Package ‘BayesFluxR’

October 6, 2023

Type Package
Title Implementation of Bayesian Neural Networks
Version 0.1.2
Maintainer Enrico Wegner <e.wegner@student.maastrichtuniversity.nl>
Description Implementation of 'BayesFlux.jl' for R: It extends the famous 'Flux.jl' machine learning library to Bayesian Neural Networks. The goal is not to have the fastest production ready library, but rather to allow more people to be able to use and research on Bayesian Neural Networks.
License MIT + file LICENSE
Encoding UTF-8
RoxygenNote 7.2.3
Imports JuliaCall (>= 0.17.5), stats
Suggests testthat (>= 3.0.0)
Config/testthat/edition 3
NeedsCompilation no
Author Enrico Wegner [aut, cre]
Repository CRAN
Date/Publication 2023-10-06 14:10:03 UTC

R topics documented:

  .install_pkg .......................................................... 2
  .julia_project_status ............................................. 3
  .set_seed ............................................................ 3
  .using ................................................................. 4
  BayesFluxR_setup .................................................. 4
  bayes_by_backprop ................................................ 5
  BNN ................................................................. 6
  BNN.totparams ....................................................... 7
  Chain ................................................................. 8
Index

/install_pkg  Installs Julia packages if needed

Description

Installs Julia packages if needed
.julia_project_status

Usage

.install_pkg(...)  

Arguments

... strings of package names

---

.julia_project_status  Obtain the status of the current Julia project

Description

Obtain the status of the current Julia project

Usage

.julia_project_status()

---

.set_seed  Set a seed both in Julia and R

Description

Set a seed both in Julia and R

Usage

.set_seed(seed)

Arguments

seed seed to be used

Value

No return value, called for side effects.

Examples

## Not run:
## Needs previous call to `BayesFluxR_setup` which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)  
.set_seed(123)

## End(Not run)
BayesFluxR_setup

.Description

Loads Julia packages

.Usage

 .using(...) 

.Arguments

... strings of package names

BayesFluxR_setup

Set up of the Julia environment needed for BayesFlux

.Description

This will set up a new Julia environment in the current working directory or another folder if provided. This environment will then be set with all Julia dependencies needed.

.Usage

BayesFluxR_setup( 
    pkg_check = TRUE, 
    nthreads = 4, 
    seed = NULL, 
    env_path = getwd(), 
    installJulia = FALSE, 
    ...
)

.Arguments

pkg_check (Default=TRUE) Check whether needed Julia packages are installed
nthreads (Default=4) How many threads to make available to Julia
seed Seed to be used.
env_path The path to were the Julia environment should be created. By default, this is the current working directory.
installJulia (Default=TRUE) Whether to install Julia
...
Other parameters passed on to julia_setup
bayes_by_backprop

Value

No return value, called for side effects.

Examples

```r
## Not run:
## Time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)

## End(Not run)
```

bayes_by_backprop  Use Bayes By Backprop to find Variational Approximation to BNN.

Description

This was proposed in Blundell, C., Cornebise, J., Kavukcuoglu, K., & Wierstra, D. (2015, June). Weight uncertainty in neural network. In International conference on machine learning (pp. 1613-1622). PMLR.

Usage

```r
bayes_by_backprop(
  bnn,
  batchsize,
  epochs,
  mc_samples = 1,
  opt = opt.ADAM(),
  n_samples_convergence = 10
)
```

Arguments

- **bnn**: a BNN obtained using `BNN`
- **batchsize**: batch size
- **epochs**: number of epochs to run for
- **mc_samples**: samples to use in each iteration for the MC approximation usually one is enough.
- **opt**: An optimiser. These all start with ‘opt.’. See for example `opt.ADAM`
- **n_samples_convergence**: At the end of each iteration convergence is checked using this many MC samples.
Value

- 'juliavar' - julia variable storing VI
- 'juliacode' - julia representation of function call
- 'params' - variational family parameters for each iteration
- 'losses' - BBB loss in each iteration

Examples

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(RNN(5, 1))
like <- likelihood.sequitoone_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
data <- matrix(rnorm(10*1000), ncol = 10)
# Choosing sequences of length 10 and predicting one period ahead
tensor <- tensor_embed_mat(data, 10*1+1)
x <- tensor[1:10, , , drop = FALSE]
# Last value in each sequence is the target value
y <- tensor[11, , ,]
bnn <- BNN(x, y, like, prior, init)
vi <- bayes_by_backprop(bnn, 100, 100)
vi_samples <- vi.get_samples(vi, n = 1000)
## End(Not run)
```

BNN

Create a Bayesian Neural Network

Description

Create a Bayesian Neural Network

Usage

BNN(x, y, like, prior, init)

Arguments

- `x` - For a Feedforward structure, this must be a matrix of dimensions variables x observations; For a recurrent structure, this must be a tensor of dimensions sequence_length x number_variables x number_sequences; In general, the last dimension is always the dimension over which will be batched.
BNN.totparams

y
like
prior
init

A vector or matrix with observations.
Likelihood; See for example likelihood.feedforward_normal
Prior; See for example prior.gaussian
Initialiser; See for example initialise.allsame

Value

List with the following content

- `juliavar` - the julia variable containing the BNN
- `juliacode` - the string representation of the BNN
- `x` - x
- `juliax` - julia variable holding x
- `y` - y
- `juliay` - julia variable holding y

Examples

```r
## Not run:
## Needs previous call to 'BayesFluxR_setup' which is time
## consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
sampler <- sampler.SGLD()
ch <- mcmc(bnn, 10, 1000, sampler)

## End(Not run)
```

BNN.totparams

Obtain the total parameters of the BNN

Description

Obtain the total parameters of the BNN

Usage

```r
BNN.totparams(bnn)
```
Arguments

bnn A BNN formed using BNN

Value

The total number of parameters in the BNN

Examples

## Not run:
## Needs previous call to `BayesFluxR_setup` which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
BNN.totparams(bnn)

## End(Not run)
### Dense

Create a Dense layer with `in_size` inputs and `out_size` outputs using `act` activation function

#### Description

Create a Dense layer with `in_size` inputs and `out_size` outputs using `act` activation function

#### Usage

Dense(in_size, out_size, act = c("identity", "sigmoid", "tanh", "relu"))

#### Arguments

- **in_size**: Input size
- **out_size**: Output size
- **act**: Activation function

#### Value

A list with the following content

- in_size - Input Size
- out_size - Output Size
- activation - Activation Function
- julia - Julia code representing the Layer
find_mode

Description

Find the MAP of a BNN using SGD

Usage

find_mode(bnn, optimiser, batchsize, epochs)

Arguments

- **bnn**: a BNN obtained using BNN
- **optimiser**: an optimiser. These start with `opt.`. See for example `opt.ADAM`
- **batchsize**: batch size
- **epochs**: number of epochs to run for

Value

Returns a vector. Use `posterior_predictive` to obtain a prediction using this MAP estimate.

Examples

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time
## consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 5, "relu"))

## End(Not run)

find_mode(bnn, optimiser, batchsize, epochs)
```

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time
## consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 5, "relu"))

## End(Not run)
```
**Gamma**

Create a Gamma Prior

**Description**

Creates a Gamma prior in Julia using Distributions.jl

**Usage**

```julia
Gamma(shape = 2, scale = 2)
```

**Arguments**

- `shape`  
  shape parameter
- `scale`  
  scale parameter

**Value**

A list with the following content

- `juliavar` - julia variable containing the distribution
- `juliacode` - julia code used to create the distribution

**Examples**

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
## End(Not run)
```

---

**get_random_symbol**

Creates a random string that is used as variable in julia

**Description**

Creates a random string that is used as variable in julia

**Usage**

```r
get_random_symbol()
```
**initialise.allsame**  
Initialises all parameters of the network, all hyper parameters of the prior and all additional parameters of the likelihood by drawing random values from ‘dist’.

**Description**
Initialises all parameters of the network, all hyper parameters of the prior and all additional parameters of the likelihood by drawing random values from ‘dist’.

**Usage**

```
initialise.allsame(dist, like, prior)
```

**Arguments**
- `dist` A distribution; See for example `Normal`
- `like` A likelihood; See for example `likelihood.feedforward_normal`
- `prior` A prior; See for example `prior.gaussian`

**Value**
A list containing the following

- ‘juliavar’ - julia variable storing the initialiser
- ‘juliacode’ - julia code used to create the initialiser

**Examples**

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
BNN.totparams(bnn)
## End(Not run)
```
Description

Creates and Inverse Gamma prior in Julia using Distributions.jl

Usage

InverseGamma(shape = 2, scale = 2)

Arguments

shape  shape parameter
scale  scale parameter

Value

A list with the following content

• juliavar - julia variable containing the distribution
• juliacode - julia code used to create the distribution

See Also

Gamma

Examples

## Not run:
## Needs previous call to 'BayesFluxR_setup' which is time
## consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, InverseGamma(2.0, 0.5))

## End(Not run)
likelihood.feedforward_normal

Use a Normal likelihood for a Feedforward network

Description
This creates a likelihood of the form
\[ y_i \sim \text{Normal}(\text{net}(x_i), \sigma) \quad \forall i = 1, \ldots, N \]
where the \( x_i \) is fed through the network in a standard feedforward way.

Usage
likelihood.feedforward_normal(chain, sig_prior)

Arguments
- **chain**: Network structure obtained using `link{Chain}`
- **sig_prior**: A prior distribution for sigma defined using `Gamma`, `link{InverGamma}`, `Truncated`, `Normal`

Value
A list containing the following
- **julivar**: Julia variable containing the likelihood
- **juliacode**: Julia code used to create the likelihood

Examples
```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time-consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
BNN.totparams(bnn)
## End(Not run)
```
Use a t-Distribution likelihood for a Feedforward network

Description
This creates a likelihood of the form
\[
\frac{y_i - net(x_i)}{\sigma} \sim T_\nu \quad \forall i = 1, ..., N
\]
where the \( x_i \) is fed through the network in the standard feedforward way.

Usage
likelihood.feedforward_tdist(chain, sig_prior, nu = 30)

Arguments
- **chain**: Network structure obtained using `link{Chain}`
- **sig_prior**: A prior distribution for sigma defined using `Gamma`, `link{InverGamma}`, `Truncated`, `Normal`
- **nu**: DF of TDist

Value
see `likelihood.feedforward_normal`

Examples
```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_tdist(net, Gamma(2.0, 0.5), nu=8)
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
BNN.totparams(bnn)
## End(Not run)
```
### Description

This creates a likelihood of the form

\[ y_i \sim \text{Normal}(\text{net}(x_i), \sigma), i = 1, \ldots, N \]

Here \( x_i \) is a subsequence which will be fed through the recurrent network to obtain the final output \( \text{net}(x_i) = \hat{y}_i \). Thus, if one has a single time series, and splits the single time series into subsequences of length \( K \) which are then used to predict the next output of the time series, then each \( x_i \) consists of \( K \) consecutive observations of the time series. In a sense one constrains the maximum memory length of the network this way.

### Usage

```r
likelihood.seqtoone_normal(chain, sig_prior)
```

### Arguments

- **chain**: Network structure obtained using `Chain`.
- **sig_prior**: A prior distribution for \( \sigma \) defined using `Gamma`, `InverGamma`, `Truncated`, `Normal`.

### Value

See `likelihood.feedforward_normal`.

### Examples

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time
## consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(RNN(5, 1))
like <- likelihood.seqtoone_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- array(rnorm(5*100*10), dim=c(10,5,100))
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
BNN.totparams(bnn)

## End(Not run)
```
Use a T-likelihood for a seq-to-one recurrent network.

Description

See `likelihood.seqtoone_normal` and `likelihood.feedforward_tdist` for details.

Usage

```r
likelihood.seqtoone_tdist(chain, sig_prior, nu = 30)
```

Arguments

- `chain`: Network structure obtained using `link{Chain}`
- `sig_prior`: A prior distribution for sigma defined using `Gamma`, `link{InverGamma}`, `Truncated`, `Normal`
- `nu`: DF of TDist

Value

See `likelihood.feedforward_normal`

Examples

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(RNN(5, 1))
like <- likelihood.seqtoone_tdist(net, Gamma(2.0, 0.5), nu=5)
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- array(rnorm(5*100*10), dim=c(10,5,100))
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
BNN.totparams(bnn)
## End(Not run)
```
**LSTM**

Create an LSTM layer with `in_size` input size, and `out_size` hidden state size

**Description**

Create an LSTM layer with `in_size` input size, and `out_size` hidden state size

**Usage**

LSTM(in_size, out_size)

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>in_size</td>
<td>Input size</td>
</tr>
<tr>
<td>out_size</td>
<td>Output size</td>
</tr>
</tbody>
</table>

**Value**

A list with the following content

- `in_size` - Input Size
- `out_size` - Output Size
- `julia` - Julia code representing the Layer

**See Also**

Dense

**Examples**

