Introduction to keras

Daniel Meier and Jürg Schelldorfer

What is keras?

An R package to fit neural networks

keras-package   *R interface to Keras*

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.

back-end uses TensorFlow

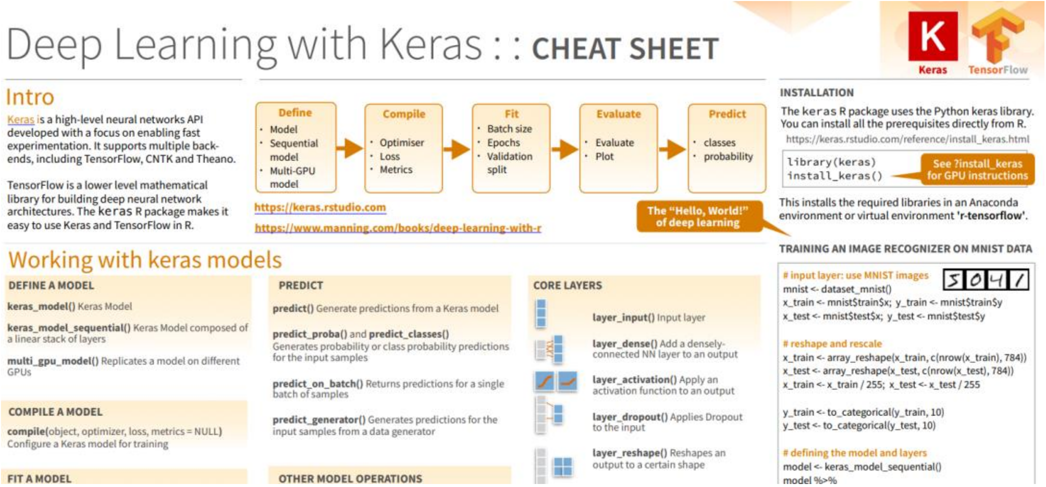
- Is capable of running on top of multiple back-ends including TensorFlow, CNTK, or Theano.

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

An R package that might not be straightforward to install and run on your computer.

Where to find help?

https://keras.rstudio.com/



https://ugoproto.github.io/ugo_r_doc/pdf/keras.pdf

And many others... It is an advantage that keras for R and Python are similar. In case of issues you can look for the solution in the (broader) Python communities.

Goal: Fit a neural network

Fitting neural networks consists of 5 steps and 5 main functions:

Define
- Model
- Sequential model
- Multi-GPU model

→

Compile
- Optimiser
- Loss
- Metrics

→

Fit
- Batch size
- Epochs
- Validation split

→

Evaluate
- Evaluate
- Plot

→

Predict
- classes
- probability

https://keras.rstudio.com

https://www.manning.com/books/deep-learning-with-r

The "Hello, World!" of deep learning

DEFINE A MODEL

keras_model() Keras Model

keras_model_sequential() Keras Model composed of a linear stack of layers

→

COMPILE A MODEL

compile(object, optimizer, loss, metrics = NULL)
Configure a Keras model for training

→

FIT A MODEL

fit(object, x = NULL, y = NULL, batch_size = NULL, epochs = 10, verbose = 1, callbacks = NULL, ...)
Train a Keras model for a fixed number of epochs (iterations)

→

EVALUATE A MODEL

evaluate(object, x = NULL, y = NULL, batch_size = NULL) Evaluate a Keras model

→

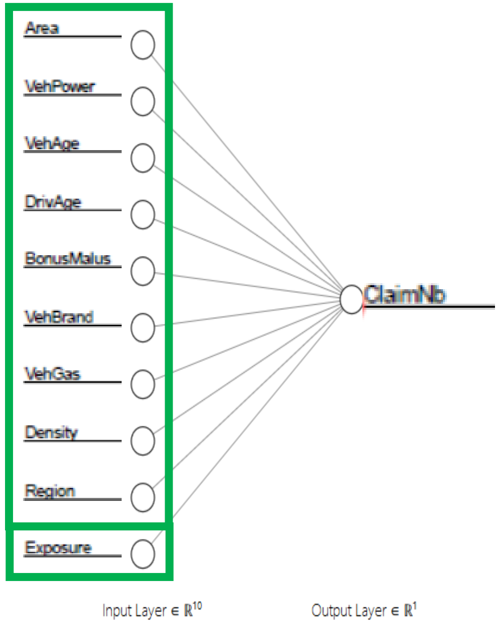
PREDICT

predict() Generate predictions from a Keras model

Model 0: Generalized Linear Model (GLM)

