ASSIGNMENT 01

COMSATS University Islamabad

Sahiwal Campus

Usama Sarwar

FA17-BS(CS)-090-B

Ms. Raheela

Pattern Recognition

October 17, 2020

Table of Contents

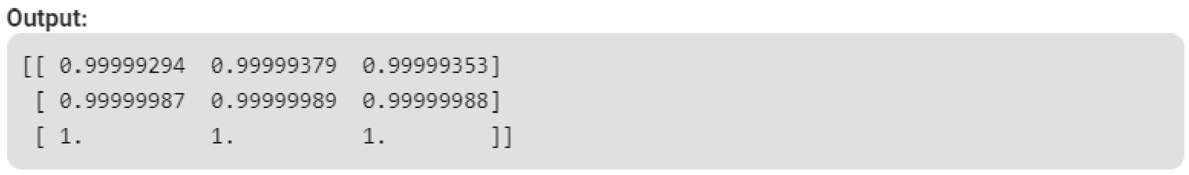
1. Implementation of Simple Artificial Neural Network

Neural networks are artificial systems that were inspired by biological neural networks. These systems learn to perform tasks by being exposed to various datasets and examples without any task-specific rules. The idea is that the system generates identifying characteristics from the data they have been passed without being programmed with a pre-programmed understanding of these datasets. Neural networks are based on computational models for threshold logic. Threshold logic is a combination of algorithms and mathematics. Neural networks are based either on the study of the brain or on the application of neural networks to artificial intelligence. The work has led to improvements in finite automata theory.

1.1 Code:

```
import numpy as np

# array of any amount of numbers. n = m
X = np.array([[1, 2, 3],
              [3, 4, 1],
              [2, 5, 3]])

# multiplication
y = np.array([[.5, .3, .2]])

# transpose of y
y = y.T

# sigma value
sigm = 2

# find the delta
delt = np.random.random((3, 3)) - 1

for j in range(100):
    # find matrix 1. 100 layers.
    m1 = (y - (1/(1 + np.exp(-(np.dot((1/(1 + np.exp(
                 -(np.dot(X, sigm))))), delt))))))*((1/(
                     1 + np.exp(-(np.dot((1/(1 + np.exp(
                 -(np.dot(X, sigm))))), delt)))))*(1-(1/(
                     1 + np.exp(-(np.dot((1/(1 + np.exp(
                 -(np.dot(X, sigm))))), delt)))))))

    # find matrix 2
    m2 = m1.dot(delt.T) * ((1/(1 + np.exp(-(np.dot(X, sigm)))))
                  * (1-(1/(1 + np.exp(-(np.dot(X, sigm)))))))
    # find delta
    delt = delt + (1/(1 + np.exp(-(np.dot(X, sigm))))).T.dot(m1)

    # find sigma
    sigm = sigm + (X.T.dot(m2))

# print output from the matrix
print(1/(1 + np.exp(-(np.dot(X, sigm)))))
```

1.2 Output



Output:

```
[[ 0.99999294  0.99999379  0.99999353]
 [ 0.99999987  0.99999989  0.99999988]
 [ 1.          1.          1.        ]]
```

2. Implementation of Artificial Neural Network for AND Logic Gate with 2-bit Binary Input

Artificial Neural Network (ANN) is a computational model dependent on the biological neural networks of animal cerebrums. ANN is modeled with three kinds of layers: an information layer, concealed layers (at least one), and a yield layer. Each layer contains hubs (like biological neurons) are called Artificial Neurons. All hubs are associated with weighted edges (like neural connections in biological minds) between two layers. Initially, with the forward engendering capacity, the yield is anticipated. At that point through backpropagation, the weight and predisposition to the hubs are refreshed to limiting the blunder in forecast to accomplish the intermingling of cost work in deciding the final yield. AND logical function truth table for 2-bit binary variables i.e., the input vector ***x: (x_1, x_2)*** and the corresponding output ***y*-**

x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

2.1 Approach:

Step1: Import the required Python libraries

Step2: Define Activation Function: Sigmoid Function

Step3: Initialize neural network parameters (weights, bias)

and define model hyperparameters (number of iterations, learning rate)

