CMPE 489 - COGS 500

Neural Networks Project Report

Uğurcan Arıkan

2014400111

Boğaziçi University Computer Engineering

Fall 2018

CMPE 489 PROJECT REPORT

Introduction

In this project, the goal was to implement a multilayer neural network with at least one hidden layer and to train it with different datasets successfully using back propagation algorithm described in the lecture.

The solution that has been implemented is a multilayer neural network with 1 hidden layer, 1 input layer and 1 output layer with all three having the ability to be assigned different number of neurons in a hardcoded way, ie user of the program may decide number of neurons each layer possesses, but cannot decide how many hidden neurons will reside between the output layer and the input layer.

The project implemented can successfully feed forward and give the output according to the current weights and the given input, can use the back propagation algorithm and learn about the function proposed to it and update its weights accordingly and can calculate the error function and give a chart of the error function after each iteration of the training.

Implementation Details

The neural network implemented in the project consists of three main classes: neuron, layer and network. Neuron is the most basic structure in the network, layer consists of neurons and network consists of layers.

Neuron

Neuron is the most basic class in the neural network. It has four variables:

bias : Bias assigned to the neuron. Default is 0.

weights : Array of weights of the inputs of the neuron

inputs : Array of input values of the neuron

output : output of the neuron according to the weights, inputs and bias

Neuron also has 8 functions:

__init__(self, bias) : initializes the neuron with the given bias

calculate_output(self, inputs) : Calculates the output of the neuron according to the given inputs

squash_with_tanh(self, net) : Squashes the given net value using tanh function

squash_with_sigmoid(self, net) : Squashes the given net value using sigmoid function

calculate_net(self) : Calculates the net of the neuron which is the weighted sum of all the inputs of the neuron

calculate_delta(self, desired_output) : Calculates the delta value of the neuron according to given desired output and the output value of the neuron itself. It is used only for the output neurons.

calculate_error(self, desired_output) : Calculates the error of the neurons output according to the desired output.

sigmoid(self, x) : Sigmoid function. Calculates the sigmoid of the given x.

Layer

Layer consists of neurons and is constitutes the network. It has 2 main variables:

bias : Bias value of the layer. It is assigned to all the neurons in the layer.

neurons: Array containing all the neurons in the layer.

Layer also has 3 functions:

__init__(self, neuron_number, bias) : Initializes the layer with the given bias and creates the neurons array whose length is the given neuron_number.

initialize_weights(self, input_number) : Initializes the weights of all the neurons in the layer according to their input and weight number. It creates weights with normal distribution whose mean is 0 and standard deviation is 1.

feed_forward(self, inputs) : Feeds forward according to the inputs and the current weights of all of the layers neurons and returns an output array.

Network

Network is the main data structure of the project. It consists of layers which in turn consists of neurons. It has 3 variables:

alpha : learning rate of the network. It is set to 0.15 as after many trials, it has been decided that it works the best with the current algorithm and datasets.

hidden_layer : Hidden layer of the network.

output_layer : Output layer of the network.

Network also consists of 4 functions:

__init__(self, input_neuron_number=3, hidden_neuron_number=4, output_neuron_number=1, hidden_bias=0, output_bias=0) : Initializes the network. Creates the hidden layer with hidden_neuron_number and hidden_bias and initializes its weights. Creates output layer with output_neuron_number and output_bias and initializes its weights.

feed_forward(self, inputs) : Feeds forward first its hidden layer with the given inputs and then output layer with the outputs obtained from the hidden layer.

backpropagate(self, inputs, desired_output) : Backpropagates according to the given input set and its desired output. It first feeds forward, then obtains delta values of the output layer, then delta values of the hidden layer and updates those layers' weights according to the backpropagation algorithm and formulas discussed in the neural network lecture.

train(self, inputs, desired_outputs) : Takes a dataset inputs which is an array of array of inputs and their desired outputs and back propagates for all the inputs given.

calculate_error(self, desired_output) : Calculates the error according to the given desired output and current outputs of the output layer neurons and returns the average of the sum of their errors.

How the Error is Minimized

Inputs and outputs are fed to the network as array of arrays in a hard coded way. Network trains with the given input and output dataset by using back propagate for each of their elements by first using feed forward, then calculating the delta values of the output layer neurons and delta values of the hidden layer neurons. Then the weights of the output layer are updated which is followed by the update of the hidden layer neurons' weights.

Output layer neuron delta = (desired_output - self.output) * (-1) * (1 - self.output * self.output)

hidden layer neuron delta =

(weighted sum of weights and their output neuron deltas) * (1 - output * output)

All the weights are updated according to the formula

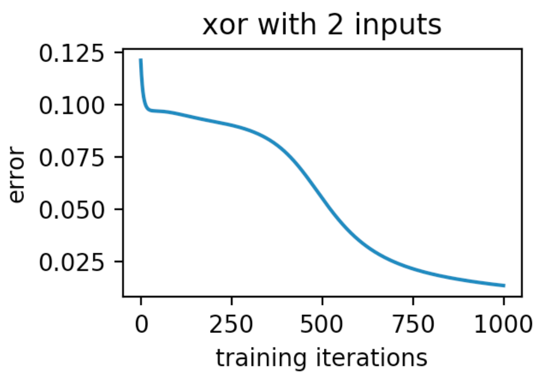
alpha * (target neuron's delta) * (source neuron's input)

