM implementation backed by [CuDNN](https://developer.nvidia.com/cudnn).

**Description**

Can only be run on GPU, with the TensorFlow backend.

**Usage**

```python
layer_cudnn_lstm(object, units, kernel_initializer = "glorot_uniform",
        recurrent_initializer = "orthogonal", bias_initializer = "zeros",
        unit_forget_bias = TRUE, kernel_regularizer = NULL,
        recurrent_regularizer = NULL, bias_regularizer = NULL,
        activity_regularizer = NULL, kernel_constraint = NULL,
        recurrent_constraint = NULL, bias_constraint = NULL,
        return_sequences = FALSE, return_state = FALSE, stateful = FALSE,
        input_shape = NULL, batch_input_shape = NULL, batch_size = NULL,
        dtype = NULL, name = NULL, trainable = NULL, weights = NULL)
```

**Arguments**

- **object**: Model or layer object
- **units**: Positive integer, dimensionality of the output space.
- **kernel_initializer**: Initializer for the kernel weights matrix, used for the linear transformation of the inputs.
- **recurrent_initializer**: Initializer for the recurrent_kernel weights matrix, used for the linear transformation of the recurrent state.
- **bias_initializer**: Initializer for the bias vector.
- **unit_forget_bias**: Boolean. If TRUE, add 1 to the bias of the forget gate at initialization. Setting it to true will also force `bias_initializer="zeros"`. This is recommended in [Jozefowicz et al.](#).
- **kernel_regularizer**: Regularizer function applied to the kernel weights matrix.
- **recurrent_regularizer**: Regularizer function applied to the recurrent_kernel weights matrix.
- **bias_regularizer**: Regularizer function applied to the bias vector.
- **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation").
- **kernel_constraint**: Constraint function applied to the kernel weights matrix.
**recurrent_constraint**  
Constraint function applied to the recurrent_kernel weights matrix.

**bias_constraint**  
Constraint function applied to the bias vector.

**return_sequences**  
Boolean. Whether to return the last output in the output sequence, or the full sequence.

**return_state**  
Boolean (default FALSE). Whether to return the last state in addition to the output.

**stateful**  
Boolean (default FALSE). If TRUE, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.

**input_shape**  
Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

**batch_input_shape**  
Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.

**batch_size**  
Fixed batch size for layer

**dtype**  
The data type expected by the input, as a string (float32, float64, int32...)

**name**  
An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.

**trainable**  
Whether the layer weights will be updated during training.

**weights**  
Initial weights for layer.

**References**
- Long short-term memory (original 1997 paper)
- Supervised sequence labeling with recurrent neural networks
- A Theoretically Grounded Application of Dropout in Recurrent Neural Networks

**See Also**
Other recurrent layers: layer_cudnn_gru, layer_gru, layer_lstm, layer_simple_rnn

---

**layer_dense**  
*Add a densely-connected NN layer to an output*

**Description**
Implements the operation: output = activation(dot(input, kernel) + bias) where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bias is TRUE). Note: if the input to the layer has a rank greater than 2, then it is flattened prior to the initial dot product with kernel.
layer_dense

Usage

layer_dense(object, units, activation = NULL, use_bias = TRUE,
kernel_initializer = "glorot_uniform", bias_initializer = "zeros",
kernel_regularizer = NULL, bias_regularizer = NULL,
activity_regularizer = NULL, kernel_constraint = NULL,
bias_constraint = NULL, input_shape = NULL,
batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
name = NULL, trainable = NULL, weights = NULL)

Arguments

object  Model or layer object
units   Positive integer, dimensionality of the output space.
activation Name of activation function to use. If you don’t specify anything, no activation
            is applied (ie. "linear" activation: a(x) = x).
use_bias Whether the layer uses a bias vector.
kernal_initialiser  Initializer for the kernel weights matrix.
bias_initialiser    Initializer for the bias vector.
kernal_regularizer  Regularizer function applied to the kernel weights matrix.
bias_regularizer    Regularizer function applied to the bias vector.
activity_regularizer Regularizer function applied to the output of the layer (its "activation")..
kernal_constraint   Constraint function applied to the kernel weights matrix.
bias constrain    Constraint function applied to the bias vector.
input_shape  Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.
batch_input_shape  Shapes, including the batch size. For instance, batch_input_shape=c(10, 32)
                    indicates that the expected input will be batches of 10 32-dimensional vectors.
batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.
batch_size  Fixed batch size for layer
dtype  The data type expected by the input, as a string (float32, float64, int32...)
name  An optional name string for the layer. Should be unique in a model (do not reuse
      the same name twice). It will be autogenerated if it isn’t provided.
trainable Whether the layer weights will be updated during training.
weights  Initial weights for layer.
Input and Output Shapes

Input shape: nD tensor with shape: (batch_size, ..., input_dim). The most common situation would be a 2D input with shape (batch_size, input_dim).

Output shape: nD tensor with shape: (batch_size, ..., units). For instance, for a 2D input with shape (batch_size, input_dim), the output would have shape (batch_size, unit).

See Also

Other core layers: layer_activation, layer_activity_regularization, layer_dropout, layer_flatten, layer_input, layer_lambda, layer_masking, layer_permute, layer_repeat_vector, layer reshape

layer_depthwise_conv_2d

Depthwise separable 2D convolution.

Description

Depthwise Separable convolutions consists in performing just the first step in a depthwise spatial convolution (which acts on each input channel separately). The depth_multiplier argument controls how many output channels are generated per input channel in the depthwise step.

Usage

layer_depthwise_conv_2d(object, kernel_size, strides = c(1, 1), padding = "valid", depth_multiplier = 1, data_format = NULL, activation = NULL, use_bias = TRUE, depthwise_initializer = "glorot_uniform", bias_initializer = "zeros", depthwise_regularizer = NULL, bias_regularizer = NULL, activity_regularizer = NULL, depthwise_constraint = NULL, bias_constraint = NULL, input_shape = NULL, batch_input_shape = NULL, batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

- **object**: Model or layer object
- **kernel_size**: An integer or list of 2 integers, specifying the width and height of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides**: An integer or list of 2 integers, specifying the strides of the convolution along the width and height. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.
- **padding**: one of "valid" or "same" (case-insensitive).
depth_multiplier
The number of depthwise convolution output channels for each input channel. The total number of depthwise convolution output channels will be equal to filters_in * depth_multiplier.

data_format
A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".

activation
Activation function to use. If you don’t specify anything, no activation is applied (ie. "linear" activation: a(x) = x).

use_bias
Boolean, whether the layer uses a bias vector.

depthwise_initializer
Initializer for the depthwise kernel matrix.

bias_initializer
Initializer for the bias vector.

depthwise_regularizer
Regularizer function applied to the depthwise kernel matrix.

bias_regularizer
Regularizer function applied to the bias vector.

activity_regularizer
Regularizer function applied to the output of the layer (its "activation").

depthwise_constraint
Constraint function applied to the depthwise kernel matrix.

bias_constraint
Constraint function applied to the bias vector.

input_shape
Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

batch_input_shape
Shapes, including the batch size. For instance, batch_input_shape=call(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.

batch_size
Fixed batch size for layer

dtype
The data type expected by the input, as a string (float32, float64, int32...)

name
An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.

trainable
Whether the layer weights will be updated during training.

weights
Initial weights for layer.

See Also
Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d,
Layer that computes a dot product between samples in two tensors.

Usage

layer_dot(inputs, axes, normalize = FALSE, batch_size = NULL,
          dtype = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

- **inputs**: A list of input tensors (at least 2).
- **axes**: Integer or list of integers, axis or axes along which to take the dot product.
- **normalize**: Whether to L2-normalize samples along the dot product axis before taking the dot product. If set to TRUE, then the output of the dot product is the cosine proximity between the two samples. **kwargs: Standard layer keyword arguments.
- **batch_size**: Fixed batch size for layer.
- **dtype**: The data type expected by the input, as a string (float32, float64, int32...)
- **name**: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- **trainable**: Whether the layer weights will be updated during training.
- **weights**: Initial weights for layer.

Value

A tensor, the dot product of the samples from the inputs.

See Also

Other merge layers: layer_add, layer_average, layer_concatenate, layer_maximum, layer_minimum, layer_multiply, layer_subtract
layer_dropout

Applies Dropout to the input.

**Description**

Dropout consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting.

**Usage**

```
layer_dropout(object, rate, noise_shape = NULL, seed = NULL,
              input_shape = NULL, batch_input_shape = NULL, batch_size = NULL,
              name = NULL, trainable = NULL, weights = NULL)
```

**Arguments**

- **object**
  Model or layer object
- **rate**
  float between 0 and 1. Fraction of the input units to drop.
- **noise_shape**
  1D integer tensor representing the shape of the binary dropout mask that will be multiplied with the input. For instance, if your inputs have shape (batch_size, timesteps, features) and you want the dropout mask to be the same for all timesteps, you can use noise_shape=c(batch_size, 1, features).
- **seed**
  integer to use as random seed.
- **input_shape**
  Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.
- **batch_input_shape**
  Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.
- **batch_size**
  Fixed batch size for layer
- **name**
  An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- **trainable**
  Whether the layer weights will be updated during training.
- **weights**
  Initial weights for layer.

**See Also**

Other core layers: `layer_activation`, `layer_activity_regularization`, `layer_dense`, `layer_flatten`, `layer_input`, `layer_lambda`, `layer_masking`, `layer_permute`, `layer_repeat_vector`, `layer_reshape`

Other dropout layers: `layer_spatial_dropout_1d`, `layer_spatial_dropout_2d`, `layer_spatial_dropout_3d`
layer_embedding

Turns positive integers (indexes) into dense vectors of fixed size.

Description

For example, list(4L, 20L) -> list(c(0.25, 0.1), c(0.6, -0.2)) This layer can only be used as the first layer in a model.

Usage

layer_embedding(object, input_dim, output_dim,
    embeddings_initializer = "uniform", embeddings_regularizer = NULL,
    activity_regularizer = NULL, embeddings_constraint = NULL,
    mask_zero = FALSE, input_length = NULL, batch_size = NULL,
    name = NULL, trainable = NULL, weights = NULL)

Arguments

object \hspace{1cm} \text{Model or layer object}
input_dim \hspace{1cm} \text{int > 0. Size of the vocabulary, i.e. maximum integer index + 1.}
output_dim \hspace{1cm} \text{int >= 0. Dimension of the dense embedding.}
embeddings_initializer \hspace{1cm} \text{Initializer for the embeddings matrix.}
embeddings_regularizer \hspace{1cm} \text{Regularizer function applied to the embeddings matrix.}
activity_regularizer \hspace{1cm} \text{activity_regularizer}
embeddings_constraint \hspace{1cm} \text{Constraint function applied to the embeddings matrix.}
mask_zero \hspace{1cm} \text{Whether or not the input value 0 is a special "padding" value that should be masked out. This is useful when using recurrent layers, which may take variable length inputs. If this is TRUE then all subsequent layers in the model need to support masking or an exception will be raised. If mask_zero is set to TRUE, as a consequence, index 0 cannot be used in the vocabulary (input_dim should equal size of vocabulary + 1).}
input_length \hspace{1cm} \text{Length of input sequences, when it is constant. This argument is required if you are going to connect Flatten then Dense layers upstream (without it, the shape of the dense outputs cannot be computed).}
batch_size \hspace{1cm} \text{Fixed batch size for layer}
name \hspace{1cm} \text{An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.}
trainable \hspace{1cm} \text{Whether the layer weights will be updated during training.}
weights \hspace{1cm} \text{Initial weights for layer.}
Input shape

2D tensor with shape: (batch_size, sequence_length).

Output shape

3D tensor with shape: (batch_size, sequence_length, output_dim).

References

- A Theoretically Grounded Application of Dropout in Recurrent Neural Networks

layer_flatten  Flattens an input

Description

Flatten a given input, does not affect the batch size.

Usage

layer_flatten(object, data_format = NULL, input_shape = NULL, 
dtype = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

object  Model or layer object

data_format  A string. one of channels_last (default) or channels_first. The order-
ing of the dimensions in the inputs. The purpose of this argument is to pre-
serve weight ordering when switching a model from one data format to another.
channels_last corresponds to inputs with shape (batch, ..., channels) 
while channels_first corresponds to inputs with shape (batch, channels, ...).
It defaults to the image_data_format value found in your Keras config file at 
~/.keras/keras.json. If you never set it, then it will be "channels_last".

input_shape  Input shape (list of integers, does not include the samples axis) which is required 
when using this layer as the first layer in a model.

dtype  The data type expected by the input, as a string (float32, float64, int32...)

name  An optional name string for the layer. Should be unique in a model (do not reuse 
the same name twice). It will be autogenerated if it isn’t provided.

trainable  Whether the layer weights will be updated during training.

weights  Initial weights for layer.

See Also

Other core layers: layer_activation, layer_activity_regularization, layer_dense, layer_dropout, 
layer_input, layer_lambda, layer_masking, layer_permute, layer_repeat_vector, layer_reshape
layer_gaussian_dropout

*Apply multiplicative 1-centered Gaussian noise.*

**Description**

As it is a regularization layer, it is only active at training time.

**Usage**

```python
def layer_gaussian_dropout(object, rate, input_shape = None,
                          batch_input_shape = None, batch_size = None, dtype = None,
                          name = None, trainable = None, weights = None)
```

**Arguments**

- **object**: Model or layer object
- **rate**: float, drop probability (as with Dropout). The multiplicative noise will have standard deviation \(\sqrt{\text{rate} / (1 - \text{rate})}\).
- **input_shape**: Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.
- **batch_input_shape**: Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors.
- **batch_size**: Fixed batch size for layer
- **dtype**: The data type expected by the input, as a string (`float32`, `float64`, `int32`...)
- **name**: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- **trainable**: Whether the layer weights will be updated during training.
- **weights**: Initial weights for layer.

**Input shape**

Arbitrary. Use the keyword argument `input_shape` (list of integers, does not include the samples axis) when using this layer as the first layer in a model.

**Output shape**

Same shape as input.

**References**

- **Dropout: A Simple Way to Prevent Neural Networks from Overfitting** Srivastava, Hinton, et al. 2014
layer_gaussian_noise

See Also

Other noise layers: layer_alpha_dropout, layer_gaussian_noise

layer_gaussian_noise  Apply additive zero-centered Gaussian noise.

Description

This is useful to mitigate overfitting (you could see it as a form of random data augmentation). Gaussian Noise (GS) is a natural choice as corruption process for real valued inputs. As it is a regularization layer, it is only active at training time.

Usage

layer_gaussian_noise(object, stddev, input_shape = NULL,
batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
name = NULL, trainable = NULL, weights = NULL)

Arguments

object  Model or layer object
stddev  float, standard deviation of the noise distribution.
input_shape  Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.
batch_input_shape  Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.
batch_size  Fixed batch size for layer
dtype  The data type expected by the input, as a string (float32, float64, int32...)
name  An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
trainable  Whether the layer weights will be updated during training.
weights  Initial weights for layer.

Input shape

Arbitrary. Use the keyword argument input_shape (list of integers, does not include the samples axis) when using this layer as the first layer in a model.

Output shape

Same shape as input.
See Also

Other noise layers: `layer_alpha_dropout`, `layer_gaussian_dropout`

---

`layer_global_average_pooling_1d`

*Global average pooling operation for temporal data.*

**Description**

Global average pooling operation for temporal data.

**Usage**

```python
layer_global_average_pooling_1d(object, data_format = "channels_last",
  batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)
```

**Arguments**

- **object**: Model or layer object
- **data_format**: One of `channels_last` (default) or `channels_first`. The ordering of the dimensions in the inputs.
- **batch_size**: Fixed batch size for layer
- **name**: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- **trainable**: Whether the layer weights will be updated during training.
- **weights**: Initial weights for layer.

**Input shape**

3D tensor with shape: `(batch_size, steps, features)`.

**Output shape**

2D tensor with shape: `(batch_size, channels)`

**See Also**

Other pooling layers: `layer_average_pooling_1d`, `layer_average_pooling_2d`, `layer_average_pooling_3d`, `layer_global_average_pooling_2d`, `layer_global_average_pooling_3d`, `layer_global_max_pooling_1d`, `layer_global_max_pooling_2d`, `layer_global_max_pooling_3d`, `layer_max_pooling_1d`, `layer_max_pooling_2d`, `layer_max_pooling_3d`
Global average pooling operation for spatial data.

### Arguments

- **object**: Model or layer object
- **data_format**: A string, one of `channels_last` (default) or `channels_first`. The ordering of the dimensions in the inputs. `channels_last` corresponds to inputs with shape `(batch, height, width, channels)` while `channels_first` corresponds to inputs with shape `(batch, channels, height, width)`. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels_last".
- **batch_size**: Fixed batch size for layer
- **name**: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- **trainable**: Whether the layer weights will be updated during training.
- **weights**: Initial weights for layer.

### Input shape

- If `data_format='channels_last'`: 4D tensor with shape: `(batch_size, rows, cols, channels)`
- If `data_format='channels_first'`: 4D tensor with shape: `(batch_size, channels, rows, cols)`

### Output shape

2D tensor with shape: `(batch_size, channels)`

### See Also

Other pooling layers: `layer_average_pooling_1d`, `layer_average_pooling_2d`, `layer_average_pooling_3d`, `layer_global_average_pooling_1d`, `layer_global_average_pooling_2d`, `layer_global_average_pooling_3d`, `layer_global_max_pooling_1d`, `layer_global_max_pooling_2d`, `layer_global_max_pooling_3d`, `layer_max_pooling_1d`, `layer_max_pooling_2d`, `layer_max_pooling_3d`
layer_global_average_pooling_3d

Global Average pooling operation for 3D data.

Description

Global Average pooling operation for 3D data.

Usage

layer_global_average_pooling_3d(object, data_format = NULL,
batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

- object: Model or layer object
- data_format: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, spatial_dim1, spatial_dim2, spatial_dim3, channels) while channels_first corresponds to inputs with shape (batch, channels, spatial_dim1, spatial_dim2, spatial_dim3). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".
- batch_size: Fixed batch size for layer
- name: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- trainable: Whether the layer weights will be updated during training.
- weights: Initial weights for layer.

