Artificial Intelligence: `Lab Assignment-7`

Group Members-
Abhishek Katara
Ujwal Tewari
Nikhil Thota
Group-7

March 18, 2018

1 Question-1

Given data points-
X =[-20], [-19], [-18],[-17],[20],[-10],[30]
y =[10], [5], [-1],[5],[20],[-10],[30]

Train the neural network on 40 neurons and then predict the trained model on-
1. Range(-20 - 20)
2. Theta and Corresponding sin value

1.1 Code and Implementation

1.1.1 Neural Network

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

X = np.array(([-20], [-19], [-18],[-17],[20],[-10],[30]), dtype=float)
y = np.array(([10], [5], [-1],[5],[20],[-10],[30]), dtype=float)

# scale units
#X = X/np.amax(X, axis=0) # maximum of X array
#y = y/100

class Neural_Network(object):
  def __init__(self):
    #parameters
    self.inputSize = 1
    self.outputSize = 1
    self.hiddenSize = 40
    #weights
    self.W1 = np.random.randn(self.inputSize, self.hiddenSize)
```

```
    # weight matrix from input to hidden layer
    self.W2 = np.random.randn(self.hiddenSize, self.outputSize)
    # weight matrix from hidden to output layer
```

1.2 Forward Propagation

```
def forward(self, X):
    #forward propagation through our network
    self.z = np.dot(X, self.W1)
    self.z2 = self.sigmoid(self.z) # activation function
    self.z3 = np.dot(self.z2, self.W2)
    #o = self.z3
    return self.z3

  def sigmoid(self, s):
    # activation function
    #return 1/(1+np.exp(-s))
    return np.tanh(s)

  def sigmoidPrime(self, s):
    #derivative of sigmoid
    #return s * (1 - s)
    return (1 - (np.tanh(s)**2))
```

1.3 Back Propagation

```
 def backward(self, X, y, o):
    # backward propgate through the network
    self.o_error = y - o # error in output
    self.o_delta = self.o_error*self.sigmoidPrime(o)
    # applying derivative of sigmoid to error

    self.z2_error = self.o_delta.dot(self.W2.T)
    # z2 error: how much our hidden layer weights contributed to output error
    self.z2_delta = self.z2_error*self.sigmoidPrime(self.z2)
    # applying derivative of sigmoid to z2 error

    self.W1 += X.T.dot(self.z2_delta)
    # adjusting first set (input --> hidden) weights
    self.W2 += self.z2.T.dot(self.o_delta)
    # adjusting second set (hidden --> output) weights

  def train (self, X, y):
    o = self.forward(X)
    self.backward(X, y, o)
```

1.4 Training the network

```
NN = Neural_Network()
for i in range(1000): # trains the NN 1,000 times
  print ("Input: \n" + str(X))
```

```
 print ("Actual Output: \n" + str(y))
 print ("Predicted Output: \n" + str(NN.forward(X)))
 print ("Loss: \n" + str(np.mean(np.square(y - NN.forward(X)))))
# mean sum squared loss
 print ("\n")
NN.train(X, y)
```

1.5 Prediction on X and Y values-Decimal Values

```
 result_array = np.array(([-20],[-19],[-18],[-17],[-16]
  ,[-15],[-14],[-13],[-12],[-11],[-10],
  [-9],[-8],[-7],[-6],[-5],[-4],[-3]
  ,[-2],[-1],[0],[1],[2],[3],[4],[5],[6],[7
  ,[8],[9],[10],[11],[12],[11],[12],[13]
  ,[14],[15],[16],[17],[18],[19],
  [20]),dtype=float)
 #X_final_pred = np.array((), dtype=float)
  print ("Predicted Output: \n" + str(NN.forward(result_array)))

