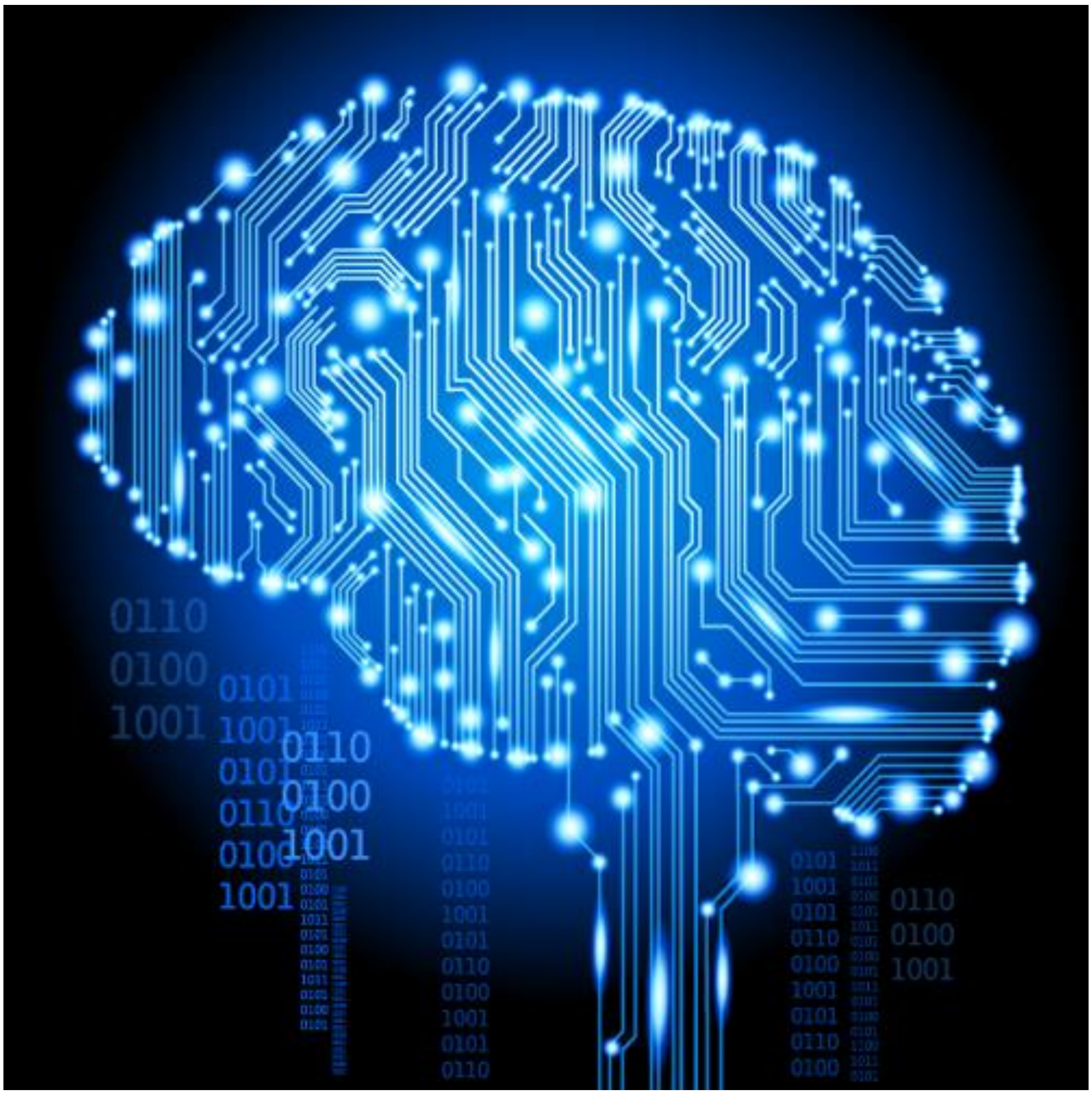
COMP3009 Machine Learning

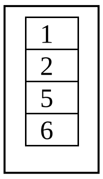
TensorFlow Introduction

Computer Based Coursework Manual – Autumn 2021

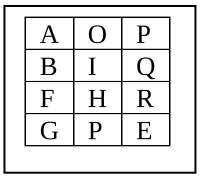


Introduction

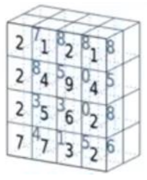
Tensors are multidimensional array which are extensions of two-dimensional matrices to data with higher dimensions and can be used to represent all types of data.



