A Neural Network Implementation for Pattern Classification using Hebb's rule

Reading input images

In [943]:
```python
import numpy as np
import os
import copy
```

File read operations

In [944]:
```python
filePath = "./pattern_data/"
```

In [945]:
```python
def read_file(fileName,read_type="col_wise"):
    file = os.path.join(filePath,fileName)
    print('Quering file {}'.format(file))

    if read_type == "row_wise":
        pattern, labels,pixel= _read_file_row_wise(file)
    else:
        pattern, labels,pixel= _read_file_col_wise(file)

    print('..Completed processing file {}...'.format(read_type))
    return pattern, labels
```

In [946]:
```python
def _read_file_row_wise(file):

    pattern, labels,pixel =[],[],[];


    with open(file,"r") as f:
        lines = f.read().splitlines()
        for line in lines:
            for ch in line:
                if ch == "#" or ch == ".":
                    pixel.append(ch)
                elif ch ==",":
                    pattern.append(pixel)
                    pixel=[]
                elif ch == 'X' or ch == 'C':
                    labels.append(ch)
                else:
                    pixel.append('0')


    return pattern, labels,pixel
```

In [947]:
```python
def _read_file_col_wise(file):
    """
    Routine to read the data columnwise from the file
    """
    pattern, labels,pixel =[],[],[];

    lists_by_line = [];
    with open(file,"r") as f:
        lines = f.read().splitlines()
        for line in lines:
            for ch in line:
                if ch == "#" or ch == ".":
                    pixel.append(ch)
                elif ch ==",":
                    lists_by_line.append(copy.copy(pixel))
                    pattern.append(copy.copy(lists_by_line))
                    pixel.clear()
                    lists_by_line.clear()
                elif ch == 'X' or ch == 'C':
                    labels.append(ch)
                else:
                    pixel.append('0')

            if(len(pixel) > 0):
                lists_by_line.append(copy.copy(pixel))
                pixel.clear()



    pattern_row_list = []
    L = len(pattern)

    for i in range(0,L):
        pattern_row_list.append(_convert_to_row_vector(pattern[i]))

    return pattern_row_list,labels,pixel
```

In [948]:
```python
def _convert_to_row_vector(pattern):
    L = len(pattern)

    sub_len = len(pattern[0])
    row_vector=[]
    row_vector.clear()
    for j in range(0,sub_len):
        i = 0
        while i < L:
            row_vector.append(pattern[i][j])
            i = i+1;

    return row_vector
```

In [949]:
```python
train_patterns,train_labels = read_file(fileName="training_data.txt",read_type="col_wise")
```

```
Quering file ./pattern_data/training_data.txt
..Completed processing file col_wise...
```

In [962]:
```python
test_patterns,test_labels = read_file(fileName="test_data.txt",read_type="col_wise")
```

```
Quering file ./pattern_data/test_data.txt
..Completed processing file col_wise...
```

Convert to Numpy Array

In [951]:
```python
def convertPatternToBipolarVectors(patterns):
    input_vector,image = [],[];
    for pattern in patterns:
        for  ch in pattern:
            if ch == "#":
                image.append(1)
            elif ch == ".":
                image.append(-1)
            else:
                image.append(0)

        input_vector.append(image)
        image=[]

    return input_vector

def convertLabelToBipolar(labels):
    output_vector =[];

    for ch in labels:
        if ch == "C":
            output_vector.append(1)
        elif ch=="X":
            output_vector.append(-1)
        else:
            output_vector.append(0)

    return output_vector
```

Proccesing Training Data

In [952]:
```python
bipolar_train = convertPatternToBipolarVectors(train_patterns)
bipolar_train_label = convertLabelToBipolar(train_labels)
bipolar_train,bipolar_train_label
```

Out[952]:
```
([[-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])
```

In [953]:
```python
def convertTrainToNumpyArray(data,labels):
    images = np.array(data)
    dim = labels[0]
    t = np.array(labels).reshape(-1,dim)
    return images,t
```

