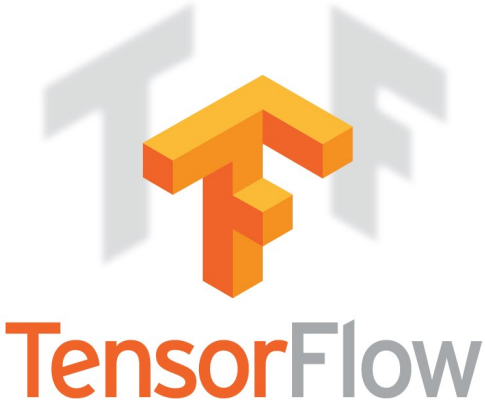
# Assignment 3: TensorFlow Neural Network Training Industrial Systems Design and Integration 282 772

Marc Alexander Sferrazza
12164165 *†




## Abstract

*In this documentation TensorFlow's neural network machine learning is used to find a function for y=f(x) given some inputs and outputs. The process involves using the Multi-Layered Perceptron (MLP) technique training and placeholders are used to provide input.*

*This work was not supported by any organization

†Faculty of Mechatronics Engineering, Massey University, Albany, Auckland, New Zealand `Progress of project: https://github.com/alex1v1a/Industrial-Systems-Design-and-Integration`

## Contents



## List of Figures

# 1. INTRODUCTION

TensorFlow is a free python language based library which is used for machine learning using neural networks to determine functions based on given inputs and outputs. As TensorFlow (TF) is supported by the community it makes as a favourable tool to learn, and is not only capable of black box systems, but also able to coordinate with complex systems with learning nodes.

In this documentation the program TensorFlow is used to train a neural network that will approximate an unknown function of y=f(x) for a given vector. Using Python 3.5+ syntax and TensorFlow 1.0+ syntax, a predefined data set csv file will be called on which to preform the learning task for an output range.

## 1.1. Installation

As TensorFlow uses Python, it supports multiple platforms including Mac, Windows, Linux, and Chrome OS. I have successfully installed Python 3.6 with a TensorFlow Python 3.5 CPU based workspace from source and have things compiled nicely; however for consistency with the lectures at Massey and to avoid any unnecessary errors I have completed these tasks using the advised programs with Visual Studio's CODE on Windows 10 OS. To setup and install TensorFlow simply follow the instructions on the TensorFlow website, or see the Getting Started with TensorFlow video on Frazer Nobles YouTube channel [1]

When setting up TensorFlow it features to automatically add to the environmental variables PATH, however the Python version and must be set to 3.5 manually (due to current version of 3.6), along with the option for CPU or GPU compiling. Visual Studios CODE is installed as the editor with a built in terminal cmd for quick compiling. Finally after setting up and installing TensorFlow with Python and CODE, a test program is written to check the status of the install with success.

```
'''Test program for TensorFlow'''

#pylint: disable=E0401
#pylint: disable=C0103

import tensorflow as tf

A = tf.placeholder(tf.float32, None, 'A')
B = tf.placeholder(tf.float32, None, 'B')

C = A + B

with tf.Session() as s:

    ans = s.run(C, {A: 2.5, B: 2.0})

    print(ans)
```

Figure 1: Test file to check the working status for TensorFlow install, Result = 4.5

# 2. METHOD

Using TensorFlows neural network training MLP method, an approximation can be made for the y=f(x) function. As discussed in the lectures to revise what was mentioned, the training method involves the input to segment blocks, where it is compared and examined with the relationship of output in a set shape space. The learning rate is the increment of which will define how well the relationship can be made with the smoothest function; this over iterations with a certain node size (mutations) and can give a higher accuracy with more reliable results of many iterations.

### 2.1. Multi-Layered Perceptron (MLP) Training

Multi-Layered Perceptron (MLP) is used..... and trained

Placeholders are used to provide input

Using this training method ....

### 2.2. Importing the CSV

A csv file, containing pairs of x and y values is read into the .....

### 2.3. Nodes

when you have a slow learning rate, you need more iterations,

but if you have a higher learning rate you waste alot of time getting stuck in local minimuns

lower learning rate, learns more finely...

The more node you have the more variations you have

The more different types of variations it will try

If you have 3 nodes you have

source code can be adapted to plot the given x and y values and the neural network's inferred output

### 2.4. Base Optimizer

## 3. RESULTS

Original Data set & Inferred Output is shown

An image illustrating the original data set and the inferred output of your program.

