**My Third Blog**

This blog post shows how a TensorFlow API can be used to build a model.

Google’s TensorFlow team is spending lots of efforts on developing easier-to-use TensorFlow APIs. This example uses TF Estimator to build a model to predict housing prices .

**Introduction to TensorFlow**

TensorFlow is an open source platform for machine learning. TensorFlow derives its name from the multidimensional arrays or lists known as tensors, which are used by the neural networks for different operations. In TensorFlow, a model is represented as a dataflow graph where nodes represent set of operations and the edges that connect the nodes in a graph represents multidimensional array(vector or matrix), what is known as tensors. These arrays with different dimensions and ranks go as input to the network. According to Google compared to many high-level libraries and APIs, TensorFlow is faster, smarter and more flexible and can be easily adaptable to new areas and products. TensorFlow is available on different operating systems such as Linux, Windows, MacOS and also on mobile operating platforms like iOS and Android. One of the salient features of TensorFlow is that it is capable of running on multiple CPUs and GPUs.

As a Google product it includes a variety of machine learning and deep learning algorithms. TensorFlow can train and run deep neural networks for handwritten digit classification, image recognition, word embedding and many more. Although TensorFlow has many areas of operations, its main area of research and development has been in the applications of Deep Neural Networks (DNN).

**TensorFlow Basics**

Tensors are the central part of TensorFlow. As discussed above tensors and the operations are connected to each other in a computational graph. There are certain programming elements in TensorFlow that are essential for writing any TensorFlow code like constants, variables, placeholder and session. Each element has its own syntax and functionalities. Where constants, variables and placeholders are used to assign and store values, a session is used to run a computational graph.

The below section explains some of operations using TensorFlow.

*Constants and use of session*

import tensorflow as tf

a = tf.constant(5)

b=tf.constant(4)

result = tf.add(a,b,name='add\_a\_b')

#To evaluate result and get the output we have to run the code under a 'session'. First create an instance of a session object from tf.Session class.

sess = tf.Session()

sess.run(result)

**result : 9**

*Variables and placeholders*

There are two main types of tensor objects in a Graph.

* Variables
* Placeholders

*Variables*

During optimization process variables can hold the values of weights and biases throughout the session. Variables need to be initialized. Variables are need to specify initial value and the data type.

*Placeholder*

Placeholders are initially empty and are used to feed in the training time. We need to specify the shape and datatype to the placeholder. A ‘shape = None’ indicates that the placeholder can get any arbitrary value. Initially values are not assigned to a variable it feed through the ‘feed\_dict’ argument. In the following program I have used TensorFlow’s global\_variables\_

intializer() to initialize the variable.

import tensorflow as tf

# Use of tf.global\_variables\_initializer()

my\_var = tf.random\_uniform((4,4),0,1)

make\_var = tf.Variable(initial\_value=my\_var)

init = tf.global\_variables\_initializer()

sess = tf.Session()

sess.run(init)

sess.run(make\_var)

**The result shows**

array([[0.4332559 , 0.36361885, 0.01518774, 0.26752925], [0.8444054 , 0.13574994, 0.07237113, 0.43475437],[0.7892722 , 0.30322468, 0.65823495, 0.11253977], [0.82987416, 0.77820754, 0.91608214, 0.80677783]],

dtype=float32)

**Linear Regression**

Linear Regression is a very common statistical method and the most basic forms of the machine learning algorithm that is used to predict numerical values. The following paragraph shows how to use TensorFlow’s variable, placeholder and session to find a best fit line which forms a linear separation.

For example, for given some data points of x and corresponding y we need to learn the relationship between them. In a simple dataset with only one feature and one output to predict, the form of the equation can be y = w\*x + b. From the equation w is a vector called weights

and b is a scalar called bias. The Weights and Bias are called the parameters of the model.

**Full code is available on GitHub…..**

import numpy as np

import tensorflow as tf

x\_batch = np.linspace(0,2,100)

y\_batch = np.linspace(0,2,100)

# Now create our model by creating placeholders x and y, so that we can feed our #training examples x and y into the optimizer during the training process.

X = tf.placeholder("float")

Y = tf.placeholder("float")

………………

………………..