Input shape

- If data_format='channels_last': 5D tensor with shape: (batch_size, spatial_dim1, spatial_dim2, spatial_dim3, channels)
- If data_format='channels_first': 5D tensor with shape: (batch_size, channels, spatial_dim1, spatial_dim2, spatial_dim3)

Output shape

2D tensor with shape: (batch_size, channels)

See Also

Other pooling layers: layer_average_pooling_1d, layer_average_pooling_2d, layer_average_pooling_3d,
layer_global_average_pooling_1d, layer_global_average_pooling_2d, layer_global_average_pooling_3d,
layer_global_max_pooling_1d, layer_global_max_pooling_2d, layer_global_max_pooling_3d,
layer_max_pooling_1d, layer_max_pooling_2d, layer_max_pooling_3d,
layer_max_pooling_3d
layer_global_max_pooling_1d

Global max pooling operation for temporal data.

Description

Global max pooling operation for temporal data.

Usage

layer_global_max_pooling_1d(object, data_format = "channels_last",
batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

object Model or layer object
data_format One of channels_last (default) or channels_first. The ordering of the dimensions in the inputs.
batch_size Fixed batch size for layer
name An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
trainable Whether the layer weights will be updated during training.
weights Initial weights for layer.

Input shape

3D tensor with shape: (batch_size, steps, features).

Output shape

2D tensor with shape: (batch_size, channels)

See Also

Other pooling layers: layer_average_pooling_1d, layer_average_pooling_2d, layer_average_pooling_3d,
layer_global_average_pooling_1d, layer_global_average_pooling_2d, layer_global_average_pooling_3d,
layer_global_max_pooling_2d, layer_global_max_pooling_3d, layer_max_pooling_1d, layer_max_pooling_2d,
layer_max_pooling_3d
layer_global_max_pooling_2d

Global max pooling operation for spatial data.

Description
Global max pooling operation for spatial data.

Usage
layer_global_max_pooling_2d(object, data_format = NULL,
                             batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments
- object: Model or layer object
- data_format: A string, one of channels_last (default) or channels_first. The ordering
  of the dimensions in the inputs. channels_last corresponds to inputs with shape
  (batch, height, width, channels) while channels_first corresponds to inputs with shape
  (batch, channels, height, width). It defaults to the image_data_format value found in your
  Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".
- batch_size: Fixed batch size for layer
- name: An optional name string for the layer. Should be unique in a model (do not reuse
  the same name twice). It will be autogenerated if it isn't provided.
- trainable: Whether the layer weights will be updated during training.
- weights: Initial weights for layer.

Input shape
- If data_format='channels_last': 4D tensor with shape: (batch_size, rows, cols, channels)
- If data_format='channels_first': 4D tensor with shape: (batch_size, channels, rows, cols)

Output shape
2D tensor with shape: (batch_size, channels)

See Also
Other pooling layers: layer_average_pooling_1d, layer_average_pooling_2d, layer_average_pooling_3d,
layer_global_average_pooling_1d, layer_global_average_pooling_2d, layer_global_average_pooling_3d,
layer_global_max_pooling_1d, layer_global_max_pooling_2d, layer_global_max_pooling_3d, layer_max_pooling_1d,
layer_max_pooling_2d, layer_max_pooling_3d
layer_global_max_pooling_3d

Global Max pooling operation for 3D data.

Description

Global Max pooling operation for 3D data.

Usage

layer_global_max_pooling_3d(object, data_format = NULL,
batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

object Model or layer object
data_format A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, spatial_dim1, spatial_dim2, spatial_dim3, channels) while channels_first corresponds to inputs with shape (batch, channels, spatial_dim1, spatial_dim2, spatial_dim3). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".
batch_size Fixed batch size for layer
name An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
trainable Whether the layer weights will be updated during training.
weights Initial weights for layer.

Input shape

- If data_format='channels_last': 5D tensor with shape: (batch_size, spatial_dim1, spatial_dim2, spatial_dim3, channels)
- If data_format='channels_first': 5D tensor with shape: (batch_size, channels, spatial_dim1, spatial_dim2, spatial_dim3)

Output shape

2D tensor with shape: (batch_size, channels)

See Also

Other pooling layers: layer_average_pooling_1d, layer_average_pooling_2d, layer_average_pooling_3d, layer_global_average_pooling_1d, layer_global_average_pooling_2d, layer_global_average_pooling_3d, layer_global_max_pooling_1d, layer_global_max_pooling_2d, layer_max_pooling_1d, layer_max_pooling_2d, layer_max_pooling_3d
Description

There are two variants. The default one is based on 1406.1078v3 and has reset gate applied to hidden state before matrix multiplication. The other one is based on original 1406.1078v1 and has the order reversed.

Usage

layer_gru(object, units, activation = "tanh",
recurrent_activation = "hard_sigmoid", use_bias = TRUE,
return_sequences = FALSE, return_state = FALSE,
go_backwards = FALSE, stateful = FALSE, unroll = FALSE,
reset_after = FALSE, kernel_initializer = "glorot_uniform",
recurrent_initializer = "orthogonal", bias_initializer = "zeros",
kernel_regularizer = NULL, recurrent_regularizer = NULL,
bias_regularizer = NULL, activity_regularizer = NULL,
kernel_constraint = NULL, recurrent_constraint = NULL,
bias_constraint = NULL, dropout = 0, recurrent_dropout = 0,
input_shape = NULL, batch_input_shape = NULL, batch_size = NULL,
dtype = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

object Model or layer object
units Positive integer, dimensionality of the output space.
activation Activation function to use. Default: hyperbolic tangent (tanh). If you pass NULL, no activation is applied (ie. "linear" activation: a(x) = x).
recurrent_activation Activation function to use for the recurrent step.
use_bias Boolean, whether the layer uses a bias vector.
return_sequences Boolean. Whether to return the last output in the output sequence, or the full sequence.
return_state Boolean (default FALSE). Whether to return the last state in addition to the output.
go_backwards Boolean (default FALSE). If TRUE, process the input sequence backwards and return the reversed sequence.
stateful Boolean (default FALSE). If TRUE, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.
unroll Boolean (default FALSE). If TRUE, the network will be unrolled, else a symbolic loop will be used. Unrolling can speed-up a RNN, although it tends to be more memory-intensive. Unrolling is only suitable for short sequences.
reset_after  GRU convention (whether to apply reset gate after or before matrix multiplication). FALSE = "before" (default), TRUE = "after" (CuDNN compatible).

kernel_initializer
  Initializer for the kernel weights matrix, used for the linear transformation of the inputs.

recurrent_initializer
  Initializer for the recurrent_kernel weights matrix, used for the linear transformation of the recurrent state.

bias_initializer
  Initializer for the bias vector.

kernel_regularizer
  Regularizer function applied to the kernel weights matrix.

recurrent_regularizer
  Regularizer function applied to the recurrent_kernel weights matrix.

bias_regularizer
  Regularizer function applied to the bias vector.

activity_regularizer
  Regularizer function applied to the output of the layer (its "activation").

kernel_constraint
  Constraint function applied to the kernel weights matrix.

recurrent_constraint
  Constraint function applied to the recurrent_kernel weights matrix.

bias_constraint
  Constraint function applied to the bias vector.

dropout
  Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs.

recurrent_dropout
  Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state.

input_shape
  Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

batch_input_shape
  Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.

batch_size
  Fixed batch size for layer

dtype
  The data type expected by the input, as a string (float32, float64, int32...)

name
  An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.

trainable
  Whether the layer weights will be updated during training.

weights
  Initial weights for layer.
Details

The second variant is compatible with CuDNNGRU (GPU-only) and allows inference on CPU. Thus it has separate biases for kernel and recurrent_kernel. Use reset_after = TRUE and recurrent_activation = "sigmoid".

Input shapes

3D tensor with shape (batch_size, timesteps, input_dim), (Optional) 2D tensors with shape (batch_size, output_dim).

Output shape

• if return_state: a list of tensors. The first tensor is the output. The remaining tensors are the last states, each with shape (batch_size, units).
• if return_sequences: 3D tensor with shape (batch_size, timesteps, units).
• else, 2D tensor with shape (batch_size, units).

Masking

This layer supports masking for input data with a variable number of timesteps. To introduce masks to your data, use an embedding layer with the mask_zero parameter set to TRUE.

Statefulness in RNNs

You can set RNN layers to be 'stateful', which means that the states computed for the samples in one batch will be reused as initial states for the samples in the next batch. This assumes a one-to-one mapping between samples in different successive batches.

To enable statefulness:

• Specify stateful=TRUE in the layer constructor.
• Specify a fixed batch size for your model. For sequential models, pass batch_input_shape = c(...) to the first layer in your model. For functional models with 1 or more Input layers, pass batch_shape = c(...) to all the first layers in your model. This is the expected shape of your inputs including the batch size. It should be a vector of integers, e.g. c(32, 10, 100).
• Specify shuffle = FALSE when calling fit().

To reset the states of your model, call reset_states() on either a specific layer, or on your entire model.

Initial State of RNNs

You can specify the initial state of RNN layers symbolically by calling them with the keyword argument initial_state. The value of initial_state should be a tensor or list of tensors representing the initial state of the RNN layer.

You can specify the initial state of RNN layers numerically by calling reset_states with the keyword argument states. The value of states should be a numpy array or list of numpy arrays representing the initial state of the RNN layer.
layer_input

References

- Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation
- On the Properties of Neural Machine Translation: Encoder-Decoder Approaches
- Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling
- A Theoretically Grounded Application of Dropout in Recurrent Neural Networks

See Also

Other recurrent layers: layer_cudnn_gru, layer_cudnn_lstm, layer_lstm, layer_simple_rnn

layer_input  Input layer

Description

Layer to be used as an entry point into a graph.

Usage

layer_input(shape = NULL, batch_shape = NULL, name = NULL, dtype = NULL, sparse = FALSE, tensor = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>shape</td>
<td>Shape, not including the batch size. For instance, shape=c(32) indicates that the expected input will be batches of 32-dimensional vectors.</td>
</tr>
<tr>
<td>batch_shape</td>
<td>Shape, including the batch size. For instance, shape = c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_shape = list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.</td>
</tr>
<tr>
<td>name</td>
<td>An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.</td>
</tr>
<tr>
<td>dtype</td>
<td>The data type expected by the input, as a string (float32, float64, int32...)</td>
</tr>
<tr>
<td>sparse</td>
<td>Boolean, whether the placeholder created is meant to be sparse.</td>
</tr>
<tr>
<td>tensor</td>
<td>Existing tensor to wrap into the Input layer. If set, the layer will not create a placeholder tensor.</td>
</tr>
</tbody>
</table>

Value

A tensor

See Also

Other core layers: layer_activation, layer_activity_regularization, layer_dense, layer_dropout, layer_flatten, layer_lambda, layer_masking, layer_permute, layer_repeat_vector, layer_reshape
layer_lambda  Wraps arbitrary expression as a layer

Description

Wraps arbitrary expression as a layer

Usage

layer_lambda(object, f, output_shape = NULL, mask = NULL,
arguments = NULL, input_shape = NULL, batch_input_shape = NULL,
batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL,
weights = NULL)

Arguments

object        Model or layer object
f             The function to be evaluated. Takes input tensor as first argument.
output_shape  Expected output shape from the function (not required when using TensorFlow
              back-end).
mask          mask
arguments     optional named list of keyword arguments to be passed to the function.
input_shape   Dimensionality of the input (integer) not including the samples axis. This argument
               is required when using this layer as the first layer in a model.
batch_input_shape  Shapes, including the batch size. For instance, batch_input_shape=c(10, 32)
                   indicates that the expected input will be batches of 10 32-dimensional vectors.
batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number
                   of 32-dimensional vectors.
batch_size    Fixed batch size for layer
dtype         The data type expected by the input, as a string (float32, float64, int32...)
name          An optional name string for the layer. Should be unique in a model (do not reuse
              the same name twice). It will be autogenerated if it isn’t provided.
trainable     Whether the layer weights will be updated during training.
weights       Initial weights for layer.

Input shape

Arbitrary. Use the keyword argument input_shape (list of integers, does not include the samples axis) when using this layer as the first layer in a model.

Output shape

Arbitrary (based on tensor returned from the function)
layer_locally_connected_1d

See Also

Other core layers: layer_activation, layer_activity_regularization, layer_dense, layer_dropout, layer_flatten, layer_input, layer_masking, layer_permute, layer_repeat_vector, layer_reshape

Description

layer_locally_connected_1d() works similarly to layer_conv_1d(), except that weights are unshared, that is, a different set of filters is applied at each different patch of the input.

Usage

layer_locally_connected_1d(object, filters, kernel_size, strides = 1L, padding = "valid", data_format = NULL, activation = NULL, use_bias = TRUE, kernel_initializer = "glorot_uniform", bias_initializer = "zeros", kernel_regularizer = NULL, bias_regularizer = NULL, activity_regularizer = NULL, kernel_constraint = NULL, bias_constraint = NULL, batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Model or layer object</td>
</tr>
<tr>
<td>filters</td>
<td>Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution).</td>
</tr>
<tr>
<td>kernel_size</td>
<td>An integer or list of a single integer, specifying the length of the 1D convolution window.</td>
</tr>
<tr>
<td>strides</td>
<td>An integer or list of a single integer, specifying the stride length of the convolution. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.</td>
</tr>
<tr>
<td>padding</td>
<td>Currently only supports &quot;valid&quot; (case-insensitive). &quot;same&quot; may be supported in the future.</td>
</tr>
<tr>
<td>data_format</td>
<td>A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be &quot;channels_last&quot;.</td>
</tr>
<tr>
<td>activation</td>
<td>Activation function to use. If you don’t specify anything, no activation is applied (i.e. &quot;linear&quot; activation: a(x) = x).</td>
</tr>
<tr>
<td>use_bias</td>
<td>Boolean, whether the layer uses a bias vector.</td>
</tr>
</tbody>
</table>
kernel_initializer
   
   Initializer for the kernel weights matrix.

bias_initializer
   
   Initializer for the bias vector.

kernel_regularizer
   
   Regularizer function applied to the kernel weights matrix.

bias_regularizer
   
   Regularizer function applied to the bias vector.

activity_regularizer
   
   Regularizer function applied to the output of the layer (its "activation").

kernel_constraint
   
   Constraint function applied to the kernel matrix.

bias_constraint
   
   Constraint function applied to the bias vector.

batch_size
   
   Fixed batch size for layer

name
   
   An optional name string for the layer. Should be unique in a model (do not reuse
   the same name twice). It will be autogenerated if it isn’t provided.

trainable
   
   Whether the layer weights will be updated during training.

weights
   
   Initial weights for layer.

Input shape

3D tensor with shape: (batch_size, steps, input_dim)

Output shape

3D tensor with shape: (batch_size, new_steps, filters) steps value might have changed
due to padding or strides.

See Also

Other locally connected layers: layer_locally_connected_2d

layer_locally_connected_2d

Locally-connected layer for 2D inputs.

Description

layer_locally_connected_2d works similarly to layer_conv_2d(), except that weights are un-
shared, that is, a different set of filters is applied at each different patch of the input.
layer_locally_connected_2d

Usage

layer_locally_connected_2d(object, filters, kernel_size, strides = c(1L, 1L), padding = "valid", data_format = NULL, activation = NULL, use_bias = TRUE, kernel_initializer = "glorot_uniform", bias_initializer = "zeros", kernel_regularizer = NULL, bias_regularizer = NULL, activity_regularizer = NULL, kernel_constraint = NULL, bias_constraint = NULL, batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

object Model or layer object
filters Integer, the dimensionality of the output space (i.e. the number output of filters in the convolution).
kernel_size An integer or list of 2 integers, specifying the width and height of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
strides An integer or list of 2 integers, specifying the strides of the convolution along the width and height. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.
padding Currently only supports "valid" (case-insensitive). "same" may be supported in the future.
data_format A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, width, height, channels) while channels_first corresponds to inputs with shape (batch, channels, width, height). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".
activation Activation function to use. If you don't specify anything, no activation is applied (i.e. "linear" activation: $a(x) = x$).
use_bias Boolean, whether the layer uses a bias vector.
kernel_initializer
bias_initializer
kernel_regularizer
bias_regularizer
activity_regularizer
kernel_constraint
Constraint function applied to the kernel matrix.
layer_lstm

**bias_constraint**
Constraint function applied to the bias vector.

**batch_size**
Fixed batch size for layer.

**name**
An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.

**trainable**
Whether the layer weights will be updated during training.

**weights**
Initial weights for layer.

**Input shape**
4D tensor with shape: `(samples, channels, rows, cols)` if `data_format='channels_first'` or 4D tensor with shape: `(samples, rows, cols, channels)` if `data_format='channels_last'`.

**Output shape**
4D tensor with shape: `(samples, filters, new_rows, new_cols)` if `data_format='channels_first'` or 4D tensor with shape: `(samples, new_rows, new_cols, filters)` if `data_format='channels_last'`. rows and cols values might have changed due to padding.

