

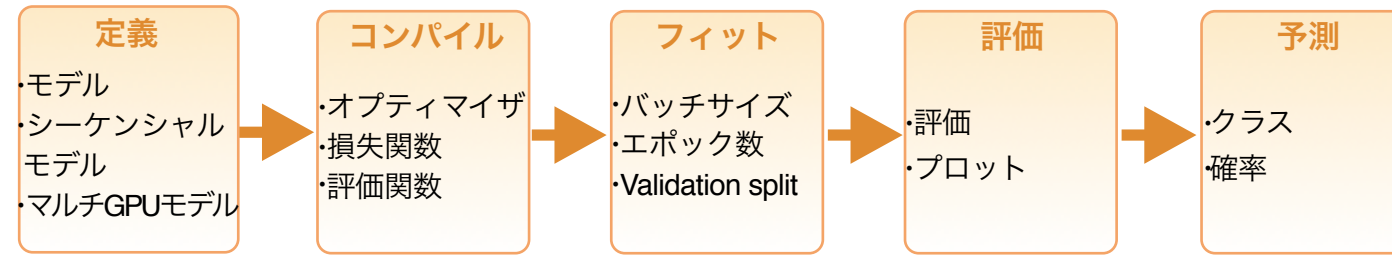
Deep Learning with Keras : : CHEAT SHEET



イントロ

Keras は高レベルなニューラルネットワーク API です。迅速な実験を可能にすることに重点を置いて開発されました。TensorFlow, CNTK, Theano といった複数のバックエンドをサポートしています。

TensorFlow は深層ニューラルネットワークを記述するために利用する低レベルな数学です。Keras R パッケージは Keras と TensorFlow を R 内で簡単に利用することができます。



<https://keras.rstudio.com>

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

インストール

keras R パッケージは Python の keras ライブラリを使用しています。依存するものは全て R だけでインストールできます。

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

```
library(keras)
install_keras()
```

GPU版については
?keras_install を参照

必要なライブラリが Anaconda 環境か 'r-tensorflow' という名前の virtual env 環境にインストールされます。

kerasモデルの操作

モデル定義

`keras_model()` Keras モデル

`keras_model_sequential()` 線形に積み上げた Keras モデル

`multi_gpu_model()` デルを複数の GPU へ複製 (データ並列化)

モデルコンパイル

`compile(object, optimizer, loss, metrics = NULL)` 学習用に Keras モデルを定義する

学習

`fit(object, x = NULL, y = NULL, batch_size = NULL, epochs = 10, verbose = 1, callbacks = NULL, ...)` 指定したエポック数の分だけ Keras モデルを学習させる

`fit_generator()` ジェネレータを用いてバッチ毎に生成されたデータでモデルを訓練する

`train_on_batch()` `test_on_batch()` 単一バッチデータに対して一度、勾配の更新/モデル評価を行う

モデルの評価

`evaluate(object, x = NULL, y = NULL, batch_size = NULL)` Keras モデルを評価する

`evaluate_generator()` データジェネレータを用いてモデルを評価する

予測

`predict()` Keras モデルを用いて予測を行う

`predict_proba()` `predict_classes()` 入力サンプルに対して確率/クラス確率を生成

`predict_on_batch()` サンプルの単一バッチに対する予測値を返す

`predict_generator()` ジェネレータから生成されたデータに対して予測値を返す

その他のモデル操作

`summary()` Keras モデルのサマリを表示

`export_savedmodel()` 保存したモデルをエクスポート

`get_layer()` 名前かインデックスで層を取り出す
`pop_layer()` モデルの最終層を削除

`save_model_hdf5(); load_model_hdf5()` HDF5 ファイルとしてモデルを保存/読み込み

`serialize_model(); unserialize_model()` R オブジェクトとしてモデルをシリアライズ

`clone_model()` モデルインスタンスをクローン

`freeze_weights(); unfreeze_weights()` 重み付けを固定/固定解除

所謂 "Hello, World!" の
ディープラーニング版

基本レイヤー

`layer_input()` 入力層

`layer_dense()` 出力層に全結合ニューラルネット層を追加

`layer_activation()` 出力に対して活性化関数を適用

`layer_dropout()` 入力に対してドロップアウトを適用

`layer_reshape()` 出力を任意の形へ変形

`layer_permute()` 与えられたパターンに従い入力の組み合わせを生成

`layer_repeat_vector()` 入力を n 回繰り返す

`layer_lambda(object, f)` 任意の関数を適用する層

`layer_activity_regularization()` 入力に基づいてコスト関数を更新

`layer_masking()` スキップされるタイムステップを特定するためマスク値を使い入力をマスクする

`layer_flatten()` 入力を1次元に変換

MNIST画像でのトレーニング例

```
# 入力層: MNIST画像
mnist <- dataset_mnist()
x_train <- mnist$train$x; y_train <- mnist$train$y
x_test <- mnist$test$x; y_test <- mnist$test$y

# 変形・リスケール
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)

# モデルと層の定義
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')




# コンパイル (損失関数とオプティマイザの定義)
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)

# 学習 (フィッティング)
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)
```







様々なレイヤー






畳込み層(CNN)

