Experiment-6

Aim: - Evaluate the output of a single layer neural network of width 3 using Sigmoid activation function. Use the following two dimensional data

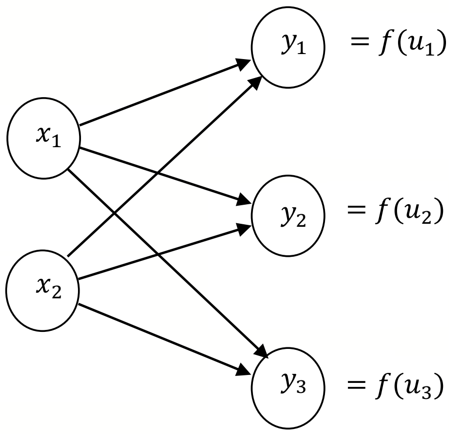
$x = (x_1, x_2)$
(0.5, -1.66)
(-1.0, -0.51)
(0.78, -0.65)
(0.04, -0.20)

Apparatus: -

- Laptop Configuration CPU, GPU, Core, clock etc.
- Laptop Configuration
    - Macbook
    - 8GB RAM
    - 8-Core CPU
    - 7-Core GPU
- Libraries Used: -
    - TensorFlow 1.40
    - NumPy
    - PyLab
    - Matplotlib
- Coding Environment: - Python 3.7.

Theory: -

$$\boldsymbol{u = W^T X + b}$$



$$\boldsymbol{where, f(u) = \frac{1}{1 + e^{-u}}}$$

Python CODE: -

```python
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import numpy as np
import pylab as plt
from mpl_toolkits.mplot3d import Axes3D

import os
if not os.path.isdir('figures'):
    os.makedirs('figures')

tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

no_features = 2
no_classes = 3

SEED = 10
np.random.seed(SEED)

# data
X = np.array([[0.5, -1.66],[-1.0, -0.51],[0.78, -0.65],[0.04, -0.20]])

# Model parameters
w = tf.Variable(np.random.normal(0., 0.1, (no_features, no_classes)),
dtype=tf.float32)
b = tf.Variable(0.1*np.random.rand(no_classes), dtype=tf.float32)

# Model input and output
x = tf.placeholder(tf.float32, X.shape)


u = tf.matmul(x, w) + b
y = tf.sigmoid(u)

print('x:{}'.format(X))

# training loop
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init) # reset values to wrong
w_, b_ = sess.run([w, b])
print('w: {}, b: {}'.format(w_, b_))

u_, y_ = sess.run([u, y], {x: X})
print('u:{}'.format(u_))
print('p:{}'.format(y_))
```

OUTPUT: -

```
x:[[ 0.5  -1.66] [-1.   -0.51] [ 0.78 -0.65] [ 0.04 -0.2 ]]
w: [[ 0.13315865  0.0715279  -0.15454003] [-0.00083839  0.0621336  -0.07200856]],
b: [0.01691108 0.00883398 0.06853598]
u:[[ 0.08488213 -0.05854384  0.11080017] [-0.11582    -0.09438206  0.25980037]
 [ 0.12131978  0.0242389  -0.00519968] [ 0.02240511 -0.00073162  0.07675609]]
p:[[0.5212078  0.4853682  0.5276717 ] [0.47107735 0.476422   0.5645872 ]
 [0.5302928  0.5060594  0.49870008] [0.50560105 0.4998171  0.5191796 ]]
```

Precautions: -

- Ensure that your system is connected to a stable power supply (if not using MAC).
- Make sure you have all the required Libraries.
- Save the code periodically.
- Don’t close the terminal if using Jupyter Notebook.

Experiment-7

Aim: - To train a linear regression model using a single layered neural network.

Apparatus: -

- Laptop Configuration CPU, GPU, Core, clock etc.
- Laptop Configuration
  - Macbook
  - 8GB RAM
  - 8-Core CPU
  - 7-Core GPU
- Libraries Used: -
  - TensorFlow 1.40
  - NumPy
  - PyLab
  - Matplotlib
- Coding Environment: - Python 3.7.

Theory: -

Deep Feed Forward Neural Network: - feedforward neural network consists of several layers of neurons where activation propagate from input layer to the output layer. the layer between the input and output layer are referred to as hidden layers.

