DEEP LEARNING ASSIGNMENT 1

Artificial Neural Network from Scratch

Karthikeyan K 2018103549

Theory of ANN

An artificial neural network is a supervised learning algorithm which means that we provide it the input data containing the independent variables and the output data that contains the dependent variable. For instance, in our example our independent variables are X1, X2 and X3. The dependent variable is Y.

In the beginning, the ANN makes some random predictions, these predictions are compared with the correct output and the error(the difference between the predicted values and the actual values) is calculated. The function that finds the difference between the actual value and the propagated values is called the cost function. The cost here refers to the error. Our objective is to minimize the cost function. Training a neural network basically refers to minimizing the cost function. We will see how we can perform this task.

A neural network executes in two phases: Feed Forward phase and Back Propagation phase. Let us discuss both these steps in detail.

In [40]:
```
import pandas as pd
import numpy as np
import h5py

#for calculating accuracy score for the model
from sklearn.metrics import accuracy_score
```

Loading the DataSet

The dataset is comprised of photos of dogs and cats provided as a subset of photos from a much larger dataset of anually annotated photos. The dataset was developed as a partnership between Petfinder.com and Microsoft.

The DataSet is splitted into Train and Test. Train DataSet is used to train the Model and Test Dataset is used to Test the model.

In [41]:
```
def load_dataset():

    train_dataset = h5py.File('DataSets/train_catvnoncat.h5', "r")
    train_set_x_orig = np.array(train_dataset["train_set_x"][:])

    train_set_y_orig = np.array(train_dataset["train_set_y"][:])

    test_dataset = h5py.File('DataSets/test_catvnoncat.h5', "r")

    test_set_x_orig = np.array(test_dataset["test_set_x"][:])

    test_set_y_orig = np.array(test_dataset["test_set_y"][:])

    classes = np.array(test_dataset["list_classes"][:])
```

```
    return train_set_x_orig, train_set_y_orig, test_set_x_orig, test_set_y_orig, classes
```

In [42]:
```
X_train, y_train, X_test, y_test, classes = load_dataset()
```

Reshaping the DataSet

In [43]:
```
X_train = X_train.reshape(X_train.shape[0], -1)/255.
X_test = X_test.reshape(X_test.shape[0], -1)/255.
y_train = y_train.reshape(-1, 1)
y_test = y_test.reshape(-1, 1)
```

Converting the DataSet into Pandas DataFrame

In [44]:
```
X_train_dataFrame = pd.DataFrame(X_train)
```

In [45]:
```
X_train_dataFrame.head()
```

Out[45]:

	0	1	2	3	4	5	6	7	8	9	...	12
0	0.066667	0.121569	0.219608	0.086275	0.129412	0.231373	0.098039	0.137255	0.243137	0.098039	...	0.000
1	0.768627	0.752941	0.745098	0.756863	0.729412	0.713725	0.737255	0.701961	0.682353	0.835294	...	0.313
2	0.321569	0.278431	0.266667	0.349020	0.325490	0.325490	0.392157	0.384314	0.407843	0.415686	...	0.670
3	0.003922	0.086275	0.007843	0.003922	0.054902	0.007843	0.003922	0.050980	0.003922	0.015686	...	0.105
4	0.035294	0.035294	0.019608	0.039216	0.035294	0.023529	0.035294	0.035294	0.023529	0.035294	...	0.066

5 rows × 12288 columns

Sigmoid Function

Applies sigmoid function to an array

In [46]:
```
def sigmoid(Z):

  return 1/(1+np.power(np.e, -Z))
```

sigmoid prime

Applies differentiation of sigmoid function to an array

In [47]:
```
def sigmoid_prime(Z):

    return (1-np.power(Z, 2))
```

Forward propagation

In order to generate some output, the input data should be fed in the forward direction only. The data should not flow in reverse direction during output generation otherwise it would form a cycle and the output could never be

generated. Such network configurations are known as feed-forward network. The feed-forward network helps in forward propagation.