-  **layer_conv_1d()** 1次元畳込み
例: 一時的な畳込み
-  **layer_conv_2d_transpose()** 2次元のTransposed畳込み 2D (逆畳込み)
-  **layer_conv_2d()** 2次元畳込み
例: 画像の空間畳込み
-  **layer_conv_3d_transpose()** 3次元のTransposed畳込み (逆畳込み)
-  **layer_conv_3d()** 3次元畳込み
例: 立体の空間畳込み
- layer_conv_lstm_2d()** LSTM畳込み
- layer_separable_conv_2d()** 深度方向(空間方向)へ分離して畳込み
-  **layer_upsampling_1d()**
layer_upsampling_2d()
layer_upsampling_3d() アップサンプリング層
-  **layer_zero_padding_1d()**
layer_zero_padding_2d()
layer_zero_padding_3d() ゼロ埋め層
-  **layer_cropping_1d()**
layer_cropping_2d()
layer_cropping_3d() 切り落とし層

プーリング層

-  **layer_max_pooling_1d()**
layer_max_pooling_2d()
layer_max_pooling_3d() 1D~3DのMaxプーリング
-  **layer_average_pooling_1d()**
layer_average_pooling_2d()
layer_average_pooling_3d() 1D~3Dの平均プーリング
-  **layer_global_max_pooling_1d()**
layer_global_max_pooling_2d()
layer_global_max_pooling_3d() 特徴マップ全体へのMaxプーリング
-  **layer_global_average_pooling_1d()**
layer_global_average_pooling_2d()
layer_global_average_pooling_3d() 特徴マップ全体への平均プーリング

活性化層

-  **layer_activation(object, activation)** 出力に活性化関数を適用
-  **layer_activation_leaky_relu()** LeakyReLU
-  **layer_activation_parametric_relu()** 傾きがパラメータ可変されたReLU
-  **layer_activation_thresholded_relu()** 閾値を用いたReLU
-  **layer_activation_elu()** ELU(指数的線形ユニット)

ドロップアウト層

-  **layer_dropout()** ロップアウトの適用
-  **layer_spatial_dropout_1d()**
layer_spatial_dropout_2d()
layer_spatial_dropout_3d() 1D~3D版空間的ドロップアウト

再帰層(RNN)

-  **layer_simple_rnn()** 出力層が入力層へ全結合したRNN層
- layer_gru()** ゲート付き再帰ユニット - [Cho et al.](#)
- layer_cudnn_gru()** CuDNNを使った高速版GRU
- layer_lstm()** Long-Short Term Memoryユニット - ([Hochreiter 1997](#))
- layer_cudnn_lstm()** CuDNNを用いた高速版LSTM

局所接続層

- layer_locally_connected_1d()**
layer_locally_connected_2d() 畳込みと同様. ただし、重みは共有されず入力パッチ毎に異なったフィルタが適用される

前処理

シーケンスの前処理

- pad_sequences()** 各シーケンスを同じ長さに揃える (長さは最長のシーケンスに揃えられる)
- skipgrams()** 単語インデックスのシーケンスを生成する
- make_sampling_table()** 順位ベースに確率付けされた単語のサンプリングテーブルを生成する

テキストの前処理

- text_tokenizer()** トークン化ユーティリティ
- fit_text_tokenizer()** トークナイザの持つボキャブラリを更新する
- save_text_tokenizer(); load_text_tokenizer()** トークナイザをファイルへ保存する
- texts_to_sequences(); texts_to_sequences_generator()** 文章中の各文をシーケンスへ変換する
- texts_to_matrix(); sequences_to_matrix()** シーケンスのリストを行列へ変換する
- text_one_hot()** 文を単語インデックスへOne-hotエンコーディングする
- text_hashing_trick()** 文を固定長ハッシュ空間のインデックスのシーケンスへ変換する
- text_to_word_sequence()** 文を単語またはトークンのシーケンスへ変換する

画像の前処理

- image_load()** PILフォーマットとして画像を読み込む
- flow_images_from_data()**
flow_images_from_directory() 元画像とラベルまたはディレクトリの情報から画像を拡張(水増し)/正規化するバッチを生成する
- image_data_generator()** リアルタイムにデータを拡張するミニバッチを生成する
- fit_image_data_generator()** 画像データ生成器の内部統計量をサンプルデータに基づいて計算する
- generator_next()** 次のアイテムを取り出す
- image_to_array(); image_array_resize()**
image_array_save() 3D配列表現



学習済みモデル

Keras applicationとは事前に学習で得られた重みが利用できる深層学習モデルです. これらのモデルは予測, 特徴抽出, ファインチューニングに用いることが可能です.

- application_xception(); xception_preprocess_input()** Xception v1 モデル
- application_inception_v3(); inception_v3_preprocess_input()** Inception v3 モデル(ImageNetで重みを計算済み)
- application_inception_resnet_v2(); inception_resnet_v2_preprocess_input()** Inception-ResNet v2 モデル(ImageNetで重みを計算済み)
- application_vgg16(); application_vgg19()** VGG16・VGG19モデル
- application_resnet50()** ResNet50モデル
- application_mobilenet(); mobilenet_preprocess_input(); mobilenet_decode_predictions(); mobilenet_load_model_hdf5()** MobileNet モデルアーキテクチャ

IMAGENET

[ImageNet](#)とはディープラーニングでよく使われる巨大なラベル付きデータセットです

- imagenet_preprocess_input()**
imagenet_decode_predictions() テンソルをImageNet用に画像のバッチへ変換する前処理や、予測のデコードを行う

コールバック

コールバックは学習の各段階で適用される関数の事です. 内部状態や学習中の統計情報を見るために使う事ができます

- callback_early_stopping()** 監視している評価値が向上しない場合に学習を早期に中断させる
- callback_learning_rate_scheduler()** 学習率のスケジューラ
- callback_tensorboard()** TensorBoardを使った可視化

