Lab0 - Warm Up

Wen-Jie Tseng (0556146)
Deep Learning and Practice - 2018 Spring Semester

1 Introduction

In this report, I will simply introduce my understanding about sigmoid function, neural networks, and backpropagation. Some descriptions about the implementation were also included. Two neural networks with one hidden layer and two hidden layers were implemented in nn-xor.py and 2lnn-xor.py respectively. Boldface in the following text represents the code segments of Python. For more details of the implementation, see the source code (nn-xor.py and 2lnn-xor.py) in .zip file.

2 Experiment Setups

2.1 Sigmoid Functions

Proof of the derivative of sigmoid function, first we have:

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{1}$$

Next, find the derivative of sigmoid function with a tricky step, add one minus one, then we can obtain:

$$\frac{d}{dx}\sigma(x) = \frac{e^{-x}}{(1+e^{-x})^2} = (\frac{1+e^{-x}-1}{1+e^{-x}})(\frac{1}{1+e^{-x}}) = (1-\sigma(x))(\sigma(x)) \tag{2}$$

The equation (2) helps us in finding gradients of neural networks.

2.2 Neural Networks

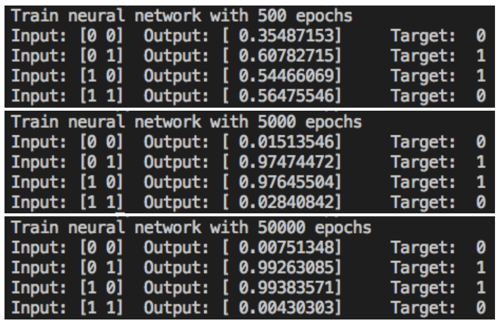
The neural networks was implemented as a **neural_network** class, which contains the neurons declared in **__init__**. There were two weight matrices, from input layer to hidden layer and from hidden layer to output layer. The activation units for input, hidden, and output layers are also declared in **__init__**. The weight matrices must be assigned some randomized values through the **randomized_matrix** function. Note that in 2lnn-xor.py, I replaced **randomized_matrix** with **numpy.random.randn**. This function can sample an array of values from standard normal distribution. At last, two arrays for recording weights in backpropagation were created. All the variables above were declared as array type in numpy.

2.3 Backpropagation

The process of backpropagation was divided into two functions, 1) **forward_propagation** and 2) **backward_propagation**. The **forward_propagation** means computing the final output of a network. The input unit has to multiply weight matrix between input layer and hidden layer. Update these values to units in hidden layer. Numpy provides arithmetic within arrays saving much efforts instead of using for loop. In **backward_propagation**, we are ultimately interesting in the gradient of each input neuron. By chain rule, we can derive the gradient of each neuron from the output layer, hidden layer, and back to input layer. Multiply the gradient and activation unit of each layer, we can update this value into weight matrix. One complete process including forward and backward is named as epoch. We can run a certain amount of epochs, and check the training result.

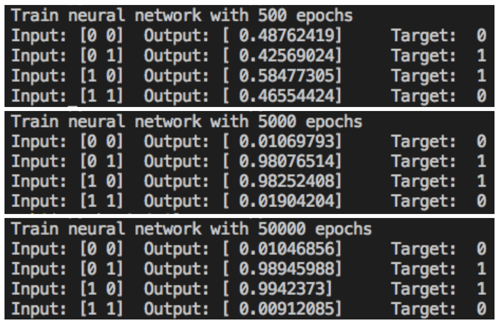
3 Results

I trained two networks with 500, 5000, and 50000 epochs, as shown in Figure 1. Both networks performed poorly with 500 epochs setting. While increasing epochs from 500 to 5000 and 50000, one can observe the outputs converged towards targets.



```
Train neural network with 500 epochs
Input: [0 0]  Output: [ 0.35487153]     Target:  0
Input: [0 1]  Output: [ 0.60782715]     Target:  1
Input: [1 0]  Output: [ 0.54466069]     Target:  1
Input: [1 1]  Output: [ 0.56475546]     Target:  0
Train neural network with 5000 epochs
Input: [0 0]  Output: [ 0.01513546]     Target:  0
Input: [0 1]  Output: [ 0.97474472]     Target:  1
Input: [1 0]  Output: [ 0.97645504]     Target:  1
Input: [1 1]  Output: [ 0.02840842]     Target:  0
Train neural network with 50000 epochs
Input: [0 0]  Output: [ 0.00751348]     Target:  0
Input: [0 1]  Output: [ 0.99263085]     Target:  1
Input: [1 0]  Output: [ 0.99383571]     Target:  1
Input: [1 1]  Output: [ 0.00430303]     Target:  0
```

(a) One hidden layers



```
Train neural network with 500 epochs
Input: [0 0]  Output: [ 0.48762419]     Target:  0
Input: [0 1]  Output: [ 0.42569024]     Target:  1
Input: [1 0]  Output: [ 0.58477305]     Target:  1
Input: [1 1]  Output: [ 0.46554424]     Target:  0
Train neural network with 5000 epochs
Input: [0 0]  Output: [ 0.01069793]     Target:  0
Input: [0 1]  Output: [ 0.98076514]     Target:  1
Input: [1 0]  Output: [ 0.98252408]     Target:  1
Input: [1 1]  Output: [ 0.01904204]     Target:  0
Train neural network with 50000 epochs
Input: [0 0]  Output: [ 0.01046856]     Target:  0
Input: [0 1]  Output: [ 0.98945988]     Target:  1
Input: [1 0]  Output: [ 0.9942373]      Target:  1
Input: [1 1]  Output: [ 0.00912085]     Target:  0
```

(b) Two hidden layers

Figure 1: Training neural networks with 500, 5000, and 50000 epochs

4 Discussion

Two hidden layers network has more parameters than one hidden layer, which means the training result should be closer to targets. However, we observed that one hidden layer took advantage of two hidden layers in this setting. I thought one possible reason may be the amount of input data, since that four input data cannot support the parameter estimation of two hidden layer network. Another issue was that the output may vary from each training. This might cause by the random weight sampling and not enough epochs.

I am a master student in computer science and my research interests is Human-Computer Interaction. For the final project, I hope we can build an artistic project such like Deep Style, producing paintings or music. Or I would like to investigate applying deep learning technique to computer vision and build a system, which has attention-like functionality and tells the user where to focus.

5 References

1. CS231: Backpropagation, Intuitions
2. "OR" example provided in Lab0 materials