Building your First Neural Network on a Structured Dataset (using Keras)

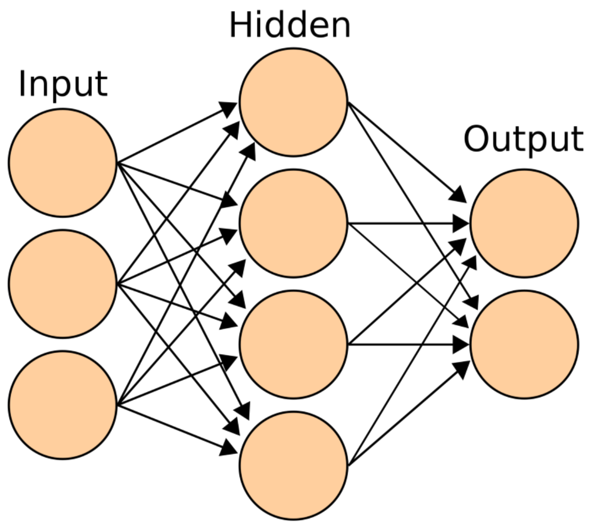
Sunil Ray Follow
Aug 20, 2018 · 8 min read

Introduction

Have you ever applied a neural network model on a structured dataset? If the answer is no, which of the following reasons are applicable for you?

1. It is very complex to apply
2. Neural Network is good for unstructured datasets like image, audio, and text and it does not perform well on structured datasets
3. It is not as easy as building a model using scikit-learn/caret
4. Training time is too high
5. Requires high computational power

In this article, I will focus on the first three reasons and showcase how easily you can apply a neural network model on a structured dataset using a popular high-level library - "keras".



Understand the problem statement

We will work on the Black Friday dataset in this article. It is a regression challenge where we need to predict the purchase amount of a customer against various products. We have been provided with various information about customer demographics (age, gender, marital status, city_type, stay_in_current_city) and product details (product_id and product category). Below is the data dictionary:

Variable	Definition
User_ID	User ID
Product_ID	Product ID
Gender	Sex of User
Age	Age in bins
Occupation	Occupation (Masked)
City_Category	Category of the City (A,B,C)
Stay_In_Current_City_Years	Number of years stay in current city
Marital_Status	Marital Status
Product_Category_1	Product Category (Masked)
Product_Category_2	Product may belongs to other category also (Masked)
Product_Category_3	Product may belongs to other category also (Masked)
Purchase	Purchase Amount (Target Variable)

The evaluation metric for this challenge is theRoot Mean Squared Error (RMSE).

Pre-requisite

In this article, I will solve this Black-Friday challenge using a Random Forest (RF) model using scikit-learn and a basic Neural Network (NN) model using keras. The idea of this article is to show how easily we can build an NN model on a structured dataset (it is similar to building a RF model using the scikit-learn library). This article assumes that you have a fair background of building a Machine Learning model, scikit-learn, and the basics of Neural Network. If you are not comfortable with these concepts, I would recommend going through the below articles first:

1. Introduction-deep-learning-fundamentals-neural-networks
2. Understanding-and-coding-neural-networks-from-scratch-in-python
3. Common Machine Learning Algorithms

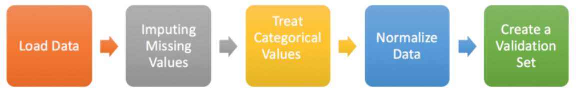
Approach to solve the problem using Machine (Deep) learning

I'm going to separate my approach broadly into four sub-sections:

1. Data Preparation
2. Model Building (Random Forest and Neural Network)
3. Evaluation
4. Prediction

Data Preparation

In this section, I will focus on basic data preparation steps like loading the dataset, imputing missing values, treating categorical variables, normalizing data and creating a validation set. I will follow the same steps for both the Random Forest and our NN model.



- **Load Data:** Here, I'll import the necessary libraries to load the dataset, combine train and test to perform preprocessing together, and also create a flag for the same.

```
#Importing Libraries for data preparation
import pandas as pd
import numpy as np


#Read Necessary files
train = pd.read_csv("train_black_friday.csv")
test = pd.read_csv("test_black_friday.csv")

#Combined both Train and Test Data set to do preprocessing
together # and set flag for both as well
train['Type'] = 'Train'
test['Type'] = 'Test'
fullData = pd.concat([train,test],axis=0)
```

