

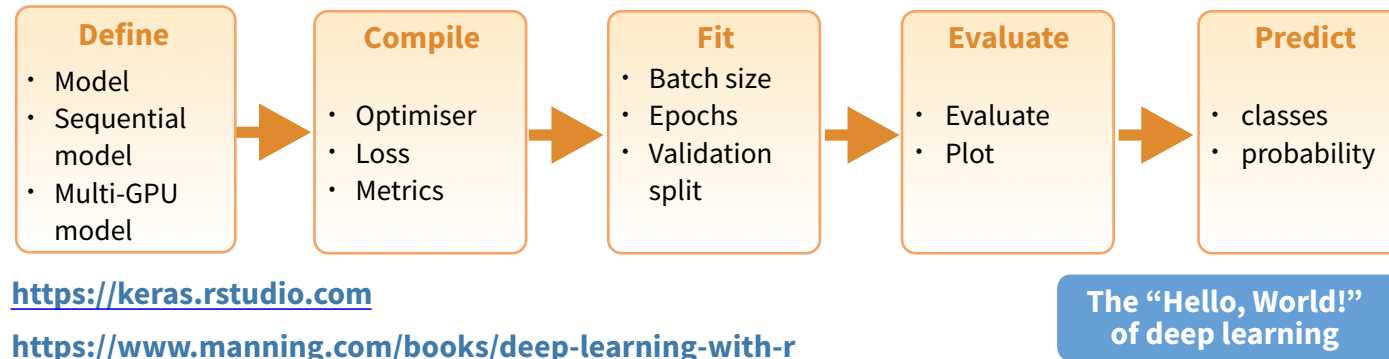
Deep Learning with Keras :: CHEAT SHEET



Intro

[Keras](#) is a high-level neural networks API developed with a focus on enabling fast experimentation. It supports multiple back-ends, 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.



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 ?keras_install for GPU instructions

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

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

`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)

`fit_generator()` Fits the model on data yielded batch-by-batch by a generator

`train_on_batch()` `test_on_batch()` Single gradient update or model evaluation over one batch of samples

EVALUATE A MODEL

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

`evaluate_generator()` Evaluates the model on a data generator

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

`summary()` Print a summary of a Keras model

`export_savedmodel()` Export a saved model

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

`pop_layer()` Remove the last layer in a model

`save_model_hdf5(); load_model_hdf5()` Save/Load models using HDF5 files

`serialize_model(); unserialize_model()`
Serialize a model to an R object

`clone_model()` Clone a model instance

`freeze_weights(); unfreeze_weights()`
Freeze and unfreeze weights

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



`layer_permute()` Permute the dimensions of an input according to a given pattern



`layer_repeat_vector()` Repeats the input n times



`layer_lambda(object, f)` Wraps arbitrary expression as a layer



`layer_activity_regularization()`
Layer that applies an update to the cost function based input activity



`layer_masking()` Masks a sequence by using a mask value to skip timesteps



`layer_flatten()` Flattens an input

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 %>%
  layer_dense(units = 256, activation = 'relu',
              input_shape = c(784)) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dense(units = 10, activation = 'softmax')
```

compile (define loss and optimizer)

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

train (fit)