After successfully configuring the Multi-Layered Perceptron (MLP), reading in the values in the provided csv

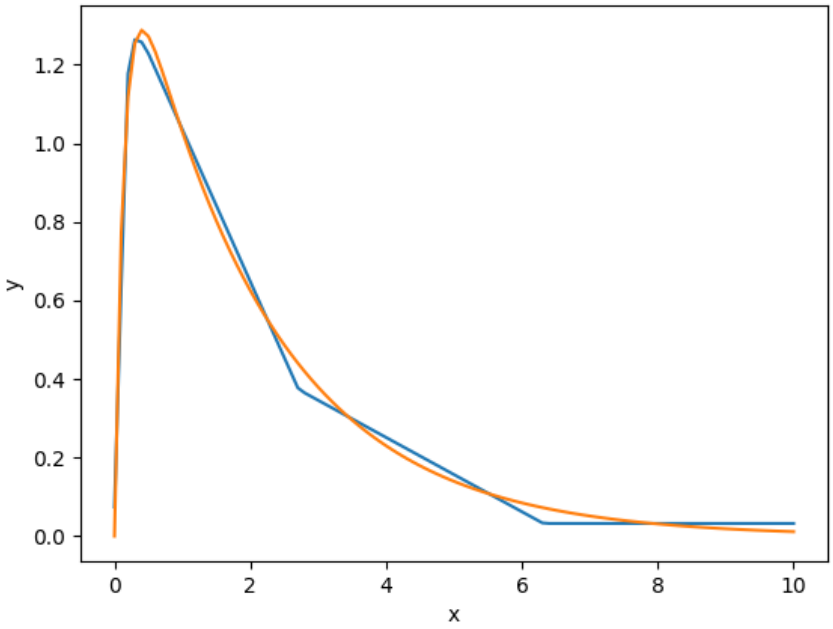



Figure 2: The final resultant is shown in compare on a graph

Once the initial results were found, to further the accuracy of the output the training of the MLP using the Adam Optimiser algorithum was used as suggested in class, please see variable for assigned learn rate in the code in the appendix. Below is the revised version using the Adam optimiser with specified nodes and learning rates as described in the logic documentation [2]. A clear difference between the optimiser reflects the accuracy for the learning of the function in this case.

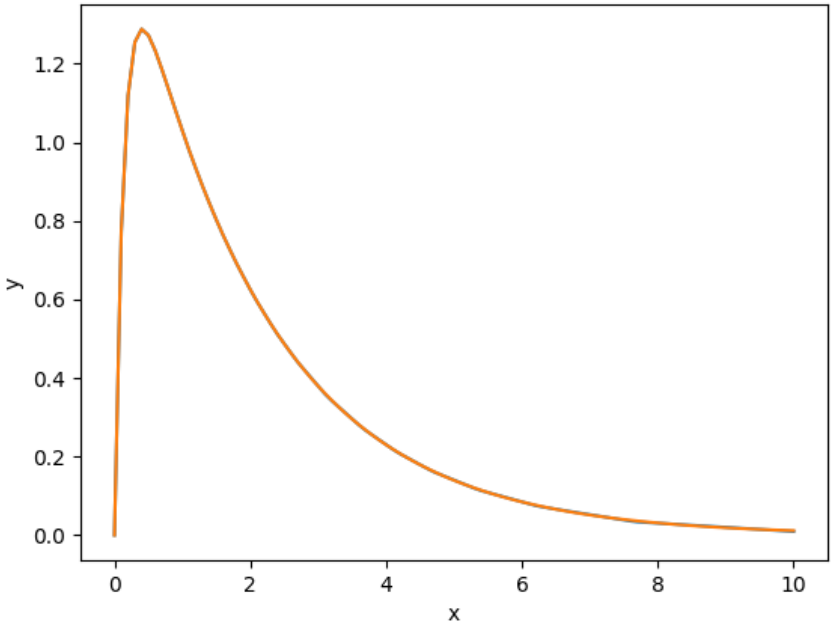



Figure 3: The final resultant is shown in compare on a graph

## 4. OUTCOMES

## 5. CONCLUSIONS

For any progress related to the report please see the public Github repo for alex1v1a or use the link in the cover page to be automatically redirected to this project. The repo provides all relative project information

I have come to a further understanding learned the fundamentals of TensorFlow with the use of such machine learning tools like Multi-Layered Perceptron using placeholders. This is a powerful tool and from the basic level of understanding is an strong aspect of the machine learning processing, it is a great skill to learn.

