QBUS6850_Tutorial_10

May 3, 2019

QBUS6850 - Machine Learning for Business

1 Tutorial 10 - Neural Networks 1

1.1 Keras

Keras provides a very high level approach to building, training and predicting using neural networks. It is built on top of other neural network backend libraries like TensorFlow or Theano. It helps to prototype neural networks very quickly.

https://keras.io

1.1.1 Installing Keras

For this tutorial you will need to install the Keras library: - **Lab Computers**: If you are using the lab computers you can get the qbus6850 environment from GRASP, it already has keras installed. - **Personal Computer**: If you want to install Keras on your laptop etc you can type the following on the Windows Command Prompt or Terminal on OS X:

```
conda install -c conda-forge keras
```

1.1.2 Keras Error (No module named...)

Occasionally you may run into one of the following errors:

```
No module named tensorflow
```

```
No module named theano
```

This is because keras configuration states that tensorflow is installed when theano was installed and vice versa.

You need to make sure the keras JSON configuration file uses the backend that was installed. The configuration file is located in either:

```
/Users/YOUR_USERNAME/.keras/keras.json
```

```
C:\Users\YOUR_USERNAME\.keras\keras.json
```

An example of the config file is below. You just need to change the backend to "tensorflow" or "theano" depending on whichever was actually installed.

```
{
    "floatx": "float32",
    "epsilon": 1e-07,
    "backend": "tensorflow",
    "image_data_format": "channels_last"
}
```

1.2 Linear Regression Neural Network

First we will look at building a simple network to perform linear regression.

The design of the network dictates the type of function it is performing. So we must carefully match the design with that of a linear regression.

We can visualise this as a neural network:

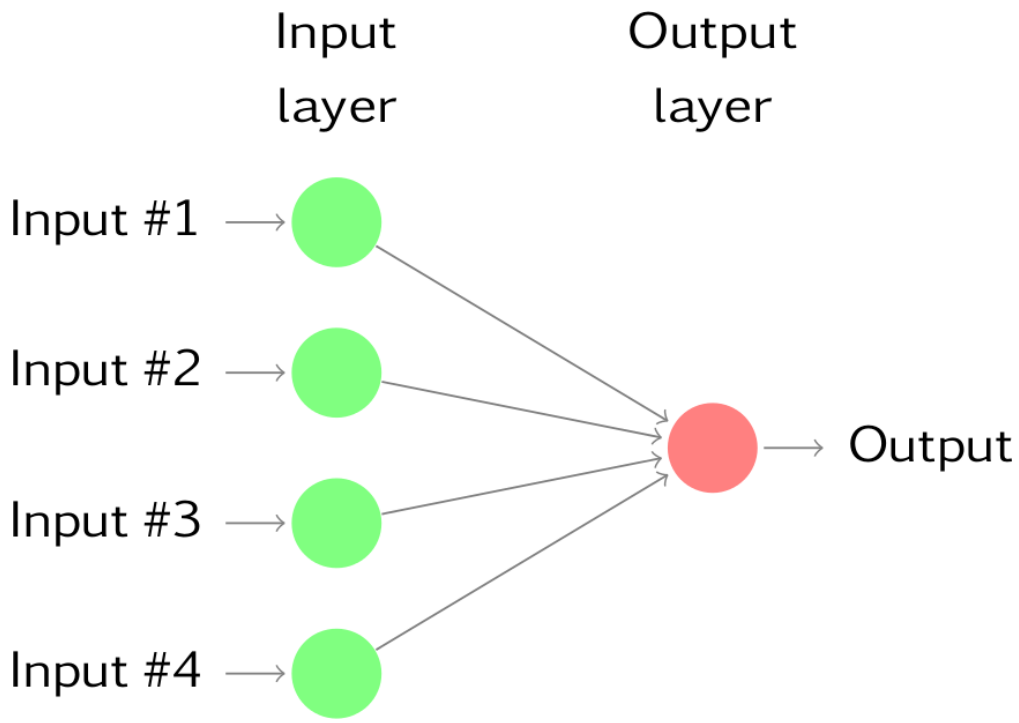

image from http://dungba.org/linear-regression-with-neural-network/

Note that in this simple case there are no hidden layers in the network.

