ECE 5332-011: Deep Learning for Medical Signal/Image data

Assignment 2: Implementation of a simple Neural Network

Spring 2019 Due: 03/06/2019

1. The perceptron algorithm (30 points)

In this part we would be seeing how the smallest functional unit in any Neural Network; the perceptron, work. The perceptron basically takes weighted inputs from more than one inputs, add them together and passes that through a non-linear activation function. The structure of a simple perceptron is shown in figure 1. Here X's are the inputs, b is the bias and Y is the output. Mathematically, the output 'y' is computed in the following manner:

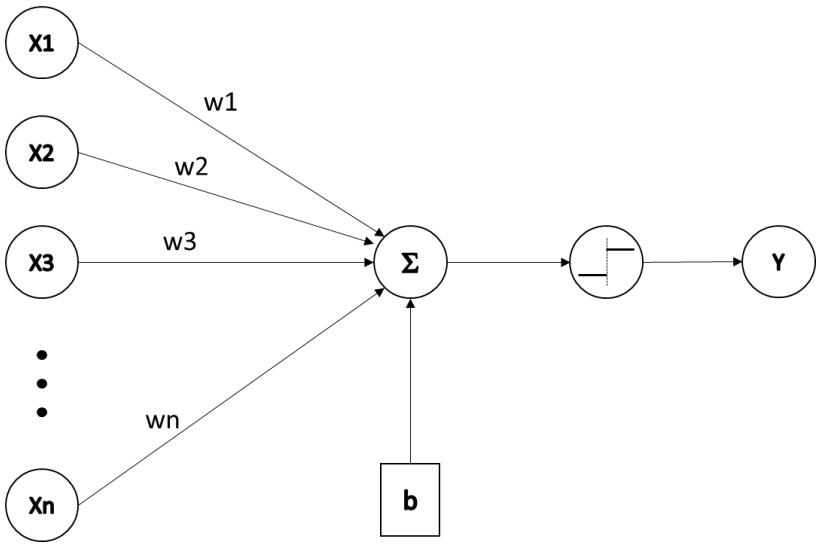

Figure 1: Perceptron

$Y = f(X1*w1 + X2*w2 + \ldots Xn*wn + b)$, where $f(z)$ is the activation function of z.

Here we would use the perceptron and implement simple 2-bit logic gates. Because the output of a logic gate can take only binary values of either 1 or 0, we will use a step activation function. A step activation outputs a 1 for an input of greater than zero and outputs a zero for all other inputs.

For training the perceptron, we would be using a single example at a time. The weight update equation is given as follow:

$$W_{k+1}^{(i)} = W_k^{(i)} + \rho * [Y_k - \bar{Y}_k] * X_k^{(i)}$$

Where 'ρ' is the learning rate, 'k' is the iteration count, and 'i' is the feature number. For a 2-bit input with 1 bias, the value of i = {1,2,3}. The whole process of training a perceptron for a 2-bit logic gate can be summarized in the following manner:

i. Define inputs and outputs
ii. Set initial weights
iii. Compute output using weights and compute error
iv. If error is not zero, update weights
v. Jump to iii until error is zero.

Use the above defined perceptron algorithm to implement 2-bit AND, OR, NAND, NOR and XOR gates. For each case vary the learning rate from 0.0001 to 1 in orders of 10. Report the average number of iterations taken for convergence.

Can all the 5 gates be implemented using this approach? If not provide a suitable reason why some logic gates cannot be implemented using this approach.

2. Neural Network

A Neural Network is simply a collection of interconnected perceptrons. Here we may have more than one layers between the input and output layers. Such layers are known as hidden layers. A simple neural network is shown in figure 2. Just like a single perceptron, a Neural Network can be trained to perform more complicated task that a single perceptron cannot do. A link to a YouTube playlist explaining Neural Networks in detail along with implementation in Python is given below:

[https://www.youtube.com/playlist?list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU]

As shown in the playlist, we would use a backpropagation algorithm to update the weights and train the network. The equation for the weight update is as shown below:

$$W_{k+1}^{(i)} = W_k^{(i)} - \rho * \frac{\partial J}{\partial W^{(i)}}$$

where J is the cost function

$$J = \frac{1}{2}\sum_{n=1}^{N}(y_n - \bar{y}_n)^2$$

$\frac{\partial J}{\partial W^{(i)}}$ is the partial derivative of the cost function w.r.t the weights after the i^{th} layer.

