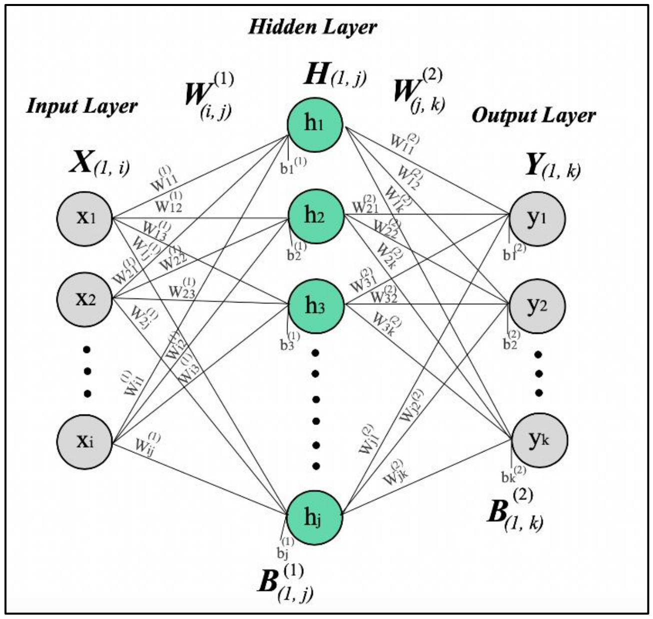
FEED FORWARD MULTILAYER NEURAL NETWORK

XOR with backpropagation
Handwritten digits recognition
Handwritten alphabet recognition

SRI SAI VIJAYA ADITYA NITTALA

ROLL NUMBER: 177163
REGISTRATION NUMBER: 841775

The architecture of a 3-layer Neural Network is shown below:



Where,

- X: Input features fed into the Neural Network.
- W: Weights associated with the activations of the Neural Network.
- B: Bias term introduced to the Neural Network.

A Neural Network can be used for classification purposes. The algorithm that is used to classify is called ***Backpropagation.*** An overview is given below:

Algorithm: Backpropagation. Neural network learning for classification or numeric prediction, using the backpropagation algorithm.

Input:

- *D*, a data set consisting of the training tuples and their associated target values;
- *l*, the learning rate;
- *network*, a multilayer feed-forward network.

Output: A trained neural network.

Method:

Initialize all weights and biases in *network*;
while terminating condition is not satisfied {
  for each training tuple $\boldsymbol{X}$ in D {
    // Propagate the inputs forward:
    for each input layer unit j {
      $O_j = I_j$; // output of an input unit is its actual input value
    for each hidden or output layer unit j {
      $I_j = \sum_i w_{ij} O_i + \theta_j$; //compute the net input of unit j with respect to the previous layer, i
      $O_j = \frac{1}{1+e^{-I_j}}$; } // compute the output of each unit j
    // Backpropagate the errors:
    for each unit j in the output layer
      $Err_j = O_j(1-O_j)(T_j - O_j)$; // compute the error
    for each unit j in the hidden layers, from the last to the first hidden layer
      $Err_j = O_j(1-O_j)\sum_k Err_k w_{jk}$; // compute the error with respect to the next higher layer, k
    for each weight w_{ij} in *network* {
      $\Delta w_{ij} = (l) Err_j O_i$; // weight increment
      $w_{ij} = w_{ij} + \Delta w_{ij}$; } // weight update
    for each bias θ_j in *network* {
      $\Delta \theta_j = (l) Err_j$; // bias increment
      $\theta_j = \theta_j + \Delta \theta_j$; } // bias update
    } }

In this assignment, using the ***backpropagation*** algorithm, the following have been implemented:

1. **Exclusive OR (XOR) gate implementation:**

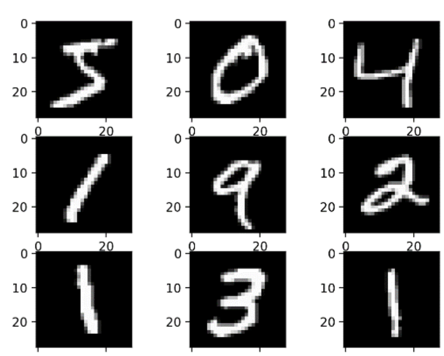
   The truth-table of an XOR gate is given below:

Input		Output
A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

Information about the implementation:

a. Number of neurons in input layer: 2
b. Number of hidden layers: 1
c. Number of neurons in the hidden layer: 2
d. Number of neurons in output layer: 1
e. Input: Dataset is essentially the truth-table of XOR but 0s are represented by 0.1 and 1s with 0.9 as sigmoid cannot compute exactly 0 or 1.
f. Number of iterations: 1000
g. Learning rate: 0.1
h. Weights are update after each forward-backward pass, i.e, not a batch update.
i. Test data: truth-table of XOR

2. **Handwritten Digit recognition:**

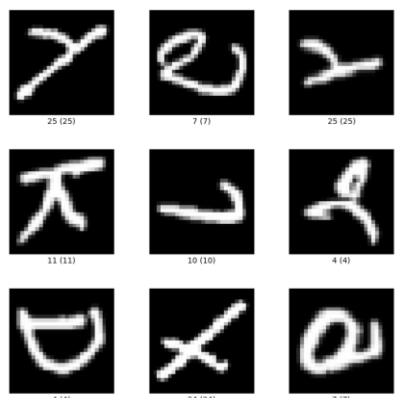


The dataset used for this application is the **MNIST** dataset. It consists of images, each of size **28x28** pixels (total 784 per image). Number of training and testing examples taken is less due to infrastructure constraints. Code attached at the end contains more information about the input.

Information about the implementation:

a. Number of neurons in input layer: 784
b. Number of hidden layers: 1
c. Number of neurons in hidden layer: 50
d. Number of neurons in output layer: 10
e. Input:
    a. Number of training examples considered: 1000
    b. Number of testing examples considered: 100
f. Learning rate: 0.01
g. Number of iterations: 10

3. **<u>Handwritten alphabet recognition</u>:**



The dataset used for this application is the **EMNIST** dataset. It is very similar to the previously described MNIST dataset. Contains images of size **28x28**, total of 784 pixels. More information is provided as comments in the code attached below.

Information about the implementation:

a. Number of neurons in input layer: 784
b. Number of hidden layers: 1
c. Number of neurons in hidden layer: 50
d. Number of neurons in output layer: 26
e. Input:
    a. Number of training examples considered: 2400
    b. Number of testing examples considered: 600
f. Learning rate: 0.01
g. Number of iterations: 10

Feed Forward Neural Networks - ML Lab assignment

Name: Sri Sai Vijaya Aditya Nittala

Roll No.: 177163

Section: A

Code includes implementation of :

- XOR gate : Adaline Backpropagation
- Hand-written digits classification
- Hand-written character classification

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

Functions required for Neural Networks

Initializing parameters for all layers

```
dim( W[l] ) = ( n[l], n[l-1] )
 dim( b[l] ) = ( 1, n[l] )
 where,
   l = current layer
   W[l] = weights of current layer
   b[l] = bias for the current layer
   n[l] = number of nodes in current layer
```

```
def initialize_parameters(layer_dims):
  parameters = {}
  L = len(layer_dims)

  for i in range(1, L):
    parameters['W' + str(i)] = np.random.randn(layer_dims[i], layer_dims[i-1])*0.01
    parameters['b' + str(i)] = np.zeros((1, layer_dims[i])) + 0.01

  return parameters
```

Forward propagation

- Activation function: sigmoid

Forward propagation equations:

```
Z[l] = W[l].X + b[l]
A[l] = g( Z[l] )
Where,
    Z = weighted sum of input and bias
    A = activations of particular layer
    l = layer
```

Backward propagation

Backward propagation equations:

```
Err(j)(output layer) = O(j)(1 - O(j))(T(j) - O(j))
Err(j)(hidden layer) = O(j)(1 - O(j))(SUM(Err(k)W(j,k)
del(W(i,j)) = (l)Err(j)O(i)
del(b(j)) = (l)Err(j)
Where,
      O: Output of a node
      W: weight
      b: bias
      i, j, k: nodes
```