- **Impute Missing Values:** Methods to treat missing values for categorical and continuous variables will be different.
  So, our first step would be identifying the ID column, target variable, categorical and continuous independent variables.
  Post this, we will create dummy flags for missing value(s). Why do this? Because sometimes missing values themselves can carry a good amount of information. Finally, we will impute the missing

values of continuous variables with the mean of that columns, and for the categorical variable, we will create a new level.

```
#Identifying ID, Categorical
ID_col = ['User_ID','Product_ID']
flag_col= ['Type']
target_col = ["Purchase"]
cat_cols=
['Gender','Age','City_Category','Stay_In_Current_City_Years'
]
num_cols= list(set(list(fullData.columns))-set(cat_cols)-
set(ID_col)-set(target_col)-set(flag_col))

# Combined numerical and Categorical variables
num_cat_cols = num_cols+cat_cols

#Create a new variable for each variable having missing
value with VariableName_NA
# and flag missing value with 1 and other with 0


for var in num_cat_cols:
    if fullData[var].isnull().any()==True:
        fullData[var+'_NA']=fullData[var].isnull()*1

#Impute numerical missing values with mean
fullData[num_cols] =
fullData[num_cols].fillna(fullData[num_cols].mean())

#Impute categorical missing values with -9999
fullData[cat_cols] = fullData[cat_cols].fillna(value =
-9999)
```

- **Treat Categorical Values:** We will create a label encoder for categorical variables.

```
#create label encoders for categorical features
from sklearn.preprocessing import LabelEncoder


for var in cat_cols:
    number = LabelEncoder()
    fullData[var] =
number.fit_transform(fullData[var].astype('str'))
```

- **Normalize Data:** Scale (normalize) the independent variables between 0 and 1. It will help us to converge comparatively faster.

```
features = list(set(list(fullData.columns))-set(ID_col)-
set(target_col))
fullData[features] =
fullData[features]/fullData[features].max()
```

- **Create a validation Set:** Here, we will segregate the train-test from the full dataset and remove the train-test flag from features list. While building our model, we have target values for the train dataset only, so we will create a validation set out of this train dataset for evaluating the model's performance. Here, I'm using *train_test_split* to divide the train dataset in training and validation in a 70:30 ratio.

```
#Creata a validation set
from sklearn.model_selection import train_test_split


train=fullData[fullData['Type']==1]
test=fullData[fullData['Type']==0]
features=list(set(list(fullData.columns))-set(ID_col)-
set(target_col)-set(flag_col))

X = train[features].values
y = train[target_col].values


X_train, X_valid, y_train, y_valid = train_test_split(X, y,
test_size=0.30, random_state=42)
```

Model Building using Random Forest

This part is fairly straightforward and I have written about this multiple times before. If you still want to review the random forest algorithm and its parameters, I would recommend going through this article: Tutorial on Tree Based Algorithms.

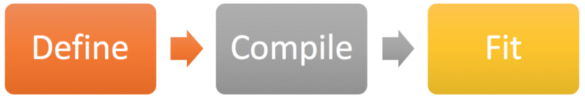
```
#import necessary libraries to build model


import random
from sklearn.ensemble import RandomForestRegressor


random.seed(42)
rf = RandomForestRegressor(n_estimators=10)
rf.fit(X_train, y_train)
```

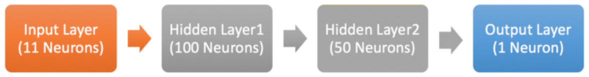
Model Building using Deep Learning Model (Keras)

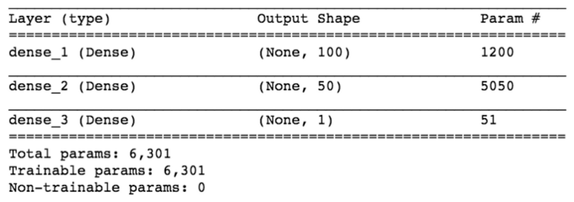
Here, I will focus on the steps to build a basic deep learning model. This will help beginners in creating their own models in the future. The steps to do this are:



- **Define Model:** For building a deep learning model, we need to define the layers (Input, Hidden, and Output). Here, we will go ahead with a sequential model, which means that we will define layers sequentially. Also, we will be going ahead with a fully connected network.
  1. First, we will focus on **defining the input layer**. This can be specified while creating the first layer with the *input dim* argument and setting it to 11 for the 11 independent variables.
  2. Next, **define the number of hidden layer(s)** along with the number of neurons and activation functions. The right number can be achieved by going through multiple iterations. Higher the number, more complex is your model. To start with, I'm simply using two hidden layers. One has 100 neurons and the other has 50 with the same activation function - "relu".
  3. Finally, we need to **define the output layer** with 1 neuron to predict the purchase amount. The problem in hand is a regression challenge so we can go ahead with a linear transformation at the output layer. Therefore, there is no need to mention any activation function (it is linear by default).

```
# Define model
model = Sequential()
model.add(Dense(100, input_dim=11, activation= "relu"))
model.add(Dense(50, activation= "relu"))
model.add(Dense(1))
model.summary() #Print model Summary
```





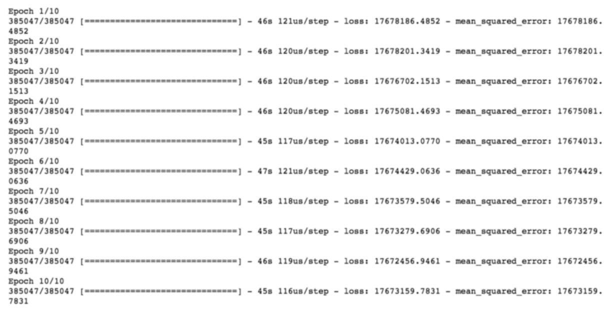
```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 100)               1200
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 51
=================================================================
Total params: 6,301
Trainable params: 6,301
Non-trainable params: 0
```