```r
## Not run:
## Needs previous call to 'BayesFluxR_setup' which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(LSTM(5, 5))
## End(Not run)
```
Use the diagonal of sample covariance matrix as inverse mass matrix.

Usage

madapter.DiagCov(adapt_steps, windowlength, kappa = 0.5, epsilon = 1e-06)

Arguments

adapt_steps Number of adaptation steps
windowlength Lookback window length for calculation of covariance
kappa How much to shrink towards the identity
epsilon Small value to add to diagonal so as to avoid numerical non-pos-def problem

Value

list containing ‘juliavar‘ and ‘juliacode‘ and all given arguments.

Examples

```r
## Not run:
## Needs previous call to 'BayesFluxR_setup' which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
madapter <- madapter.DiagCov(100, 10)
sampler <- sampler.GGMC(madapter = madapter)
ch <- mcmc(bnn, 10, 1000, sampler)
## End(Not run)
```
Use a fixed mass matrix

Description

Use a fixed mass matrix

Usage

madapter.FixedMassMatrix(mat = NULL)

Arguments

mat (Default=NULL); inverse mass matrix; If ‘NULL’, then identity matrix will be used

Value

list with ‘juliavar’ and ‘juliacode’ and given matrix or ‘NULL’

Examples

## Not run:
## Needs previous call to `BayesFluxR_setup` which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
madapter <- madapter.FixedMassMatrix()
sampler <- sampler.GGMC(madapter = madapter)
ch <- mcmc(bnn, 10, 1000, sampler)

# Providing a non-sense weight matrix
weight_matrix <- matrix(runif(BNN.totparams(bnn)^2, 0, 1),
nrow = BNN.totparams(bnn))
madapter2 <- madapter.FixedMassMatrix(weight_matrix)
sampler2 <- sampler.GGMC(madapter = madapter2)
ch2 <- mcmc(bnn, 10, 1000, sampler2)

## End(Not run)
madapter.FullCov  

Use the full covariance matrix as inverse mass matrix

Description

Use the full covariance matrix as inverse mass matrix.

Usage

madapter.FullCov(adapt_steps, windowlength, kappa = 0.5, epsilon = 1e-06)

Arguments

- adapt_steps: Number of adaptation steps
- windowlength: Lookback window length for calculation of covariance
- kappa: How much to shrink towards the identity
- epsilon: Small value to add to diagonal so as to avoid numerical non-pos-def problem

Value

See madapter.DiagCov

Examples

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
madapter <- madapter.FullCov(100, 10)
sampler <- sampler.GGMC(madapter = madapter)
ch <- mcmc(bnn, 10, 1000, sampler)

## End(Not run)
```
Use RMSprop to adapt the inverse mass matrix.

Description


Usage

```
madapter.RMSProp(adapt_steps, lambda = 1e-05, alpha = 0.99)
```

Arguments

- `adapt_steps`: number of adaptation steps
- `lambda`: see above paper
- `alpha`: see above paper

Value

list with `juliavar` and `juliacode` and all given arguments

Examples

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time
## consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
madapter <- madapter.RMSProp(100)
sampler <- sampler.GGMC(madapter = madapter)
ch <- mcmc(bnn, 10, 1000, sampler)
## End(Not run)
```
Sample from a BNN using MCMC

Usage

```
mcmc(
    bnn, 
    batchsize, 
    numsamples, 
    sampler = sampler.SGLD(stepsize_a = 1), 
    continue_sampling = FALSE, 
    start_value = NULL 
)
```

Arguments

- **bnn**: A BNN obtained using BNN
- **batchsize**: batchsize to use; Most samplers allow for batching. For some, theoretical justifications are missing (HMC)
- **numsamples**: Number of mcmc samples
- **sampler**: Sampler to use; See for example `sampler.SGLD` and all other samplers start with 'sampler.' and are thus easy to identity.
- **continue_sampling**: Do not start new sampling, but rather continue sampling For this, numsamples must be greater than the already sampled number.
- **start_value**: Values to start from. By default these will be sampled using the initialiser in 'bnn'.

Value

a list containing the 'samples' and the 'sampler' used.

Examples

```
## Not run:
## Needs previous call to 'BayesFluxR_setup' which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
```
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
sampler <- sampler.SGNHTS(1e-3)
ch <- mcmc(bnn, 10, 1000, sampler)

## End(Not run)

---

**Normal**

*Create a Normal Prior*

**Description**

Creates a Normal prior in Julia using Distributions.jl. This can then be truncated using `Truncated` to obtain a prior that could then be used as a variance prior.

**Usage**

`Normal(mu = 0, sigma = 1)`

**Arguments**

- `mu`: Mean
- `sigma`: Standard Deviation

**Value**

see `Gamma`

**Examples**