**See Also**
Other locally connected layers: `layer_locally_connected_1d`

---

**layer_lstm**
*Long Short-Term Memory unit - Hochreiter 1997.*

**Description**
For a step-by-step description of the algorithm, see this tutorial.

**Usage**
```
layer_lstm(object, units, activation = "tanh",
           recurrent_activation = "hard_sigmoid", use_bias = TRUE,
           return_sequences = FALSE, return_state = FALSE,
           go_backwards = FALSE, stateful = FALSE, unroll = FALSE,
           kernel_initializer = "glorot_uniform",
           recurrent_initializer = "orthogonal", bias_initializer = "zeros",
           unit_forget_bias = TRUE, kernel_regularizer = NULL,
           recurrent_regularizer = NULL, bias_regularizer = NULL,
           activity_regularizer = NULL, kernel_constraint = NULL,
           recurrent_constraint = NULL, bias_constraint = NULL, dropout = 0,
           recurrent_dropout = 0, input_shape = NULL,
           batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
           name = NULL, trainable = NULL, weights = NULL)
```
Arguments

- **object**: Model or layer object
- **units**: Positive integer, dimensionality of the output space.
- **activation**: Activation function to use. Default: hyperbolic tangent (tanh). If you pass NULL, no activation is applied (i.e., "linear" activation: $a(x) = x$).
- **recurrent_activation**: Activation function to use for the recurrent step.
- **use_bias**: Boolean, whether the layer uses a bias vector.
- **return_sequences**: Boolean. Whether to return the last output in the output sequence, or the full sequence.
- **return_state**: Boolean (default FALSE). Whether to return the last state in addition to the output.
- **go_backwards**: Boolean (default FALSE). If TRUE, process the input sequence backwards and return the reversed sequence.
- **stateful**: Boolean (default FALSE). If TRUE, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.
- **unroll**: Boolean (default FALSE). If TRUE, the network will be unrolled, else a symbolic loop will be used. Unrolling can speed-up a RNN, although it tends to be more memory-intensive. Unrolling is only suitable for short sequences.
- **kernel_initializer**: Initializer for the kernel weights matrix, used for the linear transformation of the inputs.
- **recurrent_initializer**: Initializer for the recurrent_kernel weights matrix, used for the linear transformation of the recurrent state.
- **bias_initializer**: Initializer for the bias vector.
- **unit_forget_bias**: Boolean. If TRUE, add 1 to the bias of the forget gate at initialization. Setting it to true will also force bias_initializer="zeros". This is recommended in Jozefowicz et al.
- **kernel_regularizer**: Regularizer function applied to the kernel weights matrix.
- **recurrent_regularizer**: Regularizer function applied to the recurrent_kernel weights matrix.
- **bias_regularizer**: Regularizer function applied to the bias vector.
- **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation").
- **kernel_constraint**: Constraint function applied to the kernel weights matrix.
- **recurrent_constraint**: Constraint function applied to the recurrent_kernel weights matrix.
**bias_constraint**
Constraint function applied to the bias vector.

**dropout**
Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs.

**recurrent_dropout**
Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state.

**input_shape**
Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

**batch_input_shape**
Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors.

**batch_size**
Fixed batch size for layer

**dtype**
The data type expected by the input, as a string (`float32`, `float64`, `int32`,...)

**name**
An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.

**trainable**
Whether the layer weights will be updated during training.

**weights**
Initial weights for layer.

**Input shapes**
3D tensor with shape `(batch_size, timesteps, input_dim)`. (Optional) 2D tensors with shape `(batch_size, output_dim)`. 

**Output shape**
- if return_state: a list of tensors. The first tensor is the output. The remaining tensors are the last states, each with shape `(batch_size, units)`.
- if return_sequences: 3D tensor with shape `(batch_size, timesteps, units)`.
- else, 2D tensor with shape `(batch_size, units)`.

**Masking**
This layer supports masking for input data with a variable number of timesteps. To introduce masks to your data, use an embedding layer with the `mask_zero` parameter set to `TRUE`.

**Statefulness in RNNs**
You can set RNN layers to be ‘stateful’, which means that the states computed for the samples in one batch will be reused as initial states for the samples in the next batch. This assumes a one-to-one mapping between samples in different successive batches.

To enable statefulness:
- Specify `stateful=TRUE` in the layer constructor.
• Specify a fixed batch size for your model. For sequential models, pass `batch_input_shape = c(\ldots)` to the first layer in your model. For functional models with 1 or more `Input` layers, pass `batch_shape = c(\ldots)` to all the first layers in your model. This is the expected shape of your inputs including the batch size. It should be a vector of integers, e.g. `c(32, 10, 100)`.

• Specify `shuffle = FALSE` when calling `fit()`.

To reset the states of your model, call `reset_states()` on either a specific layer, or on your entire model.

**Initial State of RNNs**

You can specify the initial state of RNN layers symbolically by calling them with the keyword argument `initial_state`. The value of `initial_state` should be a tensor or list of tensors representing the initial state of the RNN layer.

You can specify the initial state of RNN layers numerically by calling `reset_states` with the keyword argument `states`. The value of `states` should be a numpy array or list of numpy arrays representing the initial state of the RNN layer.

**References**

• Long short-term memory (original 1997 paper)

• Supervised sequence labeling with recurrent neural networks

• A Theoretically Grounded Application of Dropout in Recurrent Neural Networks

**See Also**

Other recurrent layers: `layer_cudnn_gru`, `layer_cudnn_lstm`, `layer_gru`, `layer_simple_rnn`

Other recurrent layers: `layer_cudnn_gru`, `layer_cudnn_lstm`, `layer_gru`, `layer_simple_rnn`

---

**layer_masking**

**Masks a sequence by using a mask value to skip timesteps.**

**Description**

For each timestep in the input tensor (dimension #1 in the tensor), if all values in the input tensor at that timestep are equal to `mask_value`, then the timestep will be masked (skipped) in all downstream layers (as long as they support masking). If any downstream layer does not support masking yet receives such an input mask, an exception will be raised.

**Usage**

```r
layer_masking(object, mask_value = 0, input_shape = NULL,
batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
name = NULL, trainable = NULL, weights = NULL)
```
**Arguments**

- **object**: Model or layer object.
- **mask_value**: float, mask value.
- **input_shape**: Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.
- **batch_input_shape**: Shapes, including the batch size. For instance, `batch_input_shape=s(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors.
- **batch_size**: Fixed batch size for layer.
- **dtype**: The data type expected by the input, as a string (`float32`, `float64`, `int32`...).
- **name**: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- **trainable**: Whether the layer weights will be updated during training.
- **weights**: Initial weights for layer.

**See Also**

Other core layers: `layer_activation`, `layer_activity_regularization`, `layer_dense`, `layer_dropout`, `layer_flatten`, `layer_input`, `layer_lambda`, `layer_permute`, `layer_repeat_vector`, `layer_reshape`.

---

**layer_maximum**

*Layer that computes the maximum (element-wise) a list of inputs.*

**Description**

It takes as input a list of tensors, all of the same shape, and returns a single tensor (also of the same shape).

**Usage**

```
layer_maximum(inputs, batch_size = NULL, dtype = NULL, name = NULL,
              trainable = NULL, weights = NULL)
```

**Arguments**

- **inputs**: A list of input tensors (at least 2).
- **batch_size**: Fixed batch size for layer.
- **dtype**: The data type expected by the input, as a string (`float32`, `float64`, `int32`...).
- **name**: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- **trainable**: Whether the layer weights will be updated during training.
- **weights**: Initial weights for layer.
layer_max_pooling_1d

Value

A tensor, the element-wise maximum of the inputs.

See Also

Other merge layers: layer_add, layer_average, layer_concatenate, layer_dot, layer_minimum, layer_multiply, layer_subtract

layer_max_pooling_1d  Max pooling operation for temporal data.

Description

Max pooling operation for temporal data.

Usage

layer_max_pooling_1d(object, pool_size = 2L, strides = NULL, padding = "valid", batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Model or layer object</td>
</tr>
<tr>
<td>pool_size</td>
<td>Integer, size of the max pooling windows.</td>
</tr>
<tr>
<td>strides</td>
<td>Integer, or NULL. Factor by which to downscale. E.g. 2 will halve the input. If NULL, it will default to pool_size.</td>
</tr>
<tr>
<td>padding</td>
<td>One of &quot;valid&quot; or &quot;same&quot; (case-insensitive).</td>
</tr>
<tr>
<td>batch_size</td>
<td>Fixed batch size for layer</td>
</tr>
<tr>
<td>name</td>
<td>An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.</td>
</tr>
<tr>
<td>trainable</td>
<td>Whether the layer weights will be updated during training.</td>
</tr>
<tr>
<td>weights</td>
<td>Initial weights for layer</td>
</tr>
</tbody>
</table>

Input shape

3D tensor with shape: (batch_size, steps, features).

Output shape

3D tensor with shape: (batch_size, downsampling_steps, features).
layer_max_pooling_2d

Max pooling operation for spatial data.

Description

Max pooling operation for spatial data.

Usage

layer_max_pooling_2d(object, pool_size = c(2L, 2L), strides = NULL,
padding = "valid", data_format = NULL, batch_size = NULL,
name = NULL, trainable = NULL, weights = NULL)

Arguments

object       Model or layer object
pool_size    integer or list of 2 integers, factors by which to downscale (vertical, horizontal).
             (2, 2) will halve the input in both spatial dimension. If only one integer is specified, the same window length will be used for both dimensions.
strides      Integer, list of 2 integers, or NULL. Strides values. If NULL, it will default to pool_size.
padding      One of "valid" or "same" (case-insensitive).
data_format   A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".
batch_size   Fixed batch size for layer
name         An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
trainable    Whether the layer weights will be updated during training.
weights      Initial weights for layer.

Input shape

- If data_format='channels_last': 4D tensor with shape: (batch_size, rows, cols, channels)
- If data_format='channels_first': 4D tensor with shape: (batch_size, channels, rows, cols)
layer_max_pooling_3d

Output shape

- If data_format='channels_last': 4D tensor with shape: (batch_size, pooled_rows, pooled_cols, channels)
- If data_format='channels_first': 4D tensor with shape: (batch_size, channels, pooled_rows, pooled_cols)

See Also

Other pooling layers: layer_average_pooling_1d, layer_average_pooling_2d, layer_average_pooling_3d, layer_global_average_pooling_1d, layer_global_average_pooling_2d, layer_global_average_pooling_3d, layer_global_max_pooling_1d, layer_global_max_pooling_2d, layer_global_max_pooling_3d, layer_max_pooling_1d, layer_max_pooling_3d

layer_max_pooling_3d  Max pooling operation for 3D data (spatial or spatio-temporal).

Description

Max pooling operation for 3D data (spatial or spatio-temporal).

Usage

layer_max_pooling_3d(object, pool_size = c(2L, 2L, 2L), strides = NULL, padding = "valid", data_format = NULL, batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Model or layer object</td>
</tr>
<tr>
<td>pool_size</td>
<td>list of 3 integers, factors by which to downscale (dim1, dim2, dim3). (2, 2, 2) will halve the size of the 3D input in each dimension.</td>
</tr>
<tr>
<td>strides</td>
<td>list of 3 integers, or NULL. Strides values.</td>
</tr>
<tr>
<td>padding</td>
<td>One of &quot;valid&quot; or &quot;same&quot; (case-insensitive).</td>
</tr>
<tr>
<td>data_format</td>
<td>A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, spatial_dim1, spatial_dim2, spatial_dim3, channels) while channels_first corresponds to inputs with shape (batch, channels, spatial_dim1, spatial_dim2, spatial_dim3) It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be &quot;channels_last&quot;.</td>
</tr>
<tr>
<td>batch_size</td>
<td>Fixed batch size for layer</td>
</tr>
<tr>
<td>name</td>
<td>An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.</td>
</tr>
<tr>
<td>trainable</td>
<td>Whether the layer weights will be updated during training.</td>
</tr>
<tr>
<td>weights</td>
<td>Initial weights for layer</td>
</tr>
</tbody>
</table>
**layer_minimum**

Layer that computes the minimum (element-wise) a list of inputs.

**Input shape**

- If `data_format='channels_last'`: 5D tensor with shape: `(batch_size, spatial_dim1, spatial_dim2, spatial_dim3, channels)`
- If `data_format='channels_first'`: 5D tensor with shape: `(batch_size, channels, spatial_dim1, spatial_dim2, spatial_dim3)`

**Output shape**

- If `data_format='channels_last'`: 5D tensor with shape: `(batch_size, pooled_dim1, pooled_dim2, pooled_dim3, channels)`
- If `data_format='channels_first'`: 5D tensor with shape: `(batch_size, channels, pooled_dim1, pooled_dim2, pooled_dim3)`

**See Also**

Other pooling layers: `layer_average_pooling_1d`, `layer_average_pooling_2d`, `layer_average_pooling_3d`, `layer_global_average_pooling_1d`, `layer_global_average_pooling_2d`, `layer_global_average_pooling_3d`, `layer_global_max_pooling_1d`, `layer_global_max_pooling_2d`, `layer_global_max_pooling_3d`, `layer_max_pooling_1d`, `layer_max_pooling_2d`

**Description**

It takes as input a list of tensors, all of the same shape, and returns a single tensor (also of the same shape).

**Usage**

```python
layer_minimum(inputs, batch_size = NULL, dtype = NULL, name = NULL,
              trainable = NULL, weights = NULL)
```

**Arguments**

- `inputs`: A list of input tensors (at least 2).
- `batch_size`: Fixed batch size for layer
- `dtype`: The data type expected by the input, as a string (float32, float64, int32...)
- `name`: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- `trainable`: Whether the layer weights will be updated during training.
- `weights`: Initial weights for layer.

**Value**

A tensor, the element-wise maximum of the inputs.

**See Also**

Other merge layers: `layer_add`, `layer_average`, `layer_concatenate`, `layer_dot`, `layer maximum`, `layer_multiply`, `layer_subtract`
layer_multiply

Layer that multiplies (element-wise) a list of inputs.

Description

It takes as input a list of tensors, all of the same shape, and returns a single tensor (also of the same shape).

Usage

layer_multiply(inputs, batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

inputs A list of input tensors (at least 2).
batch_size Fixed batch size for layer
dtype The data type expected by the input, as a string (float32, float64, int32...)
name An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
trainable Whether the layer weights will be updated during training.
weights Initial weights for layer.

Value

A tensor, the element-wise product of the inputs.

See Also

Other merge layers: layer_add, layer_average, layer_concatenate, layer_dot, layer_maximum, layer_minimum, layer_subtract

layer_permute

Permute the dimensions of an input according to a given pattern

Description

Permute the dimensions of an input according to a given pattern

Usage

layer_permute(object, dims, input_shape = NULL, batch_input_shape = NULL, batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL, weights = NULL)
layer_repeat_vector

Arguments

object  Model or layer object

dims  List of integers. Permutation pattern, does not include the samples dimension. Indexing starts at 1. For instance, (2, 1) permutes the first and second dimension of the input.

input_shape  Input shape (list of integers, does not include the samples axis) which is required when using this layer as the first layer in a model.

batch_input_shape  Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.

batch_size  Fixed batch size for layer

dtype  The data type expected by the input, as a string (float32, float64, int32...)

name  An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.

trainable  Whether the layer weights will be updated during training.

weights  Initial weights for layer.

Input and Output Shapes

Input shape: Arbitrary
Output shape: Same as the input shape, but with the dimensions re-ordered according to the specified pattern.

Note

Useful for e.g. connecting RNNs and convnets together.

See Also

Other core layers: layer_activation, layer_activity_regularization, layer_dense, layer_dropout, layer_flatten, layer_input, layer_lambda, layer_masking, layer_repeat_vector, layer_reshape

layer_repeat_vector  Repeats the input n times.

Description

Repeats the input n times.

Usage

layer_repeat_vector(object, n, batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)
layer_reshape

Arguments

- **object**: Model or layer object
- **n**: integer, repetition factor.
- **batch_size**: Fixed batch size for layer
- **name**: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- **trainable**: Whether the layer weights will be updated during training.
- **weights**: Initial weights for layer.

Input shape

2D tensor of shape (num_samples, features).

Output shape

3D tensor of shape (num_samples, n, features).

See Also

Other core layers: layer_activation, layer_activity_regularization, layer_dense, layer_dropout, layer_flatten, layer_input, layer_lambda, layer_masking, layer_permute, layer_reshape

---

layer_reshape: Reshapes an output to a certain shape.

Description

Reshapes an output to a certain shape.

Usage

layer_reshape(object, target_shape, input_shape = NULL, batch_input_shape = NULL, batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

- **object**: Model or layer object
- **target_shape**: List of integers, does not include the samples dimension (batch size).
- **input_shape**: Input shape (list of integers, does not include the samples axis) which is required when using this layer as the first layer in a model.
- **batch_input_shape**: Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.
layer_separable_conv_1d

Description

Separable convolutions consist in first performing a depthwise spatial convolution (which acts on each input channel separately) followed by a pointwise convolution which mixes together the resulting output channels. The depth_multiplier argument controls how many output channels are generated per input channel in the depthwise step. Intuitively, separable convolutions can be understood as a way to factorize a convolution kernel into two smaller kernels, or as an extreme version of an Inception block.

