Tutorial for using Keras

Author: Yu Mi

In this tutorial, we are going to introduce a python based toolbox, Keras, which can be used to build and train neural networks quickly. You may find how to install and how to build and train neural networks within this tutorial. Here are also some useful notes that may help you with your programming assignment.

Installing Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.*Reference

To install *Keras*, we may need to firstly install a backend, here I suggest installing *TensorFlow*:

```
pip3 install tensorflow
```

After installing *tensorflow*, we could use the instruction to install *keras*:

```
pip3 install keras
```

Example based on Iris dataset

To train a neural network classifier on the *Iris* dataset, I would suggest using the following libraries that could make the training work much easier:

```
pip3 install pandas sklearn numpy matplotlib
```

Here the *sklearn* stands for 'scikit-learn', which is a software machine learning library for Python. It contains various classification, regression and clustering algorithms and could be supportive in our programming. *Panda* library is also an open-source library to provide high-performance and easy-to-use data process and data analysis functions. *numpy* here provides basic processing functions of data structures. *matplotlib* would provide plotting functions to visualize data we need.

Step 1: Import libraries

To utilize the functions in *Keras*, *sklearn*,*numpy* and *pandas*, we need firstly import the libraries into Python:

```
import numpy
import pandas
import matplotlib.pyplot as plt
from keras.layers import Dense, Activation
from keras.models import Sequential, Model
from sklearn.model_selection import train_test_split
```

Here we import some modules from *keras* and *sklearn* which we will discuss in detail when we use these functions.

Step 2: Load dataset

To read dataset, we need to utilize the *pandas* library and do some simple selection from the dataset since it has a title. In this tutorial, we use a modified *iris* dataset, which contains only two values for each flower and only two of the species.

```
dataBase = pandas.read_csv("iris.csv",header=None)
dataSet  = dataBase.values
Input    = dataSet[51:,2:4].astype(float)
Output   = dataSet[51:,4]
```

Here we use *read_csv* function to read *.csv* file of the iris dataset, and select the data from the second row to the end. And we also split the first 2 columns into the **Input** list, and the last column into the **Output** list. One thing to note is that the output data is originally in string type from the *.csv* file, so we have to convert them into easy-to-process form for the computers to read, which we will discuss in the following step.

Step 3: Preprocess the data

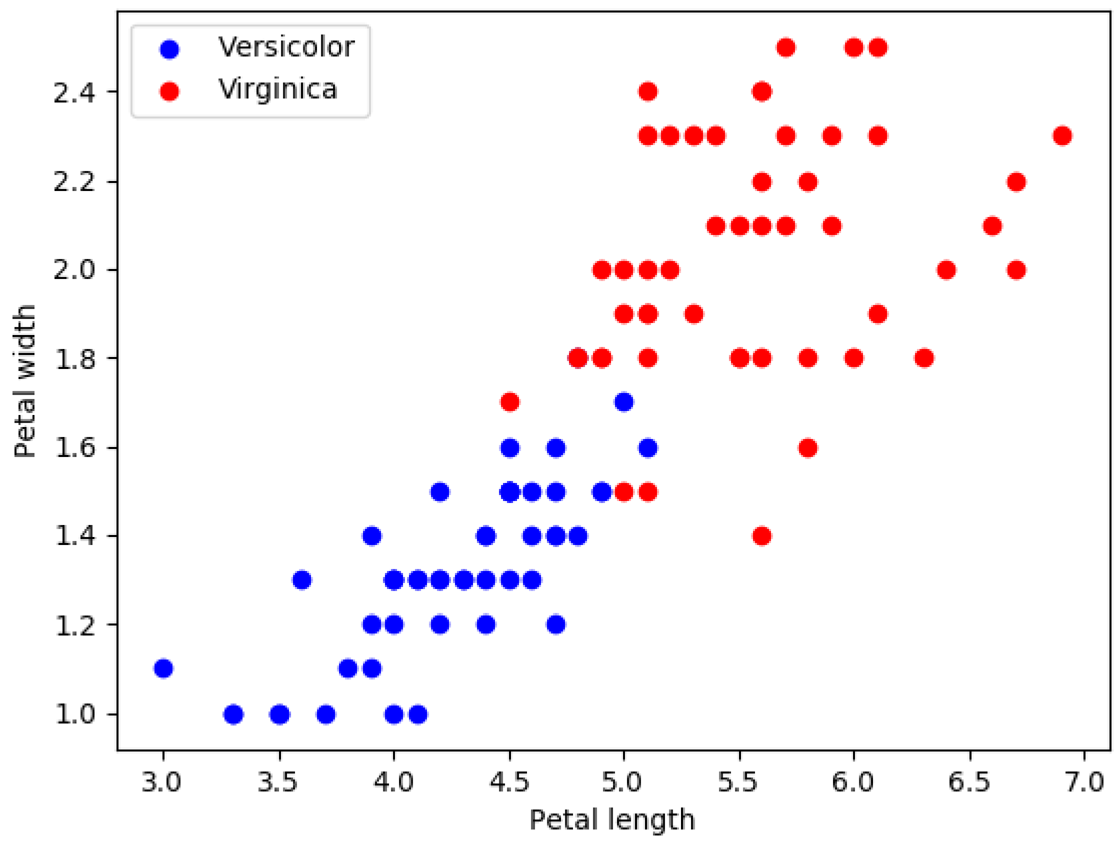
To preprocess the data in our **Output** list, we will need to convert the species string into a value indicating whether the flower belongs to a certain species, since we have only two species to classify:

```
encodeOutput = []
for i in Output:
    if i == 'versicolor':
        encodeOutput.append(0)
    else:
        encodeOutput.append(1)
```

