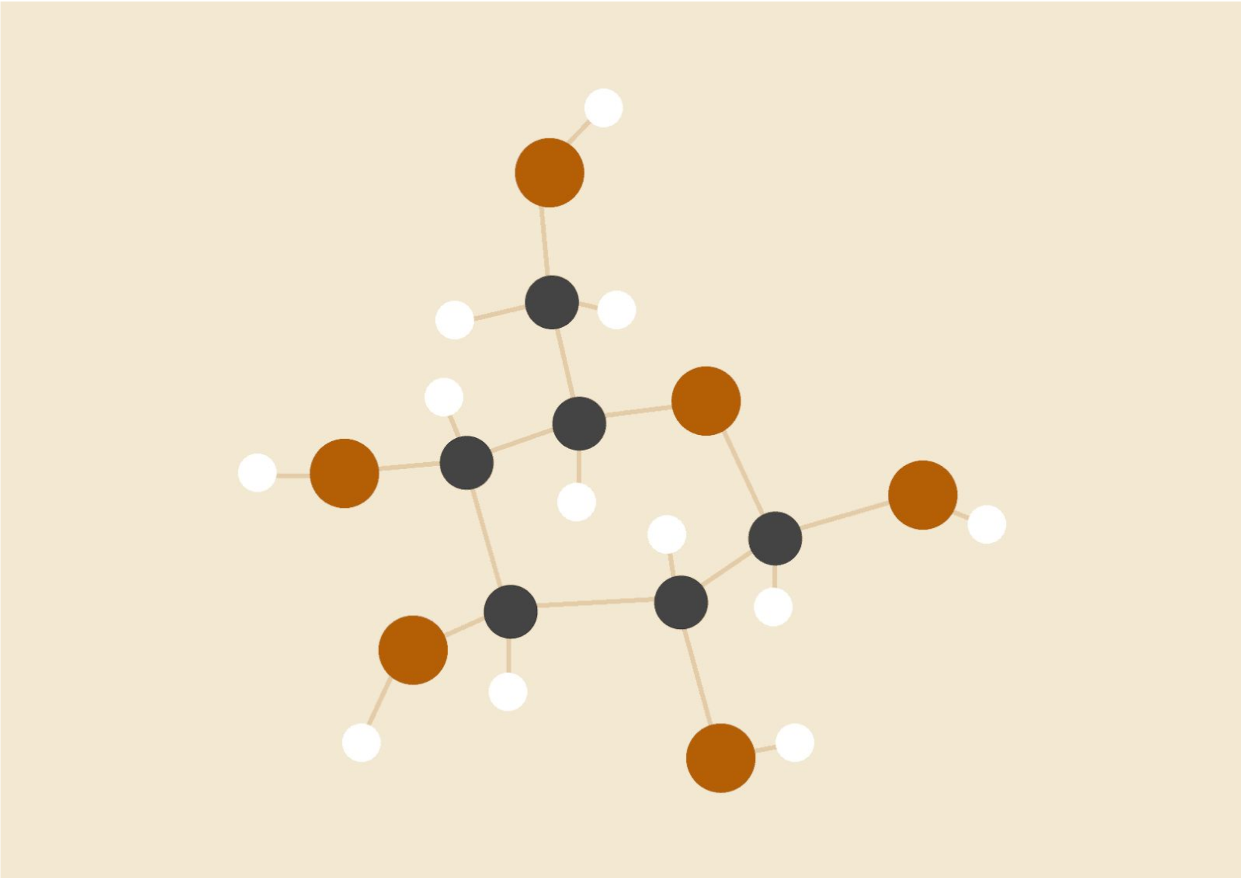
Project Report

T1- Lab Exam 1



Sumeet Padavala	01FB15ECS315
Ankit Anand	01FB15ECS041
Abhijay Gupta	01FB15ECS005

INTRODUCTION

The goal of this assignment was to create a custom estimator, and to use it to make certain predictions and analyses on the given dataset.

The Assignment focuses on building custom estimator which is similar to a pre-made DNNRegressor estimator. The problem also focused on implementing 2 more additional functions - get_layer_params() and get_layer_activations() which provide any layer's weights, biases by get_layer_params() function and also provide activation functions for the particular layer using get_layer_activations() by providing layer's id or layer's name

ESTIMATOR

The estimator we have designed implements a very basic neural network with a single hidden layer, and was created by subclassing the `tf.estimator.Estimator` class.

CREATION

At instantiation, the estimator accepts a few features:

1. `feature_columns`: A list of feature columns, similar to other estimators
2. `outputs`: The number of output logits per input
3. `hidden_units`: The number of neurons in the hidden layer (default: 2)

In addition, any other parameters are captured in a `kwargs` parameter, and passed as-is to the underlying `tf.estimator.Estimator` constructor. This allows for control various traditional Estimator features like manipulating the model directory.

MODEL FUNCTION

The model function is defined as a member function of the class, and is responsible for generating and returning output nodes for a computational graph appropriate to the requested operation.