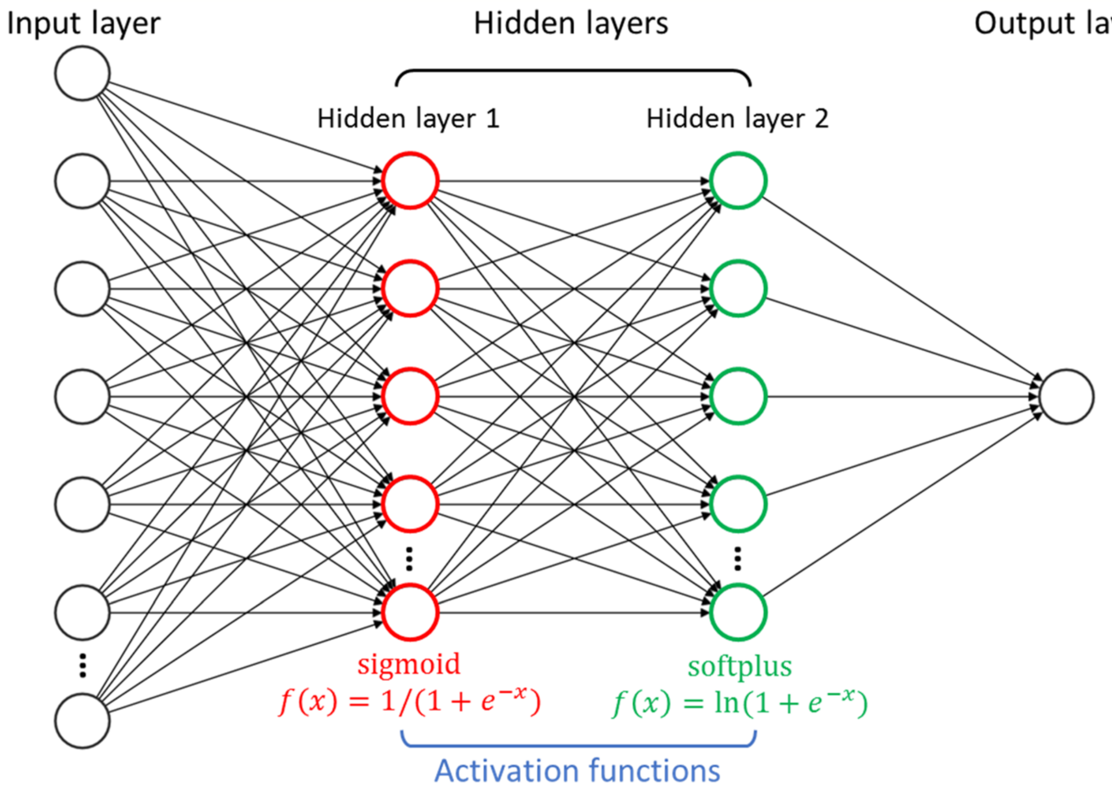
The number of layers is referred to as the depth of the feed forward network when a network has many hidden layers of neurons feed forward networks are referred to as deep feedforward neural network.

Learning in deep neural network is referred to as deep learning.

The number of neurons in a layer is referred to as the width of the layers.

The hidden layers are usually composed of perceptron (sigmoidal unit or relu units) are the output layer is usually

- a linear neuron for regression.
- a soft-Max layer for classification.



Depth =3
Width=n,m,k

$$d\epsilon R^k$$
$$y\epsilon R^k$$

Input Vector → $\boldsymbol{X = [x_1, x_2, x_3, \dots \dots \dots x_n]^T}$**.**
Output Hidden Layer → $\boldsymbol{h = [h_1, h_2, h_3, \dots \dots \dots h_m]^T}$**.**
Output Vector → $\boldsymbol{y = [y_1, y_2, y_3, \dots \dots \dots y_k]^T}$**.**

Back Propagation in a single pattern for 3 layer feedforward neural network.

$$x \xrightarrow{w,b} z \xrightarrow{g} h \xrightarrow{v,c} u \xrightarrow{f} y \longrightarrow J \longleftarrow d$$

Consider output layer
$\boldsymbol{\nabla_w J = -(d - \vec{y})}$ for output linear layer

From chain rule of differentiation

$$\boldsymbol{\nabla_h J = \frac{\partial u^T}{\partial h} * \nabla_u J}$$
$$\boldsymbol{= \mathrm{V}\nabla_h J}$$

Where $\boldsymbol{\frac{\partial u}{\partial h}}$ **is a jacobian matrix**

$$\boldsymbol{\frac{\partial u}{\partial h} = \begin{bmatrix} \frac{\partial u_1}{\partial h_1} & \cdots & \frac{\partial u_1}{\partial h_m} \\ \vdots & \ddots & \vdots \\ \frac{\partial u_k}{\partial h_1} & \cdots & \frac{\partial u_k}{\partial h_m} \end{bmatrix}}$$

$$\boldsymbol{V^T = \begin{bmatrix} V_{11} & \cdots & V_{1m} \\ \vdots & \ddots & \vdots \\ V_{k1} & \cdots & V_{km} \end{bmatrix}}$$