Let's begin by loading the data

In [ ]:
```
import pandas as pd
import numpy as np

from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split

from keras.layers.core import Dense
```

```
from keras.models import Sequential

np.random.seed(0)

df = pd.read_csv("Advertising.csv", index_col=0)

df.head()
```

Usually with Neural Networks we scale all features to be in the range (0-1) to reduce the effect of feature domination and gradient vanishing. This particularly helps for deep networks. So it is a good habit to get into.

Don't worry we can easily transform the data back into the original range later with the scaler object.

```
In [ ]:
scaler = MinMaxScaler(feature_range=(0, 1))
data = scaler.fit_transform(df.values)
df_scaled = pd.DataFrame(data, columns=df.columns)
df_scaled.head()
```

Now we can do a train/test split

```
In [ ]:
y = df_scaled["Sales"]
X = df_scaled[ df_scaled.columns.difference(["Sales"]) ]
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

1.2.1 Building a Neural Network

There are three stages to building neural networks with Keras: - defining - compiling - training

When we define a network we specify the layers, their dimensions and any activation functions. Compiling instructs Keras on how it should learn the weights by defining the loss metric and optimisation algorithm. Training performs backpropogation to actually learn the weights.

```
In [ ]:
# Define the network architecture
model = Sequential()
n_features = X_train.shape[1]

# First layer, input size is the number of input features and output size is 1
# We are doing a linear regression, and let's use linearactivation
model.add(Dense(1, input_dim=n_features, activation='linear', use_bias=True))

# Use mean squared error for the loss metric and use the ADAM backprop algorithm
model.compile(loss='mean_squared_error', optimizer='adam')

# Train the network (learn the weights)
# We need to convert from DataFrame to NumpyArray
history = model.fit(X_train.values, y_train.values,
                epochs=100, batch_size=1, verbose=2, validation_split=0)
```

1.3 Visualising a Network

Often our network can get quite complicated and a visualisation can help with understanding what is happening at each layer.

Below is an example of how to plot a model as a Figure

```
In [ ]:
from keras.utils.vis_utils import model_to_dot
from IPython.display import Image, display

dot_obj = model_to_dot(model, show_shapes = True, show_layer_names = True)
display(Image(dot_obj.create_png()))
```

You can also save your figure to an PDF or image file

```
In [ ]:
from keras.utils import plot_model

plot_model(model, to_file='model.pdf', show_shapes = True, show_layer_names = True)
```

1.4 Network Inspection

You can present a tabular summary of your neural network.

```
In [ ]: model.summary()
```

To view the weights for every layer you can use a loop. Since we only have a single layer we only have one array of weights plus the bias term.

```
In [ ]:
print(len(model.layers))
```

```
In [ ]:
for layer in model.layers:
weights = layer.get_weights()
print(weights)
```

1.5 Comparison to OLS

Now lets compare with the parameters estimated by sklearn's OLS function. The parameters are very close!

```
In [ ]:
import pandas as pd
from sklearn.linear_model import LinearRegression, RidgeCV, LassoCV
from sklearn.cross_validation import train_test_split
from sklearn.metrics import mean_squared_error

ols = LinearRegression(fit_intercept=True)
ols.fit(X_train, y_train)
```

```
ols_coefs = np.concatenate( ([ols.intercept_], ols.coef_), axis = 0 )
print("OLS Coefs: {0}".format(ols_coefs))

weights = model.layers[0].get_weights()

nn_coefs = np.concatenate( (weights[1], weights[0][:,0]) , axis = 0 )
print("NN Coefs: {0}".format(nn_coefs))
```

2 Prediction

2.1 Binary Classifier Network

To build a binary classifier network we only have one requirement: that the output is the probability of the data belonging to the class or a class label.

To achieve this for a binary classifier we just need a single neuron on the output layer. Then we need to apply an activation function to limit the output to the range [0, 1].

We can achieve this with the sigmoid activation function.