1

```
# define network and load pre-specified weights
q0 <- length(features)                    # dimension of features
```



layer_input()

```
Design  <- layer_input(shape = c(q0), dtype = 'float32', name = 'Design')
LogVol  <- layer_input(shape = c(1), dtype = 'float32', name = 'LogVol')
```

2

layer_dense()

```
Network <- Design %>%
  layer_dense(units = 1, activation = 'linear', name = 'Network',
              weights = list(array(0, dim = c(q0, 1)), array(log(lambda_hom), dim = c(1))))
```

3

layer_add()

```
Response <- list(Network, LogVol) %>%
  layer_add(name = 'Add') %>%
  layer_dense(units = 1, activation = k_exp, name = 'Response', trainable = FALSE,
              weights = list(array(1, dim = c(1, 1)), array(0, dim = c(1))))
```

4

keras_model()

```
model_sh <- keras_model(inputs = c(Design, LogVol), outputs = c(Response))
```

5

input layer

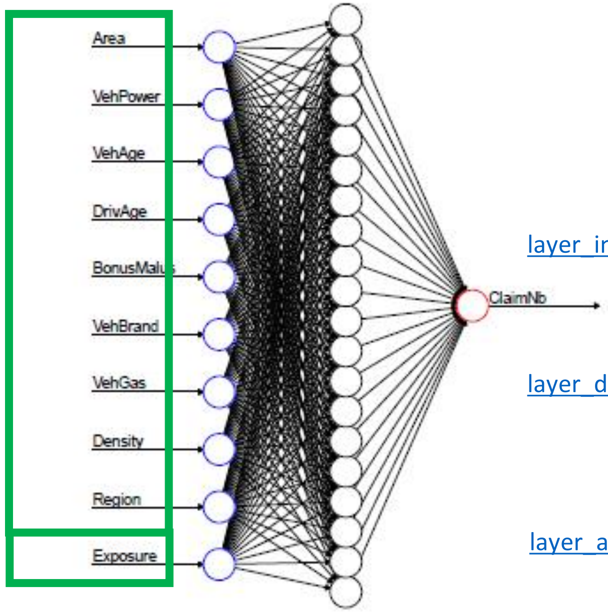
output layer

Output dimension	Trainable parameters
38 x 1	0
1 x 1	39
1 x 1	0
1 x 1	0
1 x 1	0
TOT	**39**

Model 1: Shallow neural network

1

```r
# define network and load pre-specified weights
q0 <- length(features)              # dimension of features
q1 <- 20                            # number of hidden neurons in hidden layer
```



Be careful: plotted network not correct for exposure

layer_input()

layer_dense()

layer_add()

keras_model()

```r
Design  <- layer_input(shape = c(q0), dtype = 'float32', name = 'Design')
LogVol  <- layer_input(shape = c(1), dtype = 'float32', name = 'LogVol')

Network <- Design %>%
  layer_dense(units = q1, activation = 'tanh', name = 'layer1') %>%
  layer_dense(units = 1, activation = 'linear', name = 'Network',
              weights = list(array(0, dim = c(q1, 1)), array(log(lambda_hom), dim = c(1))))

Response <- list(Network, LogVol) %>%
  layer_add(name = 'Add') %>%
  layer_dense(units = 1, activation = k_exp, name = 'Response', trainable = FALSE,
              weights = list(array(1, dim = c(1, 1)), array(0, dim = c(1))))

model_sh <- keras_model(inputs = c(Design, LogVol), outputs = c(Response))
```