Tensor of dimension= 1



Tensor of dimension=2



Tensor of dimension=3

Tensorflow is a freely available library under Python which deals with tensors. Using this library, it is possible to build a multilayer perceptron from scratch or use available APIs to build CNN or RNN. The operations in Tensorflow library are called "**op node**" and are performed inside a graph which benefits from the followings.

- Makes it possible to benefit from parallel processing units such as GPUs or multiple CPUs and even mobile operating systems.
- The portability of the graph allows to preserve the computations for immediate or later use. The graph can be saved to be executed in the future.
- All the computations in the graph are done by connecting tensors together

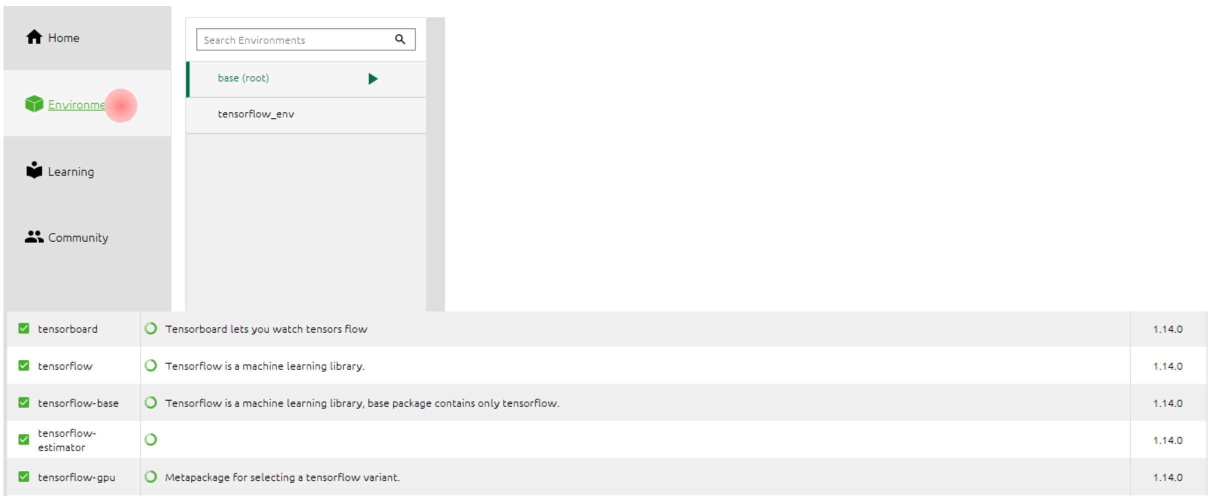
A tensor has a node and an edge. The node carries the mathematical operation and produces an endpoints output. The edges explain the input/output relationships between nodes. Currently, TensorFlow has a built-in API for:

- Linear regression: tf.estimator.LinearRegressor
- Classification:tf.estimator.LinearClassifier
- Deep learning classification: tf.estimator.DNNClassifier
- Deep learning wipe and deep: tf.estimator.DNNLinearCombinedClassifier
- Booster tree regression: tf.estimator.BoostedTreesRegressor
- Boosted tree classification: tf.estimator.BoostedTreesClassifier

Simple TensorFlow Example

```
import numpy as np
import tensorflow as tf
```

Most of available libraries under Python require installation and import to the code before they can be used in the program. It is possible to import libraries with a shorter name for brevity. In Anaconda environment, it is required to see if the tensorflow libraries shown in Fig. 1 are installed. If not you will need to install them before proceed.



✓ tensorboard	Tensorboard lets you watch tensors flow	1.14.0
✓ tensorflow	Tensorflow is a machine learning library.	1.14.0
✓ tensorflow-base	Tensorflow is a machine learning library, base package contains only tensorflow.	1.14.0
✓ tensorflow-estimator		1.14.0
✓ tensorflow-gpu	Metapackage for selecting a tensorflow variant.	1.14.0

Figure 1 Installation requirements under Anaconda for Tensorflow to work

The following lines describe how we can use TensorFlow to add two variables x_1 and x_2 together. It is required to design a graph with x_1 and x_2 and sum operator to perform such task. A TensorFlow session is then required to run the graph and calculate its output.

The nodes X_1 and X_2 are selected as to be placeholder node. The placeholder assigns a new value each time we make a calculation.

Step 1: Define the variable X_1 and X_2 as placeholders as follows.

```
X_1 = tf.placeholder(tf.float32, name = "X_1")
X_2 = tf.placeholder(tf.float32, name = "X_2")
```

The data type chosen for the nodes X_1 and X_2 are tf.float32. The nodes benefit from a name as well which will be further displayed during graphical visualizations of the model.