```
In [ ]:
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from keras.layers.normalization import BatchNormalization
from keras.layers import Dense, Dropout

np.random.seed(0)

# Load the data
bank_df = pd.read_csv("bank.csv")

X = bank_df.iloc[:, 0:-1]
y = bank_df['y_yes']

# Scale the data
scaler = MinMaxScaler(feature_range=(0, 1))

X_scaled = pd.DataFrame(scaler.fit_transform(X.values), columns=X.columns)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y)

n_features = X_train.shape[1]

# Build our classifier
model = Sequential()
model.add(Dense(64, input_dim=n_features, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
```

```
# Final output layer is a single neuron with sigmoid activation
model.add(Dense(1, activation='sigmoid'))

# Use mean squared error for the loss metric and use the ADAM backprop algorithm
model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['accuracy'])

# Train the network (learn the weights)
# We need to convert from DataFrame to NumpyArray
history = model.fit(X_train.values, y_train.values,  epochs=20, batch_size=16,
                verbose=1, validation_split=0, class_weight={0:0.0001, 1:100})
y_pred = model.predict(X_test.values)

print(classification_report(y_test, y_pred.astype(int)))
print(confusion_matrix(y_test, y_pred.astype(int)))
```

I have manually set the class weights in the previous example since we have a very imbalanced dataset. If you want a good starting point for class weights you can use sklearn to help, like so:

```
In [ ]:
from sklearn.utils import class_weight

class_weight = class_weight.compute_class_weight('balanced', np.unique(y_train), y_train)
```

2.2 Multi-Class Classification Network

In the multi class classifer case we still need class probabilities but we need a class probability for all classes and we need the sum of all probabilties to equal 1.

We can do this using the softmax activation function!

However this means we need to adjust the shape of our target variable to match the shape of the softmax output, which is a vector.

```
In [ ]:
from keras.optimizers import SGD
np.random.seed(0)

wine_df = pd.read_csv('winequality-white.csv', delimiter=";")

X = wine_df.iloc[:, :-1]
y = wine_df.iloc[:, -1]

# Convert to dummies to get same shape as softmax output
y = pd.get_dummies(y)

scaler = MinMaxScaler(feature_range=(0, 1))

X_scaled = pd.DataFrame(scaler.fit_transform(X.values), columns=X.columns)

X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, random_state=1)
```

```
n_features = X_train.shape[1]

# create model
model = Sequential()
model.add(Dense(64, activation='relu', input_dim=n_features))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
# Softmax output layer
model.add(Dense(7, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

model.fit(X_train.values, y_train.values, epochs=20, batch_size=16)

y_pred = model.predict(X_test.values)

y_te = np.argmax(y_test.values, axis = 1)
y_pr = np.argmax(y_pred, axis = 1)

print(np.unique(y_pr))

print(classification_report(y_te, y_pr))

print(confusion_matrix(y_te, y_pr))
```

2.3 Network Parameters

Sometimes tuning for your data can take a lot of time. It is part of the skill of using neural networks. Please familiarize yourself with the terminology and how each parameter affects fitting result.

Batch size is the number of samples used in each forward/backward pass of the network. You will notice the number of samples in each Epoch increasing by the batch size. It is a good idea to keep your batch size as a power of 2 e.g. 8, 16, 32, 64, 128 for speed reasons.

An Epoch is a forward/backward pass of all batches. If you increase the number of Epoch you should see an improvement in accuracy since it is refining the model parameters each time.

The validation split is the proportion of data held out for use as validation set. It provides you with an estimate of the error on the test set. But it is not used for any actual cross validation or optimisation of parameters. For small amounts of data and for low epoch's it is better to keep this number small or 0.

2.4 Cross Validation - Estimating Performance

Keras is not tightly integrated with scikit learn. So to use normal sklearn functions like cross_val_score we need to wrap Keras models with a class that behaves like an sklearn class. Or we can take a more manual approach.

2.4.1 Method 1 - Keras Wrappers

We can use cross_val_score through the use Keras' wrapper function KerasClassifier().

