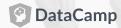
# **Keras** Cheat Sheet

## **BecomingHuman.Al**



Keras is a powerfuland easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

## A Basic Example

>>> import numpy as no

>>> 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

input\_dim=100))

>>> model.add(Dense(1, activation='sigmoid'))

>>> model.compile(optimizer='rmsprop',

loss='binary crossentropy'.

metrics=['accuracy'])

#### Data

Also see NumPv. Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the train\_test\_split module of sklearn.cross\_validation.

#### **Keras Data Sets**

>>> from keras.datasets import boston\_housing, cifar10,

>>> (x train.v train).(x test.v 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

>>> model.fit(data,labels,epochs=10,batch\_size=32)

>>> predictions = model.predict(data)

#### Other

>>> from urllib.request import urlopen

>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/ ml/machine-learning-databases/pima-indians-diabetes.data").delimiter=".")

>>> X = data[:.0:8]

>>> y = data [:,8]

## **Model Architecture**

#### Sequential Model

>>> from keras models import Sequential

>>> model = Sequential()

>>> model2 = Sequential()

>>> model3 = Sequential()

#### Multilaver Perceptron (MLP)

#### Binary Classification

>>> 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'))

#### **Multi-Class Classification**

>>> 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'))

>>> model.add(Dense(64.activation='relu'.input\_dim=train\_data.shape[1]))

>>> model add(Dense(1))

#### **Convolutional Neural Network (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'))

#### **Recurrent Neural Network (RNN)**

>>> from keras.klavers 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'))

## **Inspect Model**

>>> model.output shape

>>> model.summarv()

>>> model.get\_config()

>>> model.get weights()

Model output shape Model summary representation Model configuration

List all weight tensors in the model

### **Prediction**

>>> model3.predict(x\_test4, batch\_size=32)

>>> model3.predict classes(x test4,batch size=32)

## **Model Training**

>>> model3.fit(x train4,

**Model Fine-tuning** 

**Optimization Parameters** 

>>> from keras.optimizers import RMSprop

>>> opt = RMSprop(lr=0.0001, decay=1e-6)

>>> from keras.callbacks import EarlyStopping

y\_train4, batch\_size=32,

**MLP: Binary Classification** >>> model.compile(optimizer='adam

>>> model.compile(optimizer='rmspro

>>> model.compile(optimizer='rmsprop',

**Recurrent Neural Network** 

Save/ Reload Models

>>> from keras.models import load model

>>> my\_model = load\_model('my\_model.h5')

>>> model3 save('model\_file.h5')

**MLP: Regression** 

>>> model3.compile(loss='bina

Early Stopping

>>> model3.fit(x\_train4,

Compile Model

>>> model2.compile(loss='categorical crossentropy',

optimizer=opt,

>>> early\_stopping\_monitor = EarlyStopping(patience=2)

metrics=['accuracy'])

epochs=15, validation\_data=(x\_test4,y\_test4),

metrics=['accuracy'])

loss='categorical\_cro metrics=['accuracy'])

metrics=['mae'])

optimizer='adam', metrics=['accuracy'])

**MLP: Multi-Class Classification** 

callbacks=[early stopping monitor])

y\_train4, batch size=32. verbose=1

validation\_data=(x\_test4,y\_test4))

## **Evaluate Your** Model's Performance

>>> score = model3.evaluate(x test

## **Preprocessing**

## Sequence Padding

>>> from keras.preprocessing import sequence

>>> x\_train4 = sequence.pad\_sequences(x\_train4,maxlen=80)

>>> x\_test4 = sequence.pad\_sequences(x\_test4,maxlen=80)

#### One-Hot Encoding

>>> 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)

#### Train and Test Sets

>>> from sklearn.model selection import train test split

>>> X\_train5,X\_test5,y\_train5,y\_test5 = train\_test\_split(X,

test\_size=0.33, random\_state=42)

#### Standardization/Normalization

>>> from sklearn.preprocessing import StandardScaler

>>> scaler = StandardScaler().fit(x train2)

>>> standardized\_X = scaler.transform(x\_train2)

>>> standardized\_X\_test = scaler.transform(x\_test2)