plt.plot(result_array, NN.forward(result_array))
plt.xlabel('Input (X)')
plt.ylabel('Prediction(Y_)')
plt.title('Prediction on given data')
plt.grid(True)
plt.savefig("test.png")
plt.show()
```

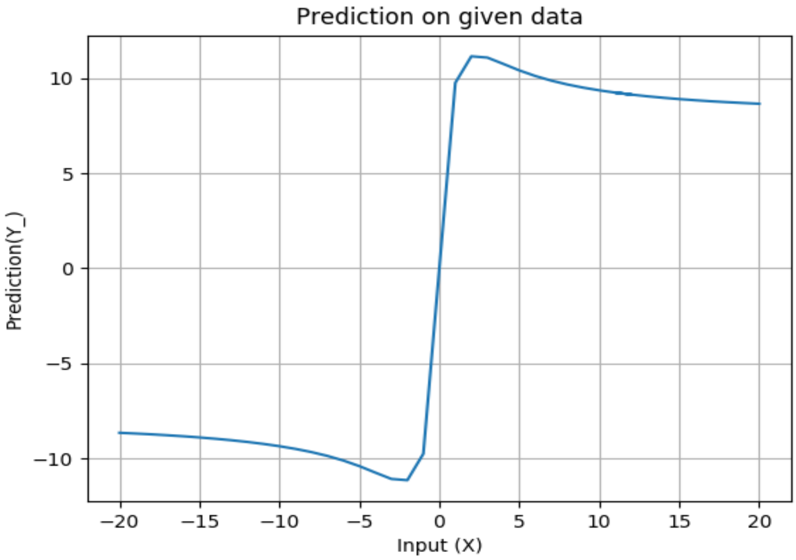

Figure 1: Decimal Value prediction

1.6 Prediction on Theta and Sin Values

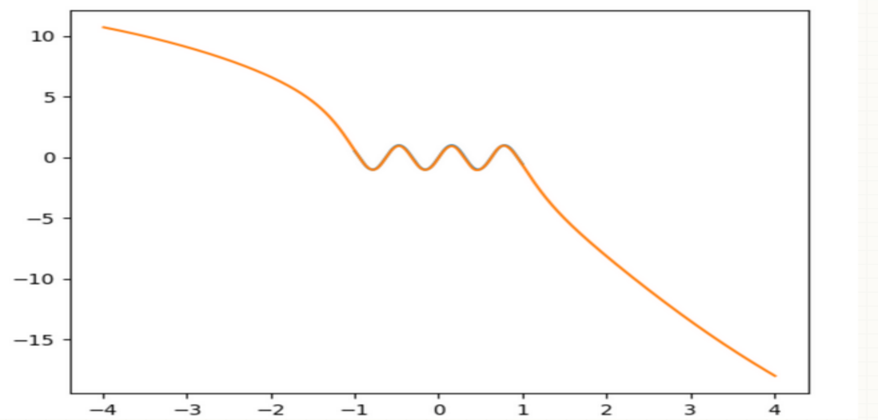

Figure 2: 100 iterations

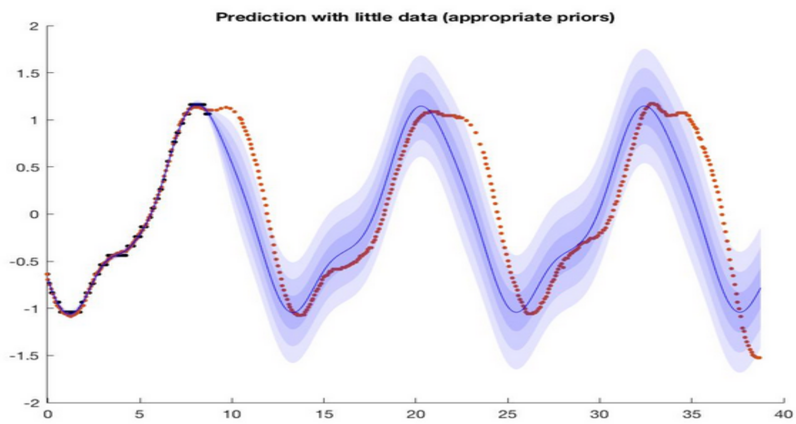

Figure 3: 1000 iterations

2 Neural Network Prediction on Climate Data

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.metrics import explained_variance_score, \
    mean_absolute_error, \
    median_absolute_error
from sklearn.model_selection import train_test_split
df = pd.read_csv('end-part2_df.csv').set_index('date')

df.describe().T
df = df.drop(['mintempm', 'maxtempm'], axis=1)
y = df['meantempm']
X_train, X_tmp, y_train, y_tmp = train_test_split(X, y, test_size=0.2, random_s
X_test, X_val, y_test, y_val = train_test_split(X_tmp, y_tmp, test_size=0.5, ra
X_train.shape, X_test.shape, X_val.shape
print("Training instances   {},
        Training features   {}".
        format(X_train.shape[0],
        X_train.shape[1]))
print("Validation instances {},
       Validation features {}"
       .format(X_val.shape[0],
       X_val.shape[1]))
print("Testing instances
       {}, Testing features
       {}".format(X_test.shape[0],
       X_test.shape[1]))

feature_cols = [tf.feature_column.numeric_column(col)
                for col in X.columns]

regressor = tf.estimator.
            DNNRegressor(feature_columns=feature_cols,
                         hidden_units=[50, 50],
                         model_dir='tf_wx_model')

def wx_input_fn(X, y=None, num_epochs=None,
                shuffle=True, batch_size=400):
    return tf.estimator.
    inputs.pandas_input_fn(x=X,
                           y=y,
                          num_epochs=num_epochs,
                          shuffle=shuffle,
                           batch_size=batch_size)

evaluations = []
STEPS = 400
```

```
for i in range(100):
    regressor.train(
    input_fn=wx_input_fn(X_train, y=y_train)
    , steps=STEPS)
    evaluation = regressor.evaluate(
    input_fn=wx_input_fn(X_val, y_val,
                         num_epochs=1,
                         shuffle=False),
                         steps=1)
    evaluations.append(regressor.
    evaluate(input_fn=wx_input_fn(X_val,
              y_val,num_epochs=1,shuffle=False)))


# manually set the parameters of the figure to and appropriate size
plt.rcParams['figure.figsize'] = [14, 10]

loss_values = [ev['loss'] for ev in evaluations]
training_steps = [ev['global_step'] for ev in evaluations]

plt.scatter(x=training_steps, y=loss_values)
plt.xlabel('Training steps (Epochs = steps / 2)')
plt.ylabel('Loss (SSE)')
plt.show()
```