In [954]:
```python
patterns_train,t = convertTrainToNumpyArray(bipolar_train,bipolar_train_label)
```

In [955]:
```python
bipolar_test = convertPatternToBipolarVectors(test_patterns)
print('Training Images dimensions',patterns_train.shape)
print('Training Labels dimensions',t.shape)
```

```
Training Images dimensions (2, 15)
Training Labels dimensions (2, 1)
```

Processing Test Data

In [1308]:
```python
test_patternC = bipolar_test[0]
test_patternX = bipolar_test[1]
print("C: {}\n X :{}".format(test_patternC,test_patternX))
```

```
C: [-1, 1, 1, 1, -1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1]
 X :[1, 1, -1, 1, 1, -1, -1, 1, -1, -1, 1, 1, -1, 1, 1]
```

In [740]:
```python
def convertTestToNumpyArray(data):
    images = np.array(data)
    return images
```

Flip Test Patterns by K pixels: Valid Training Data

Recursive Inplementation of k flips for valid character array

In [963]:
```python
def combinationsByKFlip(vector,start,k,end,combination_list,result_list):
    if vector not in combination_list:
        combination_list.insert(start,vector)


    if k == 0:
        result_list.append(vector)
        return result_list

    for i in range(start,end):
        vector = _flipElement(vector,i)
        combinationsByKFlip(vector,i+1,k-1,end,combination_list,result_list)
        vector =  combination_list[start]

    return result_list


def _flipElement(vector,i):
    vectorCopy = copy.copy(vector)
    if vector[i] == 1:
        vectorCopy[i] = -1
    elif vector[i] == -1:
        vectorCopy[i] = 1
    elif vector[i] == 0:
        vectorCopy[i] = 0
    return vectorCopy
```

Test for Pattern C

In [1315]:
```python
combination_list=[]
result_list = []
start = 0
length = len(vector)
k_flipped = 15
vector = test_patternX
```

In [1220]:
```python
flippedPatterns_test_set = combinationsByKFlip(vector,start,k_flipped,length,combination_list,resul
t_list)
combination_list.clear()
print("Total Test Data by flipping {1} elements are {0}".format(len(flippedPatterns_test_set),k_fli
pped))
patterns_flipped_test= convertTestToNumpyArray(flippedPatterns_test_set)
```

```
Total Test Data by flipping 15 elements are 1
```

To Create Corrupted Data
Generate combinations for corrupted training data

In [828]:
```python
def getCombinationsForUndetermined(pattern,path,index,misclassifiedLength):
    if misclassifiedLength == 0:
        temp =copy.deepcopy(path)
        combinations.append(temp)
        return
    for i in range(index,15):
        path.append(i)
        getCombinationsForUndetermined(pattern,path,i+1,misclassifiedLength-1)
        path.pop()


def undeterminedFlipping(combinationIndex,pattern):
    for i in range(len(combinationIndex)):
        temp2 = copy.deepcopy(pattern)
        for j in range(len(combinationIndex[i])):
            num = 0*temp2.item(combinationIndex[i][j])
            temp2.itemset(combinationIndex[i][j],num)
            undeterminedPatterns.append(temp2)
```

In [1309]:
```python
test_patternX = convertTestToNumpyArray(test_patternX)
vector = test_patternX
k_undetermined = 15
combinations=[]
combinations.clear()
undeterminedPatterns=[]

getCombinationsForUndetermined(vector,[],0,k_undetermined)
undeterminedPatterns.clear()
len(combinations)
```

Out[1309]: 1

In [1310]:
```python
undeterminedFlipping(combinations,vector)
print("Total Corrupted Test Data by using  {1} undetermined elements are {0}".format(len(undetermin
edPatterns),k_undetermined))
```

```
Total Corrupted Test Data by using  15 undetermined elements are 15
```