Step 2: Define the operation node

```
Y = tf.add(X_1, X_2, name = "Y")
```

Using this command a node is defined which performs addition operation between its input variables. We will pass in the X_1 and X_2 nodes to the addition node. It tells tensorflow to link those nodes in the computational graph, so we are asking it to pull the values from x and y and add the result. Let's also give the addition node the name Y. It is the entire definition for our simple computational graph.

Step 3: Execute the operation

To execute operations in the graph, we have to create a session which is done by tf.Session(). Having made the session, the computations are done using command run on the values of the parameters assigned for X_1 and X_2. The parameters are assigned using feed_dict. Consider the following example in which the values 1,2,3 and 4,5,6 are assigned to X_1 and X_2 and the addition operation are performed on them. The results are printed out. We should see 5, 7 and 9 as the results of the operation.

```
X_1 = tf.placeholder(tf.float32, name = "X_1")
X_2 = tf.placeholder(tf.float32, name = "X_2")
Y = tf.add(X_1, X_2, name = "Y")
with tf.Session() as session:
    result = session.run(Y, feed_dict={X_1:[1,2,3], X_2:[4,5,6]})
    print(result)
[ 5. 7. 9.]
```

Linear Regression Using TensorFlow

Linear Regression is a very common statistical method that allows us to learn a function or relationship from a given set of continuous data. For example, we are given some data points of x and corresponding y and we need to learn the relationship between them that is called a hypothesis. In case of Linear regression, the hypothesis is a straight line, i.e,

$h(x) = wx + b$

where w is a vector called Weights and b is a scalar called Bias. The Weights and Bias are called the parameters of the model.

All we need to do is estimate the value of w and b from the given set of data such that the resultant hypothesis produces the least cost J which is defined by the following cost function

$$J(w, b) = \frac{1}{2n} \sum_{i=1}^{n} \left(y_i - h(x_i)\right)^2$$

where *n* is the number of data points in the given dataset. This cost function is also called Mean Squared Error. For finding the optimized value of the parameters for which *J* is minimum, we will be using a commonly used optimizer algorithm called Gradient Descent. Following is the pseudo-code for Gradient Descent:

```
Repeat until convergence {
        $w = w - a\frac{\partial J}{\partial w}$
        $b = b - a\frac{\partial J}{\partial b}$
}
```

where α is called learning rate which tunes the convergence speed towards the extremum of the objective function.

For implementation, it is required to import data to perform the linear regression. However, here for education purpose we assume that we know the linear relationship between input and output of the linear function, and we use such a linear function to generate data.

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

In order to make the random numbers predictable, we will define fixed seeds for both Numpy and Tensorflow.

```
np.random.seed(101)
tf.set_random_seed(101)
```

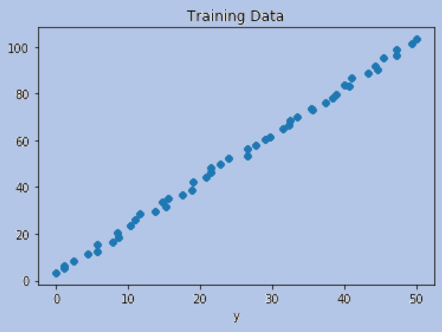
Now, let us generate some random data for training the Linear Regression Model.

```
# Genrating random linear data
# There will be 50 data points ranging from 0 to 50
x = np.linspace(0, 50, 50)
y = 2*x+3

# Adding noise to the random linear data
x += np.random.uniform(-1, 1, 50)
y += np.random.uniform(-1, 1, 50)
```

```
n = len(x) # Number of data points
Let us visualize the training data.
# Plot of Training Data
plt.scatter(x, y)
plt.xlabel('x')
plt.xlabel('y')
plt.title("Training Data")
plt.show()
```

Output:



Now we will start creating our model by defining the 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")
```

Now we will declare two trainable Tensorflow Variables for the Weights and Bias and initializing them randomly using np.random.randn().

```
W = tf.Variable(np.random.randn(), name = "W")
b = tf.Variable(np.random.randn(), name = "b")
```

Now we will define the hyperparameters of the model, the Learning Rate and the number of Epochs.

```
learning_rate = 0.01
training_epochs = 1000
```

Now, we will be building the Hypothesis, the Cost Function, and the Optimizer. We won’t be implementing the Gradient Descent Optimizer manually since it is built inside Tensorflow. After that, we will be initializing the Variables.