At. Hidden layer: -

$$\boldsymbol{\nabla_h J = \frac{\partial u^T}{\partial h} * \nabla_u J = \mathrm{V}\nabla_h J}$$

Where g(z)= $\boldsymbol{\frac{1}{1+e^{-z}}}$

Deep Learning back Propagation Algorithm: -

$\boldsymbol{for\ a\ given\ input\ patten\ \vec{X}, \vec{d}}$

1) $\boldsymbol{Set\ learning\ Rate\ \ \alpha}$
2) $\boldsymbol{initialize\ (w, b, v, c)}$
3) $\boldsymbol{epoc\ until\ the\ Convergence}$

$\boldsymbol{for\ each\ pattern\ in\ \vec{X}, \vec{d}}$

$$\boldsymbol{\vec{Z} = W^T\vec{x} + \vec{b}}$$

$$\vec{h} = g(\vec{z})$$
$$\vec{u} = V^T\vec{h} + \vec{c}$$
$$\vec{y} = f(\vec{u})$$
$$\nabla_w J = -(d - \vec{y})$$
$$V \leftarrow V - \alpha\vec{h}\,(\nabla_u J)^T$$
$$c \leftarrow c - \alpha(\nabla_u J)$$
$$W \leftarrow W - \alpha\vec{h}\,(\nabla_z J)^T$$
$$b \leftarrow b - \alpha(\nabla_z J)$$

Python CODE: -

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import numpy as np
import pylab as plt

import os
if not os.path.isdir('figures'):
    os.makedirs('figures')

tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

num_features = 2
num_labels = 2
num_hidden = 3

num_data = 8

lr = 0.05
num_iters = 20000

SEED = 10
np.random.seed(SEED)

# data
# generate training data
X = np.random.rand(num_data, num_features)
Y = 2*np.random.rand(num_data, num_labels) - 1

print('x:{}'.format(X))
print('y:{}'.format(Y))


# initialization routines for bias and weights
def init_bias(n = 1):
    return(tf.Variable(np.zeros(n), dtype=tf.float32))

def init_weights(n_in=1, n_out=1, logistic=True):
    W_values = np.asarray(np.random.uniform(low=-np.sqrt(6. / (n_in + n_out)),
```

```
high=np.sqrt(6. / (n_in + n_out)),
size=(n_in, n_out)))
    if logistic == True:
        W_values *= 4
    return(tf.Variable(W_values, dtype=tf.float32))

#Define variables:
V = init_weights(num_hidden, num_labels)
c = init_bias(num_labels)
W = init_weights(num_features, num_hidden)
b = init_bias(num_hidden)

# Model input and output
x = tf.placeholder(tf.float32, X.shape)
d = tf.placeholder(tf.float32, Y.shape)

z = tf.matmul(x, W) + b
h = tf.nn.sigmoid(z)
y = tf.matmul(h, V) + c


cost = tf.reduce_mean(tf.reduce_sum(tf.square(d - y),axis=1))


grad_u = -(d - y)
grad_V = tf.matmul(tf.transpose(h), grad_u)
grad_c = tf.reduce_sum(grad_u, axis=0)

dh = h*(1-h)
grad_z = tf.matmul(grad_u, tf.transpose(V))*dh
grad_W = tf.matmul(tf.transpose(x), grad_z)
grad_b = tf.reduce_sum(grad_z, axis=0)

W_new = W.assign(W - lr*grad_W)
b_new = b.assign(b - lr*grad_b)
V_new = V.assign(V - lr*grad_V)
c_new = c.assign(c - lr*grad_c)

# training loop
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init) # reset values to wrong
W_, b_ = sess.run([W, b])
print('W: {}, b: {}'.format(W_, b_))
V_, c_ = sess.run([V, c])
print('V:{}, c:{}'.format(V_, c_))