In [1311]:
```python
undeterminedPatterns = convertTestToNumpyArray(undeterminedPatterns)
```

In [814]:
```python
print("Datatype of complete training data.....")
print("Test Pattern: {0} \nFlipped Pattern Test Data type: {1} \nCorrupted Test Data type: {2}".form
at(type(test_patternC),type(patterns_flipped_test),type(undeterminedPatterns)))
```

```
Datatype of complete training data.....
Test Pattern: <class 'numpy.ndarray'>
Flipped Pattern Test Data type: <class 'numpy.ndarray'>
Corrupted Test Data type: <class 'numpy.ndarray'>
```

In [713]:
```python
print("Pattern :{} \nFlipped Patterns: {}\nUndermined Patterns:{}".format(test_patternC,patterns_fli
pped_test[0:2],undeterminedPatterns[0:2]))
```

```
Pattern :[-1, 1, 1, 1, -1, 1, -1, -1, -1, 1, 1, -1, -1, -1, 1]
Flipped Patterns: [[ 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]]
Undermined Patterns:[[ 0  1  1  1 -1  1 -1 -1 -1  1  1 -1 -1 -1  1]
 [-1  0  1  1 -1  1 -1 -1 -1  1  1 -1 -1 -1  1]]
```

Convert Test Data Combinations to Numpy Array

In [847]:
```python
print("Train Data Dimenions......")
print("Pattern :{} \nFlipped Patterns: {}\nUndermined Patterns:{}".format(test_patternC.shape,patter
ns_flipped_test.shape,undeterminedPatterns.shape))
```

```
Train Data Dimenions......
Pattern :(15,)
Flipped Patterns: (1365, 15)
Undermined Patterns:(1365, 15)
```

Neural Network Implementation using Hebb's Rule

Define the dimensions for a two layered Neural Network Perceptron Model

In [956]:
```python
dimensions = [patterns_train.shape[1],t.shape[1]]
parameters = {}
```

Define the Neural Network

In [1328]:
```python
class NeuralNetwork:
    def __init__(self,dimensions):
        self.parameters = parameters

        print('...Initializing Parameters for the model...')
        parameters["W"] = np.zeros([dimensions[0],dimensions[1]])
        parameters["b"] = np.zeros([dimensions[1],1])

        print('Weight Matrix defined as {}'.format(parameters["W"].shape))
        print('Bias Vector defined as {}'.format(parameters["b"].shape))

    def affineForward(self,X,W,b):
        print('.....Staring affineForward.....')
        yin = np.dot(X,W) + b
        #print(yin)
        return yin

    def feedForwardPropagation(self,X):
        print('....Start FeedForwardPropagation....')

        W = self.parameters["W"]
        b = self.parameters["b"]
        yin = self.affineForward(X,W,b)

       # print ('Shape of output {}'.format(yin.shape))
        return yin

    def computeGradient(self,t,S):
        print('....Computing Gradients.....')
        dw = np.dot(S.T,t)
        db = np.sum(t,axis=0).reshape(1,-1)

        #print('Delta W:{}'.format(dw.shape))
        #print('Delta b:{}'.format(db.shape))
        return dw,db

    def updateParameters(self,dw,db):
        print('....Updating Weight and bias.....')
        parameters = self.parameters

        parameters["W"] = np.add(parameters["W"],dw)
        parameters["b"] = np.add(parameters["b"],db)

        #print("W")
        #print(parameters["W"])
        #print("b")
        #print(parameters["b"])

    def activation_unit(self,x):
        theta = 0;
        x[x > theta] = 1
        x[x < theta] = -1
        x[x == 0] = 0
        return x

    def train(self,X,t,numOfIterations):
        S = X
        y = t
        print('......Training Data Model......')
        for i in range(1,numOfIterations):
            print('Starting iteration {}*********'.format(i))
            yin = self.feedForwardPropagation(X)
            dw,db = self.computeGradient(y,S)
            self.updateParameters(dw,db)

        print('...Training ends here....')


    def predict(self,X):
        t = self.feedForwardPropagation(X)
        t = self.activation_unit(t)
        return t
```