Out[3]:

	count	mean	std	min	25%	50%	75%	max
meantempm	997.0	13.129388	10.971591	-17.0	5.0	15.0	22.00	32.00
maxtempm	997.0	19.509529	11.577275	-12.0	11.0	22.0	29.00	38.00
mintempm	997.0	6.438315	10.957267	-27.0	-2.0	7.0	16.00	26.00
meantempm_1	997.0	13.109328	10.984613	-17.0	5.0	15.0	22.00	32.00
meantempm_2	997.0	13.088265	11.001106	-17.0	5.0	14.0	22.00	32.00
meantempm_3	997.0	13.066199	11.017312	-17.0	5.0	14.0	22.00	32.00
meandewptm_1	997.0	6.440321	10.596265	-22.0	-2.0	7.0	16.00	24.00
meandewptm_2	997.0	6.420261	10.606550	-22.0	-2.0	7.0	16.00	24.00
meandewptm_3	997.0	6.393180	10.619083	-22.0	-2.0	7.0	16.00	24.00
meanpressurem_1	997.0	1016.139418	7.582453	989.0	1011.0	1016.0	1021.00	1040.00
meanpressurem_2	997.0	1016.142427	7.584185	989.0	1011.0	1016.0	1021.00	1040.00
meanpressurem_3	997.0	1016.151454	7.586988	989.0	1011.0	1016.0	1021.00	1040.00