To make the data we encoded and the input data easier for us to understand, we could also plot out the data points:

```
plt.scatter(Input[0:50,0],Input[0:50,1],c='b',label='Versicolor')
plt.scatter(Input[50:100,0],Input[50:100,1],c='r',label='Virginica')
plt.xlabel('Pental length')
plt.ylabel('Pental width')
plt.legend(loc='upper left')
plt.savefig('inputFigure.png')
```

The figure '*inputFigure.png*' is shown as follows:



We also split the dataset into training set and validation set with *train_test_split* function:

```
inputTrain,inputVal,outputTrain,outputVal =
train_test_split(Input,encodeOutput,test_size=0.25,shuffle=True)
```

Step 4: Define the model

To define the model, I would like to suggest that we use a *function* of python to pack the model structure:

```
def modelNN():
    model = Sequential()
    model.add(Dense(1,input_dim=2,activation='sigmoid'))
    model.compile(optimizer='rmsprop',loss='mean_squared_error',metrics=['accuracy'])
    return model
```

Here is the most important part that everyone would like to play with.

First, we declare the neural network model to be a sequential model, where the neural network should be a feed-forward network. Such structure of model could be most commonly used in our assignment.

After that, we add an output layer which contains 1 node, corresponding to the 2 kinds of output species of the flowers.

Finally, we will also compile the model to let it work.

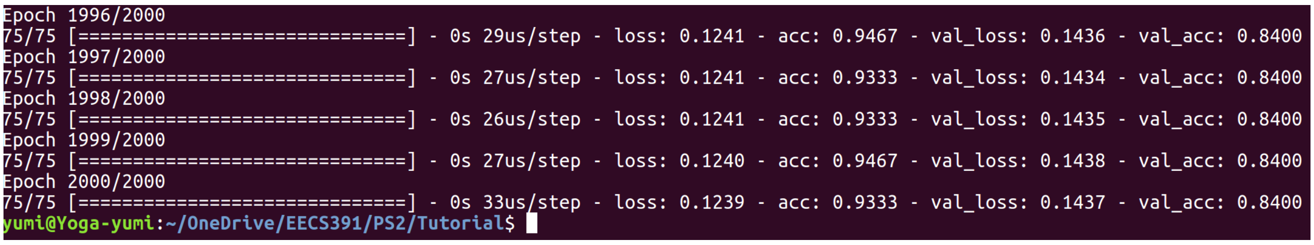
To check other details of the Keras model, I would like to suggest you read introductions of Layers, Activation functions, Loss functions and Optimizers.

Step 5: Let your neural network fit the data

After defining the model, we can use the *Keras* to make the neural network fit the data.

```
model = modelNN()
model.fit(x=inputTrain,y=outputTrain,epochs=2000, validation_data=(inputVal,outputVal))
```

Here we need to specify the training input, training output and validation data. The result on my laptop is shown as follows:



```
Epoch 1996/2000
75/75 [==============================] - 0s 29us/step - loss: 0.1241 - acc: 0.9467 - val_loss: 0.1436 - val_acc: 0.8400
Epoch 1997/2000
75/75 [==============================] - 0s 27us/step - loss: 0.1241 - acc: 0.9333 - val_loss: 0.1434 - val_acc: 0.8400
Epoch 1998/2000
75/75 [==============================] - 0s 26us/step - loss: 0.1241 - acc: 0.9333 - val_loss: 0.1435 - val_acc: 0.8400
Epoch 1999/2000
75/75 [==============================] - 0s 27us/step - loss: 0.1240 - acc: 0.9467 - val_loss: 0.1438 - val_acc: 0.8400
Epoch 2000/2000
75/75 [==============================] - 0s 33us/step - loss: 0.1239 - acc: 0.9333 - val_loss: 0.1437 - val_acc: 0.8400
yumi@Yoga-yumi:~/OneDrive/EECS391/PS2/Tutorial$
```

As you may find, the converge speed of such neural network may differ among each training round, it is your work to find out the best parameters for the neural network.