Train Model

In [1329]:
```python
model = NeuralNetwork(dimensions)
```

```
...Initializing Parameters for the model...
Weight Matrix defined as (15, 1)
Bias Vector defined as (1, 1)
```

In [1330]:
```python
model.train(patterns_train,t,3)
```

```
......Training Data Model......
Starting iteration 1*********
....Start FeedForwardPropagation....
.....Staring affineForward.....
....Computing Gradients.....
....Updating Weight and bias.....
Starting iteration 2*********
....Start FeedForwardPropagation....
.....Staring affineForward.....
....Computing Gradients.....
....Updating Weight and bias.....
...Training ends here....
```

Validation of Training Dataset

In [1331]:
```python
y_train = model.predict(patterns_train)
```

```
....Start FeedForwardPropagation....
.....Staring affineForward.....
```

In [1332]:
```python
y_train
```

Out[1332]:
```
array([[ 1.],
       [-1.]])
```

Prediction Test dataset available are: test_patternC,patterns_flipped_test,undeterminedPatterns

In [608]:
```python
"""
Use Test data set :
test_patternC,
patterns_flipped_test,
undeterminedPatterns
"""
```

Out[608]: '\nUse Test data set : \ntest_patternC,\npatterns_flipped_test,\nundeterminedPatterns\n'

Testing pattern C

In [623]:
```python
#Validating for valid patterns
correct_pred = model.predict(test_patternC)
print("Predicted out for valid test data {}".format(correct_pred))
```

```
....Start FeedForwardPropagation....
Predicted out for valid test data [[1.]]
```

Predictions for Flipped and Corrupted Data Sets C

In [966]:
```python
misclassfies_via_flip = {}
```

In [967]:
```python
def countMisclassification(output, pattern,test_data,estimator="flipped"):
    output = output.astype(int).tolist()
    misclassified_list =[]
    misclassified_input_set = []
    if pattern == 'C':
        label = 1
    elif pattern == 'X':
        label = -1

    missclassified_list = list(filter(lambda x:x != label,output))
    count = len(missclassified_list)
    if estimator == "undermined":
        for i in range(0,len(output)):
            if(output[i] != label):
                misclassified_input_set.append(test_data[i])

    return count,misclassified_input_set
```

In [1038]:
```python
#Analyzing Flipped Patterns

flipped_pred = model.predict(patterns_flipped_test)
flipped_pred = flipped_pred[:,0]
count,misfit = countMisclassification(flipped_pred,pattern='C',test_data=patterns_flipped_test,esti
mator="flipped")

if count > 1:
    print('Misclassficiation found at k = {} flipped pixels.'.format(k_flipped))
    print('Total {} Misclassifications found.'.format(count))
else:
    print('Neural Network classified correctly for k = {} flipped pixels'.format(k_flipped))

misclassfies_via_flip[k_flipped]=count
```

```
....Start FeedForwardPropagation....
Neural Network classified correctly for k = 15 flipped pixels
```

In [1039]:
```python
print("Misclassfication count for K flipped pixesl \n",misclassfies_via_flip)
```

```
Misclassfication count for K flipped pixesl
 {1: 0, 2: 0, 3: 0, 4: 70, 5: 546, 6: 1890, 7: 3830, 8: 5055, 9: 4585, 10: 2947, 11: 1365, 12: 4
55, 13: 105, 14: 15, 15: 1}
```