Usage

layer_separable_conv_1d(object, filters, kernel_size, strides = 1, padding = "valid", data_format = "channels_last", dilation_rate = 1, depth_multiplier = 1, activation = NULL, use_bias = TRUE, depthwise_initializer = "glorot_uniform", pointwise_initializer = "glorot_uniform", bias_initializer = "zeros", depthwise_regularizer = NULL, pointwise_regularizer = NULL, bias_regularizer = NULL, activity_regularizer = NULL, depthwise_constraint = NULL, pointwise_constraint = NULL, bias_constraint = NULL, input_shape = NULL, batch_input_shape = NULL, batch_size = NULL, dtype = NULL, name = NULL, trainable = NULL, weights = NULL)
### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Model or layer object</td>
</tr>
<tr>
<td>filters</td>
<td>Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).</td>
</tr>
<tr>
<td>kernel_size</td>
<td>An integer or list of 2 integers, specifying the width and height of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.</td>
</tr>
<tr>
<td>strides</td>
<td>An integer or list of 2 integers, specifying the strides of the convolution along the width and height. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.</td>
</tr>
<tr>
<td>padding</td>
<td>one of &quot;valid&quot; or &quot;same&quot; (case-insensitive).</td>
</tr>
<tr>
<td>data_format</td>
<td>A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be &quot;channels_last&quot;.</td>
</tr>
<tr>
<td>dilation_rate</td>
<td>an integer or list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any stride value != 1.</td>
</tr>
<tr>
<td>depth_multiplier</td>
<td>The number of depthwise convolution output channels for each input channel. The total number of depthwise convolution output channels will be equal to filters_in * depth_multiplier.</td>
</tr>
<tr>
<td>activation</td>
<td>Activation function to use. If you don’t specify anything, no activation is applied (i.e. &quot;linear&quot; activation: a(x) = x).</td>
</tr>
<tr>
<td>use_bias</td>
<td>Boolean, whether the layer uses a bias vector.</td>
</tr>
<tr>
<td>depthwise_initializer</td>
<td>Initializer for the depthwise kernel matrix.</td>
</tr>
<tr>
<td>pointwise_initializer</td>
<td>Initializer for the pointwise kernel matrix.</td>
</tr>
<tr>
<td>bias_initializer</td>
<td>Initializer for the bias vector.</td>
</tr>
<tr>
<td>depthwise_regularizer</td>
<td>Regularizer function applied to the depthwise kernel matrix.</td>
</tr>
<tr>
<td>pointwise_regularizer</td>
<td>Regularizer function applied to the pointwise kernel matrix.</td>
</tr>
<tr>
<td>bias_regularizer</td>
<td>Regularizer function applied to the bias vector.</td>
</tr>
<tr>
<td>activity_regularizer</td>
<td>Regularizer function applied to the output of the layer (its &quot;activation&quot;).</td>
</tr>
</tbody>
</table>
layer_separable_conv_2d

depthwise_constraint
   Constraint function applied to the depthwise kernel matrix.

pointwise_constraint
   Constraint function applied to the pointwise kernel matrix.

bias_constraint
   Constraint function applied to the bias vector.

input_shape
   Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

batch_input_shape
   Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.

batch_size
   Fixed batch size for layer.

dtype
   The data type expected by the input, as a string (float32, float64, int32...)

name
   An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.

trainable
   Whether the layer weights will be updated during training.

weights
   Initial weights for layer.

Input shape

3D tensor with shape: (batch, channels, steps) if data_format='channels_first' or 3D tensor with shape: (batch, steps, channels) if data_format='channels_last'.

Output shape

3D tensor with shape: (batch, filters, new_steps) if data_format='channels_first' or 3D tensor with shape: (batch, new_steps, filters) if data_format='channels_last'. new_steps values might have changed due to padding or strides.

See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_depthwise_conv_2d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

layer_separable_conv_2d

Separable 2D convolution.
Description

Separable convolutions consist in first performing a depthwise spatial convolution (which acts on each input channel separately) followed by a pointwise convolution which mixes together the resulting output channels. The depth_multiplier argument controls how many output channels are generated per input channel in the depthwise step. Intuitively, separable convolutions can be understood as a way to factorize a convolution kernel into two smaller kernels, or as an extreme version of an Inception block.

Usage

```
layer_separable_conv_2d(object, filters, kernel_size, strides = c(1, 1),
  padding = "valid", data_format = NULL, dilation_rate = 1,
  depth_multiplier = 1, activation = NULL, use_bias = TRUE,
  depthwise_initializer = "glorot_uniform",
  pointwise_initializer = "glorot_uniform", bias_initializer = "zeros",
  depthwise_regularizer = NULL, pointwise_regularizer = NULL,
  bias_regularizer = NULL, activity_regularizer = NULL,
  depthwise_constraint = NULL, pointwise_constraint = NULL,
  bias_constraint = NULL, input_shape = NULL,
  batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
  name = NULL, trainable = NULL, weights = NULL)
```

Arguments

- **object**: Model or layer object
- **filters**: Integer, the dimensionality of the output space (i.e. the number of output filters in the convolution).
- **kernel_size**: An integer or list of 2 integers, specifying the width and height of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- **strides**: An integer or list of 2 integers, specifying the strides of the convolution along the width and height. Can be a single integer to specify the same value for all spatial dimensions. Specifying any stride value != 1 is incompatible with specifying any dilation_rate value != 1.
- **padding**: One of "valid" or "same" (case-insensitive).
- **data_format**: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, height, width, channels) while channels_first corresponds to inputs with shape (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".
- **dilation_rate**: An integer or list of 2 integers, specifying the dilation rate to use for dilated convolution. Can be a single integer to specify the same value for all spatial dimensions. Currently, specifying any dilation_rate value != 1 is incompatible with specifying any stride value != 1.
depth_multiplier
The number of depthwise convolution output channels for each input channel. The total number of depthwise convolution output channels will be equal to filters_in * depth_multiplier.

activation
Activation function to use. If you don’t specify anything, no activation is applied (ie. "linear" activation: a(x) = x).

use_bias
Boolean, whether the layer uses a bias vector.

depthwise_initializer
Initializer for the depthwise kernel matrix.

pointwise_initializer
Initializer for the pointwise kernel matrix.

bias_initializer
Initializer for the bias vector.

depthwise_regularizer
Regularizer function applied to the depthwise kernel matrix.

pointwise_regularizer
Regularizer function applied to the pointwise kernel matrix.

bias_regularizer
Regularizer function applied to the bias vector.

activity_regularizer
Regularizer function applied to the output of the layer (its "activation").

depthwise_constraint
Constraint function applied to the depthwise kernel matrix.

pointwise_constraint
Constraint function applied to the pointwise kernel matrix.

bias_constraint
Constraint function applied to the bias vector.

input_shape
Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

batch_input_shape
Shapes, including the batch size. For instance, batch_input_shape=(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.

batch_size
Fixed batch size for layer.

dtype
The data type expected by the input, as a string (float32, float64, int32...)

name
An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.

trainable
Whether the layer weights will be updated during training.

weights
Initial weights for layer.

Input shape
4D tensor with shape: (batch, channels, rows, cols) if data_format='channels_first' or 4D tensor with shape: (batch, rows, cols, channels) if data_format='channels_last'.
Output shape

4D tensor with shape: (batch, filters, new_rows, new_cols) if data_format='channels_first'
or 4D tensor with shape: (batch, new_rows, new_cols, filters) if data_format='channels_last'.
rows and cols values might have changed due to padding.

See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d, layer_conv_3d, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_deeplineconv_2d, layer_separable_conv_1d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

layer_simple_rnn  Fully-connected RNN where the output is to be fed back to input.

Description

Fully-connected RNN where the output is to be fed back to input.

Usage

layer_simple_rnn(object, units, activation = "tanh", use_bias = TRUE,
return_sequences = FALSE, return_state = FALSE,
go_backwards = FALSE, stateful = FALSE, unroll = FALSE,
kernel_initializer = "glorot_uniform",
recurrent_initializer = "orthogonal", bias_initializer = "zeros",
kernel_regularizer = NULL, recurrent_regularizer = NULL,
bias_regularizer = NULL, activity_regularizer = NULL,
kernel_constraint = NULL, recurrent_constraint = NULL,
bias_constraint = NULL, dropout = 0, recurrent_dropout = 0,
input_shape = NULL, batch_input_shape = NULL, batch_size = NULL,
dtype = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Model or layer object</td>
</tr>
<tr>
<td>units</td>
<td>Positive integer, dimensionality of the output space.</td>
</tr>
<tr>
<td>activation</td>
<td>Activation function to use. Default: hyperbolic tangent (tanh). If you pass</td>
</tr>
<tr>
<td></td>
<td>NULL, no activation is applied (ie. &quot;linear&quot; activation: $a(x) = x$).</td>
</tr>
<tr>
<td>use_bias</td>
<td>Boolean, whether the layer uses a bias vector.</td>
</tr>
<tr>
<td>return_sequences</td>
<td>Boolean. Whether to return the last output in the output sequence, or the</td>
</tr>
<tr>
<td></td>
<td>full sequence.</td>
</tr>
<tr>
<td>return_state</td>
<td>Boolean (default FALSE). Whether to return the last state in addition to</td>
</tr>
<tr>
<td></td>
<td>the output.</td>
</tr>
</tbody>
</table>
go_backwards  Boolean (default FALSE). If TRUE, process the input sequence backwards and return the reversed sequence.

stateful  Boolean (default FALSE). If TRUE, the last state for each sample at index i in a batch will be used as initial state for the sample of index i in the following batch.

unroll  Boolean (default FALSE). If TRUE, the network will be unrolled, else a symbolic loop will be used. Unrolling can speed-up a RNN, although it tends to be more memory-intensive. Unrolling is only suitable for short sequences.

kernel_initializer  

recurrent_initializer  

bias_initializer  

kernel_regularizer  

recurrent_regularizer  

bias_regularizer  

activity_regularizer  

kernel_constraint  

recurrent_constraint  

bias_constraint  

dropout  Float between 0 and 1. Fraction of the units to drop for the linear transformation of the inputs.

recurrent_dropout  Float between 0 and 1. Fraction of the units to drop for the linear transformation of the recurrent state.

input_shape  Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.

batch_input_shape  Shapes, including the batch size. For instance, batch_input_shape=c(10, 32) indicates that the expected input will be batches of 10 32-dimensional vectors. batch_input_shape=list(NULL, 32) indicates batches of an arbitrary number of 32-dimensional vectors.

batch_size  Fixed batch size for layer

dtype  The data type expected by the input, as a string (float32, float64, int32...)

name  An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
trainable Whether the layer weights will be updated during training.
weights Initial weights for layer.

Input shapes
3D tensor with shape (batch_size, timesteps, input_dim), (Optional) 2D tensors with shape (batch_size, output_dim).

Output shape
- if return_state: a list of tensors. The first tensor is the output. The remaining tensors are the last states, each with shape (batch_size, units).
- if return_sequences: 3D tensor with shape (batch_size, timesteps, units).
- else, 2D tensor with shape (batch_size, units).

Masking
This layer supports masking for input data with a variable number of timesteps. To introduce masks to your data, use an embedding layer with the mask_zero parameter set to TRUE.

Statefulness in RNNs
You can set RNN layers to be 'stateful', which means that the states computed for the samples in one batch will be reused as initial states for the samples in the next batch. This assumes a one-to-one mapping between samples in different successive batches.

To enable statefulness:
- Specify stateful=TRUE in the layer constructor.
- Specify a fixed batch size for your model. For sequential models, pass batch_input_shape = c(…) to the first layer in your model. For functional models with 1 or more Input layers, pass batch_shape = c(…) to all the first layers in your model. This is the expected shape of your inputs including the batch size. It should be a vector of integers, e.g. c(32, 10, 100).
- Specify shuffle = FALSE when calling fit().

To reset the states of your model, call reset_states() on either a specific layer, or on your entire model.

Initial State of RNNs
You can specify the initial state of RNN layers symbolically by calling them with the keyword argument initial_state. The value of initial_state should be a tensor or list of tensors representing the initial state of the RNN layer.

You can specify the initial state of RNN layers numerically by calling reset_states with the keyword argument states. The value of states should be a numpy array or list of numpy arrays representing the initial state of the RNN layer.

References
- A Theoretically Grounded Application of Dropout in Recurrent Neural Networks
layer_spatial_dropout_1d

Spatial 1D version of Dropout.

Description

This version performs the same function as Dropout, however it drops entire 1D feature maps instead of individual elements. If adjacent frames within feature maps are strongly correlated (as is normally the case in early convolution layers) then regular dropout will not regularize the activations and will otherwise just result in an effective learning rate decrease. In this case, layer_spatial_dropout_1d will help promote independence between feature maps and should be used instead.

Usage

layer_spatial_dropout_1d(object, rate, batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

object            Model or layer object
rate              float between 0 and 1. Fraction of the input units to drop.
batch_size        Fixed batch size for layer
name              An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
trainable         Whether the layer weights will be updated during training.
weights           Initial weights for layer.

Input shape

3D tensor with shape: (samples, timesteps, channels)

Output shape

Same as input

References

- Efficient Object Localization Using Convolutional Networks

See Also

Other dropout layers: layer_dropout, layerSpatialDropout_2d, layerSpatialDropout_3d
**layer_spatial_dropout_2d**

*Spatial 2D version of Dropout.*

**Description**

This version performs the same function as Dropout, however it drops entire 2D feature maps instead of individual elements. If adjacent pixels within feature maps are strongly correlated (as is normally the case in early convolution layers) then regular dropout will not regularize the activations and will otherwise just result in an effective learning rate decrease. In this case, `layer_spatial_dropout_2d` will help promote independence between feature maps and should be used instead.

**Usage**

```python
layer_spatial_dropout_2d(object, rate, data_format = NULL,
                        batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)
```

**Arguments**

- **object**: Model or layer object
- **rate**: float between 0 and 1. Fraction of the input units to drop.
- **data_format**: 'channels_first' or 'channels_last'. In 'channels_first' mode, the channels dimension (the depth) is at index 1, in 'channels_last' mode is it at index 3. It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be "channels_last".
- **batch_size**: Fixed batch size for layer
- **name**: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- **trainable**: Whether the layer weights will be updated during training.
- **weights**: Initial weights for layer.

**Input shape**

4D tensor with shape: (samples, channels, rows, cols) if data_format='channels_first' or 4D tensor with shape: (samples, rows, cols, channels) if data_format='channels_last'.

**Output shape**

Same as input

**References**

- [Efficient Object Localization Using Convolutional Networks](https://arxiv.org/abs/1603.09025)

**See Also**

Other dropout layers: `layer_dropout, layer_spatial_dropout_1d, layer_spatial_dropout_3d`
layer_spatial_dropout_3d

Layer 3D version of Dropout.

Description

This version performs the same function as Dropout, however it drops entire 3D feature maps instead of individual elements. If adjacent voxels within feature maps are strongly correlated (as is normally the case in early convolution layers) then regular dropout will not regularize the activations and will otherwise just result in an effective learning rate decrease. In this case, layer_spatial_dropout_3d will help promote independence between feature maps and should be used instead.

Usage

layer_spatial_dropout_3d(object, rate, data_format = NULL,
batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)

Arguments

object Model or layer object
rate float between 0 and 1. Fraction of the input units to drop.
data_format 'channels_first' or 'channels_last'. In 'channels_first' mode, the channels dimension (the depth) is at index 1, in 'channels_last' mode is it at index 4. It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".
batch_size Fixed batch size for layer
name An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
trainable Whether the layer weights will be updated during training.
weights Initial weights for layer.

Input shape

5D tensor with shape: (samples, channels, dim1, dim2, dim3) if data_format='channels_first'
or 5D tensor with shape: (samples, dim1, dim2, dim3, channels) if data_format='channels_last'.

Output shape

Same as input

References

- Efficient Object Localization Using Convolutional Networks

See Also

Other dropout layers: layer_dropout, layer_spatial_dropout_1d, layer_spatial_dropout_2d
**layer_subtract**  
*Layer that subtracts two inputs.*

**Description**

It takes as input a list of tensors of size 2, both of the same shape, and returns a single tensor,  
(inputs[[1]] - inputs[[2]]), also of the same shape.

**Usage**

```python
layer_subtract(inputs, batch_size = NULL, dtype = NULL, name = NULL,  
trainable = NULL, weights = NULL)
```

**Arguments**

- **inputs**  
  A list of input tensors (exactly 2).

- **batch_size**  
  Fixed batch size for layer

- **dtype**  
  The data type expected by the input, as a string (float32, float64, int32...)

- **name**  
  An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.

- **trainable**  
  Whether the layer weights will be updated during training.

- **weights**  
  Initial weights for layer.

**Value**

A tensor, the difference of the inputs.

**See Also**

Other merge layers: layer_add, layer_average, layer_concatenate, layer_dot, layer_maximum,  
layer_minimum, layer_multiply

---

**layer_upsampling_1d**  
*Upsampling layer for 1D inputs.*

**Description**

Repeats each temporal step size times along the time axis.

**Usage**

```python
layer_upsampling_1d(object, size = 2L, batch_size = NULL,  
name = NULL, trainable = NULL, weights = NULL)
```
layer_upsampling_2d

Arguments

object  Model or layer object
size    integer. Upsampling factor.
batch_size  Fixed batch size for layer
name    An optional name string for the layer. Should be unique in a model (do not reuse
        the same name twice). It will be autogenerated if it isn’t provided.
trainable  Whether the layer weights will be updated during training.
weights  Initial weights for layer.

Input shape

3D tensor with shape: (batch, steps, features).

Output shape

3D tensor with shape: (batch, upsampled_steps, features).

See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose,
layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d,
layer_depthwise_conv_2d, layer_separable_conv_1d, layer_separable_conv_2d, layer_upsampling_2d,
layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

layer_upsampling_2d  Upsampling layer for 2D inputs.

Description

Repeats the rows and columns of the data by size[0] and size[1] respectively.

Usage

layer_upsampling_2d(object, size = c(2L, 2L), data_format = NULL,
interpolation = "nearest", batch_size = NULL, name = NULL,
trainable = NULL, weights = NULL)

Arguments

object  Model or layer object
size    int, or list of 2 integers. The upsampling factors for rows and columns.
data_format  A string, one of channels_last (default) or channels_first. The ordering
             of the dimensions in the inputs. channels_last corresponds to inputs with
             shape (batch, height, width, channels) while channels_first corresponds to inputs with shape
             (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at
             ~/.keras/keras.json. If you never set it, then it will be "channels_last".
layer_upsampling_3d

interpolation A string, one of nearest or bilinear. Note that CNTK does not support yet the bilinear upscaling and that with Theano, only size=(2, 2) is possible.

batch_size Fixed batch size for layer

name An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.

trainable Whether the layer weights will be updated during training.

weights Initial weights for layer.

Input shape

4D tensor with shape:

- If `data_format` is "channels_last": (batch, rows, cols, channels)
- If `data_format` is "channels_first": (batch, channels, rows, cols)

Output shape

4D tensor with shape:

- If `data_format` is "channels_last": (batch, upsampled_rows, upsampled_cols, channels)
- If `data_format` is "channels_first": (batch, channels, upsampled_rows, upsampled_cols)

See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_depthwise_conv_2d, layer_separable_conv_1d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

layer_upsampling_3d Upsampling layer for 3D inputs.

Description

Repeats the 1st, 2nd and 3rd dimensions of the data by `size[[0]], size[[1]]` and `size[[2]]` respectively.