- **Compile Model:** At this stage, we will configure the model for training. We will set the optimizer to change the weights and biases, and the loss function and metric to evaluate the model's performance. Here, we will use "adam" as the optimizer, "mean squared error" as the loss metric. Depending on the type of problem we are solving, we can change our loss and metrics. For binary classification, we use "binary-crossentropy" as a loss function.

```
# Compile model
model.compile(loss= "mean_squared_error" , optimizer="adam",
metrics=["mean_squared_error"])
```

- **Fit Model:** Now, the final step of model building is fitting the model on the training dataset (which is actually 70% of the full dataset). We need to provide both independent and dependent variables along with the number of training iterations, i.e. epochs. Here, we have taken 10 epochs.

```
# Fit Model
model.fit(X_train, y_train, epochs=10)
```



```
Epoch 1/10
385047/385047 [==============================] - 46s 121us/step - loss: 17678186.4852 - mean_squared_error: 17678186.
4852
Epoch 2/10
385047/385047 [==============================] - 46s 120us/step - loss: 17678201.3419 - mean_squared_error: 17678201.
3419
Epoch 3/10
385047/385047 [==============================] - 46s 120us/step - loss: 17676702.1513 - mean_squared_error: 17676702.
1513
Epoch 4/10
385047/385047 [==============================] - 46s 120us/step - loss: 17675081.4693 - mean_squared_error: 17675081.
4693
Epoch 5/10
385047/385047 [==============================] - 45s 117us/step - loss: 17674013.0770 - mean_squared_error: 17674013.
0770
Epoch 6/10
385047/385047 [==============================] - 47s 121us/step - loss: 17674429.0636 - mean_squared_error: 17674429.
0636
Epoch 7/10
385047/385047 [==============================] - 45s 118us/step - loss: 17673579.5046 - mean_squared_error: 17673579.
5046
Epoch 8/10
385047/385047 [==============================] - 45s 117us/step - loss: 17673279.6906 - mean_squared_error: 17673279.
6906
Epoch 9/10
385047/385047 [==============================] - 46s 119us/step - loss: 17672456.9461 - mean_squared_error: 17672456.
9461
Epoch 10/10
385047/385047 [==============================] - 45s 116us/step - loss: 17673159.7831 - mean_squared_error: 17673159.
7831
```

Evaluation

Now that we have built the model using Random Forest and Neural Network techniques, the next step is to evaluate the performance on the validation dataset for both the models.

- **Evaluation for Random Forest Model:** We will get the predictions on the validation dataset and do the evaluation with actual target values (y_valid). We get the root mean squared error as ~3106.

```
from sklearn.metrics import mean_squared_error
pred=rf.predict(X_valid)
score = np.sqrt(mean_squared_error(y_valid,pred))
print (score)
3106.5008973291074

# Evaluation while fitting the model
model.fit(X_train, y_train, epochs=10, validation_data=
(X_valid, y_valid))
```

- **Evaluation for Neural Network Model:** Similarly, we will get the predictions on the validation dataset using the neural network model and calculate the root mean squared error. RMSE with a basic NN model comes out to be ~4214. This is a fairly basic model, you can go ahead and tune the hyper-parameters to build a more complex network. *You can pass validation data as an argument while fitting the NN model as well to look at the validation score after each epoch.*

```
pred= model.predict(X_valid)
score = np.sqrt(mean_squared_error(y_valid,pred))
print (score)
4213.954523194906
```

Prediction

After evaluating the model and finalizing the model parameters, we can go ahead with the prediction on the test data. Below is the code to do this using both random forest and NN models.

```
#Select the independent variables for test dataset
X_test = test[features].values
```

```
#Prediction using Random Forest
y_test_rf = rf.predict(X_test)

#Prediction using Neural Network
y_test_nn = model.predict(X_test)
```

What’s Next

The idea of this article was to show how easily we can build an NN model on a structured dataset so we have not focused on other aspects of improving the model’s predictions. Below is my list of ideas which you can apply to build on the neural network:

- Impute missing values after looking at the variable to variable relationships
- Feature Engineering (Product Ids may have some information about the purchase amount)
- Select right hyper-parameters
- Build a more complex network by adding more hidden layers
- Use regularization
- Train for more number of epochs
- Take Ensemble of both RF with NN model

Summary

In this article, we have discussed the different stages of model building like data preparation, model building, evaluation and finally prediction. We also looked at how we can apply a neural network model on a structured dataset using keras.

We further learned about different steps of model building in keras like defining, compiling, and fitting the model. Performance of a simple NN model was impressive and we can improve on that after making it into a more complex network. In the next article, we will look at how we can apply a RNN model on a time series challenge.

If you have any questions or suggestions, please feel free to connect with me in the comments section below.