```
# Hypothesis
y_pred = tf.add(tf.multiply(X, W), b)

# Mean Squared Error Cost Function
cost = tf.reduce_sum(tf.pow(y_pred-Y, 2)) / (2 * n)

# Gradient Descent Optimizer
optimizer =
tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# Global Variables Initializer
init = tf.global_variables_initializer()
```

Now we will begin the training process inside a Tensorflow Session.

```
# Starting the Tensorflow Session
with tf.Session() as sess:

    # Initializing the Variables
    sess.run(init)

    # Iterating through all the epochs
    for epoch in range(training_epochs):

        # Feeding each data point into the optimizer using Feed
Dictionary
        for (_x, _y) in zip(x, y):
            sess.run(optimizer, feed_dict = {X : _x, Y : _y})

        # Displaying the result after every 50 epochs
        if (epoch + 1) % 50 == 0:
            # Calculating the cost a every epoch
            c = sess.run(cost, feed_dict = {X : x, Y : y})
            print("Epoch", (epoch + 1), ": cost =", c, "W =",
sess.run(W), "b =", sess.run(b))

    # Storing necessary values to be used outside the Session
    training_cost = sess.run(cost, feed_dict ={X: x, Y: y})
    weight = sess.run(W)
    bias = sess.run(b)
```

```
Output:
Epoch 50 : cost = 1.026182 W = 2.0337152 b = 1.6083914
Epoch 100 : cost = 0.98001426 W = 2.031029 b = 1.7395157
Epoch 150 : cost = 0.9434225 W = 2.0286198 b = 1.8571068
Epoch 200 : cost = 0.91447514 W = 2.0264595 b = 1.962564
Epoch 250 : cost = 0.8916278 W = 2.0245218 b = 2.0571349
Epoch 300 : cost = 0.8736412 W = 2.0227845 b = 2.1419451
Epoch 350 : cost = 0.8595225 W = 2.0212262 b = 2.2180045
Epoch 400 : cost = 0.84847975 W = 2.0198288 b = 2.2862182
Epoch 450 : cost = 0.8398797 W = 2.0185757 b = 2.3473935
Epoch 500 : cost = 0.8332128 W = 2.0174518 b = 2.4022524
Epoch 550 : cost = 0.8280766 W = 2.0164437 b = 2.4514508
Epoch 600 : cost = 0.8241466 W = 2.01554 b = 2.4955711
Epoch 650 : cost = 0.82116836 W = 2.0147293 b = 2.535138
Epoch 700 : cost = 0.8189347 W = 2.0140023 b = 2.5706244
Epoch 750 : cost = 0.8172847 W = 2.0133505 b = 2.602443
Epoch 800 : cost = 0.81608796 W = 2.0127656 b = 2.6309857
Epoch 850 : cost = 0.8152423 W = 2.0122414 b = 2.656576
Epoch 900 : cost = 0.8146687 W = 2.0117712 b = 2.6795323
Epoch 950 : cost = 0.8143 W = 2.0113497 b = 2.7001097
Epoch 1000 : cost = 0.8140866 W = 2.010971 b = 2.718578
```

Now let us look at the result.

```
# Calculating the predictions
predictions = weight * x + bias
print("Training cost =", training_cost, "Weight =", weight, "bias =",
bias, '\n')
```

```
Output:
```

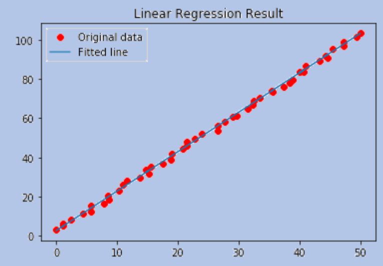
```
Training cost = 5.3110332 Weight = 1.0199214 bias = 0.02561663
```

Note that in this case both the Weight and bias are scalars. This is because, we have considered only one dependent variable in our training data. If we have m dependent variables in our training dataset, the Weight will be an m-dimensional vector while bias will be a scalar.

Finally, we will plot our result.

```
# Plotting the Results
plt.plot(x, y, 'ro', label ='Original data')
plt.plot(x, predictions, label ='Fitted line')
plt.title('Linear Regression Result')
plt.legend()
plt.show()
```

Output:



Further reading:

Please kindly consider referring to the books and webpages for more information and tutorials. The TensorFlow content of this manual is mainly taken from the following webpages.
https://www.guru99.com/linear-regression-tensorflow.html
https://www.guru99.com/what-is-tensorflow.html#1