$$\frac{\partial J}{\partial W^{(i)}} = (a^{(i)})^T \bullet \delta^{(i+1)};$$

$$\delta^{(i)} = [\delta^{(i+1)} \bullet (W^{(i)})^T] * f\,`(z^{(i)});$$

$$\delta^{(out)} = (\bar{Y} - Y) * f\,`(z^{(out)})$$

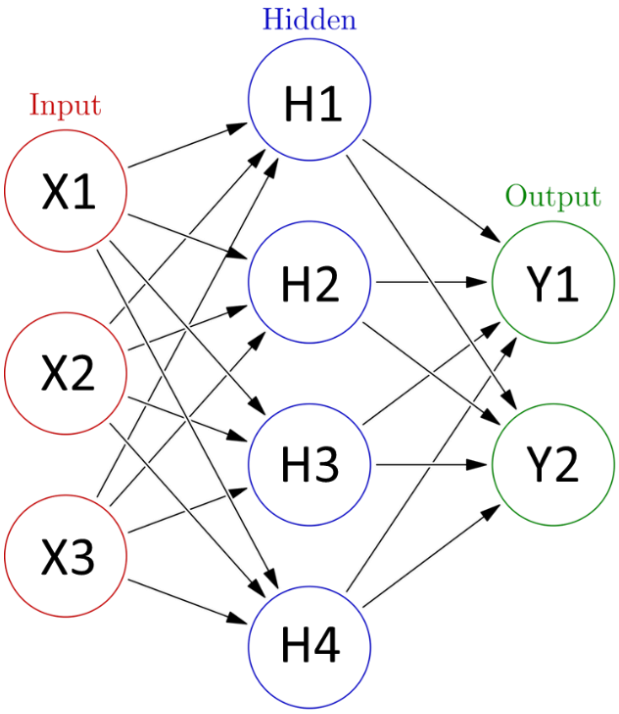

Figure 2: Simple Neural Network

Here $a^{(i)}$ is the output of the i^{th} layer; $\delta^{(i)}$ is the error at the i^{th} layer; $z^{(i)}$ is the input to the i^{th} layer and f `(z) is the derivative of the activation function f (z), w.r.t *'z'*. For this part of the assignment we would use sigmoid activation function. The symbol • is a matrix multiplication while * is the element wise scalar multiplication between two matrices respectively.

a. 2-bit logic gates **(20 points)**

Here we would try to implement 2-bit logic gates using a neural network. Unlike a traditional logic gate where we have a single output, the neural network will treat this as a classification problem and give 2 outputs; p(0) and p(1). Where p(x) denotes probability of event to be 'x'. Thus, the output of this neural network logic gate would be the one with the highest probability.

Train a simple neural network with one hidden layer and try and implement 2-bit AND, OR, NAND, NOR and XOR gates. Play around with the number of nodes in the hidden layers. Report the average number of iterations taken for convergence for each logic gate.

Can all the 5 gates be implemented using this approach? If not provide a suitable reason why some logic gates cannot be implemented using this approach.

[**BONUS**: Try and implement a simple 2-bit binary adder which takes 2 input bits and output a sum bit (S) and a carry bit (C). Describe the structure of the network and report different parameters used to implement it.]

b. Classification of Iris dataset **(20 points)**

In this part we would train a model to classify the Fisher's iris dataset. The input data has 4 measurement and the output should be one of the 3 categories of flower. So, there should be 3 nodes in the output layer each outputting the probability of a given input to be in a particular category of flower. Train a separate network for each of the 4 Training/Test cases and report the accuracy. Cases : {50/100, 75/75, 100/50, 125/25}

c. Deep network **(30 points)**

In this part, we would approach the same Fisher iris classification problem but with a network with 2 hidden layers. Repeat the same 4 Training/Test cases and compare the accuracy obtained by adding an additional hidden layer. Does the addition of a hidden layer impact the average number of iterations used to train the network?

[Note: Use randomizer to create training and test set from the iris dataset.]

Submission:

You are expected to work in groups of three. Please upload a single zip file containing a brief report either in word or PDF and a MATLAB or python code for each part of the assignment. Please include necessary figures and tables in the report with proper captions.