Step4: Forward Propagation

Step5: Backward Propagation

Step6: Update weight and bias parameters

Step7: Train the learning model

Step8: Plot Loss value vs Epoch

Step9: Test the model performance

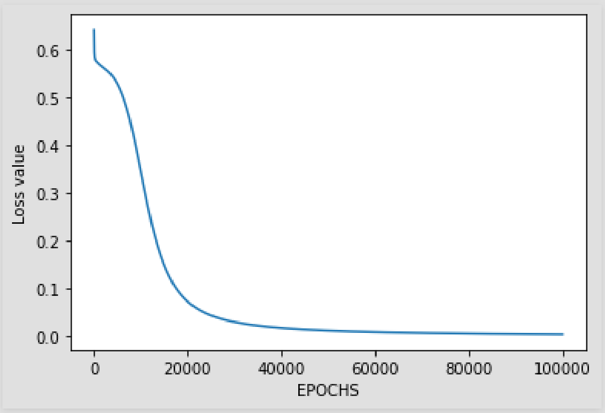
3. Python Code

```
# import Python Libraries
import numpy as np
from matplotlib import pyplot as plt

# Sigmoid Function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Initialization of the neural network parameters
# Initialized all the weights in the range of between 0 and 1
# Bias values are initialized to 0
def initializeParameters(inputFeatures, neuronsInHiddenLayers,
outputFeatures):
    W1 = np.random.randn(neuronsInHiddenLayers, inputFeatures)
    W2 = np.random.randn(outputFeatures, neuronsInHiddenLayers)
    b1 = np.zeros((neuronsInHiddenLayers, 1))
    b2 = np.zeros((outputFeatures, 1))

    parameters = {"W1" : W1, "b1": b1,
                  "W2" : W2, "b2": b2}
    return parameters

# Forward Propagation
def forwardPropagation(X, Y, parameters):
    m = X.shape[1]
    W1 = parameters["W1"]
    W2 = parameters["W2"]
    b1 = parameters["b1"]
    b2 = parameters["b2"]

    Z1 = np.dot(W1, X) + b1
    A1 = sigmoid(Z1)
    Z2 = np.dot(W2, A1) + b2
    A2 = sigmoid(Z2)

    cache = (Z1, A1, W1, b1, Z2, A2, W2, b2)
    logprobs = np.multiply(np.log(A2), Y) + np.multiply(np.log(1 - A2), (1
- Y))
    cost = -np.sum(logprobs) / m
    return cost, cache, A2

# Backward Propagation
def backwardPropagation(X, Y, cache):
    m = X.shape[1]
    (Z1, A1, W1, b1, Z2, A2, W2, b2) = cache

    dZ2 = A2 - Y
    dW2 = np.dot(dZ2, A1.T) / m
    db2 = np.sum(dZ2, axis = 1, keepdims = True)

    dA1 = np.dot(W2.T, dZ2)
    dZ1 = np.multiply(dA1, A1 * (1- A1))
    dW1 = np.dot(dZ1, X.T) / m
    db1 = np.sum(dZ1, axis = 1, keepdims = True) / m

    gradients = {"dZ2": dZ2, "dW2": dW2, "db2": db2,
                 "dZ1": dZ1, "dW1": dW1, "db1": db1}
    return gradients

# Updating the weights based on the negative gradients
def updateParameters(parameters, gradients, learningRate):
    parameters["W1"] = parameters["W1"] - learningRate * gradients["dW1"]
    parameters["W2"] = parameters["W2"] - learningRate * gradients["dW2"]
    parameters["b1"] = parameters["b1"] - learningRate * gradients["db1"]
    parameters["b2"] = parameters["b2"] - learningRate * gradients["db2"]
    return parameters

# Model to learn the AND truth table
X = np.array([[0, 0, 1, 1], [0, 1, 0, 1]]) # AND input
Y = np.array([[0, 0, 0, 1]]) # AND output

# Define model parameters
neuronsInHiddenLayers = 2 # number of hidden layer neurons (2)
inputFeatures = X.shape[0] # number of input features (2)
outputFeatures = Y.shape[0] # number of output features (1)
parameters = initializeParameters(inputFeatures, neuronsInHiddenLayers,
outputFeatures)
epoch = 100000
learningRate = 0.01
losses = np.zeros((epoch, 1))

for i in range(epoch):
    losses[i, 0], cache, A2 = forwardPropagation(X, Y, parameters)
    gradients = backwardPropagation(X, Y, cache)
    parameters = updateParameters(parameters, gradients, learningRate)

# Evaluating the performance
plt.figure()
plt.plot(losses)
plt.xlabel("EPOCHS")
plt.ylabel("Loss value")
plt.show()

# Testing
X = np.array([[1, 1, 0, 0], [0, 1, 0, 1]]) # AND input
cost, _, A2 = forwardPropagation(X, Y, parameters)
prediction = (A2 > 0.5) * 1.0
# print(A2)
print(prediction)
```

4. Output

```
Traceback (most recent call last):
  File "/home/713ef48fef500d9177705ff6d5a3dcf4.py", line 2, in <module>
    import numpy as np
  File "/usr/lib/python3/dist-packages/numpy/__init__.py", line 180, in <module>
    from . import add_newdocs
  File "/usr/lib/python3/dist-packages/numpy/add_newdocs.py", line 13, in <module>
    from numpy.lib import add_newdoc
  File "/usr/lib/python3/dist-packages/numpy/lib/__init__.py", line 8, in <module>
    from .type_check import *
  File "/usr/lib/python3/dist-packages/numpy/lib/type_check.py", line 11, in <module>
    import numpy.core.numeric as _nx
  File "/usr/lib/python3/dist-packages/numpy/core/__init__.py", line 14, in <module>
    from . import multiarray
ImportError: cannot import name 'multiarray'
```

Output:



```
[[ 0.  1.  0.  0.]]
```

Here, the model predicted output for each of the test inputs are exactly matched with the AND logic gate conventional output (***y***) according to the truth table and the cost function is also continuously converging. Hence, it signifies that the Artificial Neural Network for the AND logic gate is correctly implemented.