Implementation

```
def sigmoid(X):
  return 1/(1 + np.exp(-1*X))


def computation(X, y, parameters, eta, num_iters, batch = False):
  W1_storage = []
  W2_storage = []
  b1_storage = []
  b2_storage = []
  m = X.shape[0]    # number of training examples
```

```
  for itr in range(num_iters):
    # iterate for each training example
    for i in range(m):

      # forward pass for each example
      hidden_output = sigmoid(np.dot(X[i], parameters["W1"].T) + parameters["b1"])
      final_output = sigmoid(np.dot(hidden_output, parameters["W2"].T) + parameters["b2"])

      # backward pass for each example
      dOutput = final_output*(1 - final_output)*(y[i] - final_output)
      dHidden = hidden_output*(1 - hidden_output)*np.dot(dOutput, parameters["W2"])

      # weight changes
      dW2 = eta*dOutput.reshape(-1, 1)*hidden_output
      dW1 = eta*dHidden.reshape(-1, 1)*X[i]

      # bias changes
      db2 = eta*dOutput
      db1 = eta*dHidden

      if batch == True:
        W1_storage.append(dW1)
        W2_storage.append(dW2)
        b1_storage.append(db1)
        b2_storage.append(db2)
      else:
        parameters["W2"] += dW2
        parameters["W1"] += dW1
        parameters["b2"] += db2
        parameters["b1"] += db1

    # for batch update, parameters updated here
    if batch == True:
      parameters["W2"] += sum(W2_storage)
      parameters["W1"] += sum(W1_storage)
      parameters["b2"] += sum(b2_storage)
      parameters["b1"] += sum(b1_storage)

  parameters["W2"] = np.squeeze(parameters["W2"])
  parameters["W1"] = np.squeeze(parameters["W1"])
  parameters["b2"] = np.squeeze(parameters["b2"])
  parameters["b1"] = np.squeeze(parameters["b1"])

  return parameters
```

Training and testing model

```
def train(X, y, parameters, alpha, num_iters, batch=True):
  parameters = computation(X, y, parameters, alpha, num_iters, batch)
  return parameters


def test(X, y_test, parameters):
  y_pred = []
  counter = 0

  for i in range(X.shape[0]):
    hidden_output = sigmoid(np.dot(X[i], parameters["W1"].T) + parameters["b1"])
    final_output = sigmoid(np.dot(hidden_output, parameters["W2"].T) + parameters["b2"])
    y_pred.append(final_output)


  y_pred = np.asarray(y_pred)
  #print(y_pred)
  y_pred[y_pred < 0.5] = 0
  y_pred[y_pred >= 0.5] = 1
  #print(y_pred)
  #print(y_test)

  accuracy = np.mean(np.asarray(y_pred) == y_test)
  print("Accuracy : {} %".format(accuracy*100))
```

Hand-written digits: Loading + Formatting + Training + Testing

```
def modify_label(y, n):
  new_y = []
  for i in range(y.shape[0]):
    row = np.zeros(n)
    row[y[i, 0]] = 1.
    new_y.append(row)

  return np.asarray(new_y)


data = pd.read_csv("/content/sample_data/mnist_train_small.csv", header=None)
data = data.to_numpy()

x_train = data[:1000, 1:]
y_train = data[:1000, 0]
y_train = np.expand_dims(y_train, axis=1)
y_train = modify_label(y_train, 10)
x_train = x_train / 255.0

print("Features : \n{}".format(x_train.shape))
```

```
print("Labels : \n{}".format(y_train.shape))
print("\nDataset description : ")
print("Digits : 0-9")
print("Image size : 28x28 = 784 pixels")
print("Pixel values range : 0-255")
print("Total number of images : {}".format(x_train.shape[0]))
```

```
Features :
(1000, 784)
Labels :
(1000, 10)

Dataset description :
Digits : 0-9
Image size : 28x28 = 784 pixels
Pixel values range : 0-255
Total number of images : 1000
```

```
parameters = initialize_parameters([784, 50, 10])
print("Length of parameters dictionary : {}".format(len(parameters)))
```

```
Length of parameters dictionary : 4
```

```
print("Training model...")
parameters = train(x_train, y_train, parameters, 0.01, 10)
```

```
Training model...
```

```
print("Testing model..")
data_2 = pd.read_csv("/content/sample_data/mnist_test.csv", header=None)
data_2 = data_2.to_numpy()

x_test = data_2[:100, 1:]
y_test = data_2[:100, 0]
y_test = np.expand_dims(y_test, axis=1)
y_test = modify_label(y_test, 10)
x_test = x_test / 255.0
test(x_test, y_test, parameters)
```

```
Testing model..
Accuracy : 90.0 %
```