```r
## Not run:
## Needs previous call to 'BayesFluxR_setup' which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Truncated(Normal(0, 0.5), 0, Inf))

## End(Not run)
```
**opt.ADAM**

**ADAM optimiser**

**Description**

ADAM optimiser

**Usage**

```
opt.ADAM(eta = 0.001, beta = c(0.9, 0.999), eps = 1e-08)
```

**Arguments**

- **eta**: stepsize
- **beta**: momentum decays; must be a list of length 2
- **eps**: Flux does not document this

**Value**

see `opt.Descent`

**Examples**

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time
## consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
find_mode(bnn, opt.ADAM(), 10, 100)
## End(Not run)
```
**opt.Descent**

*Standard gradient descent*

**Description**

Standard gradient descent

**Usage**

```r
opt.Descent(eta = 0.1)
```

**Arguments**

- **eta**  
  stepsize

**Value**

list containing

- ‘julivar’ - julia variable holding the optimiser
- ‘juliocode’ - string representation

**Examples**

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time
## consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
find_mode(bnn, opt.Descent(1e-5), 10, 100)
## End(Not run)
```
opt.RMSProp

RMSProp optimiser

Description

RMSProp optimiser

Usage

opt.RMSProp(eta = 0.001, rho = 0.9, eps = 1e-08)

Arguments

eta  learning rate
rho  momentum
eps  not documented by Flux

Value

see opt.Descent

Examples

## Not run:
## Needs previous call to `BayesFluxR_setup` which is time
## consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
find_mode(bnn, opt.RMSProp(), 10, 100)

## End(Not run)
posterior_predictive  Draw from the posterior predictive distribution

Description

Draw from the posterior predictive distribution

Usage

posterior_predictive(bnn, posterior_samples, x = NULL)

Arguments

bnn  
a BNN obtained using link(BNN)
posterior_samples  
a vector or matrix containing posterior samples. This can be obtained using mcmc, or bayes_by_backprop or find_mode.
x  
input variables. If ‘NULL’ (default), training values will be used.

Value

A matrix whose columns are the posterior predictive draws.

Examples

```r
## Not run:
## Needs previous call to 'BayesFluxR_setup' which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
sampler <- sampler.SGLD()
ch <- mcmc(bnn, 10, 1000, sampler)
pp <- posterior_predictive(bnn, ch$samples)
## End(Not run)
```
prior.gaussian

Use an isotropic Gaussian prior

Description

Use a Multivariate Gaussian prior for all network parameters. Covariance matrix is set to be equal ‘sigma * I’ with ‘I’ being the identity matrix. Mean is zero.

Usage

prior.gaussian(chain, sigma)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>chain</td>
<td>Chain obtained using Chain</td>
</tr>
<tr>
<td>sigma</td>
<td>Standard deviation of Gaussian prior</td>
</tr>
</tbody>
</table>

Value

a list containing the following

- ‘juliavar’ the julia variable used to store the prior
- ‘juliacode’ the julia code

Examples

```r
## Not run:
## Needs previous call to 'BayesFluxR_setup' which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bn <- BNN(x, y, like, prior, init)
sampler <- sampler.SGLD()
ch <- mcmc(bn, 10, 1000, sampler)
## End(Not run)
```
prior.mixturescale  Scale Mixture of Gaussian Prior

Description

Uses a scale mixture of Gaussian for each network parameter. That is, the prior is given by

\[ \pi_1 \text{Normal}(0, \sigma_1) + (1 - \pi_1) \text{Normal}(0, \sigma_2) \]

Usage

```
prior.mixturescale(chain, sigma1, sigma2, pi1)
```

Arguments

- **chain**: Chain obtained using `Chain`
- **sigma1**: Standard deviation of first Gaussian
- **sigma2**: Standard deviation of second Gaussian
- **pi1**: Weight of first Gaussian

Value

A list containing the following:

- `juliavar` the julia variable used to store the prior
- `juliacode` the julia code

Examples

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.mixturescale(net, 10, 0.1, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
sampler <- sampler.SGLD()
ch <- mcmc(bnn, 10, 1000, sampler)
## End(Not run)
```
prior_predictive

Sample from the prior predictive of a Bayesian Neural Network

Description

Sample from the prior predictive of a Bayesian Neural Network

Usage

\[
\text{prior\_predictive}(\text{bnn}, n = 1)
\]