err = []
for i in range(num_iters):
```

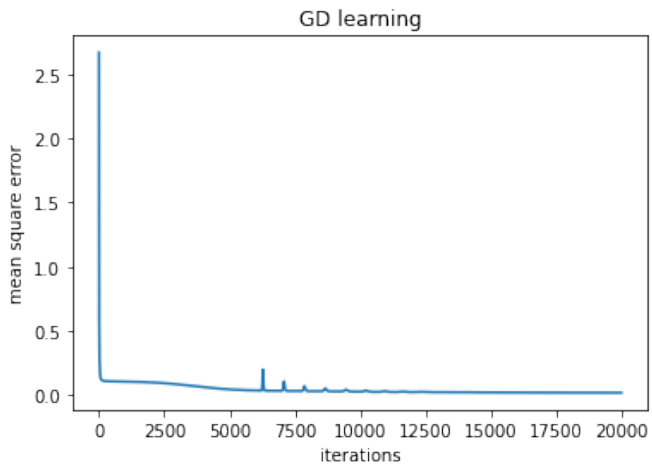
```
  if i == 0:
    z_, h_, y_, cost_, grad_u_, dh_, grad_z_, grad_V_, grad_c_, grad_W_, grad_b_ =
sess.run(
        [ z, h, y, cost, grad_u, dh, grad_z, grad_V, grad_c, grad_W, grad_b], {x:X,
d:Y})
    print('iter: {}'.format(i+1))
    print('z: {}'.format(z_))
    print('h: {}'.format(h_))
    print('y: {}'.format(y_))
    print('grad_u: {}'.format(grad_u_))
    print('dh: {}'.format(dh_))
    print('grad_z:{}'.format(grad_z_))
    print('grad_V:{}'.format(grad_V_))
    print('grad_c:{}'.format(grad_c_))
    print('grad_W:{}'.format(grad_W_))
    print('grad_b:{}'.format(grad_b_))
    print('cost: {}'.format(cost_))

  sess.run([W_new, b_new, V_new, c_new], {x:X, d:Y})
  cost_ = sess.run(cost, {x:X, d:Y})
  err.append(cost_)

  if i == 0:
    W_, b_, V_, c_ = sess.run([W, b, V, c])
    print('V: {}, c: {}'.format(V_, c_))
    print('W: {}, b: {}'.format(W_, b_))

  if not i%1000:
    print('epoch:{}, error:{}'.format(i,err[i]))


y_ = sess.run(y, {x: X})
print('y:{}'.format(y_))

# plot learning curves
plt.figure(1)
plt.plot(range(num_iters), err)
plt.xlabel('iterations')
plt.ylabel('mean square error')
plt.title('GD learning')
plt.savefig('figures/5.2_1.png')

# plot trained and predicted points
plt.figure(2)
plot_targets = plt.plot(Y[:,0], Y[:,1], 'b^', label='targeted')
plot_pred = plt.plot(y_[:,0], y_[:,1], 'ro', label='predicted')
plt.xlabel('$y_1$')
plt.ylabel('$y_2$')
plt.title('targets and predicted outputs')
plt.legend()
```

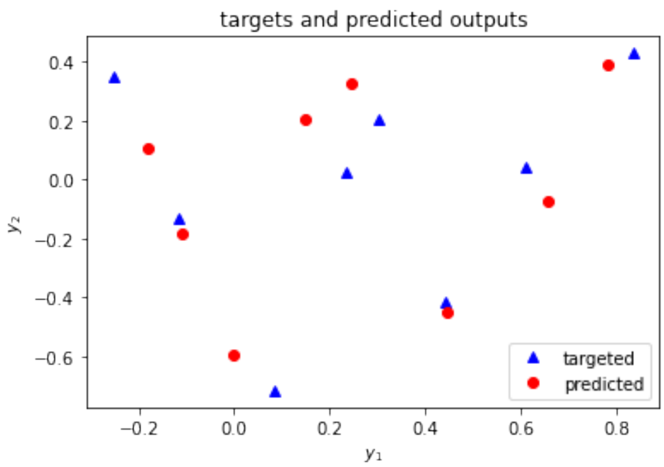
```
plt.savefig('./figures/5.2_2.png')