Character recognition: Loading + Formatting + Training + Testing

```
data_3 = pd.read_csv("/content/drive/My Drive/A_Z Handwritten Data.csv", header=None)
print(data_3.describe())
```

```
data_3 = data_3.to_numpy()
X = data_3[:3000, 1:]/255.0
y = data_3[:3000, 0]
y = np.expand_dims(y, axis=1)
y = modify_label(y, 26)
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

print("Features : \n{}".format(x_train.shape))
print("Labels : \n{}".format(y_train.shape))
print("\nDataset description : ")
print("Alphabets : A-Z")
print("Image size : 28x28 = 784 pixels")
print("Pixel values range : 0-255")
print("Total number of images : {}".format(x_train.shape[0]))
print()
print("x_train shape : {}".format(x_train.shape))
print("x_test shape : {}".format(x_test.shape))
print("y_train shape : {}".format(y_train.shape))
print("y_test shape : {}".format(y_test.shape))
```

```
                   0         1    ...            783            784
count  372451.000000  372451.0  ...  372451.000000  372451.000000
mean       13.523454       0.0  ...       0.000239       0.000011
std         6.740852       0.0  ...       0.134852       0.006554
min         0.000000       0.0  ...       0.000000       0.000000
25%        10.000000       0.0  ...       0.000000       0.000000
50%        14.000000       0.0  ...       0.000000       0.000000
75%        18.000000       0.0  ...       0.000000       0.000000
max        25.000000       0.0  ...      82.000000       4.000000

[8 rows x 785 columns]
Features :
(2400, 784)
Labels :
(2400, 26)

Dataset description :
Alphabets : A-Z
Image size : 28x28 = 784 pixels
Pixel values range : 0-255
Total number of images : 2400

x_train shape : (2400, 784)
x_test shape : (600, 784)
y_train shape : (2400, 26)
y_test shape : (600, 26)
```

```
parameters = initialize_parameters([784, 50, 26])
print("Length of parameters dictionary : {}".format(len(parameters)))
```

```
Length of parameters dictionary : 4
```

```
print("Training model...")
parameters = train(x_train, y_train, parameters, 0.01, 10)
```

```
Training model...
```

```
print("Testing model...")
test(x_test, y_test, parameters)
```

```
Testing model...
Accuracy : 100.0 %
```

XOR: Loading + Formatting + Training + Testing

```
x_train = np.array([[0.1, 0.1], [0.1, 0.9], [0.9, 0.1], [0.9, 0.9]])
x_test = np.array([[0.1, 0.1], [0.1, 0.9], [0.9, 0.1], [0.9, 0.9]])
y_train = np.array([[0.1], [0.9], [0.9], [0.1]])
y_test = np.array([0, 1, 1, 0])

print("x_train shape : {}".format(x_train.shape))
print("x_test shape : {}".format(x_test.shape))
print("y_train shape : {}".format(y_train.shape))
print("y_test shape : {}".format(y_test.shape))
```

```
x_train shape : (4, 2)
x_test shape : (4, 2)
y_train shape : (4, 1)
y_test shape : (4,)
```

XOR

```
parameters = initialize_parameters([2, 2, 1])
print("Length of parameters dictionary : {}".format(len(parameters)))
```

```
Length of parameters dictionary : 4
```

```
print("Training model...")
parameters = train(x_train, y_train, parameters, 0.1, 4000)
print(parameters)
```

```
Training model...
{'W1': array([[-211.53738831, -211.46129849],
       [-143.86872965, -144.01884906]]), 'b1': array([ 38.43477338, -24.88660562]), 'W2
```

```
print("Testing model..")
test(x_test, y_test, parameters)
```

```
Testing model..
Accuracy : 75.0 %
```