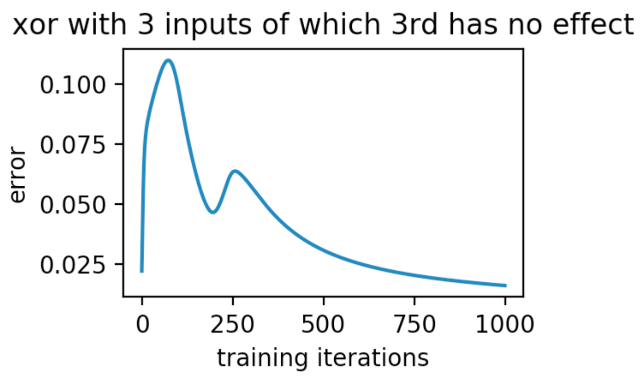
How it was tested

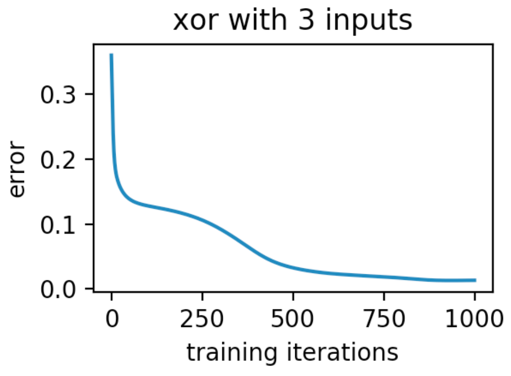
In all the tests above, network is given a set of inputs and outputs and was trained with them for 1000 iterations. At each iteration, network was also given a single input to feed forward and its error was calculated. Each error value obtained in those iterations are kept and plotted against the iteration/training number.

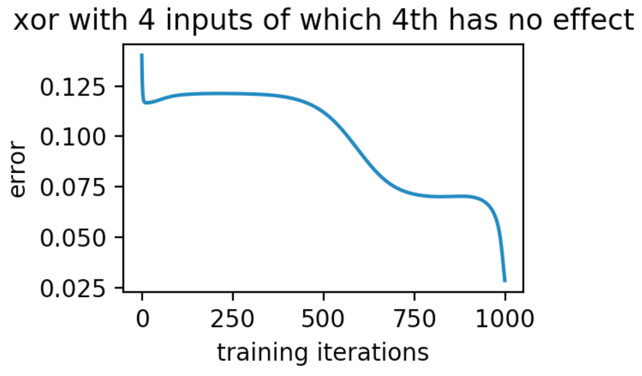
What was Network Able to Learn

As expected and as it is the case in the single layer neural networks, implemented neural network had no problem learning the “OR” and “AND” functions. The main goal was being able to learn “XOR” function which is a task single layered neural networks fail. After several tests involving xor of 2 inputs, xor of 3 inputs, xor of first 2 inputs and 3rd input which has no effect to the output and xor of first 2 inputs and 3rd and 4th inputs which have no effect to the output, it has been concluded that the network was able to learn the “XOR” function successfully and error is minimized as the training continued.







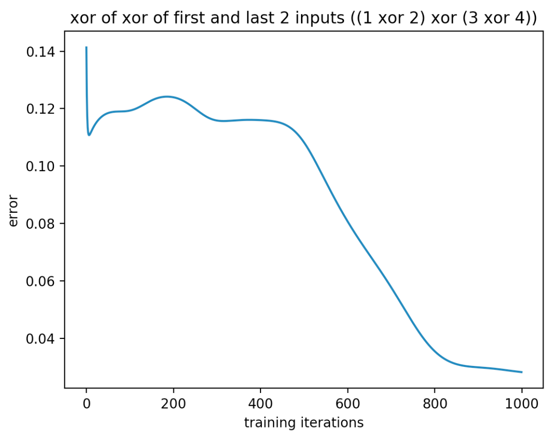


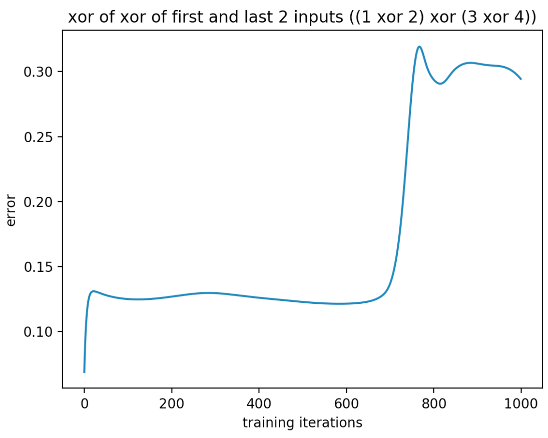
What was NOT Network Able to Learn

To make the things harder, I have also tested the network with 4 inputs and 1 outputs whose value is equal to xor of the first and last inputs xored, in other words, output was

(input1 XOR input2) XOR (input3 XOR input4)

Network was not that successful at learning this new algorithm in some cases. After several trials, it failed in almost half of them. Below there is 1 chart for each cases.





Notes about Training

As the weights are randomly initialized, there might be some cases where error is not minimized very successfully as expected but those cases were very rare (1 out of 20 in my own experiences).

Also number of training iterations helps the network to some extent, higher number of training results better network learning to some number and then starts to degrade the networks learning. In my experiences with the network I have built, the optimal number was around 1000 iterations. After around 5000 training iterations, network's learning seriously degraded and resulted higher error values.

How to Run

run with **python3 main.py** in the same directory as the main file. It was written with python 3.6.1 in MacOS Mojave 10.14.2.

It has **numpy** and **matplotlib** dependencies. They should be installed with commands

pip3 install numpy and **pip3 install matplotlib**

Conclusion

To conclude, I have successfully built a multilayer neural network with 1 hidden layer and managed to update its weights with the back propagation algorithm discussed in the lecture, in order to minimize the error function, again discussed in the lecture.