```
In [ ]:
from sklearn.model_selection import cross_val_score
from keras.wrappers.scikit_learn import KerasClassifier

np.random.seed(0)


def create_model():
    model = Sequential()
    model.add(Dense(64, activation='relu', input_dim=n_features))
    model.add(Dropout(0.5))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(7, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
                  metrics=['accuracy'])
    return model

sk_model = KerasClassifier(build_fn=create_model, epochs=20, batch_size=16, verbose=0)

print("Running...")
cv_scores = cross_val_score(sk_model, X_train.values, y_train.values,
                scoring = 'neg_log_loss')

print(cv_scores)
```

2.4.2 Method 2 - Manual K-Folds

```
from sklearn.model_selection import KFold

np.random.seed(0)
n_features = X_train.shape[1]

model = Sequential()
model.add(Dense(64, activation='relu', input_dim=n_features))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')

kf = KFold(3, shuffle=False)
cv_scores = list()

print("Running...")
```

```
for train_index, val_index in kf.split(X_train.T):
    X_cvtrain, X_cvval = X_train.iloc[train_index, :], X_train.iloc[val_index, :]
    y_cvtrain, y_cvval = y_train.iloc[train_index], y_train.iloc[val_index]
    model.fit(X_cvtrain.values, y_cvtrain.values, epochs=20, batch_size=16, verbose=0)
    score = model.evaluate(X_cvval.values, y_cvval.values, verbose=0)
    # First item in scores is the loss. We specified the loss as crossentropy
    # so we take the negative of it.
    cv_scores.append(-score)

print(cv_scores)
```

2.5 Cross Validation - Hyper Parameter Optimisation

2.5.1 Method 1 - GridSearchCV

```
In [ ]:
from sklearn.model_selection import GridSearchCV

def create_model(optimizer='rmsprop'):
    model = Sequential()
    model.add(Dense(64, activation='relu', input_dim=n_features))
    model.add(Dropout(0.5))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(7, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
    return model

model = KerasClassifier(build_fn=create_model, verbose=0)
optimizers = ['rmsprop']
epochs = [5, 10, 15]
batches = [128]

param_grid = dict(optimizer=optimizers, epochs=epochs, batch_size=batches, verbose=['2'])
grid = GridSearchCV(estimator=model, param_grid=param_grid)
grid.fit(X_train.values, y_train.values)
```

2.5.2 Cross Validation - Optimising Layers

You can also combine GridSearchCV and the method below to optimise the hyperparameters and layers at the same time

```
In [ ]:
total_layers = [
            Dense(64, activation='relu', input_dim=n_features),
            Dropout(0.5),
            Dense(64, activation='relu'),
            Dropout(0.5)
        ]
```

```
final_layer = Dense(7, activation='softmax')

class model_builder:

    def __init__(self, layers, final_layer):
        self.layers = layers
        self.final_layer = final_layer

    def __call__(self):
        self.model = Sequential()

        for layer in self.layers:
            self.model.add(layer)

        self.model.add(self.final_layer)

        self.model.compile(loss='categorical_crossentropy', optimizer='adam',
                           metrics=['accuracy'])

        return self.model


cv_mean_scores = list()
print("Running...")
for i in range(1, len(total_layers)):

    layers = total_layers[:i]

    mdl_builder = model_builder(layers, final_layer)

    sk_model = KerasClassifier(build_fn=mdl_builder, epochs=20, batch_size=16, verbose=1)

    cv_scores = cross_val_score(sk_model, X_train.values, y_train.values,
                                scoring = 'neg_log_loss')

    cv_mean_scores.append(np.mean(cv_scores))

# Put a new line in and then print out mean cv scores
# Each score is the CV score for a network with different layers
print("\n")
print(cv_mean_scores)
```