## References


[1] D. F. K. Noble, "Getting started with tensorflow." https://www.youtube.com/watch?v=Q-FF_0NAT3s. [Online; accessed 19-August-2017].

[2] A. D. Tensor Flow Training Tutorials, "Adam optimizer." https://www.tensorflow.org/api_docs/python/tf/train/AdamOptimizer. [Online; accessed 24-September-2017].

[3] D. F. K. Noble, "Assessment 3 resources - stream." http://stream.massey.ac.nz/mod/folder/view.php?id=2052029. [Online; accessed 14-August-2017].

[4] D. F. K. Noble, "Multi-layered perceptron (mlp) training." https://github.com/FKNoble/tensorflow_projects/blob/master/tutorials/tutorial_4.py. [Online; accessed 20-September-2017].

[5]

## APPENDIX

```
# pylint: disable=C0413
# pylint: disable=C0103
# pylint: disable=E0401

import os
import tensorflow as tf # TesnorFlow Librarys
import csv # For csv importing etc
import matplotlib.pyplot as plt # For graphing and plotting etc

# Adjustable Values, Nodes, Learning Rate, and epochy
nodeSize = 300 # More variation, more mutation
learnRate = 0.005 # Lower the learning rate the higher accuracy
iteration = 300000 # How many times it wil go through - epochy

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3' # Suppresses warnings

# Import csv file, assign coloums as inputX, inputY respectivly, no
    determined allocation size for memory
inputX = []
inputY = []
# Open and read in data_set.csv
with open('data_set.csv','r') as csvfile:
    plots = csv.reader(csvfile, delimiter=',')
    for row in plots:
    # Add each line of the csv to the row array for X and Y
        inputX.append([float(row[0])])
        inputY.append([float(row[1])])

# Placeholders are used, tensorflow diag 32 type with shape of nil, 1 -
    labeled x and y respectivly
x = tf.placeholder(dtype=tf.float32, shape=[None, 1], name='x')
y = tf.placeholder(dtype=tf.float32, shape=[None, 1], name='y')

# MLP
def mlp_layer(in_x, w_shape, b_shape):
    '''mlp_layer'''
  # Using the get variable function with the passed through same shapes and
      random initializer with no conditions
    W = tf.get_variable(name='W', shape=w_shape, dtype=tf.float32,
                        initializer=tf.random_uniform_initializer())
    b = tf.get_variable(name='b', shape=b_shape, dtype=tf.float32,
                        initializer=tf.random_uniform_initializer())
  # Assign the output of the added variables with shape b
    out_y = tf.add(tf.matmul(in_x, W), b)
  # Return the addition of the layer as passthrough
    return out_y

# Layer 1 computation
with tf.variable_scope('layer_1') as vs:
    h = mlp_layer(x, [1, nodeSize], [nodeSize])
    h = tf.nn.relu(h)
    vs.reuse_variables()

# Layer 2 computation
with tf.variable_scope('layer_2') as vs:
    y_ = mlp_layer(h, [nodeSize, 1], [1])
```

```
# Loss value for the mean squared error
loss = tf.losses.mean_squared_error(y, y_)
# Training of the MLP using the Adam Optimizer as suggested in class,
    please see variaible for assigned learn rate
training = tf.train.AdamOptimizer(learning_rate=learnRate).minimize(loss)

# We must initalize the tensorflow with both global and local
init = [tf.global_variables_initializer(), tf.local_variables_initializer()
    ]

# Begin the learing session after everything is setup and imported
with tf.Session() as s:
  # Run the initializer for the session
    s.run(init)

  # After evaluating the shape, this section was removed and shaped on the
      import of the csv
    # x1 = s.run(tf.reshape(inputX, [101,1]))
    # y2 = s.run(tf.reshape(inputY, [101,1]))

  # Begin the loop for the function of the assigned iteration steps.
    for i in range(0, iteration, 1):
    # Start the training
        l, _ = s.run([loss, training], feed_dict={x: inputX, y: inputY})

    # Only print the 10 cycles of iterations - epochy ~ 1st Jan 1970 lol...
        if i % (iteration/10) == 0:
      # Print the trained value..
            print(l)

  # Assign the learning points to use for the plot
    p = s.run(y_, {x:inputX})
  # Plot the learning points
    plt.plot(inputX,p)
  # Plot the origional csv values
    plt.plot(inputX, inputY)
  # Label Axis etc...
    plt.xlabel('x')
    plt.ylabel('y')
  # Show plot
    plt.show()
```