**The result shows**

Epoch 850 : cost = None W = 0.7997131 b = 0.53510934

Epoch 900 : cost = None W = 0.7943789 b = 0.5284129

Epoch 950 : cost = None W = 0.78932154 b = 0.52195704

Epoch 1000 : cost = None W = 0.78452885 b = 0.51573145

**Introduction to TF Estimator**

As discussed above, TensorFlow is an open source deep learning framework for researchers and developers. Because of its flexible tools, libraries and resources it lets developers to easily build and deploy ML applications. So as compared to TensorFlow, TF Estimators stores a bunch of pre-made models from TensorFlow. Some high level APIs of TensorFlow are TF Learn, TF Slim, Sonnet, PrettyTensor and Keras.

Yes coming back to Estimators, it is another high level of TensorFlow API that simplifies the ML process by encapsulating four main functionalities for training, evaluation, prediction and export for serving. There are two types of Estimators: 1. *Premade Estimators* and 2. *Custom Estimators*. one can write his own [custom Estimators](https://www.tensorflow.org/guide/custom_estimators) or use pre-built estimators to build a model.

According to the TensorFlow documentation, the difference between the premade and custom Estimator is on the model function. An estimator needs to get the data from the input function.

Estimators keep the model functions. It implements the model. It defines the method or tell how to train the model or what will be the behaviour during evaluation or export time. The difference between working with pre-made Estimators and custom Estimators is:

* With pre-made Estimators, someone already wrote the model function.
* With custom Estimators, one must write the model function.

**Build model function**

To create a model function, the function requires two parameters as *features and labels.* In the program the *input function* is the model function. The input function return a *tuple of features and labels*. Features are a dictionary of feature column names and maps them to the tensor. Labels are an array of values. The structure of an input function given below:

Data

Input function

Estimator

Train

Evaluate

Predict

For tf.estimator we can take input from numpy array(say numpy\_input\_fn) or pandas dataframe input function(say pandas\_input\_fn). pandas\_input\_fn has many parameters but for input function the model uses ( x: as the number of features, y: as the labels, batch\_size : a number specifying the batch size, shuffle : whether shuffle data or not, num\_epoch : number of times to run training data). pandas\_input\_fn returns an input function to the tensorflow model. Here we create two input functions. One is for training and the other one is for prediction that takes the training and test set features and labels respectively.

﻿#Create input function for training....

input\_func = tf.estimator.inputs.pandas\_input\_fn(x=x\_train, y = y\_train, batch\_size=10,num\_epochs=5, shuffle=True)

﻿# Create input function for predicting/ testing....

pred\_input\_func = tf.estimator.inputs.pandas\_input\_fn(x=x\_test, batch\_size=len(x\_test), shuffle=False)

In this model, the focus is mainly on dealing with pre-made estimators. The advantage of using pre-made estimators are:

* It handles the implementation details
* Not worrying about creation of session, initializing variables or other low-level details
* Experiment with different model architectures with minimal code exchange, for example tf.estimator.DNNClassifier trains classification model based on dense, feed-forward neural networks.
* Easy to experiment with new features.

**Premade Estimators**

Some of pre-built estimators of TensorFlow are:

* tf.contrib.learn.KMeansClustering
* tf.contrib.learn.DNNClassifier
* tf.contrib.learn.DNNRegressor
* tf.contrib.learn.DNNLinearCombinedRegressor
* tf.contrib.learn.DNNLinearCombinedClassifier
* tf.contrib.learn.LinearClassifier
* tf.contrib.learn.LinearRegressor
* tf.contrib.learn.LogisticRegressor

**Workflow in build a TensorFlow pre-made estimators**

1. Loading the libraries and datasets.
2. Split the data into train and test set.
3. Introducing Feature Columns

* Numeric feature columns
* Categorical feature columns

1. Build Input Function
2. Instantiate the model
3. Train the model
4. predict the model

**Census data classifier using tf.estimator**

We’re going to build a census data classifier using tf.estimator. The dataset we are using is census data set which is having fourteen features are *age, workclass, education, education\_*

*num ,marital\_status, occupation, relationship, race, gender, capital\_gain, capital\_loss, hours\_per\_week, native\_country, income\_bracket*.

