Report : Approach and results

Sanketh Rangreji	01FB15ECS267
Shahid Ikram	01FB15ECS274
Sondhi Nishant	01FB15ECS298
Sumanth Rao	01FB15ECS314

We developed a class called MyDNN that wraps the custom estimator and training process. We built a custom estimator by creating our own model function that creates a simple neural network with two units in the input layer, two units in the hidden layer and a single regression unit in the final layer. Within the class we create an estimator model by passing this model function to tf.estimator.Estimator() function that returns an estimator model.

The inputs to the model were normalised and passed to the training and testing functions.

Calling the train function on an object of our class with relevant parameters(as shown below) will create an Estimator object, by making a call to the constructor with our model function and will store it in our model attribute.

```
def train(self, input_fn, steps):
    self.model = tf.estimator.Estimator(
        model_fn = self.my_dnn_regression_fn,
        model_dir = self.savepoint,
        params = {
            "feature_columns": self.feature_columns,
            "learning_rate": 0.01,
            "optimizer": tf.train.AdamOptimizer,
            "hidden_units": self.hidden_units,
        })

    return self.model.train(input_fn = input_fn, steps = steps)
```

To get the model weights and biases we made use of get_variable_values() with the appropriate arguments obtained from get_variable_names() . From this we obtain the weights and biases matrices for each layer.

```
def get_weights(self, layer_name):
    # print(self.model.get_variable_names())
    # print("LAYER : " + layer_name)
    # print(self.model.get_variable_value(layer_name + '/kernel'))

    return self.model.get_variable_value(layer_name + '/kernel')

def get_bias(self, layer_name):
    # print("LAYER : " + layer_name)
    # print(self.model.get_variable_value(layer_name + '/bias'))

    return self.model.get_variable_value(layer_name + '/bias')
```

We have for now only supported relu as the activation units. This can be easily supported by adding a list of activations that the user can provide as parameters for each hidden layer. In building the hidden layer we can simply say activation = activations[i] when running it in the for loop. Following is the code for computing the activations and corresponding results(a1*w1 + b1,etc):

```
def getActivation(self, x):
    wts = self.get_weights("Layer1")
    bias = self.get_bias("Layer1")

    # Dot product
    a = np.matmul(x, wts) + bias

    # RELU
    if(a[0][0] <= 0):
        a[0][0] = 0
    if(a[0][1] <= 0):
        a[0][1] = 0

    return a

def predict(self, x1, x2):
    wts = self.get_weights("output_layer").flatten()
    bias = self.get_bias("output_layer")
    x = np.array([x1, x2])
    x = x.reshape(1, 2)
    a = self.getActivation(x)

    a = a.flatten()
    return np.array([np.matmul(a, wts).sum()]) + bias
```

Results :

We applied a train test split of 85-15 % and we were able to get the following results on the test set after evaluation for the two output columns or y values on which we are supposed to train.

Result on using 3rd column as target attribute :

```
In [10]: #test the model with 15% split for the column 3 as target
         FILE = test2
         my_model1.evaluate(input_fn = lambda: my_input_fn(FILE, True, 8))

         INFO:tensorflow:Starting evaluation at 2018-09-18-18:18:20
         INFO:tensorflow:Restoring parameters from ./model1/model.ckpt-15000
         INFO:tensorflow:Finished evaluation at 2018-09-18-18:18:25
         INFO:tensorflow:Saving dict for global step 15000: global_step = 15000, loss = 0.0227103, rmse = 0.0532803

Out[10]: {'global step': 15000, 'loss': 0.022710284, 'rmse': 0.053280253}
```

Result on using 4th column as target attribute :

```
In [11]: #test the model with 15% split for the column 4 as target
         FILE = test1
         my_model2.evaluate(input_fn = lambda: my_input_fn(FILE, True, 8))

         INFO:tensorflow:Starting evaluation at 2018-09-18-18:18:27
         INFO:tensorflow:Restoring parameters from ./model2/model.ckpt-15000
         INFO:tensorflow:Finished evaluation at 2018-09-18-18:18:32
         INFO:tensorflow:Saving dict for global step 15000: global_step = 15000, loss = 22.9558, rmse = 1.69395

Out[11]: {'global step': 15000, 'loss': 22.955769, 'rmse': 1.6939514}
```

Result on changing number of hidden units to 3 and column 3 as target:

```
In [20]: #test the model with 15% split for the column 3 as target
         FILE = test2
         my_modelh3_1.evaluate(input_fn = lambda: my_input_fn(FILE, True, 8))

         INFO:tensorflow:Starting evaluation at 2018-09-18-18:25:08
         INFO:tensorflow:Restoring parameters from ./model3/model.ckpt-30000
         INFO:tensorflow:Finished evaluation at 2018-09-18-18:25:13
         INFO:tensorflow:Saving dict for global step 30000: global_step = 30000, loss = 0.0223335, rmse = 0.0528364

Out[20]: {'global step': 30000, 'loss': 0.02233352, 'rmse': 0.052836444}
```

Result on changing number of hidden units to 3 and column 4 as target:

```
In [21]: #test the model with 15% split for the column 4 as target
         FILE = test1
         my_modelh3_2.evaluate(input_fn = lambda: my_input_fn(FILE, True, 8))

         INFO:tensorflow:Starting evaluation at 2018-09-18-18:25:33
         INFO:tensorflow:Restoring parameters from ./model4/model.ckpt-30000
         INFO:tensorflow:Finished evaluation at 2018-09-18-18:25:37
         INFO:tensorflow:Saving dict for global step 30000: global step = 30000, loss = 21.7854, rmse = 1.6502

Out[21]: {'global step': 30000, 'loss': 21.785374, 'rmse': 1.6502036}
```