

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?



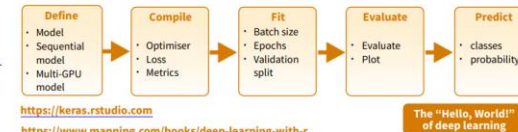
<https://keras.rstudio.com/>

Deep Learning with Keras : : CHEAT SHEET

Intro

Keras is a high-level neural networks API developed with a focus on enabling fast experimentation. It supports multiple backends, including TensorFlow, CNTK and Theano.

TensorFlow is a lower level mathematical library for building deep neural network architectures. The Keras R package makes it easy to use Keras and TensorFlow in R.



<https://keras.rstudio.com>

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

Working with keras models

DEFINE A MODEL

`keras_model()` Keras Model

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

`multi_gpu_model()` Replicates a model on different GPUs

COMPILE A MODEL

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

FIT A MODEL

PREDICT

`predict()` Generate predictions from a Keras model

`predict_proba()` and `predict_classes()` Generates probability or class probability predictions for the input samples

`predict_on_batch()` Returns predictions for a single batch of samples

`predict_generator()` Generates predictions for the input samples from a data generator

OTHER MODEL OPERATIONS

CORE LAYERS

`layer_input()` Input layer

`layer_dense()` Add a densely-connected NN layer to an output

`layer_activation()` Apply an activation function to an output

`layer_dropout()` Applies Dropout to the input

`layer_reshape()` Reshapes an output to a certain shape

INSTALLATION

The keras R package uses the Python keras library. You can install all the prerequisites directly from R.

https://keras.rstudio.com/reference/install_keras.html

```
library(keras)
```

```
install_keras()
```

See 'install_keras' for GPU instructions

This installs the required libraries in an Anaconda environment or virtual environment 'r-tensorflow'.

TRAINING AN IMAGE RECOGNIZER ON MNIST DATA

```
# input layer: use MNIST images
mnist <- dataset_mnist()
x_train <- mnist$train$x; y_train <- mnist$train$y
x_test <- mnist$test$x; y_test <- mnist$test$y
```

```
# reshape and rescale
x_train <- array_reshape(x_train, c(nrow(x_train), 784))
x_test <- array_reshape(x_test, c(nrow(x_test), 784))
x_train <- x_train / 255; x_test <- x_test / 255
```

```
y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)
```

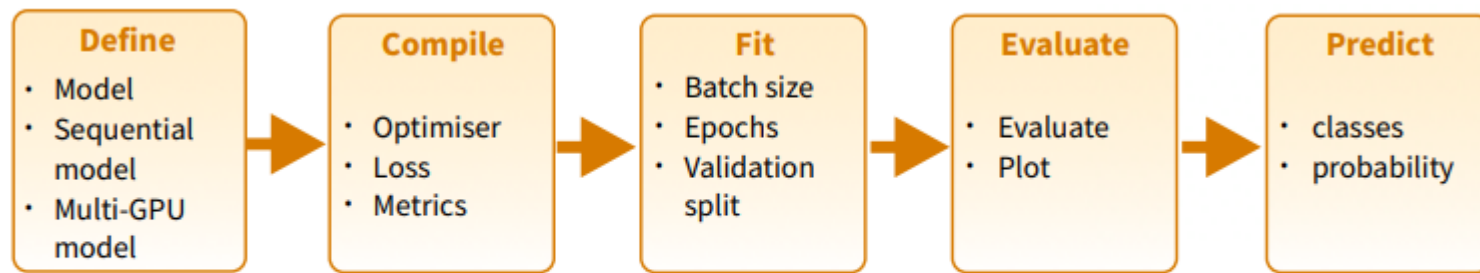
```
# defining the model and layers
model <- keras_model_sequential()
model %>%
```

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:



<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)