**Let’s start coding…Full code is available on**

1. Loading the libraries and datasets

# First import TensorFlow and the libraries the program need

import tensorflow as tf

import pandas as pd

import tensorflow.compat.v1 as tf

census = pd.read\_csv('/Users/chinu/myworkspace/census\_data.csv')

census.head()

census['income\_bracket']

def label\_fix(label):

if label.strip() == '<=50K':

return 0

else:

return 1

census['income\_bracket'] = census['income\_bracket'].apply(label\_fix)

census.head()

1. Split the data into train and test set

from sklearn.model\_selection import train\_test\_split

x\_data = census.drop(['income\_bracket'], axis=1)

y\_label = census['income\_bracket']

x\_train, x\_test,y\_train,y\_test = train\_test\_split(x\_data, y\_label,test\_size = 0.3,random\_state=101)

1. Introducing Feature Columns

***Define the columns in data that contain features.***

When we build an Estimator model, we pass it a list of feature columns that describe each of the features. TensorFlow provides many types of feature columns. Among all those we will use..

* + 1. Numeric columns - It is used to represent real valued features.
    2. Categorical columns - Data represented as string. Strings cannot directly feed to a model. Instead, we must first map them to numeric values. We use 'categorical\_column\_with\_vocabulary\_list' where categorical vocabulary columns provide a way to represent strings as a one-hot vector.
    3. Hashed feature columns - Another way to represent a categorical column with a large number of values is to use a 'categorical\_column\_with\_hash\_bucket'. This feature column calculates a hash value of the input, then selects one of the

hash\_bucket\_size buckets to encode a string.

# Categorical columns

marital\_status = tf.feature\_column.categorical\_column\_with\_hash\_bucket("marital\_status",

hash\_bucket\_size=1000)

relationship = tf.feature\_column.categorical\_column\_with\_hash\_bucket("relationship",

hash\_bucket\_size=1000)

occupation = tf.feature\_column.categorical\_column\_with\_hash\_bucket("occupation",

hash\_bucket\_size=1000)

workclass = tf.feature\_column.categorical\_column\_with\_hash\_bucket("workclass",

hash\_bucket\_size=1000)

education = tf.feature\_column.categorical\_column\_with\_hash\_bucket("education",

hash\_bucket\_size=1000)

native\_country = tf.feature\_column.categorical\_column\_with\_hash\_bucket("native\_country", hash\_bucket\_size=1000)

# Numerical Columns.

age = tf.feature\_column.numeric\_column('age')

education\_num = tf.feature\_column.numeric\_column('education\_num')

capital\_gain = tf.feature\_column.numeric\_column('capital\_gain')

capital\_loss = tf.feature\_column.numeric\_column('capital\_loss')

hours\_per\_week = tf.feature\_column.numeric\_column('hours\_per\_week')

feat\_cols =[gender,marital\_status,relationship,occupation,workclass,education,native\_country,

age,education\_num,capital\_gain,capital\_loss,hours\_per\_week]

1. Build Input Function

input\_func = tf.estimator.inputs.pandas\_input\_fn(x=x\_train, y = y\_train, batch\_size=10,

num\_epochs=5, shuffle=True)

1. Instantiate the model

model = tf.estimator.LinearClassifier(feature\_columns = feat\_cols)

1. Train the model

model.train(input\_fn=input\_func, steps=5000)

1. Predict the model

pred\_input\_func = tf.estimator.inputs.pandas\_input\_fn( x=x\_test, batch\_size

=len(x\_test), shuffle=False)

pred\_gen = model.predict(input\_fn = pred\_input\_func)

prediction = list(pred\_gen)

final\_pred = [pred['class\_ids'][0] for pred in prediction]

from sklearn.metrics import classification\_report

print(classification\_report(y\_test,final\_pred))

**Results**

The output shows an accuracy of 82% which is not so bad.

**Next steps**

* The model can be run on different datasets.
* Model can try with some other premade estimators.
* Create own custom estimator.

**References**

* <https://www.tensorflow.org/get_started/estimator>
* <https://www.tensorflow.org/get_started/input_fn>
* <https://medium.com/learning-machine-learning/introduction-to-tensorflow-estimators-part-1-39f9eb666bc7>
* <https://medium.com/google-developer-experts/demystify-the-tensorflow-apis-57d2b0b8b6c0>