Arguments

- **bnn**: BNN obtained using \texttt{BNN}
- **n**: Number of samples

Value

matrix of prior predictive samples; Columns are the different samples

Examples

```r
## Not run:
## Needs previous call to 'BayesFluxR_setup' which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
pp <- prior_predictive(bnn, n = 10)
## End(Not run)
```

RNN

Create a RNN layer with ‘in\_size’ input, ‘out\_size’ hidden state and ‘act’ activation function

Description

Create a RNN layer with ‘in\_size’ input, ‘out\_size’ hidden state and ‘act’ activation function
Usage

```r
RNN(in_size, out_size, act = c("sigmoid", "tanh", "identity", "relu"))
```

Arguments

- `in_size`: Input size
- `out_size`: Output size
- `act`: Activation function

Value

A list with the following content

- `in_size`: Input Size
- `out_size`: Output Size
- `activation`: Activation Function
- `julia`: Julia code representing the Layer

See Also

- `Dense`

Examples

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(RNN(5, 5, "tanh"))
## End(Not run)
```

---

```r
sadapter.Const
Use a constant stepsize in mcmc
```

Description

Use a constant stepsize in mcmc

Usage

```r
sadapter.Const(l)
```

Arguments

- `1`: stepsize
sadapter.DualAverage

Value
list with 'juliavar', 'juliacode' and the given arguments

Examples
### Not run:
```r
## Needs previous call to 'BayesFluxR_setup' which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
sadapter <- sadapter.Const(1e-5)
sampler <- sampler.GGMC(sadapter = sadapter)
ch <- mcmc(bnn, 10, 1000, sampler)
```
## End(Not run)

sadapter.DualAverage  Use Dual Averaging like in STAN to tune stepsize

Description
Use Dual Averaging like in STAN to tune stepsize

Usage
```r
sadapter.DualAverage(
  adapt_steps,
  initial_stepsize = 1,
  target_accept = 0.65,
  gamma = 0.05,
  t0 = 10,
  kappa = 0.75
)
```

Arguments
- adapt_steps number of adaptation steps
- initial_stepsize initial stepsize
- target_accept target acceptance ratio
- gamma See STAN manual NUTS paper
t0 See STAN manual or NUTS paper
kappa See STAN manual or NUTS paper

Value
list with ‘juliavar’, ‘juliacode’, and all given arguments

Examples

```r
## Not run:
## Needs previous call to 'BayesFluxR_setup' which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
sadapter <- sadapter.DualAverage(100)
sampler <- sampler.GGMC(sadapter = sadapter)
ch <- mcmc(bnn, 10, 1000, sampler)
## End(Not run)
```

sampler.AdaptiveMH

*Adaptive Metropolis Hastings as introduced in*

**Description**


**Usage**

`sampler.AdaptiveMH(bnn, t0, sd, eps = 1e-06)`

**Arguments**

- `bnn` BNN obtained using `BNN`
- `t0` Number of iterators before covariance adaptation will be started. Also the look-back period for covariance adaptation.
- `sd` Tuning parameter; See paper
- `eps` Used for numerical reasons. Increase this if pos-def-error thrown.
sampler.GGMC

Value

A list with ‘juliavar’, ‘juliacode’, and all given arguments

Examples

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bn <- BNN(x, y, like, prior, init)
sampler <- sampler.AdaptiveMH(bnn, 10, 1)
ch <- mcmc(bnn, 10, 1000, sampler)
## End(Not run)
```

sampler.GGMC

**Gradient Guided Monte Carlo**

Description


Usage

```r
sampler.GGMC(
  beta = 0.1,
  l = 1,
  sadapter = sadapter.DualAverage(1000),
  madapter = madapter.FixedMassMatrix(),
  steps = 3
)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta</td>
<td>See paper</td>
</tr>
<tr>
<td>l</td>
<td>Stepsize</td>
</tr>
<tr>
<td>sadapter</td>
<td>Stepsize adapter; Not used in original paper</td>
</tr>
<tr>
<td>madapter</td>
<td>Mass adapter; Not used in original paper</td>
</tr>
<tr>
<td>steps</td>
<td>Number of steps before accept/reject</td>
</tr>
</tbody>
</table>
Value

a list with 'juliavar', 'juliacode' and all provided arguments.

Examples

