

## Keras のカンペ

`クラス keras` は簡単に深層学習モデルを試すことができるフレームワークです。

### 基本的な使い方

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
data = np.random.random((1000,100))
labels = np.random.randint(2,size=(1000,1))
model = Sequential()
model.add(Dense(32,
                activation='relu',
                input_dim=100))
model.add(Dense(1,
                activation='sigmoid'))
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
model.fit(data,labels,epochs=10,batch_size=32)
predictions = model.predict(data)
```

### データの準備

訓練データとテストデータとも `Numpy` の `array` として保存してください。サイキットラーン `scikit-learn` のモジュール `sklearn.cross_validation` を使って分割できます。

```
from keras.datasets import boston_housing,
                           mnist,
                           cifar10,
                           imdb

(x_train,y_train),(x_test,y_test) = mnist.load_data()
(x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
(x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
(x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
num_classes = 10
```

### 別の場所からデータを取得

```
from urllib.request import urlopen
data = np.loadtxt(urlopen("http://archive.ics.uci.edu/ml/\
machine-learning-databases/pima-indians-diabetes/\
pima-indians-diabetes.data"),delimiter=",")
X = data[:,0:8]
y = data[:,8]
```

### 前処理

#### 系列処理のためのパディング

```
from keras.preprocessing import sequence
x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

#### ワンホットベクトル化

```
from keras.utils import to_categorical
Y_train = to_categorical(y_train, num_classes)
Y_test = to_categorical(y_test, num_classes)
Y_train3 = to_categorical(y_train3, num_classes)
Y_test3 = to_categorical(y_test3, num_classes)
```

## 訓練データとテストデータへデータを分割

```
from sklearn.model_selection import train_test_split
X_train5, X_test5, y_train5, y_test5 = train_test_split(X,
                                                         y,
                                                         test_size=0.33,
                                                         random_state=42)
```

## 標準化, 規格化

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler().fit(x_train2)
standardized_X = scaler.transform(x_train2)
standardized_X_test = scaler.transform(x_test2)
```

## モデル

### Sequential モデル

```
from keras.models import Sequential
model = Sequential()
model2 = Sequential()
model3 = Sequential()
```

## 多層パーセプトロン

### 二値分類

```
from keras.layers import Dense
model.add(Dense(12,
                input_dim=8,
                kernel_initializer='uniform',
                activation='relu'))
model.add(Dense(8, kernel_initializer='uniform', activation='relu'))
model.add(Dense(1, kernel_initializer='uniform', activation='sigmoid'))
```

### 多クラス分類

```
from keras.layers import Dropout
model.add(Dense(512, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
```

### 回帰

```
model.add(Dense(64, activation='relu', input_dim=train_data.shape[1]))
model.add(Dense(1))
```

## 畳込みニューラルネットワーク CNN

```

from keras.layers import Activation, Conv2D, MaxPooling2D, Flatten
model2.add(Conv2D(32, (3, 3), padding='same', input_shape=x_train.shape[1:]))
model2.add(Activation('relu'))
model2.add(Conv2D(32, (3, 3)))
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.25))
model2.add(Conv2D(64, (3, 3), padding='same'))
model2.add(Activation('relu'))
model2.add(Conv2D(64, (3, 3)))
model2.add(Activation('relu'))
model2.add(MaxPooling2D(pool_size=(2, 2)))
model2.add(Dropout(0.25))
model2.add(Flatten())
model2.add(Dense(512))
model2.add(Activation('relu'))
model2.add(Dropout(0.5))
model2.add(Dense(num_classes))
model2.add(Activation('softmax'))

```

## リカレントニューラルネットワーク RNN

```

from keras.layers import Embedding, LSTM
model3.add(Embedding(20000, 128))
model3.add(LSTM(128, dropout=0.2, recurrent_dropout=0.2))
model3.add(Dense(1, activation='sigmoid'))

```

## モデルの照合

```

model.output_shape      # 出力層の構成
model.summary()         # 作成したモデル表示
model.get_config()       # 作成したモデルの構成
model.get_weights()     # モデルの全結合係数リスト

```

## モデルのコンパイル(訓練実施前に実施)

```

# 多層パーセプトロン 2 値分類
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
# 多層パーセプトロン 多値分類
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
# 多層パーセプトロン 回帰
model.compile(optimizer='rmsprop',
              loss='mse',
              metrics=['mae'])

# リカレントニューラルネットワーク
model3.compile(loss='binary_crossentropy',
               optimizer='adam',
               metrics=['accuracy'])

```

## 学習の実施

```

model3.fit(x_train4,
          y_train4,
          batch_size=32,
          epochs=15,
          verbose=1,
          validation_data=(x_test4, y_test4))

```

## モデルの評価

```

model3.evaluate(x_test,
               y_test,
               batch_size=32)

```

## 訓練済モデルによる予測

```
model3.predict(x_test4, batch_size=32)
model3.predict_classes(x_test4, batch_size=32)
```

## モデルの保存と読み込み

```
from keras.models import load_model
model3.save('model_file.h5')
my_model = load_model('my_model.h5')
```

## ファインチューニング

```
from keras.optimizers import RMSprop
opt = RMSprop(lr=0.0001, decay=1e-6)
model2.compile(loss='categorical_crossentropy',
               optimizer=opt,
               metrics=['accuracy'])
```