plt.show()
```

OUTPUT: -

```
x:[[0.77132064 0.02075195]
 [0.63364823 0.74880388]
 [0.49850701 0.22479665]
 [0.19806286 0.76053071]
 [0.16911084 0.08833981]
 [0.68535982 0.95339335]
 [0.00394827 0.51219226]
 [0.81262096 0.61252607]]
y:[[ 0.44351063 -0.41624786]
 [ 0.83554825  0.42915157]
 [ 0.08508874 -0.7156599 ]
 [-0.25331848  0.34826723]
 [-0.11633365 -0.13197201]
 [ 0.23553396  0.02627649]
 [ 0.30079436  0.20207791]
 [ 0.61044639  0.0432943 ]]
W: [[-3.9708016   1.1067251   0.41702417]
 [ 2.798091   -2.6382916   3.1272793 ]],b: [0. 0. 0.]
V:[[ 3.5812194 -1.5841355]
 [-3.5890346 -1.7465771]
 [-3.3828716  2.8804188]], c:[0. 0.]
iter: 1
z: [[-3.0046954   0.79889023  0.3865565 ]
 [-0.42087007 -1.2742885   2.6059654 ]
 [-1.350471   -0.0413689   0.9108914 ]
 [ 1.3415658  -1.7873006   2.460989  ]
 [-0.42432272 -0.04590698  0.34678656]
 [-0.0537467  -1.7568247   3.267339  ]
 [ 1.4174827  -1.3469429   1.6034148 ]
 [-1.5128529  -0.7166744   2.2544227 ]]
h: [[0.0472142  0.689737   0.5954535 ]
 [0.39630857 0.21852402 0.9312445 ]
 [0.20579337 0.48965925 0.71318257]
 [0.79274726 0.143404   0.9213613 ]
 [0.3954828  0.48852527 0.58583814]
 [0.48656657 0.14718847 0.96329117]
 [0.8049435  0.20637062 0.8324951 ]
 [0.18051638 0.32812572 0.9050313 ]]
y: [[-4.3207483   0.43568286]
 [-2.515303    1.6728985 ]
 [-3.4330177   0.8730323 ]
 [-0.79252726  1.1476213 ]
 [-2.3188386   0.20771378]
 [-2.0444534   1.7468184 ]
 [-0.6742163   0.7623527 ]
 [-3.5927906   1.7478099 ]]
grad_u: [[-4.764259    0.85193074]
 [-3.350851    1.243747  ]
 [-3.5181065   1.5886922 ]
 [-0.53920877  0.7993541 ]
 [-2.2025049   0.3396858 ]
 [-2.2799873   1.720542  ]
 [-0.97501063  0.5602748 ]
 [-4.203237    1.7045156 ]]
dh: [[0.04498502 0.21399987 0.24088863]
```

```
 [0.2392481  0.17077127 0.06402819]
 [0.16344245 0.24989307 0.20455319]
 [0.16429904 0.12283929 0.07245463]
 [0.23907617 0.24986833 0.24263181]
 [0.24981956 0.12552403 0.03536129]
 [0.15700947 0.16378179 0.13944702]
 [0.14793022 0.22045922 0.08594963]]
grad_z:[[-0.82823855  3.3407793
4.4734926 ]
 [-3.3423908   1.6827835   0.9551732 ]
 [-2.4705658   2.4619045   3.370505  ]
 [-0.525315    0.06622333  0.29898757]
 [-2.0143988   1.826932    2.045199  ]
 [-2.7207117   0.6499501   0.44798464]
 [-0.68758816  0.41285852  0.6849863 ]
 [-2.62619     2.6694293   1.6441073 ]]
grad_V:[[ -6.2283707   3.2239394]
 [ -8.810296    2.8460538]
 [-17.06557     7.400489 ]]
grad_c:[-21.833166   8.808743]
grad_W:[[-8.434512   7.808766   7.7868166]
 [-8.207522   4.560802   3.7588198]]
grad_b:[-15.215399  13.110861  13.920435]
cost: 10.873991966247559
V: [[ 3.892638  -1.7453325]
 [-3.1485198 -1.8888798]
 [-2.529593   2.5103943]],
c: [ 1.0916584  -0.44043714]
W: [[-3.549076    0.7162868   0.02768332]
 [ 3.208467   -2.8663316   2.9393382 ]],
b: [ 0.76076996 -0.655543   -0.69602174]
epoch:0, error:2.667658805847168
epoch:1000, error:0.10762323439121246
          ⋮
          ⋮
          ⋮
epoch:18000, error:0.02155356854200363
epoch:19000, error:0.021127980202436447
y:[[ 0.44688845 -0.44957975]
 [ 0.7834601   0.39290723]
 [-0.00315142 -0.5953219 ]
 [-0.18091488  0.10645691]
 [-0.11047626 -0.18300903]
 [ 0.15007544  0.20265695]
 [ 0.24650598  0.32641682]
 [ 0.6579385  -0.07345203]]
```





Precautions: -

- Ensure that your system is connected to a stable power supply (if not using MAC).
- Make sure you have all the required Libraries.
- Save the code periodically.
- Don't close the terminal if using Jupyter Notebook.

Experiment-8

Aim: - To train a multiclass classification model on iris dataset using a single layered neural network. Shuffle the data 25 times and plot the graph of experiment no vs classification error

Apparatus: -

- Laptop Configuration CPU, GPU, Core, clock etc.
- Laptop Configuration
  - Macbook
  - 8GB RAM
  - 8-Core CPU
  - 7-Core GPU
- Libraries Used: -
  - TensorFlow 1.40
  - NumPy
  - PyLab
  - Matplotlib
- Coding Environment: - Python 3.7.

Theory: -

\# **Cross Entropy: -** consider model that produces data belong to k classes with levels 1-k.
at class probability $P_{1\text{-}k}$

assume n_1, n_2, n_3, n_4……. n_k, number of data points were observed for each class and data point are independent of one another

the likelihood/probability P(data/model) of data given by the model

$$\boldsymbol{P(data/model) = P_1^{n_1}.P_2^{n_2}.P_3^{n_3} \dots\dots\dots P_k^{n_k}}$$

$$= \prod_{i=1}^{k} \boldsymbol{P_i^{n_i}}$$

The principle of maximum likelihood is a method of obtaining the optimum value of the parameters that define a model

the negative likelihood or cross entropy is given by

$$\mathbf{-log}\,(\boldsymbol{P(data/model)} = -\sum_{i=1}^{k} \boldsymbol{n_i log\, P_i}$$

dividing cross entropy by $\boldsymbol{N = n_1 + n_2 + \cdots + n_k}$

cross entropy

$$= -\sum_{i=1}^{k} \boldsymbol{\frac{n_i}{N} log\, P_i}$$

$$= -\sum_{i=1}^{k} \boldsymbol{q_i log\, P_i}$$

where $\boldsymbol{q_i = \frac{n_i}{N}}$ given probability of class K given by data

it can be shown that the cross entropy is minimum when $\boldsymbol{A_c = q_i}$ for all K

Python CODE: -

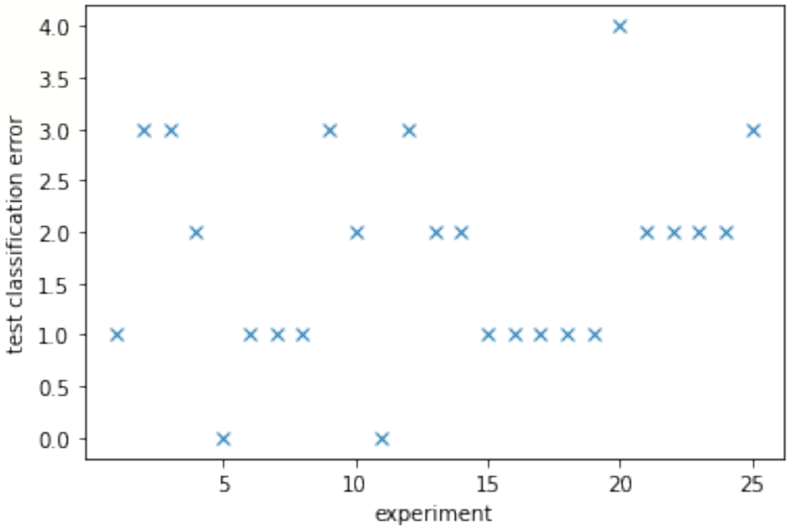
```python
from sklearn import datasets
import numpy as np
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()
import pylab as plt
import os
if not os.path.isdir('figures'):
    os.makedirs('figures')

tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

no_iters = 1000
no_labels = 3
no_features = 4
no_exps = 25

seed = 10

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

def ffn(x, hidden_units):

  # Hidden
  with tf.name_scope('hidden'):
    weights = tf.Variable(
      tf.truncated_normal([no_features, hidden_units],
                            stddev=1.0 / np.sqrt(float(no_features))),
        name='weights')
    biases = tf.Variable(tf.zeros([hidden_units]),
                         name='biases')
    hidden = tf.nn.relu(tf.matmul(x, weights) + biases)

  # output
  with tf.name_scope('linear'):
    weights = tf.Variable(
        tf.truncated_normal([hidden_units, no_labels],
                            stddev=1.0 / np.sqrt(float(hidden_units))),
        name='weights')
    biases = tf.Variable(tf.zeros([no_labels]),
                         name='biases')
    logits = tf.matmul(hidden, weights) + biases

  return logits


def main():
    # input data
```

```
iris = datasets.load_iris()
iris.data -= np.mean(iris.data, axis=0)
n = iris.data.shape[0]

print(iris.target.shape)

X = iris.data
no_data = len(iris.data)
Y = np.zeros((no_data, no_labels))
for i in range(no_data):
    Y[i, iris.target[i]] = 1


x = tf.placeholder(tf.float32, [None, no_features])
y_ = tf.placeholder(tf.float32, [None, no_labels])

y = ffn(x, 5)

# Create the model

cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits_v2(labels=y_, logits=y))

error = tf.reduce_sum(tf.cast(tf.not_equal(tf.argmax(y, axis=1), tf.argmax(y_,
axis=1)), dtype=tf.int32))

train = tf.train.GradientDescentOptimizer(0.05).minimize(cross_entropy)


err = []
for exp in range(no_exps):

    idx = np.arange(n)
    np.random.shuffle(idx)
    XX, YY = X[idx], Y[idx]
    x_train, y_train, x_test, y_test = XX[:100], YY[:100], XX[100:], YY[100:]