Performs forward propagation and calculates output value

In [48]:
```
def forward_prop(X, params):

    w = params['w']

    b = params['b']
    z = np.dot(X, w) + b

    a = sigmoid(z)

    return {'z': z, 'a': a}
```

Back propagation

Backpropagation is the essence of neural network training. It is the method of fine-tuning the weights of a neural network based on the error rate obtained in the previous epoch (i.e., iteration). Proper tuning of the weights allows you to reduce error rates and make the model reliable by increasing its generalization

Performs backward propagation and calculates dw and db

In [49]:
```
def backward_prop(X, y, cache):

    z = cache['z']

    a = cache['a']
    m = X.shape[0]

    dz = a - y
    dw = (1./m)*np.dot(X.T, dz)

    db = (1./m)*np.sum(dz)

    #Dictionary containing gradients 'dz', 'dw' and 'db'

    return {'dz': dz, 'dw': dw, 'db': db}
```

Updating the weights of the layer

Updates weights of the layers

In [50]:
```
def update_weights(params, changes, learning_rate=0.01):

    w = params['w']
    b = params['b']
    dw = changes['dw']
    db = changes['db']

    w -= learning_rate*dw
    b -= learning_rate*db

    #Dictionary containing updated weights and biases
    #The keys for weights and bias arrays in the dict is 'w' and 'b'
```

```
    return {'w': w, 'b': b}
```

Calculating the Loss

Calculate the entropy loss

In [51]:
```
def calculate_loss(cache, y):

    a = cache['a']
    m = y.shape[0]
    return -1/m*np.sum(xlogy(y, a) + xlogy(1-y, 1-a))
```

Initializing the Artificial Neural Network

Initializes random weights and bias

In [52]:
```
def initialize_nn(X):

    np.random.seed(999)

    w = np.random.randn(X.shape[1], 1) * 0.01
    b = 0

    return {'w': w, 'b': b}
```

Training the model

In [53]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(209, 12288)
(209, 1)
(50, 12288)
(50, 1)
```

Epoch

In terms of artificial neural networks, an epoch refers to one cycle through the full training dataset. Usually, training a neural network takes more than a few epochs. In other words, if we feed a neural network the training data for more than one epoch in different patterns, we hope for a better generalization when given a new "unseen" input (test data). An epoch is often mixed up with an iteration

Learning Rate

In machine learning and statistics, the learning rate is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function. Since it influences to what extent newly acquired information overrides old information, it metaphorically represents the speed at which a machine learning model "learns". In the adaptive control literature, the learning rate is commonly referred to as gain

Epoch is set to 1000 and Learning Rate is set to 0.005

In [58]:
```
epochs = 1000
learning_rate = 5e-3
```

In [59]:
```
params = initialize_nn(X_train)

for i in range(epochs):

    cache = forward_prop(X_train, params)

    loss = calculate_loss(cache, y_train)

    updates = backward_prop(X_train, y_train, cache)

    params = update_weights(params, updates, learning_rate=learning_rate)

    if i%(epochs/10) == 0:

        print('Epoch: {}\tLoss:{:.5f}'.format(i, loss), end='')
        train_cache = np.where(cache['a']>0.5, 1, 0)

        print('\tTraining accuracy:{:.5f}'.format(accuracy_score(y_train, train_cache)), 
        test_cache = forward_prop(X_test, params)['a']

        test_cache = np.where(test_cache>=0.5, 1, 0)

        print('\tTesting accuracy:{:.5f}'.format(accuracy_score(y_test, test_cache)))
```

```
Epoch: 0        Loss:0.72463    Training accuracy:0.45455       Testing accuracy:0.34000
Epoch: 100      Loss:0.58224    Training accuracy:0.68421       Testing accuracy:0.34000
Epoch: 200      Loss:0.46335    Training accuracy:0.81818       Testing accuracy:0.44000
Epoch: 300      Loss:0.37246    Training accuracy:0.89952       Testing accuracy:0.62000
Epoch: 400      Loss:0.32947    Training accuracy:0.91866       Testing accuracy:0.70000
Epoch: 500      Loss:0.30155    Training accuracy:0.92823       Testing accuracy:0.72000
Epoch: 600      Loss:0.27835    Training accuracy:0.93780       Testing accuracy:0.74000
Epoch: 700      Loss:0.25866    Training accuracy:0.94258       Testing accuracy:0.74000
Epoch: 800      Loss:0.24167    Training accuracy:0.95694       Testing accuracy:0.74000
Epoch: 900      Loss:0.22683    Training accuracy:0.96172       Testing accuracy:0.76000
```

In [ ]:

In [ ]: