Deep Learning with Keras Cheat Sheet (2021), Python for Data Science

The absolute basics for beginners learning Keras for Deep Learning in 2021

Christopher Zita Apr 28 · 5 min read ★

Photo from Unsplash by Ashim D'Silva

Check out the sections below to learn how to optimize Keras to create various deep learning models.

Sections:

Basic Example

The code below demonstrates the basic steps of using Keras to create and run a deep learning model on a set of data.

The steps in the code include: loading the data, preprocessing the data, creating the model, adding layers to the model, fitting the model on the data, using the trained model to make predictions on the test set, and finally evaluating the performance of the model.

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> from keras.datasets import boston_housing
>>> (x_train, y_train),(x_test, y_test) = boston_housing.load_data()
>>> model = Sequential()
>>> model.add(Dense(12,activation='relu',input_dim= 8))
>>> model.add(Dense(1))
>>> model.compile(optimizer='rmsprop',loss='mse',metrics=['mae']))
>>> model.fit(x_train,y_train,batch_size=32,epochs=15)
>>> model.predict(x_test, batch_size=32)
>>> score = model.evaluate(x_test,y_test,batch_size=32)
```

split the data in training and test sets, for which you can also resort to the train_test_split module of sklearn.cross_validation.

For an example dataset, you can use the Boson Housing dataset that is incorporated in the Keras library.

```
>>> from keras.datasets import boston_housing
>>> (x_train, y_train),(x_test, y_test) = boston_housing.load_data()
```

Preprocessing

After a dataset is imported it may not be ready for a model to be fit onto it. In this case, you have to do some preprocessing to prepare the data for the model.

Encoding Categorical Features

Encode's target labels with values between 0 and n_classes-1.

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
```

Train and Test Sets

Splits the dataset into training and test sets for both the X and y variables.

```
>>> from sklearn.model_selection import train_test_split
>>> X_train,X_test,y_train,y_test = train_test_split(X, y,
    test_size=0.33,random_state=42)
```

Standardization

Standardize's the features by removing the mean and scaling to unit variance.

```
>>> standardized_X = scaler.transform(x_train)
>>> standardized_X_test = scaler.transform(x_test)
```

Model Architecture

In this section, you'll learn how to build different deep learning models layer by layer.

Sequential Model

A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor. This is typically the first layer you would add in any model.

```
>>> from keras.models import Sequential
>>> model = Sequential()
```

Artificial Neural Network (ANN)

Deep learning model used for **classification** and **regression.**

Binary Classification:

The code below is an example of an ANN model used for binary classification (identifying a class as 0 or 1). The code adds 3 layers to the model. The first is the input layer and has 12 nodes, the second layer has 8 nodes, and the last layer has 1 which is the output node or what is predicted.

```
>>> 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'))
>>>
```

Multi-Class Classification:

The code below is an example of an ANN model used for multi-class classification. The code adds 4 layers to the model. The first is the input layer and has 52 nodes, the second is a dropout layer used to reduce overfitting, the third layer has 52 nodes, and the last layer has 10 nodes which are the probabilities that a certain observation goes into one of 10 different classes.

```
>>> from keras.layers import Dropout
>>> model.add(Dense(52,activation='relu'),input_shape=(78,))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(52,activation='relu'))
>>> model.add(Dense(10,activation='softmax'))
```

Regression:

The code below is an example of an ANN model used for regression. The code adds 2 layers to the model. The first is the input layer which has 64 nodes, and the second is the output layer having just 1 node which is the value predicted.

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

Convolution Neural Network (CNN)

Deep learning model used for the **classification of pictures.** The model is fairly complicated and includes quite a bit of layers which you can see in the code below. That code gives a basic example of what a CNN model looks like.

If you want to have an understanding of what each layer does check out the explanations in the Keras documentation.

```
(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)

Deep learning model used for the **time series.**

The code below is an example of an RNN model used for time series. The code adds 3 layers to the model. The first layer is the input layer, the second layer is something called Long Short Term Memory, and the third layer is the output layer or what's predicted from the model.

```
>>> from keras.klayers import Embedding,LSTM
>>> model.add(Embedding(20000,128))
>>> model.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model.add(Dense(1,activation='sigmoid'))
```

Compile Model

After a model is constructed you would then compile the model. Compiling just refers to specifying what training configurations (optimizer, loss, metrics) are going to be

ANN: Binary Classification

Use these training configurations for an ANN model used for binary classification.

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

ANN: Multi-Class Classification

Use these training configurations for an ANN model used for multi-class classification.

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

ANN: Regression

Use these training configurations for an ANN model used for regression.

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

Use these training configurations for an RNN model.

```
>>> model.compile(loss='binary_crossentropy',
                  optimizer='adam',
                  metrics=['accuracy'])
```

Model Training

After compiling the model, you would now fit it onto your training set. Batch size determines how many observations are inputted into the model at one time, and

```
>>> model.fit(x_train,
              y_train,
              batch_size=32,
              epochs=15)
```

Prediction

Predicting the test set using the trained model

```
>>> model.predict(x_test, batch_size=32)
```

Evaluate Your Model's Performace

Determine how well the model performed on the test set.

```
>>> score = model.evaluate(x_test,y_test,batch_size=32)
```

Save/Reload Models

Deep learning models can take quite a long time to train and run, so when they are finished you can save and load them again so you don't have to go through that process.

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

Python is the top dog when it comes to data science for now and in the foreseeable future. Knowledge of Keras, one of its most powerful libraries for Deep Learning, is often a requirement for Data Scientists today.

Use this cheat sheet as a guide in the beginning and come back to it when needed, and you'll be well on your way to mastering the Keras library.

If you want to learn more about the Keras library and functions I didn't cover check out the documentation for **Keras**, as there are still plenty of useful functions you should learn.

Join my email list with 2k+ people to get The Complete Python for Data Science Cheat Sheet Booklet for Free.