Homework 1 Solutions
CSCE4613

Morgan Maness and Chris Troupe

Program Output

Training model
Predicting
Neural Network prediction for example (1, 1) is 0
Neural Network prediction for example (0, 1) is 1
Neural Network prediction for example (1, 0) is 1
Neural Network prediction for example (0, 0) is 0

The Software

My Code

Morgan Maness and Chris Troupe
12/4/2019
CSCE 4613
Assignment 4

```
import numpy as np

def sig(x):
return 1/(1 + np.exp(-x))
sigmoid activation function

def initialize(first, hidden, output):
initializeing parameters
weighted paramaters
W1 = np.random.randn(hidden, first)
W2 = np.random.randn(output, hidden)
biases
b1 = np.zeros((hidden, 1))
b2 = np.zeros((output, 1))

parameters =
"W1": W1,
"b1" : b1,
"W2": W2,
"b2" : b2

return parameters

def forwardProp(X, parameters):
takes neural network input matrix and paramaters dictionary
again initiliazing paramaters
W1 = parameters["W1"]
b1 = parameters["b1"]
W2 = parameters["W2"]
b2 = parameters["b2"]
start calculations of network A2 to be used in back propagation
Z1 = np.dot(W1, X) + b1
A1 = np.tanh(Z1)
Z2 = np.dot(W2, A1) + b2
A2 = sig(Z2)
setting paramaters to be used later
cache =
"A1": A1,
"A2": A2

return A2, cache

def backwardProp(X, Y, cache, parameters):
initialize needed variabls
A1 = cache["A1"]
A2 = cache["A2"]
W2 = parameters["W2"]
calculate for backward propagation
dZ2 = A2 - Y
dW2 = np.dot(dZ2, A1.T)/m
db2 = np.sum(dZ2, axis=1, keepdims=True)/m
dZ1 = np.multiply(np.dot(W2.T, dZ2), 1-np.power(A1, 2))
dW1 = np.dot(dZ1, X.T)/m
db1 = np.sum(dZ1, axis=1, keepdims=True)/m
setting paramaters to be used later
grads =
"dW1": dW1,
"db1": db1,
"dW2": dW2,
"db2": db2

return grads

def updateParam(parameters, grads, learningRate):
```

```
updating paramaters and making the model learn using learnRate
W1 = parameters["W1"]
b1 = parameters["b1"]
W2 = parameters["W2"]
b2 = parameters["b2"]
dW1 = grads["dW1"]
db1 = grads["db1"]
dW2 = grads["dW2"]
db2 = grads["db2"]

   W1 = W1 - learningRate*dW1
W2 = W2 - learningRate*dW2
b1 = b1 - learningRate*db1
b2 = b2 - learningRate*db2
newParam =
"W1": W1,
"W2": W2,
"b1" : b1,
"b2" : b2

return newParam

   def model(X, Y, first, hidden, output, numIterators, learnRate):
calculates and returns trained paramaters of the model
parameters = initialize(first, hidden, output)
for i in range(0, numIterators+1):
a2, cache = forwardProp(X, parameters)
grads = backwardProp(X, Y, cache, parameters)
parameters = updateParam(parameters, grads, learnRate)
return parameters

   def predict(X, parameters):
a2, cache = forwardProp(X, parameters)
yhat = a2
yhat = np.squeeze(yhat)
if(yhat ¿= 0.5):
output = 1
else:
output = 0
return output

   np.random.seed(2)
The 4 training examples by columns
X = np.array([[0, 0, 1, 1], [0, 1, 0, 1]])
The outputs of the XOR for every example in X
Y = np.array([[0, 1, 1, 0]])
No. of training examples
m = X.shape[1]

   Set the hyperparameters
first = 2 No. of neurons in first layer
hidden = 2 No. of neurons in hidden layer
output = 1 No. of neurons in output layer
iterators = 1000
learnRate = 0.3

   trained = model(X, Y, first, hidden, output, iterators, learnRate)
print('Training model')

   Test 2X1 vector to calculate the XOR of its elements.
aTest = np.array([[1], [1]])
bTest = np.array([[0], [1]])
cTest = np.array([[1], [0]])
dTest = np.array([[0], [0]])

   print('Predicting')
print('Neural Network prediction for example (:d, :d) is :d'.format( aTest[0][0], aTest[1][0], predict(aTest, trained)))
print('Neural Network prediction for example (:d, :d) is :d'.format( bTest[0][0], bTest[1][0], predict(bTest, trained)))
print('Neural Network prediction for example (:d, :d) is :d'.format( cTest[0][0], cTest[1][0], predict(cTest, trained)))
print('Neural Network prediction for example (:d, :d) is :d'.format( dTest[0][0], dTest[1][0], predict(dTest, trained)))
```

Design

My program starts out by creating the sigmoid activation function. Then it initializes some of the first used variables. Then I go on to forward propagation, using the variables I just initialized and returning the calculation and the cache. Then I do backwards propagation, returning the calculated variables. I then have a way to update my paramaters, then I go to training my model. once my model is trained i go to being able to predict.

Nodes

The number of layers or nodes can be changed by changed by changing the variables labeled first and hidden, which correspond to my first two layers.

The Training Process

Network Configuration

I decided to use this network configuration because it seemed to be the most straight forward implementation of the project

Weights

The weights are initialized in variables called b1 and b2, which are created in the class initialize.

Iterations

My program ran 1000 interations

Using Training Data

The training data was used in my variables to set the variables that would be able to calculate the the predictions.

Creating Training Data

The training data was created using the matrices labeled X and Y

The End Results

Convergence

The network converged 100 percent because all of the predictions were correct

Correctness

All of the training data is classified correctly