    # train
    with tf.Session() as sess:
        tf.global_variables_initializer().run()


        for i in range(no_iters):
            train.run(feed_dict={x:x_train, y_: y_train})
        err.append(error.eval(feed_dict={x:x_test, y_:y_test}))


    print('exp %d error %g'%(exp, err[exp]))

print('* mean error = %g *'% np.mean(err))
```

```
    plt.figure(1)
    plt.plot(np.arange(no_exps)+1, err, marker = 'x', linestyle = 'None')
    plt.xticks([5, 10, 15, 20, 25])
    plt.xlabel('experiment')
    plt.ylabel('test classification error')
    plt.savefig('./figures/6.1a_1.png')


    plt.show()

if __name__ == '__main__':
    main()
```

OUTPUT: -

```
(150,)
exp 0 error 1
exp 1 error 3
exp 2 error 3iter:900, error:2
       :
exp 22 error 2
exp 23 error 2
exp 24 error 3
* mean error = 1.76 *
```



Precautions: -

- Ensure that your system is connected to a stable power supply (if not using MAC).
- Make sure you have all the required Libraries.
- Save the code periodically.
- Don't close the terminal if using Jupyter Notebook.

Experiment-9

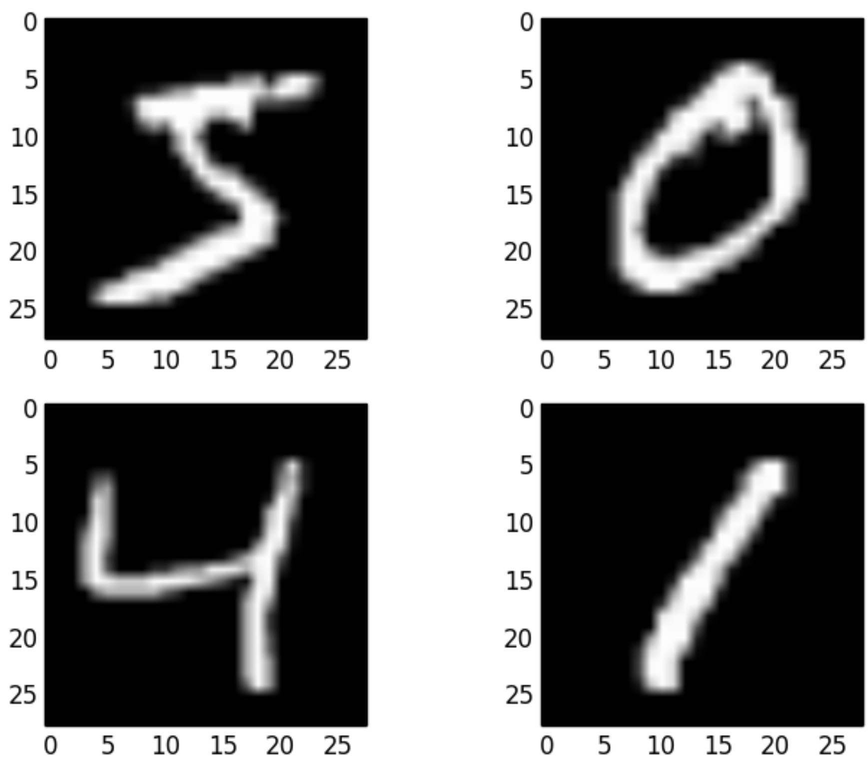
Aim: - Implementing CNN in Python with TensorFlow for MNIST digit recognition
Apparatus: -

- Laptop Configuration CPU, GPU, Core, clock etc.
- Laptop Configuration
  - Macbook
  - 8GB RAM
  - 8-Core CPU
  - 7-Core GPU
- Libraries Used: -
  - TensorFlow 1.40
  - NumPy
  - PyLab
  - Matplotlib
- Coding Environment: - Python 3.7.

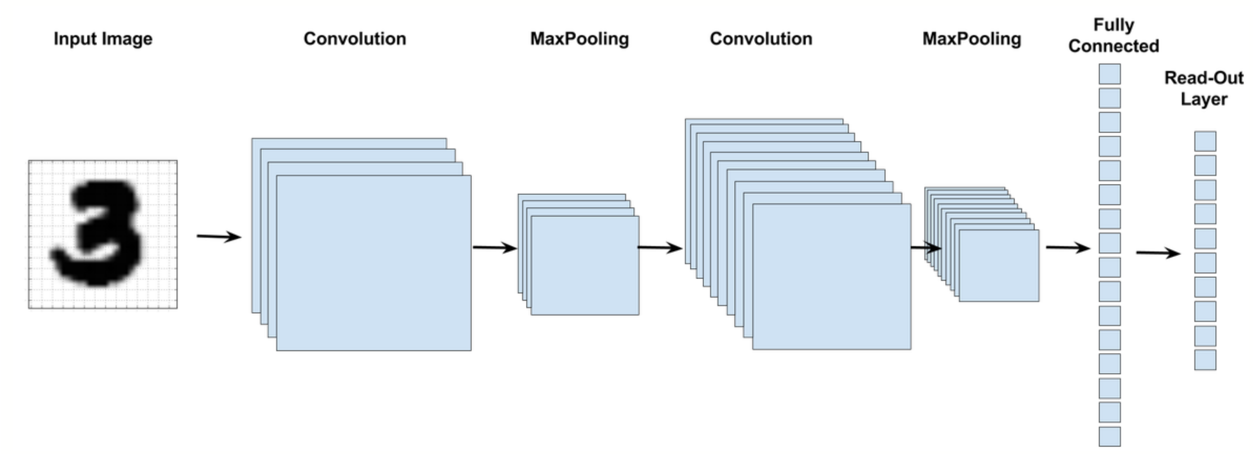
Theory: -

MNIST ("Modified National Institute of Standards and Technology") is the de facto "hello world" dataset of computer vision. Since its release in 1999, this classic dataset of handwritten images has served as the basis for benchmarking classification algorithms. As new machine learning techniques emerge, MNIST remains a reliable resource for researchers and learners alike.

MNIST is a dataset consisting of 60,000+ images of handwritten digits for training and another 10,000 for testing. Each training example comes with an associated label (0 to 9) indicating what digit it is. Each digit will be a black and white image of 28 X 28 pixels.



Architecture of CNN



A convolutional neural network is different from a standard artificial neural network, and may involve convolutional, pooling, fully connected and softmax layers. Let's understand each of these layers.

Python CODE: -

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot = True)

n_classes = 10
batch_size = 128

x = tf.placeholder('float', [None, 784])
y = tf.placeholder('float')

keep_rate = 0.8