Usage

```
layer_upsampling_3d(object, size = c(2L, 2L, 2L), data_format = NULL,
                   batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)
```
layer_zero_padding_1d

Arguments

- **object**: Model or layer object
- **size**: int, or list of 3 integers. The upsampling factors for dim1, dim2 and dim3.
- **data_format**: A string, one of channels_last (default) or channels_first. The ordering of the dimensions in the inputs. channels_last corresponds to inputs with shape (batch, spatial_dim1, spatial_dim2, spatial_dim3, channels) while channels_first corresponds to inputs with shape (batch, channels, spatial_dim1, spatial_dim2, spatial_dim3). It defaults to the image_data_format value found in your Keras config file at ~/.keras/keras.json. If you never set it, then it will be "channels_last".
- **batch_size**: Fixed batch size for layer
- **name**: An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- **trainable**: Whether the layer weights will be updated during training.
- **weights**: Initial weights for layer.

Input shape

5D tensor with shape:

- If data_format is "channels_last": (batch, dim1, dim2, dim3, channels)
- If data_format is "channels_first": (batch, channels, dim1, dim2, dim3)

Output shape

5D tensor with shape:

- If data_format is "channels_last": (batch, upsampled_dim1, upsampled_dim2, upsampled_dim3, channels)
- If data_format is "channels_first": (batch, channels, upsampled_dim1, upsampled_dim2, upsampled_dim3)

See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_depthwise_conv_2d, layer_separable_conv_1d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d

layer_zero_padding_1d  Zero-padding layer for 1D input (e.g. temporal sequence).

Description

Zero-padding layer for 1D input (e.g. temporal sequence).
Usage

layer_zero_padding_1d(object, padding = 1L, batch_size = NULL,
name = NULL, trainable = NULL, weights = NULL)

Arguments

object Model or layer object
padding int, or list of int (length 2)
  • If int: How many zeros to add at the beginning and end of the padding
    dimension (axis 1).
  • If list of int (length 2): How many zeros to add at the beginning and at the
    end of the padding dimension ((left_pad, right_pad)).
batch_size Fixed batch size for layer
name An optional name string for the layer. Should be unique in a model (do not reuse
the same name twice). It will be autogenerated if it isn’t provided.
trainable Whether the layer weights will be updated during training.
weights Initial weights for layer.

Input shape

3D tensor with shape (batch, axis_to_pad, features)

Output shape

3D tensor with shape (batch, padded_axis, features)

See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose,
layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d,
layer_depthwise_conv_2d, layer_separable_conv_1d, layer_separable_conv_2d, layer_separable_conv_3d, layer_upsampling_1d,
layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_2d, layer_zero_padding_3d

layer_zero_padding_2d Zero-padding layer for 2D input (e.g. picture).

Description

This layer can add rows and columns of zeros at the top, bottom, left and right side of an image

tensor.

Usage

layer_zero_padding_2d(object, padding = c(1L, 1L), data_format = NULL,
batch_size = NULL, name = NULL, trainable = NULL, weights = NULL)
Arguments

object  Model or layer object
padding  int, or list of 2 ints, or list of 2 lists of 2 ints.
  • If int: the same symmetric padding is applied to width and height.
  • If list of 2 ints: interpreted as two different symmetric padding values for
    height and width: (symmetric_height_pad, symmetric_width_pad).
  • If list of 2 lists of 2 ints: interpreted as ((top_pad, bottom_pad), (left_pad, right_pad))

data_format  A string, one of channels_last (default) or channels_first. The ordering
  of the dimensions in the inputs. channels_last corresponds to inputs with
  shape (batch, height, width, channels) while channels_first corresponds to inputs with shape
  (batch, channels, height, width). It defaults to the image_data_format value found in your Keras config file at
  ~/.keras/keras.json. If you never set it, then it will be "channels_last".

batch_size  Fixed batch size for layer

name  An optional name string for the layer. Should be unique in a model (do not reuse
  the same name twice). It will be autogenerated if it isn’t provided.

trainable  Whether the layer weights will be updated during training.

weights  Initial weights for layer.

Input shape

4D tensor with shape:

• If data_format is "channels_last": (batch, rows, cols, channels)
• If data_format is "channels_first": (batch, channels, rows, cols)

Output shape

4D tensor with shape:

• If data_format is "channels_last": (batch, padded_rows, padded_cols, channels)
• If data_format is "channels_first": (batch, channels, padded_rows, padded_cols)

See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d_transpose, layer_conv_2d, layer_conv_3d_transpose,
layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d,
layer_depthwise_conv_2d, layer_separable_conv_1d, layer_separable_conv_2d, layer_upsampling_1d,
layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d, layer_zero_padding_3d
layer_zero_padding_3d  Zero-padding layer for 3D data (spatial or spatio-temporal).

Description
Zero-padding layer for 3D data (spatial or spatio-temporal).

Usage
layer_zero_padding_3d(object, padding = c(1L, 1L, 1L),
                     data_format = NULL, batch_size = NULL, name = NULL,
                     trainable = NULL, weights = NULL)

Arguments
- object: Model or layer object
- padding: int, or list of 3 ints, or list of 3 lists of 2 ints.
  - If int: the same symmetric padding is applied to width and height.
  - If list of 3 ints: interpreted as three different symmetric padding values:
    (symmetric_dim1_pad, symmetric_dim2_pad, symmetric_dim3_pad).
  - If list of 3 lists of 2 ints: interpreted as ((left_dim1_pad, right_dim1_pad),
    (left_dim2_pad, right_dim2_pad), (left_dim3_pad, right_dim3_pad)).
- data_format: A string, one of channels_last (default) or channels_first. The ordering of
  the dimensions in the inputs. channels_last corresponds to inputs with shape
  (batch, spatial_dim1, spatial_dim2, spatial_dim3, channels) while
  channels_first corresponds to inputs with shape (batch, channels, spatial_dim1, spatial_dim2,
  spatial_dim3). It defaults to the image_data_format value found in your Keras config file at
  ~/.keras/keras.json. If you never set it, then it will be "channels_last".
- batch_size: Fixed batch size for layer
- name: An optional name string for the layer. Should be unique in a model (do not reuse
  the same name twice). It will be autogenerated if it isn’t provided.
- trainable: Whether the layer weights will be updated during training.
- weights: Initial weights for layer.

Input shape
5D tensor with shape:
- If data_format is "channels_last": (batch, first_axis_to_pad, second_axis_to_pad, third_axis_to_pad, depth)
- If data_format is "channels_first": (batch, depth, first_axis_to_pad, second_axis_to_pad, third_axis_to_pad)

Output shape
5D tensor with shape:
- If data_format is "channels_last": (batch, first_padded_axis, second_padded_axis, third_axis_to_pad, depth)
- If data_format is "channels_first": (batch, depth, first_padded_axis, second_padded_axis, third_axis_to_pad)
See Also

Other convolutional layers: layer_conv_1d, layer_conv_2d, layer_conv_3d, layer_conv_2d_transpose, layer_conv_3d_transpose, layer_conv_3d, layer_conv_lstm_2d, layer_cropping_1d, layer_cropping_2d, layer_cropping_3d, layer_depthwise_conv_2d, layer_separable_conv_1d, layer_separable_conv_2d, layer_upsampling_1d, layer_upsampling_2d, layer_upsampling_3d, layer_zero_padding_1d, layer_zero_padding_2d

---

```python
loss_mean_squared_error

Model loss functions

Description

Model loss functions

Usage

loss_mean_squared_error(y_true, y_pred)
loss_mean_absolute_error(y_true, y_pred)
loss_mean_absolute_percentage_error(y_true, y_pred)
loss_mean_squared_logarithmic_error(y_true, y_pred)
loss_squared_hinge(y_true, y_pred)
loss_hinge(y_true, y_pred)
loss_categorical_hinge(y_true, y_pred)
loss_logcosh(y_true, y_pred)
loss_categorical_crossentropy(y_true, y_pred)
loss_sparse_categorical_crossentropy(y_true, y_pred)
loss_binary_crossentropy(y_true, y_pred)
loss_kullback_leibler_divergence(y_true, y_pred)
loss_poisson(y_true, y_pred)
loss_cosine_proximity(y_true, y_pred)
```
Arguments

- **y_true**: True labels (Tensor)
- **y_pred**: Predictions (Tensor of the same shape as y_true)

Details

Loss functions are to be supplied in the `loss` parameter of the `compile.keras.engine.training.Model()` function.

Loss functions can be specified either using the name of a built in loss function (e.g. `loss = binary_crossentropy`), a reference to a built in loss function (e.g. `loss = loss_binary_crossentropy()`) or by passing an arbitrary function that returns a scalar for each data-point and takes the following two arguments:

- **y_true**: True labels (Tensor)
- **y_pred**: Predictions (Tensor of the same shape as y_true)

The actual optimized objective is the mean of the output array across all datapoints.

Categorical Crossentropy

When using the categorical_crossentropy loss, your targets should be in categorical format (e.g. if you have 10 classes, the target for each sample should be a 10-dimensional vector that is all-zeros except for a 1 at the index corresponding to the class of the sample). In order to convert integer targets into categorical targets, you can use the Keras utility function `to_categorical()`:

```r
categorical_labels <- to_categorical(int_labels, num_classes = NULL)
```

**loss_logcosh**

\[
\log(\cosh(x)) \approx \frac{(x^2)}{2} \quad \text{for small } x \\
\log(\cosh(x)) \approx \frac{|x|}{2} - \log(2) \quad \text{for large } x
\]

This means that `logcosh` works mostly like the mean squared error, but will not be so strongly affected by the occasional wildly incorrect prediction. However, it may return NaNs if the intermediate value \( \cosh(y_{pred} - y_{true}) \) is too large to be represented in the chosen precision.

See Also

`compile.keras.engine.training.Model()`

---

**make_sampling_table**

Generates a word rank-based probabilistic sampling table.

Description

Generates a word rank-based probabilistic sampling table.

Usage

```r
make_sampling_table(size, sampling_factor = 1e-05)
```
Argument

- **size**
  - Int, number of possible words to sample.

- **sampling_factor**
  - The sampling factor in the word2vec formula.

Details

Used for generating the `sampling_table` argument for `skipgrams()`. `sampling_table[[i]]` is the probability of sampling the word i-th most common word in a dataset (more common words should be sampled less frequently, for balance).

The sampling probabilities are generated according to the sampling distribution used in word2vec:

\[ p(\text{word}) = \min(1, \sqrt{\frac{\text{word\_frequency}}{\text{sampling\_factor}}} / \frac{\text{word\_frequency}}{\text{sampling\_factor}}) \]

We assume that the word frequencies follow Zipf's law (\(s=1\)) to derive a numerical approximation of frequency(rank):

\[ \text{frequency}(\text{rank}) \sim \frac{1}{(\text{rank} \times (\log(\text{rank}) + \gamma) + 1/2 - 1/(12*\text{rank}))} \]

where \(\gamma\) is the Euler-Mascheroni constant.

Value

An array of length `size` where the i-th entry is the probability that a word of rank i should be sampled.

Note

The word2vec formula is: \( p(\text{word}) = \min(1, \sqrt{\frac{\text{word\_frequency}}{\text{sampling\_factor}}} / \frac{\text{word\_frequency}}{\text{sampling\_factor}}) \)

See Also

Other text preprocessing: `pad_sequences`, `skipgrams`, `text_hashing_trick`, `text_one_hot`, `text_to_word_sequence`
Usage

    metric_binary_accuracy(y_true, y_pred)
    metric_binary_crossentropy(y_true, y_pred)
    metric_categorical_accuracy(y_true, y_pred)
    metric_categorical_crossentropy(y_true, y_pred)
    metric_cosine_proximity(y_true, y_pred)
    metric_hinge(y_true, y_pred)
    metric_kullback_leibler_divergence(y_true, y_pred)
    metric_mean_absolute_error(y_true, y_pred)
    metric_mean_absolute_percentage_error(y_true, y_pred)
    metric_mean_squared_error(y_true, y_pred)
    metric_mean_squared_logarithmic_error(y_true, y_pred)
    metric_poisson(y_true, y_pred)
    metric_sparse_categorical_crossentropy(y_true, y_pred)
    metric_squared_hinge(y_true, y_pred)
    metric_top_k_categorical_accuracy(y_true, y_pred, k = 5)
    metric_sparse_top_k_categorical_accuracy(y_true, y_pred, k = 5)
    custom_metric(name, metric_fn)

Arguments

    y_true     True labels (tensor)
    y_pred     Predictions (tensor of the same shape as y_true).
    k          An integer, number of top elements to consider.
    name       Name of custom metric
    metric_fn  Custom metric function

Custom Metrics

You can provide an arbitrary R function as a custom metric. Note that the y_true and y_pred parameters are tensors, so computations on them should use backend tensor functions.
Use the `custom_metric()` function to define a custom metric. Note that a name ('mean_pred') is provided for the custom metric function: this name is used within training progress output. See below for an example.

If you want to save and load a model with custom metrics, you should also specify the metric in the call the `load_model_hdf5()`. For example: `load_model_hdf5("my_model.h5", c('mean_pred' = metric_mean_pred))`

Alternatively, you can wrap all of your code in a call to `with_custom_object_scope()` which will allow you to refer to the metric by name just like you do with built-in Keras metrics.

Documentation on the available backend tensor functions can be found at [https://keras.rstudio.com/articles/backend.html#backend-functions](https://keras.rstudio.com/articles/backend.html#backend-functions).

### Metrics with Parameters

To use metrics with parameters (e.g. `metric_top_k_categorical_accuracy()`) you should create a custom metric that wraps the call with the parameter. See below for an example.

**Note**

Metric functions are to be supplied in the `metrics` parameter of the `compile.keras.engine.training.Model()` function.

**Examples**

```r
## Not run:

# create metric using backend tensor functions
metric_mean_pred <- custom_metric("mean_pred", function(y_true, y_pred) {
  k_mean(y_pred)
})

model %>% compile(
  optimizer = optimizer_rmsprop(),
  loss = loss_binary_crossentropy,
  metrics = c('accuracy', metric_mean_pred)
)

# create custom metric to wrap metric with parameter
metric_top_3_categorical_accuracy <-
  custom_metric("top_3_categorical_accuracy", function(y_true, y_pred) {
    metric_top_k_categorical_accuracy(y_true, y_pred, k = 3)
  })

model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = metric_top_3_categorical_accuracy
)

## End(Not run)
```
model_to_json

Model configuration as JSON

Description
Save and re-load models configurations as JSON. Note that the representation does not include the weights, only the architecture.

Usage
model_to_json(object)
model_from_json(json, custom_objects = NULL)

Arguments
- object: Model object to save
- json: JSON with model configuration
- custom_objects: Optional named list mapping names to custom classes or functions to be considered during deserialization.

See Also
Other model persistence: get_weights, model_to_yaml, save_model_hdf5, save_model_weights_hdf5, serialize_model

model_to_yaml

Model configuration as YAML

Description
Save and re-load models configurations as YAML. Note that the representation does not include the weights, only the architecture.

Usage
model_to_yaml(object)
model_from_yaml(yaml, custom_objects = NULL)

Arguments
- object: Model object to save
- yaml: YAML with model configuration
- custom_objects: Optional named list mapping names to custom classes or functions to be considered during deserialization.
multi_gpu_model

Replicates a model on different GPUs.

Description
Replicates a model on different GPUs.

Usage
multi_gpu_model(model, gpus = NULL, cpu_merge = TRUE,
cpu_relocation = FALSE)

Arguments
model  A Keras model instance. To avoid OOM errors, this model could have been built
       on CPU, for instance (see usage example below).
gpus   NULL to use all available GPUs (default). Integer >= 2 or list of integers, number
       of GPUs or list of GPU IDs on which to create model replicas.
cpu_merge A boolean value to identify whether to force merging model weights under the
           scope of the CPU or not.
cpu_relocation A boolean value to identify whether to create the model's weights under the
                scope of the CPU. If the model is not defined under any preceding device scope,
                you can still rescue it by activating this option.

Details
Specifically, this function implements single-machine multi-GPU data parallelism. It works in the
following way:

- Divide the model’s input(s) into multiple sub-batches.
- Apply a model copy on each sub-batch. Every model copy is executed on a dedicated GPU.
- Concatenate the results (on CPU) into one big batch.

E.g. if your batch_size is 64 and you use gpus=2, then we will divide the input into 2 sub-batches
of 32 samples, process each sub-batch on one GPU, then return the full batch of 64 processed
samples.

This induces quasi-linear speedup on up to 8 GPUs.

This function is only available with the TensorFlow backend for the time being.

See Also
Other model persistence: get_weights, model_to_json, save_model_hdf5, save_model_weights_hdf5,
serialize_model
**multi_gpu_model**

**Value**
A Keras model object which can be used just like the initial model argument, but which distributes its workload on multiple GPUs.

**Model Saving**
To save the multi-gpu model, use `save_model_hdf5()` or `save_model_weights_hdf5()` with the template model (the argument you passed to `multi_gpu_model`), rather than the model returned by `multi_gpu_model`.