In [1044]:
```python
misclassfies_via_undetermined = {}
```

In [1137]:
```python
#Analyzing Undetermined Patterns

undetermined_pred = model.predict(undeterminedPatterns)
undetermined_pred = undetermined_pred[:,0]
count,misfits = countMisclassification(undetermined_pred,pattern='C',test_data=undeterminedPatterns
,estimator="undermined")

if count > 1:
    print('Misclassficiation found at k = {} undertemined pixels.'.format(k_undetermined))
    print('Total {} Misclassifications found.'.format(count))
    misclassfies_via_undetermined[k_undetermined]=count,misfits[0]
else:
    print('Neural Network classified correctly for k = {} undertemined pixels'.format(k_undetermine
d))
    misclassfies_via_undetermined[k_undetermined]=count
```

```
....Start FeedForwardPropagation....
Misclassficiation found at k = 15 undertemined pixels.
Total 15 Misclassifications found.
```

In [1138]:
```python
print("Misclassfication count for K undertmined pixesl \n",misclassfies_via_undetermined)
```

```
Misclassfication count for K undertmined pixesl
 {1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: (8, array([ 0,  1,  0,  1,  0,  0, -1,  0, -1,
0,  1,  0, -1,  0,  1])), 9: (63, array([ 0,  0,  0,  1,  0,  0, -1,  0, -1,  0,  1,  0, -1,  0,
  1])), 10: (210, array([ 0,  0,  0,  0,  0,  0, -1,  0, -1,  0,  1,  0, -1,  0,  1])), 11: (38
5, array([ 0,  0,  0,  0,  0,  0,  0,  0, -1,  0,  1,  0, -1,  0,  1])), 12: (420, array([ 0,
0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  0, -1,  0,  1])), 13: (273, array([ 0,  0,  0,  0,  0,
0,  0,  0,  0,  0,  0,  0, -1,  0,  1])), 14: (98, array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1])), 15: (15, array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]))}
```

Analysis for pattern X

In [1142]:
```python
misclassfies_via_flip_X = {}
```

In [1221]:
```python
#Analyzing Flipped Patterns

flipped_pred = model.predict(patterns_flipped_test)
flipped_pred = flipped_pred[:,0]
count,misfit = countMisclassification(flipped_pred,pattern='X',test_data=patterns_flipped_test,esti
mator="flipped")

if count > 1:
    print('Misclassficiation found at k = {} flipped pixels.'.format(k_flipped))
    print('Total {} Misclassifications found.'.format(count))
else:
    print('Neural Network classified correctly for k = {} flipped pixels'.format(k_flipped))

misclassfies_via_flip_X[k_flipped]=count
```

```
....Start FeedForwardPropagation....
Neural Network classified correctly for k = 15 flipped pixels
```

In [1222]:
```python
print("Misclassfication count for K flipped pixesl for pattern X \n",misclassfies_via_flip_X)
```

```
Misclassfication count for K flipped pixesl for pattern X
 {1: 0, 2: 0, 3: 0, 4: 70, 5: 546, 6: 1890, 7: 3830, 8: 5055, 9: 4585, 10: 2947, 11: 1365, 12: 4
55, 13: 105, 14: 15, 15: 1}
```

In [1227]:
```python
misclassfies_via_undetermined_X = {}
```

In [1312]:
```python
#Analyzing Undetermined Patterns

undetermined_pred = model.predict(undeterminedPatterns)
undetermined_pred = undetermined_pred[:,0]
count,misfits = countMisclassification(undetermined_pred,pattern='X',test_data=undeterminedPatterns
,estimator="undermined")

if count > 1:
    print('Misclassficiation found at k = {} undertemined pixels for pattern X.'.format(k_undetermi
ned))
    print('Total {} Misclassifications found.'.format(count))
    misclassfies_via_undetermined_X[k_undetermined]=count,misfits[0]
else:
    print('Neural Network classified correctly for k = {} undertemined pixels for pattern X'.format
(k_undetermined))
    misclassfies_via_undetermined_X[k_undetermined]=count
```

```
....Start FeedForwardPropagation....
Misclassficiation found at k = 15 undertemined pixels for pattern X.
Total 15 Misclassifications found.
```