```r
## Not run:
## Needs previous call to 'BayesFluxR_setup' which is time
## consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
sadapter <- sadapter.DualAverage(100)
sampler <- sampler.GGMC(sadapter = sadapter)
ch <- mcmc(bnn, 10, 1000, sampler)
## End(Not run)
```

sampler.HMC  

Standard Hamiltonian Monte Carlo (Hybrid Monte Carlo).

Description

Allows for the use of stochastic gradients, but the validity of doing so is not clear.

Usage

```r
sampler.HMC(
  l,
  path_len,
  sadapter = sadapter.DualAverage(1000),
  madapter = madapter.FixedMassMatrix()
)
```

Arguments

- `l`  
  stepsize
- `path_len`  
  number of leapfrog steps
- `sadapter`  
  Stepsize adapter
- `madapter`  
  Mass adapter
Details

This is motivated by parts of the discussion in Neal, R. M. (1996). Bayesian Learning for Neural Networks (Vol. 118). Springer New York. https://doi.org/10.1007/978-1-4612-0745-0

Value

a list with ‘juliavar’, ‘juliacode’, and all given arguments

Examples

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
sadapter <- sadapter.DualAverage(100)
sampler <- sampler.HMC(1e-3, 3, sadapter = sadapter)
ch <- mcmc(bnn, 10, 1000, sampler)
```

## End(Not run)

sampler.SGLD


Description

Stepsizes will be adapted according to

\[ a(b + t)^{-\gamma} \]

Usage

```r
sampler.SGLD(
  stepsize_a = 0.1,
  stepsize_b = 0,
  stepsize_gamma = 0.55,
  min_stepsize = -Inf
)
```
Arguments

- `stepsizes_a`: See eq. above
- `stepsizes_b`: See eq. above
- `stepsizes_gamma`: See eq. above
- `min_stepsize`: Do not decrease stepsize beyond this

Value

A list with ‘juliavar’, ‘juliacode’, and all given arguments

Examples

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
sampler <- sampler.SGLD()
ch <- mcmc(bnn, 10, 1000, sampler)
```

sampler.SGNHTS

Stochastic Gradient Nose-Hoover Thermostat as proposed in

Description


Usage

```r
sampler.SGNHTS(
  1,
  sigmaA = 1,
  xi = 1,
  mu = 1,
  madapter = madapter.FixedMassMatrix()
)
```
Arguments

- l: Step size
- sigmaA: Diffusion factor
- xi: Thermostat
- mu: Free parameter of thermostat
- madapter: Mass Adapter; Not used in original paper and thus has no theoretical backing

Details

This is similar to SGNHT as proposed in Ding, N., Fang, Y., Babbush, R., Chen, C., Skeel, R. D., & Neven, H. (2014). Bayesian sampling using stochastic gradient thermostats. Advances in neural information processing systems, 27.

Value

A list with 'juliavar', 'juliacode' and all arguments provided

Examples

```r
## Not run:
## Needs previous call to 'BayesFluxR_setup' which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
sampler <- sampler.SGNHTS(1e-3)
ch <- mcmc(bnn, 10, 1000, sampler)
## End(Not run)
```

summary.BNN

Print a summary of a BNN

Description

Print a summary of a BNN

Usage

```r
## S3 method for class 'BNN'
summary(object, ...)
```
Arguments

- object: A BNN created using BNN

... Not used

---

**tensor_embed_mat**

*Embed a matrix of timeseries into a tensor*

---

**Description**

This is used when working with recurrent networks, especially in the case of seq-to-one modelling. Creates overlapping subsequences of the data with length `len_seq`. Returned dimensions are `seq_len x num_vars x num_subsequences`.

**Usage**

```r
tensor_embed_mat(mat, len_seq)
```

**Arguments**

- **mat**: Matrix of time series
- **len_seq**: subsequence length

**Value**

A tensor of dimension: `len_seq x num_vars x num_subsequences`

**Examples**

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(RNN(5, 1))
like <- likelihood.seqtoone_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
data <- matrix(rnorm(5*1000), ncol = 5)
# Choosing sequences of length 10 and predicting one period ahead
tensor <- tensor_embed_mat(data, 10+1)
x <- tensor[1:10, , , drop = FALSE]
# Last value in each sequence is the target value
y <- tensor[11,1,]
bnn <- BNN(x, y, like, prior, init)
BNN.totparams(bnn)

## End(Not run)
```
to_bayesplot

Convert draws array to conform with 'bayesplot'

Description

BayesFluxR returns draws in a matrix of dimension params x draws. This cannot be used with the 'bayesplot' package which expects an array of dimensions draws x chains x params.