**See Also**

**Examples**
```r
## Not run:

library(keras)
library(tensorflow)

num_samples <- 1000
height <- 224
width <- 224
num_classes <- 1000

# Instantiate the base model (or "template" model).
# We recommend doing this with under a CPU device scope,
# so that the model's weights are hosted on CPU memory.
# Otherwise they may end up hosted on a GPU, which would
# complicate weight sharing.
with(tf$device("/cpu:0"), {
  model <- application_xception(
    weights = NULL,
    input_shape = c(height, width, 3),
    classes = num_classes
  )
})

# Replicates the model on 8 GPUs.
# This assumes that your machine has 8 available GPUs.
parallel_model <- multi_gpu_model(model, gpus = 8)
parallel_model %>% compile(
  loss = "categorical_crossentropy",
  optimizer = "rmsprop"
)
```
Normalize a matrix or nd-array

Description

Normalize a matrix or nd-array

Usage

normalize(x, axis = -1, order = 2)

Arguments

x Matrix or array to normalize

axis Axis along which to normalize. Axis indexes are 1-based (pass -1 to select the last axis).

order Normalization order (e.g. 2 for L2 norm)

Value

A normalized copy of the array.
optimizer_adadelta  
*Adadelta optimizer.*

**Description**

Adadelta optimizer as described in *ADADELTA: An Adaptive Learning Rate Method.*

**Usage**

```python
optimizer_adadelta(lr = 1, rho = 0.95, epsilon = NULL, decay = 0,
clipnorm = NULL, clipvalue = NULL)
```

**Arguments**

- `lr`  
  float >= 0. Learning rate.
- `rho`  
  float >= 0. Decay factor.
- `epsilon`  
  float >= 0. Fuzz factor. If NULL, defaults to `k_epsilon()`. 
- `decay`  
  float >= 0. Learning rate decay over each update.
- `clipnorm`  
  Gradients will be clipped when their L2 norm exceeds this value.
- `clipvalue`  
  Gradients will be clipped when their absolute value exceeds this value.

**Note**

It is recommended to leave the parameters of this optimizer at their default values.

**See Also**

Other optimizers: `optimizer_adagrad`, `optimizer_adamax`, `optimizer_adam`, `optimizer_nadam`, `optimizer_rmsprop`, `optimizer_sgd`
Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lr</td>
<td>float &gt;= 0. Learning rate.</td>
</tr>
<tr>
<td>epsilon</td>
<td>float &gt;= 0. Fuzz factor. If NULL, defaults to k_epsilon().</td>
</tr>
<tr>
<td>decay</td>
<td>float &gt;= 0. Learning rate decay over each update.</td>
</tr>
<tr>
<td>clipnorm</td>
<td>Gradients will be clipped when their L2 norm exceeds this value.</td>
</tr>
<tr>
<td>clipvalue</td>
<td>Gradients will be clipped when their absolute value exceeds this value.</td>
</tr>
</tbody>
</table>

Note

It is recommended to leave the parameters of this optimizer at their default values.

See Also

Other optimizers: `optimizer_adadelta`, `optimizer_adamax`, `optimizer_adam`, `optimizer_nadam`, `optimizer_rmsprop`, `optimizer_sgd`

 optimizer_adam  Adam optimizer

Description

Adam optimizer as described in Adam - A Method for Stochastic Optimization.

Usage

optimizer_adam(lr = 0.001, beta_1 = 0.9, beta_2 = 0.999,
                epsilon = NULL, decay = 0, amsgrad = FALSE, clipnorm = NULL,
                clipvalue = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lr</td>
<td>float &gt;= 0. Learning rate.</td>
</tr>
<tr>
<td>beta_1</td>
<td>The exponential decay rate for the 1st moment estimates. float, 0 &lt; beta &lt; 1. Generally close to 1.</td>
</tr>
<tr>
<td>beta_2</td>
<td>The exponential decay rate for the 2nd moment estimates. float, 0 &lt; beta &lt; 1. Generally close to 1.</td>
</tr>
<tr>
<td>epsilon</td>
<td>float &gt;= 0. Fuzz factor. If NULL, defaults to k_epsilon().</td>
</tr>
<tr>
<td>decay</td>
<td>float &gt;= 0. Learning rate decay over each update.</td>
</tr>
<tr>
<td>amsgrad</td>
<td>Whether to apply the AMSGrad variant of this algorithm from the paper &quot;On the Convergence of Adam and Beyond&quot;.</td>
</tr>
<tr>
<td>clipnorm</td>
<td>Gradients will be clipped when their L2 norm exceeds this value.</td>
</tr>
<tr>
<td>clipvalue</td>
<td>Gradients will be clipped when their absolute value exceeds this value.</td>
</tr>
</tbody>
</table>
optimizer_adamax

References

- Adam - A Method for Stochastic Optimization
- On the Convergence of Adam and Beyond

Note

Default parameters follow those provided in the original paper.

See Also

Other optimizers: optimizer_adadelta, optimizer_adagrad, optimizer_adamax, optimizer_nadam, optimizer_rmsprop, optimizer_sgd

optimizer_adamax  Adamax optimizer

Description

Adamax optimizer from Section 7 of the Adam paper. It is a variant of Adam based on the infinity norm.

Usage

optimizer_adamax(lr = 0.002, beta_1 = 0.9, beta_2 = 0.999, epsilon = NULL, decay = 0, clipnorm = NULL, clipvalue = NULL)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>lr</td>
<td>float &gt;= 0. Learning rate.</td>
</tr>
<tr>
<td>beta_1</td>
<td>The exponential decay rate for the 1st moment estimates. float, 0 &lt; beta &lt; 1. Generally close to 1.</td>
</tr>
<tr>
<td>beta_2</td>
<td>The exponential decay rate for the 2nd moment estimates. float, 0 &lt; beta &lt; 1. Generally close to 1.</td>
</tr>
<tr>
<td>epsilon</td>
<td>float &gt;= 0. Fuzz factor. If NULL, defaults to k_epsilon().</td>
</tr>
<tr>
<td>decay</td>
<td>float &gt;= 0. Learning rate decay over each update.</td>
</tr>
<tr>
<td>clipnorm</td>
<td>Gradients will be clipped when their L2 norm exceeds this value.</td>
</tr>
<tr>
<td>clipvalue</td>
<td>Gradients will be clipped when their absolute value exceeds this value.</td>
</tr>
</tbody>
</table>

See Also

Other optimizers: optimizer_adadelta, optimizer_adagrad, optimizer_adam, optimizer_nadam, optimizer_rmsprop, optimizer_sgd
optimizer_nadam  

*Nesterov Adam optimizer*

**Description**

Much like Adam is essentially RMSprop with momentum, Nadam is Adam RMSprop with Nesterov momentum.

**Usage**

```python
optimizer_nadam(lr = 0.002, beta_1 = 0.9, beta_2 = 0.999,
epsilon = NULL, schedule_decay = 0.004, clipnorm = NULL,
clipvalue = NULL)
```

**Arguments**

- `lr`  
  float $\geq$ 0. Learning rate.

- `beta_1`  
  The exponential decay rate for the 1st moment estimates. float, $0 < \beta < 1$. Generally close to 1.

- `beta_2`  
  The exponential decay rate for the 2nd moment estimates. float, $0 < \beta < 1$. Generally close to 1.

- `epsilon`  
  float $\geq$ 0. Fuzz factor. If `NULL`, defaults to `k_epsilon()`.

- `schedule_decay`  
  Schedule decay.

- `clipnorm`  
  Gradients will be clipped when their L2 norm exceeds this value.

- `clipvalue`  
  Gradients will be clipped when their absolute value exceeds this value.

**Details**

Default parameters follow those provided in the paper. It is recommended to leave the parameters of this optimizer at their default values.

**See Also**

*On the importance of initialization and momentum in deep learning.*

Other optimizers: `optimizer_adadelta, optimizer_adagrad, optimizer_adamax, optimizer_adam, optimizer_rmsprop, optimizer_sgd`
optimizer_rmsprop

RMSProp optimizer

Description

RMSProp optimizer

Usage

```python
optimizer_rmsprop(lr = 0.001, rho = 0.9, epsilon = NULL, decay = 0, clipnorm = NULL, clipvalue = NULL)
```

Arguments

- `lr`  
  float >= 0. Learning rate.
- `rho`  
  float >= 0. Decay factor.
- `epsilon`  
  float >= 0. Fuzz factor. If NULL, defaults to k_epsilon().
- `decay`  
  float >= 0. Learning rate decay over each update.
- `clipnorm`  
  Gradients will be clipped when their L2 norm exceeds this value.
- `clipvalue`  
  Gradients will be clipped when their absolute value exceeds this value.

Note

It is recommended to leave the parameters of this optimizer at their default values (except the learning rate, which can be freely tuned).

This optimizer is usually a good choice for recurrent neural networks.

See Also

Other optimizers: optimizer_adadelta, optimizer_adagrad, optimizer_adamax, optimizer_adam, optimizer_nadam, optimizer_sgd

optimizer_sgd

Stochastic gradient descent optimizer

Description

Stochastic gradient descent optimizer with support for momentum, learning rate decay, and Nesterov momentum.

Usage

```python
optimizer_sgd(lr = 0.01, momentum = 0, decay = 0, nesterov = FALSE, clipnorm = NULL, clipvalue = NULL)
```
Arguments

- **lr**: float >= 0. Learning rate.
- **momentum**: float >= 0. Parameter that accelerates SGD in the relevant direction and dampens oscillations.
- **decay**: float >= 0. Learning rate decay over each update.
- **nesterov**: boolean. Whether to apply Nesterov momentum.
- **clipnorm**: Gradients will be clipped when their L2 norm exceeds this value.
- **clipvalue**: Gradients will be clipped when their absolute value exceeds this value.

Value

Optimizer for use with `compile.keras.engine.training.Model`.

See Also

Other optimizers: optimizer_adadelta, optimizer_adagrad, optimizer_adamax, optimizer_adam, optimizer_nadam, optimizer_rmsprop

---

**Description**

Pads sequences to the same length

**Usage**

```python
pad_sequences(sequences, maxlen = NULL, dtype = "int32",
             padding = "pre", truncating = "pre", value = 0)
```

**Arguments**

- **sequences**: List of lists where each element is a sequence
- **maxlen**: int, maximum length of all sequences
- **dtype**: type of the output sequences
- **padding**: 'pre' or 'post', pad either before or after each sequence.
- **truncating**: 'pre' or 'post', remove values from sequences larger than maxlen either in the beginning or in the end of the sequence
- **value**: float, padding value
Details

This function transforms a list of num_samples sequences (lists of integers) into a matrix of shape (num_samples, num_timesteps). num_timesteps is either the maxlen argument if provided, or the length of the longest sequence otherwise.

Sequences that are shorter than num_timesteps are padded with value at the end.

Sequences longer than num_timesteps are truncated so that they fit the desired length. The position where padding or truncation happens is determined by the arguments padding and truncating, respectively.

Pre-padding is the default.

Value

Matrix with dimensions (number_of_sequences, maxlen)

See Also

Other text preprocessing: make_sampling_table, skipgrams, text_hashing_trick, text_one_hot, text_to_word_sequence

Description

Plots metrics recorded during training.

Usage

## S3 method for class 'keras_training_history'
plot(x, y, metrics = NULL,
     method = c("auto", "ggplot2", "base"),
     smooth = getOption("keras.plot.history.smooth", TRUE),
     theme_bw = getOption("keras.plot.history.theme_bw", FALSE), ...)

Arguments

- **x**: Training history object returned from fit.keras.engine.training.Model().
- **y**: Unused.
- **metrics**: One or more metrics to plot (e.g. c('loss', 'accuracy')). Defaults to plotting all captured metrics.
- **method**: Method to use for plotting. The default "auto" will use ggplot2 if available, and otherwise will use base graphics.
smooth  Whether a loess smooth should be added to the plot, only available for the ggplot method. If the number of epochs is smaller than ten, it is forced to false.

theme_bw  Use ggplot2::theme_bw() to plot the history in black and white.

Additional parameters to pass to the plot method.

description

pop_layer  Remove the last layer in a model

description

Remove the last layer in a model

Usage

pop_layer(object)

Arguments

object  Keras model object

See Also


predict.keras.engine.training.Model

Description

Generate predictions from a Keras model

Usage

## S3 method for class 'keras.engine.training.Model'
predict(object, x,
        batch_size = NULL, verbose = 0, steps = NULL, ...)
predict_generator

Arguments

object  Keras model
x      Input data (vector, matrix, or array)
batch_size Integer. If unspecified, it will default to 32.
verbose Verbosity mode, 0 or 1.
steps   Total number of steps (batches of samples) before declaring the evaluation round finished. Ignored with the default value of NULL.

Value

vector, matrix, or array of predictions

See Also


predict_generator  Generates predictions for the input samples from a data generator.

Description

The generator should return the same kind of data as accepted by predict_on_batch().

Usage

predict_generator(object, generator, steps, max_queue_size = 10, workers = 1, verbose = 0)

Arguments

object  Keras model object
generator Generator yielding batches of input samples.
steps    Total number of steps (batches of samples) to yield from generator before stopping.
max_queue_size Maximum size for the generator queue. If unspecified, max_queue_size will default to 10.
workers Maximum number of threads to use for parallel processing. Note that parallel processing will only be performed for native Keras generators (e.g. flow_images_from_directory()) as R based generators must run on the main thread.
verbose Verbosity mode, 0 or 1.
predict_on_batch

Value

Numpy array(s) of predictions.

Raises

ValueError: In case the generator yields data in an invalid format.

See Also


predict_on_batch  Returns predictions for a single batch of samples.

Description

Returns predictions for a single batch of samples.

Usage

predict_on_batch(object, x)

Arguments

object  Keras model object
x  Input data (vector, matrix, or array)

Value

array of predictions.

See Also

predict_proba

Generates probability or class probability predictions for the input samples.

Description

Generates probability or class probability predictions for the input samples.

Usage

predict_proba(object, x, batch_size = NULL, verbose = 0, steps = NULL)
predict_classes(object, x, batch_size = NULL, verbose = 0, steps = NULL)

Arguments

- **object**: Keras model object
- **x**: Input data (vector, matrix, or array)
- **batch_size**: Integer. If unspecified, it will default to 32.
- **verbose**: Verbosity mode, 0 or 1.
- **steps**: Total number of steps (batches of samples) before declaring the evaluation round finished. The default NULL is equal to the number of samples in your dataset divided by the batch size.

Details

The input samples are processed batch by batch.

See Also

regularizer_l1  \hspace{1cm} L1 and L2 regularization

Description
L1 and L2 regularization

Usage
regularizer_l1(l = 0.01)
regularizer_l2(l = 0.01)
regularizer_l1_l2(l1 = 0.01, l2 = 0.01)

Arguments
- l  \hspace{1cm} Regularization factor.
- l1  \hspace{1cm} L1 regularization factor.
- l2  \hspace{1cm} L2 regularization factor.

reset_states  \hspace{1cm} Reset the states for a layer

Description
Reset the states for a layer

Usage
reset_states(object)

Arguments
- object  \hspace{1cm} Model or layer object

See Also
Other layer methods: count_params, get_config, get_input_at, get_weights
save_model_hdf5  

Save/Load models using HDF5 files

**Description**

Save/Load models using HDF5 files

**Usage**

```r
save_model_hdf5(object, filepath, overwrite = TRUE, 
                  include_optimizer = TRUE)

load_model_hdf5(filepath, custom_objects = NULL, compile = TRUE)
```

**Arguments**

- `object`: Model object to save
- `filepath`: File path
- `overwrite`: Overwrite existing file if necessary
- `include_optimizer`: If TRUE, save optimizer's state.
- `custom_objects`: Mapping class names (or function names) of custom (non-Keras) objects to class/functions (for example, custom metrics or custom loss functions).
- `compile`: Whether to compile the model after loading.

**Details**

The following components of the model are saved:

- The model architecture, allowing to re-instantiate the model.
- The model weights.
- The state of the optimizer, allowing to resume training exactly where you left off. This allows you to save the entirety of the state of a model in a single file.

Saved models can be re-instantiated via `load_model_hdf5()`. The model returned by `load_model_hdf5()` is a compiled model ready to be used (unless the saved model was never compiled in the first place or `compile = FALSE` is specified).

As an alternative to providing the `custom_objects` argument, you can execute the definition and persistence of your model using the `with_custom_object_scope()` function.

**Note**

The `serialize_model()` function enables saving Keras models to R objects that can be persisted across R sessions.
save_model_weights_hdf5

Save/Load model weights using HDF5 files

Description

Save/Load model weights using HDF5 files

Usage

save_model_weights_hdf5(object, filepath, overwrite = TRUE)

load_model_weights_hdf5(object, filepath, by_name = FALSE, skip_mismatch = FALSE, reshape = FALSE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>object</td>
<td>Model object to save/load</td>
</tr>
<tr>
<td>filepath</td>
<td>Path to the file</td>
</tr>
<tr>
<td>overwrite</td>
<td>Whether to silently overwrite any existing file at the target location</td>
</tr>
<tr>
<td>by_name</td>
<td>Whether to load weights by name or by topological order.</td>
</tr>
<tr>
<td>skip_mismatch</td>
<td>Logical, whether to skip loading of layers where there is a mismatch in the number of weights, or a mismatch in the shape of the weight (only valid when by_name = FALSE).</td>
</tr>
<tr>
<td>reshape</td>
<td>Reshape weights to fit the layer when the correct number of values are present but the shape does not match.</td>
</tr>
</tbody>
</table>

Details

The weight file has:

- `layer_names` (attribute), a list of strings (ordered names of model layers).
- For every layer, a group named `layer.name`
- For every such layer group, a group attribute `weight_names`, a list of strings (ordered names of weights tensor of the layer).
- For every weight in the layer, a dataset storing the weight value, named after the weight tensor.

For `load_model_weights()`, if `by_name` is `FALSE` (default) weights are loaded based on the network’s topology, meaning the architecture should be the same as when the weights were saved. Note that layers that don’t have weights are not taken into account in the topological ordering, so adding or removing layers is fine as long as they don’t have weights.

If `by_name` is `TRUE`, weights are loaded into layers only if they share the same name. This is useful for fine-tuning or transfer-learning models where some of the layers have changed.

See Also

Other model persistence: `get_weights`, `model_to_json`, `model_to_yaml`, `save_model_weights_hdf5`, `serialize_model`
save_text_tokenizer

See Also

Other model persistence: get_weights, model_to_json, model_to_yaml, save_model_hdf5, serialize_model

save_text_tokenizer  Save a text tokenizer to an external file

Description

Enables persistence of text tokenizers alongside saved models.

Usage

save_text_tokenizer(object, filename)

load_text_tokenizer(filename)

Arguments

object  Text tokenizer fit with fit_text_tokenizer()
filename  File to save/load

Details

You should always use the same text tokenizer for training and prediction. In many cases however prediction will occur in another session with a version of the model loaded via load_model_hdf5(). In this case you need to save the text tokenizer object after training and then reload it prior to prediction.