The function first creates the model :

1. An input layer created from the given `feature_columns.`
2. A hidden layer with `hidden_units` neurons.
3. An output layer with `outputs` logits.

After this, the model function works differently based on the mode of operation.

If the requested mode is `PREDICT`, the function returns a reference to the the output layer in a "predictions" dictionary.

In both of the `TRAIN` and `EVALUATE` modes, the model first creates a loss operation using mean-squared error.

In `TRAIN` mode, an additional training operation is generated using the Adagrad optimizer.

LAYER PARAMS

To extract the parameters of each layer, we have included a member function called `get_layer_params`. This function takes the name of a layer (either `hidden` or output) and returns the kernel and bias as a 2-tuple.

This function utilizes the convenience function `get_variable_value` built into the base Estimator class, which extracts the value of the variable from the latest checkpoint in the model's directory.

INPUT DATA

The data was provided as a CSV file with four numeric columns which we have named `x1`, `x2`, `y1`, and `y2`.

The input data is extracted and processed using the `tf.data` API. We have written three separate functions providing three separate views of the data. Each function returns a `tf.data.Dataset` instance created using the `tf.contrib.data.CsvDataset` class, with various successive operations to transform the data as necessary.

In all cases, the input is specified with a pair of numeric feature columns with the keys `x1` and `x2`. The label format differs as necessary.

1. `input_fn_y1()`: Produces a single column Y1 as the output label.
2. `input_fn_y2()`: Produces a single column Y1 as the output label.
3. `input_fn_both()`: Produces a list of values of Y1 and Y2 as the output label.

In addition, we have defined two higher-order functions to handle splitting the dataset into training and testing datasets:

1. `train_data(input_fn)`: Returns a function that returns the first 85% (rounded down) of the dataset returned by `input_fn`.
2. `test_data(input_fn):` Returns a function that returns the last 15% (rounded

down) of the dataset returned by `input_fn`.

Both of these operations are slightly expensive, since they go through a dataset an extra time to determine its size each time. Due to the small size of this dataset, we have found the performance penalty to be negligible. In real-world cases, we recommend that the dataset size be measured beforehand.

MODELS

We have created four separate models for each of the given cases:

1. d_y1: This model predicts y1, using 2 hidden units.
2. d_y2: This model predicts y2, using 2 hidden units.
3. d_both: This model predicts y1 and y2, using 2 hidden units.
4. d_both_large: This model predicts y1 and y2, using 3 hidden units.

DATA

model/layer	kernel	bias
d_y1/hidden	[[-0.58 2.15] [-0.24 -1.75]]	[0.00 0.27]
d_y1/output	[[0.18] [2.63]]	[-3.69]
d_y2/hidden	[[1.76 1.53] [-4.66 -3.85]]	[1.24 1.16]
d_y2/output	[[2.11] [1.46]]	[1.05]
d_both/hidden	[[0.55 1.85] [-0.05 -2.56]]	[-0.92 1.28]
d_both/output	[[2.21 -0.23] [2.72 2.36]]	[-3.76 0.92]
d_both_large/hidden	[[-0.03 2.23 -0.19] [-0.78 -2.40 -0.42]]	[-0.20 1.15 0.00]
d_both_large/output	[[0.45 -0.08] [2.49 1.44] [0.86 0.58]]	[-3.60 1.58]

model	loss
d_y1	33.03469
d_y2	1029.249
d_both	550.42
d_both_arge	556.4562

RESULTS

In the first case, we observed through trial and error a positive correlation between the output y1 and the square of the input x1, as well as a negative correlation between the output y1 and the square of the input x2. As a result, we believe the final equation to be:

$$y1 \sim x2^2 - x1^2$$

We were unable to determine coefficients for the equation.

In the second case, we found a direct proportionality between x1 and y2, and an inverse proportionality between x2 and y2, giving us a relation:

$$y2 \sim x1/x2$$

We were unable to determine an equation for the final case.

CHALLENGES FACED

Over the course of this assignment, we faced a number of challenges. Most were resolved, some are still uncertain.

1. We had some difficulty in correctly designing the Estimator's model function, primarily due to a fundamental misunderstanding in its role and the role of checkpointing. As it turns out, the model function is simply responsible for building a computation graph, and checkpoint saving/loading is handled later.
2. Splitting the dataset into the test and train sets was rather tricky, since the `tf.data` API has no native support for it, and we did not wish to create a pair of helper functions for each input function. In the end, our higher-order function approach is sufficient, but it is too expensive for large datasets, and completely useless for streaming and other dynamic data.
3. The given equation (activation * weight + bias) was not understandable. It is unclear how multiplying the activation of a neuron with its weights could result in any useful quantity. We have attempted to derive the requested equation f(x1,x2)=y1 and f(x1,x2)=y2 by a combination of analyzing proportionality and trial-and-error instead. Our results are inconclusive.

REFERENCES

1. Tensorflow official documentation (creating custom estimators) https://www.tensorflow.org/guide/custom_estimators
2. Google Developer Blog https://developers.googleblog.com/2017/12/creating-custom-estimators-in-tensorflow.html