Usage

to_bayesplot(ch, param_names = NULL)

Arguments

ch Chain of draws obtained using mcmc
param_names If 'NULL', the parameter names will be of the form 'param_1', 'param_2', etc. If 'param_names' is a string, the parameter names will start with the string with the number of the parameter attached to it. If 'param_names' is a vector, it has to provide a name for each paramter in the chain.

Value

Returns an array of dimensions draws x chains x params.

Examples

## Not run:
## Needs previous call to 'BayesFluxR_setup' which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
x <- matrix(rnorm(5*100), nrow = 5)
y <- rnorm(100)
bnn <- BNN(x, y, like, prior, init)
sampler <- sampler.SGLD()
ch <- mcmc(bnn, 10, 1000, sampler)
ch <- to_bayesplot(ch)
library(bayesplot)
mcmc_intervals(ch, pars = paste0("param_", 1:10))

## End(Not run)
Truncated

Description

Truncates a Julia Distribution between ‘lower’ and ‘upper’.

Usage

Truncated(dist, lower, upper)

Arguments

- dist: A Julia Distribution created using `Gamma, InverseGamma`
- lower: lower bound
- upper: upper bound

Value

- see `Gamma`

Examples

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time
## consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(Dense(5, 1))
like <- likelihood.feedforward_normal(net, Truncated(Normal(0, 0.5), 0, Inf))
## End(Not run)
```

---

vi.get_samples

Description

Draw samples form a variational family.

Usage

`vi.get_samples(vi, n = 1)`
**vi.get_samples**

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vi</td>
<td>obtained using <code>bayes_by_backprop</code></td>
</tr>
<tr>
<td>n</td>
<td>number of samples</td>
</tr>
</tbody>
</table>

**Value**

a matrix whose columns are draws from the variational posterior

**Examples**

```r
## Not run:
## Needs previous call to `BayesFluxR_setup` which is time consuming and requires Julia and BayesFlux.jl
BayesFluxR_setup(installJulia=TRUE, seed=123)
net <- Chain(RNN(5, 1))
like <- likelihood.seqtoone_normal(net, Gamma(2.0, 0.5))
prior <- prior.gaussian(net, 0.5)
init <- initialise.allsame(Normal(0, 0.5), like, prior)
data <- matrix(rnorm(10*1000), ncol = 10)
# Choosing sequences of length 10 and predicting one period ahead
tensor <- tensor_embed_mat(data, 10+1)
x <- tensor[1:10, , , drop = FALSE]
# Last value in each sequence is the target value
y <- tensor[11,,]
bnn <- BNN(x, y, like, prior, init)
vi <- bayes_by_backprop(bnn, 100, 100)
vi_samples <- vi.get_samples(vi, n = 1000)
pp <- posterior_predictive(bnn, vi_samples)

## End(Not run)
```
Index

.install_pkg, 2
.julia_project_status, 3
.set_seed, 3
.using, 4

bayes_by_backprop, 5, 28, 43
BayesFluxR_setup, 4
BNN, 5, 6, 8, 10, 23, 31, 34, 40
BNN.totparams, 7

Chain, 8, 29, 30

Dense, 9, 18, 32

find_mode, 10, 28

Gamma, 11, 13–17, 24, 42
get_random_symbol, 11

initialise.allsame, 7, 12
InverseGamma, 13, 42

julia_setup, 4

likelihood.feedforward_normal, 7, 12, 14,
15–17
likelihood.feedforward_tdist, 15, 17
likelihood.seqtoone_normal, 16, 17
likelihood.seqtoone_tdist, 17
LSTM, 18

madapter.DiagCov, 19, 21
madapter.FixedMassMatrix, 20
madapter.FullCov, 21
madapter.RMSProp, 22
mcmc, 23, 28, 41

Normal, 12, 14–17, 24

opt.ADAM, 5, 10, 25
opt.Descent, 25, 26, 27
opt.RMSProp, 27

posterior_predictive, 10, 28
prior.gaussian, 7, 12, 29
prior.mixturescale, 30
prior_predictive, 31

RNN, 31

sadapter.Const, 32
sadapter.DualAverage, 33
sampler.AdaptiveMH, 34
sampler.GGMC, 35
sampler.HMC, 36
sampler.SGLD, 23, 37
sampler.SGNHTS, 38
summary.BNN, 39
tensor_embed_mat, 40
to_bayesplot, 41
Truncated, 14–17, 24, 42
vi.get_samples, 42

44