2 3 4 5

input layer

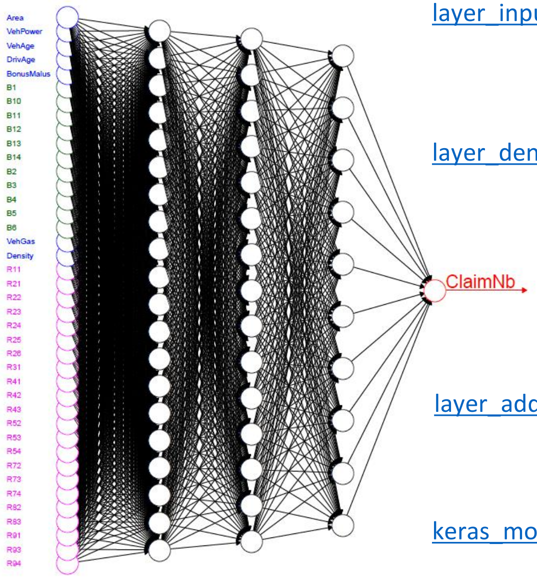
output layer

Output dimension	Trainable parameters
38 x 1	0
20 x 1	780
1 x 1	21
1 x 1	0
1 x 1	0
1 x 1	0
TOT	**801**

Model 2: Deep neural network

1

```r
# define network
q0 <- length(features)   # dimension of features
q1 <- 20                 # number of neurons in first hidden layer
q2 <- 15                 # number of neurons in second hidden layer
q3 <- 10                 # number of neurons in second hidden layer
```



layer_input()

layer_dense()

layer_add()

keras_model()

```r
Design <- layer_input(shape = c(q0), dtype = 'float32', name = 'Design')
LogVol <- layer_input(shape = c(1), dtype = 'float32', name = 'LogVol')

Network <- Design %>%
  layer_dense(units = q1, activation = 'tanh', name = 'layer1') %>%
  layer_dense(units = q2, activation = 'tanh', name = 'layer2') %>%
  layer_dense(units = q3, activation = 'tanh', name = 'layer3') %>%
  layer_dense(units = 1, activation = 'linear', name = 'Network',
              weights = list(array(0, dim = c(q3, 1)), array(log(lambda_hom), dim = c(1))))

Response <- list(Network, LogVol) %>%
  layer_add(name = 'Add') %>%
  layer_dense(units = 1, activation = k_exp, name = 'Response', trainable = FALSE,
              weights = list(array(1, dim = c(1, 1)), array(0, dim = c(1))))

model_dp <- keras_model(inputs = c(Design, LogVol), outputs = c(Response))
```

2

3

4

5

Output dimension	Trainable parameters
38 x 1	0
20 x 1	780
15 x 1	315
10 x 1	160
1 x 1	11
1 x 1	0
1 x 1	0
1 x 1	0
TOT	**1’266**

Model 4: Convolutional neural network

Output dimension	Trainable parameters
10 x 10 x 3	0
10 x 10 x 3	6
6 x 6 x 16	1216
6 x 6 x 16	32
6 x 6 x 16	0
2 x 2 x 16	6416
2 x 2 x 16	32
2 x 2 x 16	0
64 x 1	0
1 x 1	65
1 x 1	0

1

```
filterSize <- 5
numberFilters <- 16
```

layer_conv_2d()

layer_batch_normalization()

layer_activation()

layer_flatten()

layer_dense()

```
cnn <- keras_model_sequential() %>%
layer_batch_normalization() %>%
layer_conv_2d(filters = numberFilters, kernel_size = c(filterSize, filterSize),
              strides = c(1,1), padding = 'valid', data_format = 'channels_last') %>%
layer_batch_normalization() %>%
layer_activation('relu') %>%
layer_conv_2d(filters = numberFilters, kernel_size = c(filterSize, filterSize),
              strides = c(1,1), padding = 'valid', data_format = 'channels_last') %>%
layer_batch_normalization() %>%
layer_activation('relu') %>%
layer_flatten() %>%
layer_dense(1) %>%
layer_activation('sigmoid') %>%
compile(loss='mean_squared_error', optimizer='sgd')
```

2

3

4

5

6