Figure 4: Dataset

3 Hopfield Network

```
clear all;
close all;

%--------------------------------------------------------------
% Patterns to store
% D, J, C, M
%--------------------------------------------------------------
X = [1 1 1 1 -1 -1 1 -1 -1 1 -1 1 -1 -1 1 -1 1 -1 -1 1 -1 1 1 1 -1;
1 1 1 1 1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 1 -1 -1 1 -1 1 1 1 -1 -1;
-1 1 1 1 1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 1 -1 -1 -1 -1 -1 1 1 1 1;
1 -1 -1 -1 1 1 1 -1 1 1 1 -1 1 -1 1 1 -1 -1 -1 1 1 -1 -1 -1 1]';

%figure;
%imshow(reshape(-X(:,1),5,5)');

%--------------------------------------------------------------
% Learn the weights according to Hebb's rule
%--------------------------------------------------------------
[m,n] = size(X);
W = zeros(m,m);
for i = 1:n
        W = W + X(:,i)*X(:,i)';
endfor
W(logical(eye(size(W)))) = 0;
W = W/n;

%--------------------------------------------------------------
```

```
Index: 997 entries, 2015-01-04 to 2017-09-27
Data columns (total 39 columns):
meantempm             997 non-null int64
maxtempm              997 non-null int64
mintempm              997 non-null int64
meantempm_1           997 non-null float64
meantempm_2           997 non-null float64
meantempm_3           997 non-null float64
meandewptm_1          997 non-null float64
meandewptm_2          997 non-null float64
meandewptm_3          997 non-null float64
meanpressurem_1       997 non-null float64
meanpressurem_2       997 non-null float64
meanpressurem_3       997 non-null float64
maxhumidity_1         997 non-null float64
maxhumidity_2         997 non-null float64
maxhumidity_3         997 non-null float64
minhumidity_1         997 non-null float64
minhumidity_2         997 non-null float64
minhumidity_3         997 non-null float64
maxtempm_1            997 non-null float64
```