See Also

Other text tokenization: fit_text_tokenizer, sequences_to_matrix, text_tokenizer, texts_to_matrix, texts_to_sequences_generator, texts_to_sequences

Examples

## Not run:

# vectorize texts then save for use in prediction
tokenizer <- text_tokenizer(num_words = 10000)  ##
fit_text_tokenizer(tokenizer, texts)
save_text_tokenizer(tokenizer, "tokenizer")

# (train model, etc.)

# ...later in another session
tokenizer <- load_text_tokenizer("tokenizer")
```
# (use tokenizer to preprocess data for prediction)

## End(Not run)

sequences_to_matrix  

**Convert a list of sequences into a matrix.**

**Description**

Convert a list of sequences into a matrix.

**Usage**

`sequences_to_matrix(tokenizer, sequences, mode = c("binary", "count", "tfidf", "freq"))`

**Arguments**

- `tokenizer`: Tokenizer
- `sequences`: List of sequences (a sequence is a list of integer word indices).
- `mode`: one of "binary", "count", "tfidf", "freq".

**Value**

A matrix

**See Also**

Other text tokenization: `fit_text_tokenizer`, `save_text_tokenizer`, `text_tokenizer`, `texts_to_matrix`, `texts_to_sequences_generator`, `texts_to_sequences`

```
serialize_model  

**Serialize a model to an R object**

**Description**

Model objects are external references to Keras objects which cannot be saved and restored across R sessions. The `serialize_model()` and `unserialize_model()` functions provide facilities to convert Keras models to R objects for persistence within R data files.

**Usage**

`serialize_model(model, include_optimizer = TRUE)`

`unserialize_model(model, custom_objects = NULL, compile = TRUE)`
skipgrams  

Generates skipgram word pairs.

Description

Generates skipgram word pairs.

Usage

skipgrams(sequence, vocabulary_size, window_size = 4, negative_samples = 1, shuffle = TRUE, categorical = FALSE, sampling_table = NULL, seed = NULL)

Arguments

sequence  
A word sequence (sentence), encoded as a list of word indices (integers). If using a sampling_table, word indices are expected to match the rank of the words in a reference dataset (e.g. 10 would encode the 10-th most frequently occurring token). Note that index 0 is expected to be a non-word and will be skipped.

vocabulary_size  
Int, maximum possible word index + 1

window_size  
Int, size of sampling windows (technically half-window). The window of a word \( w_i \) will be \([i-window_size, i+window_size+1]\)
negative_samples
float >= 0. 0 for no negative (i.e. random) samples. 1 for same number as positive samples.

shuffle
whether to shuffle the word couples before returning them.

categorical
bool. if FALSE, labels will be integers (e.g. [0, 1, 1 . . .]), if TRUE labels will be categorical eg. [[1,0],[0,1],[0,1] . . .]

sampling_table
1D array of size vocabulary_size where the entry i encodes the probability to sample a word of rank i.

seed
Random seed

Details
This function transforms a list of word indexes (lists of integers) into lists of words of the form:

- (word, word in the same window), with label 1 (positive samples).
- (word, random word from the vocabulary), with label 0 (negative samples).

Read more about Skipgram in this gnomic paper by Mikolov et al.: Efficient Estimation of Word Representations in Vector Space

Value
List of couples, labels where:

- couples is a list of 2-element integer vectors: [word_index, other_word_index].
- labels is an integer vector of 0 and 1, where 1 indicates that other_word_index was found in the same window as word_index, and 0 indicates that other_word_index was random.
- if categorical is set to TRUE, the labels are categorical, i.e. 1 becomes [0, 1], and 0 becomes [1, 0].

See Also
Other text preprocessing: make_sampling_table, pad_sequences, text_hashing_trick, text_one_hot, text_to_word_sequence

summary.keras.engine.training.Model
Print a summary of a Keras model

Description
Print a summary of a Keras model

Usage
```r
## S3 method for class 'keras.engine.training.Model'
summary(object, 
  line_length = getOption("width"), positions = NULL, ...)
```
texts_to_matrix

Convert a list of texts to a matrix.

Description
Convert a list of texts to a matrix.

Usage

texts_to_matrix(tokenizer, texts, mode = c("binary", "count", "tfidf", "freq"))

Arguments

tokenizer Tokenizer
texts Vector/list of texts (strings).
mode one of "binary", "count", "tfidf", "freq".

Value
A matrix

See Also
Other text tokenization: fit_text_tokenizer, save_text_tokenizer, sequences_to_matrix, text_tokenizer, texts_to_sequences_generator, texts_to_sequences
texts_to_sequences  

*Transform each text in texts in a sequence of integers.*

**Description**

Only top "num_words" most frequent words will be taken into account. Only words known by the tokenizer will be taken into account.

**Usage**

`texts_to_sequences(tokenizer, texts)`

**Arguments**

- `tokenizer`: Tokenizer
- `texts`: Vector/list of texts (strings).

**See Also**

Other text tokenization: `fit_text_tokenizer`, `save_text_tokenizer`, `sequences_to_matrix`, `text_tokenizer`, `texts_to_matrix`, `texts_to_sequences_generator`

texts_to_sequences_generator

*Transforms each text in texts in a sequence of integers.*

**Description**

Only top "num_words" most frequent words will be taken into account. Only words known by the tokenizer will be taken into account.

**Usage**

`texts_to_sequences_generator(tokenizer, texts)`

**Arguments**

- `tokenizer`: Tokenizer
- `texts`: Vector/list of texts (strings).

**Value**

Generator which yields individual sequences

**See Also**

Other text tokenization: `fit_text_tokenizer`, `save_text_tokenizer`, `sequences_to_matrix`, `text_tokenizer`, `texts_to_matrix`, `texts_to_sequences`
text_hashing_trick Converts a text to a sequence of indexes in a fixed-size hashing space.

Description

Converts a text to a sequence of indexes in a fixed-size hashing space.

Usage

```
text_hashing_trick(text, n, hash_function = NULL, filters = "!"#$%&()*+,-./:;<=?@[\]^_`{|}~\t\n", lower = TRUE, split = " ")
```

Arguments

- **text**: Input text (string).
- **n**: Dimension of the hashing space.
- **hash_function**: if NULL uses python hash function, can be 'md5' or any function that takes in input a string and returns a int. Note that hash is not a stable hashing function, so it is not consistent across different runs, while 'md5' is a stable hashing function.
- **filters**: Sequence of characters to filter out such as punctuation. Default includes basic punctuation, tabs, and newlines.
- **lower**: Whether to convert the input to lowercase.
- **split**: Sentence split marker (string).

Details

Two or more words may be assigned to the same index, due to possible collisions by the hashing function.

Value

A list of integer word indices (unicity non-guaranteed).

See Also

Other text preprocessing: make_sampling_table, pad_sequences, skipgrams, text_one_hot, text_to_word_sequence
text_one_hot  

One-hot encode a text into a list of word indexes in a vocabulary of size n.

**Description**

One-hot encode a text into a list of word indexes in a vocabulary of size n.

**Usage**

```python
text_one_hot(text, n,  
              filters = "!"#$%&()%++-./:;<=>?\[\]\^_`{|}~\t\n",  
              lower = TRUE, split = " ")
```

**Arguments**

- **text**: Input text (string).
- **n**: Size of vocabulary (integer)
- **filters**: Sequence of characters to filter out such as punctuation. Default includes basic punctuation, tabs, and newlines.
- **lower**: Whether to convert the input to lowercase.
- **split**: Sentence split marker (string).

**Value**

List of integers in [1, n]. Each integer encodes a word (unicity non-guaranteed).

**See Also**

Other text preprocessing: *make_sampling_table, pad_sequences, skipgrams, text_hashing_trick, text_to_word_sequence*

text_tokenizer  

*Text tokenization utility*

**Description**

Vectorize a text corpus, by turning each text into either a sequence of integers (each integer being the index of a token in a dictionary) or into a vector where the coefficient for each token could be binary, based on word count, based on tf-idf...

**Usage**

```python
text_tokenizer(num_words = NULL,  
               filters = "!"#$%&()%++-./:;<=>?\[\]\^_`{|}~\t\n",  
               lower = TRUE, split = " ", char_level = FALSE, oov_token = NULL)
```
text_to_word_sequence

Arguments

num_words the maximum number of words to keep, based on word frequency. Only the most common num_words words will be kept.

filters a string where each element is a character that will be filtered from the texts. The default is all punctuation, plus tabs and line breaks, minus the ' character.

lower boolean. Whether to convert the texts to lowercase.

split character or string to use for token splitting.

char_level if TRUE, every character will be treated as a token

oov_token NULL or string If given, it will be added to ‘word_index” and used to replace out-of-vocabulary words during text_to_sequence calls.

Details

By default, all punctuation is removed, turning the texts into space-separated sequences of words (words maybe include the ’ character). These sequences are then split into lists of tokens. They will then be indexed or vectorized. 0 is a reserved index that won’t be assigned to any word.

Attributes

The tokenizer object has the following attributes:

- word_counts — named list mapping words to the number of times they appeared on during fit. Only set after fit_text_tokenizer() is called on the tokenizer.

- word_docs — named list mapping words to the number of documents/texts they appeared on during fit. Only set after fit_text_tokenizer() is called on the tokenizer.

- word_index — named list mapping words to their rank/index (int). Only set after fit_text_tokenizer() is called on the tokenizer.

- document_count — int. Number of documents (texts/sequences) the tokenizer was trained on. Only set after fit_text_tokenizer() is called on the tokenizer.

See Also

Other text tokenization: fit_text_tokenizer, save_text_tokenizer, sequences_to_matrix, texts_to_matrix, texts_to_sequences_generator, texts_to_sequences

---

text_to_word_sequence  Convert text to a sequence of words (or tokens).

Description

Convert text to a sequence of words (or tokens).
Usage

text_to_word_sequence(text,
    filters = "!"#$%&()++.,-;:<=>?@[\]^_\`\{}~\n",
    lower = TRUE, split = " ")

Arguments

text  Input text (string).
filters  Sequence of characters to filter out such as punctuation. Default includes basic punctuation, tabs, and newlines.
lower  Whether to convert the input to lowercase.
split  Sentence split marker (string).

Value

Words (or tokens)

See Also

Other text preprocessing: make_sampling_table, pad_sequences, skipgrams, text_hashing_trick, text_one_hot

timeseries_generator  Utility function for generating batches of temporal data.

description

Utility function for generating batches of temporal data.

Usage

timeseries_generator(data, targets, length, sampling_rate = 1,
    stride = 1, start_index = 0, end_index = NULL, shuffle = FALSE,
    reverse = FALSE, batch_size = 128)

Arguments

data  Object containing consecutive data points (timesteps). The data should be 2D, and axis 1 is expected to be the time dimension.
targets  Targets corresponding to timesteps in data. It should have same length as data.
length  Length of the output sequences (in number of timesteps).
sampling_rate  Period between successive individual timesteps within sequences. For rate r, timesteps data[i], data[i-r], ... data[i - length] are used for create a sample sequence.
time_distributed

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stride</td>
<td>Period between successive output sequences. For stride s, consecutive output samples would be centered around data[i], data[i+s], data[i+2*s], etc.</td>
</tr>
<tr>
<td>start_index, end_index</td>
<td>Data points earlier than start_index or later than end_index will not be used in the output sequences. This is useful to reserve part of the data for test or validation.</td>
</tr>
<tr>
<td>shuffle</td>
<td>Whether to shuffle output samples, or instead draw them in chronological order.</td>
</tr>
<tr>
<td>reverse</td>
<td>Boolean: if true, timesteps in each output sample will be in reverse chronological order.</td>
</tr>
<tr>
<td>batch_size</td>
<td>Number of timeseries samples in each batch (except maybe the last one).</td>
</tr>
</tbody>
</table>

**Value**

An object that can be passed to generator based training functions (e.g. `fit_generator()`).

---

**Description**

The input should be at least 3D, and the dimension of index one will be considered to be the temporal dimension.

**Usage**

```python
time_distributed(object, layer, input_shape = NULL,
   batch_input_shape = NULL, batch_size = NULL, dtype = NULL,
   name = NULL, trainable = NULL, weights = NULL)
```

**Arguments**

- **object**
  - Model or layer object
- **layer**
  - A layer instance.
- **input_shape**
  - Dimensionality of the input (integer) not including the samples axis. This argument is required when using this layer as the first layer in a model.
- **batch_input_shape**
  - Shapes, including the batch size. For instance, `batch_input_shape=c(10, 32)` indicates that the expected input will be batches of 10 32-dimensional vectors. `batch_input_shape=list(NULL, 32)` indicates batches of an arbitrary number of 32-dimensional vectors.
- **batch_size**
  - Fixed batch size for layer.
- **dtype**
  - The data type expected by the input, as a string (`float32`, `float64`, `int32`...)
- **name**
  - An optional name string for the layer. Should be unique in a model (do not reuse the same name twice). It will be autogenerated if it isn’t provided.
- **trainable**
  - Whether the layer weights will be updated during training.
- **weights**
  - Initial weights for layer.
Details

Consider a batch of 32 samples, where each sample is a sequence of 10 vectors of 16 dimensions. The batch input shape of the layer is then $(32, 10, 16)$, and the input shape, not including the samples dimension, is $(10, 16)$. You can then use `time_distributed` to apply a `layer_dense` to each of the 10 timesteps, independently.

See Also

Other layer wrappers: `bidirectional`

---

**to_categorical**

Converts a class vector (integers) to binary class matrix.

### Description

Converts a class vector (integers) to binary class matrix.

### Usage

```python
    to_categorical(y, num_classes = NULL, dtype = "float32")
```

### Arguments

- **y**: Class vector to be converted into a matrix (integers from 0 to num_classes).
- **num_classes**: Total number of classes.
- **dtype**: The data type expected by the input, as a string

### Details

E.g. for use with `loss_categorical_crossentropy`.

### Value

A binary matrix representation of the input.
**train_on_batch**  
*Single gradient update or model evaluation over one batch of samples.*

**Description**

Single gradient update or model evaluation over one batch of samples.

**Usage**

```
train_on_batch(object, x, y, class_weight = NULL, sample_weight = NULL)

test_on_batch(object, x, y, sample_weight = NULL)
```

**Arguments**

- **object**  
  Keras model object
- **x**  
  input data, as an array or list of arrays (if the model has multiple inputs).
- **y**  
  labels, as an array.
- **class_weight**  
  named list mapping classes to a weight value, used for scaling the loss function (during training only).
- **sample_weight**  
  sample weights, as an array.

**Value**

Scalar training or test loss (if the model has no metrics) or list of scalars (if the model computes other metrics). The property `model$metrics_names` will give you the display labels for the scalar outputs.

**See Also**


**use_implementation**  
*Select a Keras implementation and backend*

**Description**

Select a Keras implementation and backend
with_custom_object_scope

Description

Provide a scope with mappings of names to custom objects

Usage

with_custom_object_scope(objects, expr)

Usage

use_implementation(implementation = c("keras", "tensorflow"))

use_backend(backend = c("tensorflow", "cntk", "theano", "plaidml"))

Arguments

implementation One of "keras" or "tensorflow" (defaults to "keras").
backend One of "tensorflow", "cntk", or "theano" (defaults to "tensorflow")

Details

Keras has multiple implementations (the original keras implementation and the implementation native to TensorFlow) and supports multiple backends ("tensorflow", "cntk", "theano", and "plaidml"). These functions allow switching between the various implementations and backends.

The functions should be called after library(keras) and before calling other functions within the package (see below for an example).

The default implementation and backend should be suitable for most use cases. The "tensorflow" implementation is useful when using Keras in conjunction with TensorFlow Estimators (the tfestimators R package).

Examples

## Not run:
# use the tensorflow implementation
library(keras)
use_implementation("tensorflow")

# use the cntk backend
library(keras)
use_backend("theano")

## End(Not run)
with_custom_object_scope

Arguments

- **objects**: Named list of objects
- **expr**: Expression to evaluate

Details

There are many elements of Keras models that can be customized with user objects (e.g. losses, metrics, regularizers, etc.). When loading saved models that use these functions you typically need to explicitly map names to user objects via the `custom_objects` parameter.

The `with_custom_object_scope()` function provides an alternative that lets you create a named alias for a user object that applies to an entire block of code, and is automatically recognized when loading saved models.