keep_prob = tf.placeholder(tf.float32)

def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1,1,1,1], padding='SAME')

def maxpool2d(x):
    #                        size of window         movement of window
    return tf.nn.max_pool(x, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')


def convolutional_neural_network(x):
    weights = {'W_conv1':tf.Variable(tf.random_normal([5,5,1,32])),
               'W_conv2':tf.Variable(tf.random_normal([5,5,32,64])),
               'W_fc':tf.Variable(tf.random_normal([7*7*64,1024])),
               'out':tf.Variable(tf.random_normal([1024, n_classes]))}

    biases = {'b_conv1':tf.Variable(tf.random_normal([32])),
```

```
              'b_conv2':tf.Variable(tf.random_normal([64])),
              'b_fc':tf.Variable(tf.random_normal([1024])),
              'out':tf.Variable(tf.random_normal([n_classes]))}

    x = tf.reshape(x, shape=[-1, 28, 28, 1])

    conv1 = tf.nn.relu(conv2d(x, weights['W_conv1']) + biases['b_conv1'])
    conv1 = maxpool2d(conv1)

    conv2 = tf.nn.relu(conv2d(conv1, weights['W_conv2']) + biases['b_conv2'])
    conv2 = maxpool2d(conv2)

    fc = tf.reshape(conv2,[-1, 7*7*64])
    fc = tf.nn.relu(tf.matmul(fc, weights['W_fc'])+biases['b_fc'])
    fc = tf.nn.dropout(fc, keep_rate)

    output = tf.matmul(fc, weights['out'])+biases['out']

    return output

def train_neural_network(x):
    prediction = convolutional_neural_network(x)
    cost = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(prediction,y) )
    optimizer = tf.train.AdamOptimizer().minimize(cost)

    hm_epochs = 10
    with tf.Session() as sess:
        sess.run(tf.initialize_all_variables())

        for epoch in range(hm_epochs):
            epoch_loss = 0
            for _ in range(int(mnist.train.num_examples/batch_size)):
                epoch_x, epoch_y = mnist.train.next_batch(batch_size)
                _, c = sess.run([optimizer, cost], feed_dict={x: epoch_x, y:
epoch_y})
                epoch_loss += c

            print('Epoch', epoch, 'completed out of',hm_epochs,'loss:',epoch_loss)

        correct = tf.equal(tf.argmax(prediction, 1), tf.argmax(y, 1))

        accuracy = tf.reduce_mean(tf.cast(correct, 'float'))
        print('Accuracy:',accuracy.eval({x:mnist.test.images,
y:mnist.test.labels}))

train_neural_network(x)
```

OUTPUT: -

Precautions: -

- Ensure that your system is connected to a stable power supply (if not using MAC).
- Make sure you have all the required Libraries.
- Save the code periodically.
- Don’t close the terminal if using Jupyter Notebook.