Figure 5: Dataframe

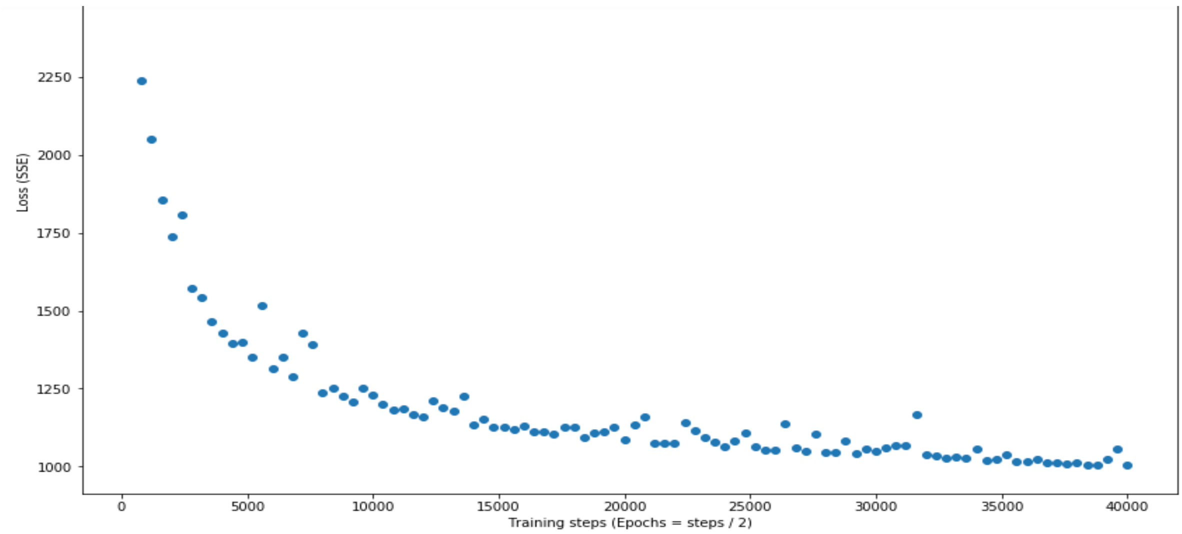

Figure 6: Loss vs Epochs

```python
pred = regressor.predict(input_fn=wx_input_fn(X_test,
                                              num_epochs=1,
                                              shuffle=False))
predictions = np.array([p['predictions'][0] for p in pred])

print("The Explained Variance: %.2f" % explained_variance_score(
                                            y_test, predictions))
print("The Mean Absolute Error: %.2f degrees Celcius" % mean_absolute_error(
                                            y_test, predictions))
print("The Median Absolute Error: %.2f degrees Celcius" % median_absolute_error(
                                            y_test, predictions))
```

```
INFO:tensorflow:Restoring parameters from tf_wx_model/model.ckpt-40000
The Explained Variance: 0.89
The Mean Absolute Error: 3.07 degrees Celcius
The Median Absolute Error: 2.45 degrees Celcius
```

Figure 7: Prediction Output on climate data-

```matlab
% Dynamical (Linear) System and fixed points
%-----------------------------------------------
x = X(:,1);
x(5)=-1*x(5);

figure(1);
subplot(1,2,1);
imshow(reshape(-X(:,1),5,5)');
subplot(1,2,2);
imshow(reshape(-x,5,5)');

y = x;
erry = 10;
while erry > 1
        yp = sign(W*y);
        erry = norm(yp-y);
        y = yp;
        figure(2);
        imshow(reshape(-y, 5, 5)');
        pause();
endwhile

%-----------------------------------------------
% Damaging 50 neurons!
%-----------------------------------------------
```

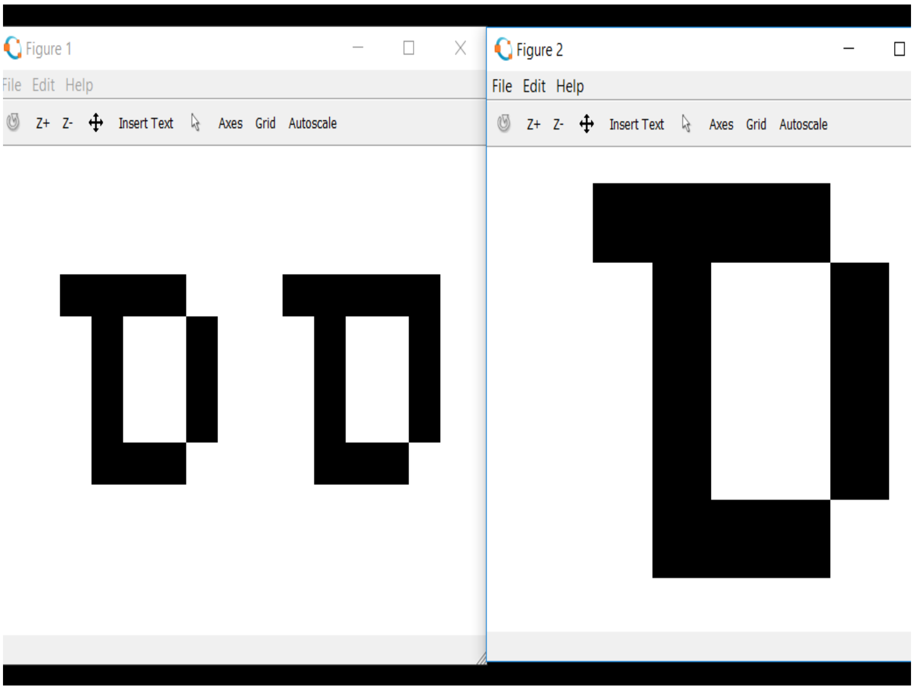

Figure 8: Hop Field Output