Examples

```r
## Not run:
# define custom metric
custom_metric <- function(y_true, y_pred) {
  metric_top_k_categorical_accuracy(y_true, y_pred, k = 3)
}

with_custom_object_scope(c(top_k_acc = sparse_top_k_cat_acc), {
  # ...define model...

  # compile model (refer to top_k_acc by name)
  model <-
    loss = "binary_crossentropy",
    optimizer = optimizer_nadam(),
    metrics = c("top_k_acc")
})

# save the model
save_model_hdf5("my_model.h5")

# loading the model within the custom object scope doesn't
# require explicitly providing the custom_object
load_model_hdf5("my_model.h5")
})

## End(Not run)
```
Index

*Topic datasets
  KerasCallback, 78
  KerasConstraint, 80
  KerasLayer, 81
  KerasWrapper, 82
  application_densenet, 11
  application_densenet1R1, 11
  application_densenet1V9, 11
  application_densenetRP1, 11
  application_inception_resnet_vR, 12
  application_inception_vS, 13
  application_mobilenet, 15
  application_mobilenet_vR, 16
  application_nasnet, 18
  application_nasnetlarge, 18
  application_nasnetmobile, 18
  application_resnet50, 19
  application_vgg, 21
  application_vgg16 (application_vgg), 21
  application_vgg19 (application_vgg), 21
  application_vgg19, 21
  backend, 23
  backend(), 35
  bidirectional, 24, 306
  callback_csv_logger, 25, 26–32
  callback_early_stopping, 25, 25, 27–32
  callback_lambda, 25, 26, 27–32
  callback_learning_rate_scheduler, 25–27, 27, 28–32
  callback_model_checkpoint, 25–27, 28, 29–32
  callbackProgbar_logger, 25–28, 29, 30–32
  callback_reduce_lr_on_plateau, 25–29, 29, 31, 32
  callback_remote_monitor, 25–30, 30, 32
  callback_tensorboard, 25–31, 31, 32
  callback_terminate_on_naan, 25–32, 32
  clone_model, 33
  compile(), 43, 46–48
  compile.keras.engine.training.Model(), 33, 44, 47, 49, 58, 60, 83, 85, 279, 286, 288–291, 299, 307
  compile.keras.engine.training.Model(), 273, 276
  constraint_maxnorm (constraints), 34
  constraint_minmaxnorm (constraints), 34
  constraint_nonneg (constraints), 34
  constraint_uniunitnorm (constraints), 34
  constraints, 34, 80
  count_params, 36, 58, 60, 61, 292
  create_layer, 36
  create_wrapper, 37
  custom_metric (metric_binary_accuracy), 274
  dataset_boston_housing, 37, 38–43
INDEX

dataset_fashion_mnist, 38, 39, 41–43
dataset_cifar10, 38, 39, 40–43
dataset_cifar100, 38, 39, 40–43
dataset_fashion_mnist, 38, 39, 41–43
dataset_imdb, 38–40, 42, 43
dataset_imdb(), 42, 43
dataset_imdb_word_index (dataset_imdb), 40
dataset_mnist, 38–41, 41, 43
dataset_reuters, 38–41, 41, 43
dataset_reuters_word_index (dataset_reuters), 42
densenet_preprocess_input
(application_densenet), 11

evaluate.keras.engine.training.Model, 34, 43, 44, 47, 49, 58, 60, 83, 85, 279, 288–292, 299, 307
evaluate_generator, 34, 44, 47, 49, 58, 60, 83, 85, 279, 288–291, 299, 307
evaluate_generator(), 55
export_savedmodel.keras.engine.training.Model

fit.keras.engine.training.Model, 34, 44, 45, 49, 58, 60, 83, 85, 279, 288–291, 299, 307
fit_generator, 34, 44, 47, 49, 58, 60, 83, 85, 279, 288–291, 299, 307
fit_generator(), 55
fit_image_data_generator, 49, 52, 54, 55, 65, 66
fit_text_tokenizer, 50, 295, 296, 299, 300, 303
fit_text_tokenizer(), 55
flow_images_from_data, 50, 51, 54, 55, 65, 66
flow_images_from_dataframe, 50, 52, 52, 55, 65, 66
flow_images_from_directory, 50, 52, 54, 55, 65, 66
flow_images_from_directory(), 48
freeze_weights, 56
from_config (get_config), 57
generator_next, 57
generator, 34, 36, 44, 47, 49, 57, 60, 61, 83, 85, 279, 288–292, 299, 307
get_file, 58
get_input_at, 36, 58, 59, 61, 292

generator_next, 57
get_config, 34, 36, 44, 47, 49, 57, 60, 61, 83, 85, 279, 288–292, 299, 307
get_file, 58
get_input_at, 36, 58, 59, 61, 292

generate_mask_at (get_input_at), 59
generate_shape_at (get_input_at), 59
get_layer, 34, 44, 47, 49, 58, 60, 83, 85, 279, 288–291, 299, 307
generate_output_at (get_input_at), 59
generate_output_mask_at (get_input_at), 59
generate_output_shape_at (get_input_at), 59
get_weights, 36, 58, 60, 61, 277, 278, 292, 294, 295, 297

hdf5_matrix, 61

image_array_resize (image_to_array), 65
image_array_save (image_to_array), 65
image_data_generator, 63
image_data_generator(), 49, 51, 53, 55, 57
image_load, 50, 52, 54, 55, 65, 66
image_to_array, 50, 52, 54, 55, 65, 65
imagenet_decode_predictions, 62
imagenet_preprocess_input, 62

inception_preprocess_input, 62

inception_v3_preprocess_input
(application_inception_v3), 13
initializer_constant, 67, 68–75
initializer_glorot_normal, 67, 68–75
initializer_glorot_uniform, 67, 68, 68, 69–75
initializer_he_normal, 67, 68, 69, 70–75
initializer_he_uniform, 67–69, 69, 70–75
initialzer_identity, 67–70, 70–71, 75, 186
initializer_lecun_normal, 67–70, 70, 71–75, 186
initializer_lecun_uniform, 67–71, 71, 72–75
initializer_ones, 67–72, 72, 73–75
initializer_orthogonal, 67–72, 72, 73–75
initializer_random_normal, 67–72, 73, 74, 75
initializer_random_normal(), 74
initializer_random_uniform, 67–73, 73, 74, 75
initializer_truncated_normal, 67–74, 74, 75
initializer_variance_scaling, 67–74, 74, 75
initializer_zeros, 67–75, 75
install_keras, 76
is_keras_available, 78
k_abs, 85
k_all, 86
k_any, 87
k_arange, 87
k_argmax, 88
k_argmin, 89
k_backend, 89
k_batch_dot, 90
k_batch_flatten, 91
k_batch_get_value, 91
k_batch_get_value(), 93
k_batch_normalization, 92
k_batch_set_value, 93
k_batch_set_value(), 92
k_bias_add, 93
k_binary_crossentropy, 94
k_cast, 95
k_cast_to_floatx, 95
k_categorical_crossentropy, 96
k_clear_session, 97
k_clip, 97
k_concatenate, 98
k_constant, 98
k_convd, 99
k_conv2d, 100
k_conv2d_transpose, 100
k_conv3d, 101
k_conv3d_transpose, 102
k_cos, 103
k_count_params, 103
k_ctc_batch_cost, 104
k_ctc_decode, 105
k_ctc_label_dense_to_sparse, 106
k_cumprod, 106
k_cumsum, 107
k_depthwise_conv2d, 108
k_dot, 108
k_dropout, 109
k_dtype, 110
k_ein, 110
k_epsilon, 111
k_equal, 111
k_eval, 112
k_exp, 113
k_expand_dims, 113
k_eye, 114
k_flatten, 115
k_floatx, 115
k_foldl, 116
kFoldr, 117
k_function, 117
k_gather, 118
k_get_session, 119
k_get_uid, 119
k_get_value, 120
k_get_variable_shape, 121
k_gradients, 121
k_greater, 122
k_greater_equal, 122
k_greater_equal(), 35
k_hard_sigmoid, 123
k_identity, 124
k_image_data_format, 124
k_in_test_phase, 125
k_in_top_k, 126
k_in_train_phase, 127
k_int_shape, 125
k_is_keras_tensor, 127
k_is_placeholder, 128
k_is_sparse, 128
k_is_tensor, 129
k_l2_normalize, 130
k_learning_phase, 130
k_less, 131
k_less_equal, 131
k_local_conv1d, 132
k_local_conv2d, 133
k_log, 134
k_logsumexp, 134
k_manual_variable_initialization, 135
k_map_fn, 136
k_max, 136
k_maximum, 137
k_mean, 138
k_min, 138
k_minimum, 139
k_moving_average_update, 140
k_ndim, 140
k_normalize_batch_in_training, 141
k_not_equal, 142
k_one_hot, 144
k_ones, 142
k_ones_like, 143
k_permute_dimensions, 144
INDEX

k_placeholder, 145
k_pool2d, 146
k_pool3d, 146
k.pow, 147
k_print_tensor, 148
k_prod, 148
k_random_binomial, 149
k_random_normal, 150
k_random_normal_variable, 150
k_random_uniform, 151
k_random_uniform_variable, 152
k_relu, 153
k_repeat, 153
k_repeat_elements, 154
k_reset_uids, 155
k_reshape, 155
k_resize_images, 156
k_resize_volumes, 156
k_reverse, 157
k_rnn, 158
k_round, 159
k_separable_conv2d, 159
k_set_epsilon (k_epsilon), 111
k_set_floatx (k_floatx), 115
k_set_image_data_format (k_image_data_format), 124
k_set_learning_phase, 160
k_set_session (k_get_session), 119
k_set_value, 161
k_shape, 161
k_sigmoid, 162
k_sign, 162
k_sin, 163
k_softmax, 164
k_softplus, 164
k_softsign, 165
k_sparse_categorical_crossentropy, 165
k_spatial_2d_padding, 166
k_spatial_3d_padding, 167
k_sqr, 167
k_square, 168
k_squeeze, 169
k_stack, 169
k_std, 170
k_stop_gradient, 171
k_sum, 171
k_switch, 172
k_tanh, 173
k_temporal_padding, 173
k_tile, 174
k_to_dense, 175
k_transpose, 175
k_truncated_normal, 176
k_update, 177
k_update_add, 177
k_update_sub, 178
k_var, 178
k_variable, 179
k_zeros, 180
k_zeros_like, 180
keras (keras-package), 9
keras-package, 9
keras_array, 82
keras_model, 34, 44, 47, 49, 58, 60, 83, 85, 279, 288–291, 299, 307
keras_model_custom, 84
keras_model_sequential, 34, 44, 47, 49, 58, 60, 83, 84, 279, 288–291, 299, 307
keras_model_sequential(), 36, 37
KerasCallback, 78, 79
KerasConstraint, 35, 80
KerasLayer, 81, 81
KerasWrapper, 82, 82
layer_activation, 181, 182, 183, 185–189, 220, 223, 225, 237, 239, 246, 252–254
layer_activation(), 10
layer_activationelu, 182, 182, 183, 185–188
layer_activation_leaky_relu, 182, 183, 185–188
layer_activation_parametric_relu, 182, 183, 184, 185–188
layer_activation_relu, 182, 182, 183, 185, 186–188
layer_activation_selu, 182, 182, 183, 185, 186–188
layer_activity_regularization, 182, 182, 183, 185, 186, 187, 188
layer_activation_softmax, 182, 182, 183, 185, 186, 187, 188
layer_activation_thresholded_relu, 182, 183, 185–187, 188
layer_activity_regularizer, 182, 182, 183, 185, 186, 187, 188
layer_activity_regularizer, 182, 189, 220, 223, 225, 237, 239, 246, 252–254
layer_add, 190, 192, 198, 222, 247, 250, 251, 265
INDEX

layer_repeat_vector, 182, 189, 220, 223, 225, 237, 239, 246, 252, 252, 254
layer_reshape, 182, 189, 220, 223, 225, 237, 239, 246, 252, 253, 253
layer_separable_conv_1d, 200, 202, 205, 207, 209, 212, 214, 215, 222, 254, 259, 266–270, 272
layer_separable_conv_2d, 200, 202, 205, 207, 209, 212, 214, 215, 222, 256, 259, 266–270, 272
layer_simple_rnn, 216, 218, 237, 245, 259
layer_spatial_dropout_1d, 223, 262, 263, 264
layer_spatial_dropout_2d, 223, 262, 263, 264
layer_spatial_dropout_3d, 223, 262, 263, 264
layer_subtract, 190, 192, 198, 222, 247, 250, 251, 265
layer_upsampling_1d, 200, 202, 205, 207, 209, 212, 214, 215, 222, 256, 259, 265, 267–270, 272
layer_upsampling_2d, 200, 202, 205, 207, 209, 212, 214, 215, 222, 256, 259, 266, 266, 268–270, 272
layer_upsampling_3d, 200, 202, 205, 207, 209, 212, 214, 215, 222, 256, 259, 266, 267, 267, 269, 270, 272
layer_zero_padding_1d, 200, 202, 205, 207, 209, 212, 214, 215, 222, 256, 259, 266–268, 268, 270, 272
layer_zero_padding_2d, 200, 202, 205, 207, 209, 212, 214, 215, 222, 256, 259, 266–269, 269, 272
layer_zero_padding_3d, 200, 202, 205, 207, 209, 212, 214, 215, 222, 256, 259, 266–270, 271
load_model_hdf5 (save_model_hdf5), 293
load_model_hdf5(), 276, 295
load_model_weights_hdf5 (save_model_weights_hdf5), 294
load_text_tokenizer (save_text_tokenizer), 295
loss_binary_crossentropy (loss_mean_squared_error), 272
loss_categorical_crossentropy (loss_mean_squared_error), 272
loss_categorical_crossentropy() ,306
loss_categorical_hinge (loss_mean_squared_error), 272
loss_cosine_proximity (loss_mean_squared_error), 272
loss_hinge (loss_mean_squared_error), 272
loss_kullback_leibler_divergence (loss_mean_squared_error), 272
loss_logcosh (loss_mean_squared_error), 272
loss_mean_absolute_error (loss_mean_squared_error), 272
loss_mean_absolute_percentage_error (loss_mean_squared_error), 272
loss_mean_squared_error (loss_mean_squared_error), 272
loss_mean_squared_logarithmic_error (loss_mean_squared_error), 272
loss_poisson (loss_mean_squared_error), 272
loss_sparse_categorical_crossentropy (loss_mean_squared_error), 272
loss_squared_hinge (loss_mean_squared_error), 272
make_sampling_table, 273, 287, 298, 301, 302, 304
metric_binary_accuracy, 274
metric_binary_crossentropy (metric_binary_accuracy), 274
metric_categorical_accuracy (metric_binary_accuracy), 274
metric_categorical_crossentropy (metric_binary_accuracy), 274
metric_cosine_proximity (metric_binary_accuracy), 274
metric_hinge (metric_binary_accuracy), 274
metric_kullback_leibler_divergence (metric_binary_accuracy), 274
metric_mean_absolute_error (metric_binary_accuracy), 274
metric_mean_absolute_percentage_error (metric_binary_accuracy), 274
metric_mean_squared_error (metric_binary_accuracy), 274
metric_mean_squared_logarithmic_error (metric_binary_accuracy), 274
metric_poisson (metric_binary_accuracy), 274
metric_squared_hinge (metric_binary_accuracy), 274

metric_sparse_categorical_crossentropy (metric_binary_accuracy), 274
metric_sparse_top_k_categorical_accuracy (metric_binary_accuracy), 274
metric_squared_hinge (metric_binary_accuracy), 274
metric_top_k_categorical_accuracy (metric_binary_accuracy), 274
mobilenet_decode_predictions (application_mobilenet), 15
mobilenet_load_model_hdf5 (application_mobilenet), 15
mobilenet_preprocess_input (application_mobilenet), 15
mobilenet_v2_decode_predictions (application_mobilenet_v2), 16
mobilenet_v2_load_model_hdf5 (application_mobilenet_v2), 16
mobilenet_v2_preprocess_input (application_mobilenet_v2), 16
model_from_json (model_to_json), 277
model_from_yaml (model_to_yaml), 277
model_to_json, 61, 277, 278, 294, 295, 297
model_to_yaml, 61, 277, 277, 294, 295, 297
multi_gpu_model, 34, 44, 47, 49, 58, 60, 83, 85, 278, 288–291, 299, 307
nasnet_preprocess_input (application_nasnet), 18
normalize, 280
optimizer_adadelta, 281, 282–286
optimizer_adagrad, 281, 281, 283–286
optimizer_adam, 281, 282, 282, 283–286
optimizer_adamax, 281–283, 283, 284–286
optimizer_nadam, 281–283, 284, 285, 286
optimizer_rmsprop, 281–284, 285, 286
optimizer_sgd, 281–285, 285
pad_sequences, 274, 286, 298, 301, 302, 304
plot(), 288
plot.keras_training_history, 287
pop_layer, 34, 44, 47, 49, 58, 60, 83, 85, 279, 288, 289–291, 299, 307
predict_classes (predict_proba), 291
predict_generator, 34, 44, 47, 49, 58, 60, 83, 85, 279, 288, 289, 289, 290, 291, 299, 307
predict_generator(), 55
predict_on_batch, 34, 44, 47, 49, 58, 60, 83, 85, 279, 288–290, 290, 291, 299, 307
predict_proba, 34, 44, 47, 49, 58, 60, 83, 85, 279, 288–290, 291, 299, 307
py_to_r(), 23
R6Class, 79–82
regularizer_l1, 292
regularizer_l1_l2 (regularizer_l1), 292
regularizer_l2 (regularizer_l1), 292
reset_states, 36, 58, 60, 61, 292
reticulate::py_install(), 77
save_model_hdf5, 61, 277, 278, 293, 295, 297
save_model_hdf5(), 35, 80, 279, 297
save_model_weights_hdf5, 61, 277, 278, 294, 297
save_model_weights_hdf5(), 35, 80, 279
save_text_tokenizer, 50, 295, 296, 299, 300, 303
sequences_to_matrix, 50, 295, 296, 299, 300, 303
sequences_to_matrix(), 50
serialize_model, 61, 277, 278, 294, 295, 296
serialize_model(), 293
set_weights (get_weights), 61
skipgrams, 274, 287, 297, 301, 302, 304
skipgrams(), 274
summary.keras.engine.training.Model, 34, 44, 47, 49, 58, 60, 83, 85, 279, 288–291, 298, 307
test_on_batch (train_on_batch), 307
text_hashing_trick, 274, 287, 298, 301, 302, 304
text_one_hot, 274, 287, 298, 301, 302, 304
text_to_word_sequence, 274, 287, 298, 301, 302, 303
text_tokenizer, 50, 295, 296, 299, 300, 302
text_tokenizer(), 50
texts_to_matrix, 50, 295, 296, 299, 300, 303
texts_to_matrix(), 50
texts_to_sequences, 50, 295, 296, 299, 300, 303
texts_to_sequences(), 50
texts_to_sequences_generator, 50, 295, 296, 299, 300, 300, 303

TimeDistributed, 24, 305
timeseries_generator, 304
to_categorical, 306
to_categorical(), 273
train_on_batch, 34, 44, 47, 49, 58, 60, 83, 85, 279, 288–291, 299, 307

unfreeze_weights (freeze_weights), 56
unserialize_model (serialize_model), 296
use_backend (use_implementation), 307
use_implementation, 307

with_custom_object_scope, 308
with_custom_object_scope(), 276, 293

xception_preprocess_input
    (application_xception), 22