```
model %>% fit(
  x_train, y_train,
  epochs = 30, batch_size = 128,
  validation_split = 0.2
)
model %>% evaluate(x_test, y_test)
model %>% predict_classes(x_test)
```

More layers

CONVOLUTIONAL LAYERS



layer_conv_1d() 1D, e.g. temporal convolution



layer_conv_2d_transpose()
Transposed 2D (deconvolution)

layer_conv_2d() 2D, e.g. spatial convolution over images



layer_conv_3d_transpose()
Transposed 3D (deconvolution)
layer_conv_3d() 3D, e.g. spatial convolution over volumes

layer_conv_lstm_2d()
Convolutional LSTM

layer_separable_conv_2d()
Depthwise separable 2D



layer_upsampling_1d()
layer_upsampling_2d()
layer_upsampling_3d()
Upsampling layer



layer_zero_padding_1d()
layer_zero_padding_2d()
layer_zero_padding_3d()
Zero-padding layer



layer_cropping_1d()
layer_cropping_2d()
layer_cropping_3d()
Cropping layer

POOLING LAYERS



layer_max_pooling_1d()
layer_max_pooling_2d()
layer_max_pooling_3d()
Maximum pooling for 1D to 3D



layer_average_pooling_1d()
layer_average_pooling_2d()
layer_average_pooling_3d()
Average pooling for 1D to 3D



layer_global_max_pooling_1d()
layer_global_max_pooling_2d()
layer_global_max_pooling_3d()
Global maximum pooling



layer_global_average_pooling_1d()
layer_global_average_pooling_2d()
layer_global_average_pooling_3d()
Global average pooling

ACTIVATION LAYERS



layer_activation(object, activation)
Apply an activation function to an output



layer_activation_leaky_relu()
Leaky version of a rectified linear unit



layer_activation_parametric_relu()
Parametric rectified linear unit



layer_activation_thresholded_relu()
Thresholded rectified linear unit



layer_activation_elu()
Exponential linear unit

DROPOUT LAYERS



layer_dropout()
Applies dropout to the input



layer_spatial_dropout_1d()
layer_spatial_dropout_2d()
layer_spatial_dropout_3d()
Spatial 1D to 3D version of dropout

RECURRENT LAYERS



layer_simple_rnn()
Fully-connected RNN where the output is to be fed back to input

layer_gru()
Gated recurrent unit - Cho et al

layer_cudnn_gru()
Fast GRU implementation backed by CuDNN

layer_lstm()
Long-Short Term Memory unit - Hochreiter 1997

layer_cudnn_lstm()
Fast LSTM implementation backed by CuDNN

LOCALLY CONNECTED LAYERS

layer_locally_connected_1d()
layer_locally_connected_2d()
Similar to convolution, but weights are not shared, i.e. different filters for each patch

Preprocessing

SEQUENCE PREPROCESSING

pad_sequences()
Pads each sequence to the same length (length of the longest sequence)

skipgrams()
Generates skipgram word pairs

make_sampling_table()
Generates word rank-based probabilistic sampling table

TEXT PREPROCESSING

text_tokenizer() Text tokenization utility

fit_text_tokenizer() Update tokenizer internal vocabulary

save_text_tokenizer(); load_text_tokenizer()
Save a text tokenizer to an external file

texts_to_sequences(); texts_to_sequences_generator()
Transforms each text in texts to sequence of integers

texts_to_matrix(); sequences_to_matrix()
Convert a list of sequences into a matrix

text_one_hot() One-hot encode text to word indices

text_hashing_trick()
Converts a text to a sequence of indexes in a fixed-size hashing space

text_to_word_sequence()
Convert text to a sequence of words (or tokens)

IMAGE PREPROCESSING

image_load() Loads an image into PIL format.

flow_images_from_data()
flow_images_from_directory()
Generates batches of augmented/normalized data from images and labels, or a directory

image_data_generator() Generate minibatches of image data with real-time data augmentation.

fit_image_data_generator() Fit image data generator internal statistics to some sample data

generator_next() Retrieve the next item

image_to_array(); image_array_resize()
image_array_save() 3D array representation



Pre-trained models

Keras applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning.

application_xception()
xception_preprocess_input()
Xception v1 model

application_inception_v3()
inception_v3_preprocess_input()
Inception v3 model, with weights pre-trained on ImageNet

application_inception_resnet_v2()
inception_resnet_v2_preprocess_input()
Inception-ResNet v2 model, with weights trained on ImageNet

application_vgg16(); application_vgg19()
VGG16 and VGG19 models

application_resnet50() ResNet50 model

application_mobilenet()
mobilenet_preprocess_input()
mobilenet_decode_predictions()
mobilenet_load_model_hdf5()
MobileNet model architecture

IMAGENET

[ImageNet](https://www.image-net.org/) is a large database of images with labels, extensively used for deep learning

imagenet_preprocess_input()
imagenet_decode_predictions()
Preprocesses a tensor encoding a batch of images for ImageNet, and decodes predictions

Callbacks

A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training.

callback_early_stopping() Stop training when a monitored quantity has stopped improving
callback_learning_rate_scheduler() Learning rate scheduler
callback_tensorboard() TensorBoard basic visualizations