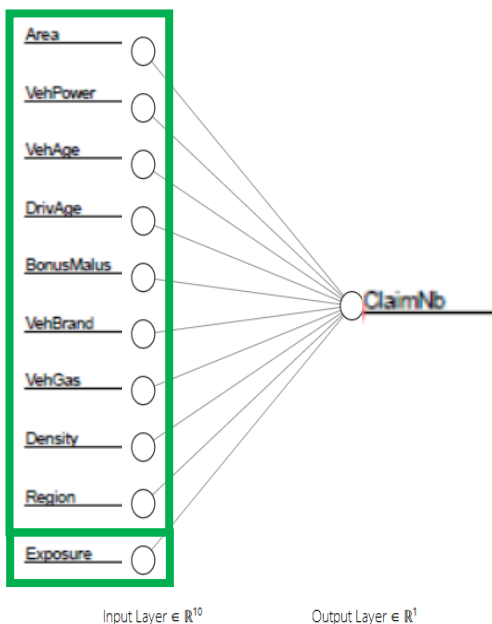
EVALUATE A MODEL

`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\(\)](#)

[layer_dense\(\)](#)

[layer_add\(\)](#)

[keras_model\(\)](#)

```
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 = 1, activation = 'linear', name = 'Network',
              weights = list(array(0, dim = c(q0, 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))))

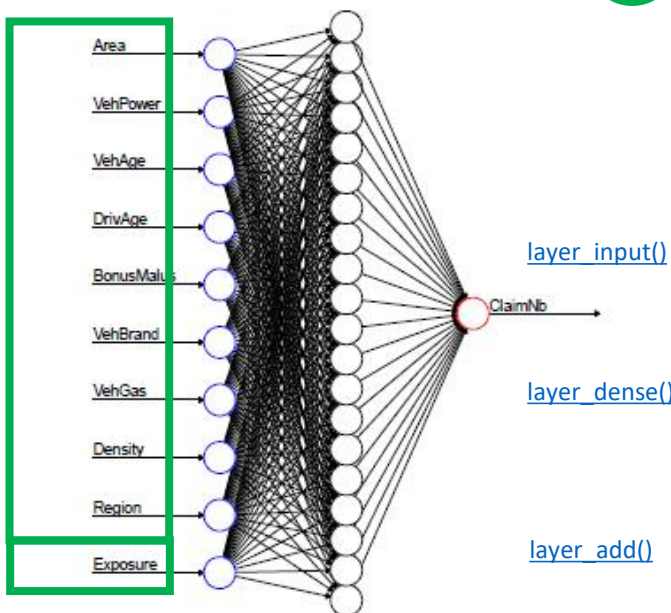
model_sh <- keras_model(inputs = c(Design, LogVol), outputs = c(Response))
```

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



Be careful: plotted network not correct for exposure

1

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

[layer_input\(\)](#)

[layer_dense\(\)](#)

[layer_add\(\)](#)

[keras_model\(\)](#)

```
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))
```

input layer

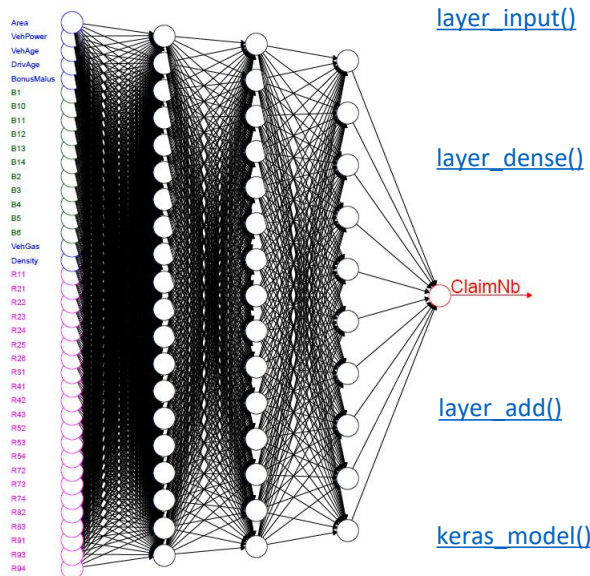
output layer

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

```
# 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
```



```
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))
```

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
```

```
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