

# Note: Neural Networks

Sun Zhao

December 31, 2012

# 1 Representation

Brain makes human to learn and gain knowledge. So, what's the structure inside brain and how does brain learn? The answer is neural networks. In biology, neural networks inside our brains consist of billions of neurons, each of which can produce and transfer biological information to others and receive from others too. In machine learning field, neural networks are algorithms that mimic our brain's neural networks. Fig. 1 shows a simplest neuron having several inputs and one output. There are weights/parameters on each edge between neurons to represent how strong the connection is. Each neuron collects all the weighted inputs, performs self-defined calculation and produces weighted output. We use an activation function to define the calculation performed on each neuron. Fig. 2 shows a logistic neuron consisting of a sigmoid activation function and an extra bias unit of  $x_0$ . Fig. 3 shows a typical neural networks including input layer, hidden layer and output layer. Neurons between layer  $i$  and layer  $i + 1$  are pair-wise connected. Calculations performed on each neuron are shown in the bottom of Fig. 3.

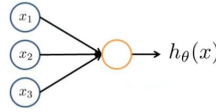


Figure 1: Neuron Model

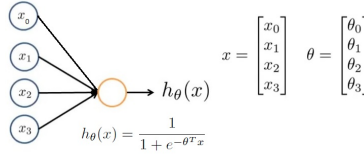


Figure 2: Logistic Neuron Unit

## 2 Motivation

Let's see an classification problem shown in Fig. 4. Obviously, there does not exists a linear curve separating the two classes. However, a logic function of  $x_1$  XOR  $x_2$  classifies the two classes perfectly. Neural networks are good at these non-linear problems, and we can build a neural network to solve the XOR classification problem. Using the logistic neuron unit in Fig. 2, we can first get three logic classifiers shown in Fig. 5. To verify the correctness of these three logic classifiers, you may draw the truth table. Combining the three logic neurons, Fig. 6 shows the final neural networks to solve the XOR classification problem with its truth table on the right.

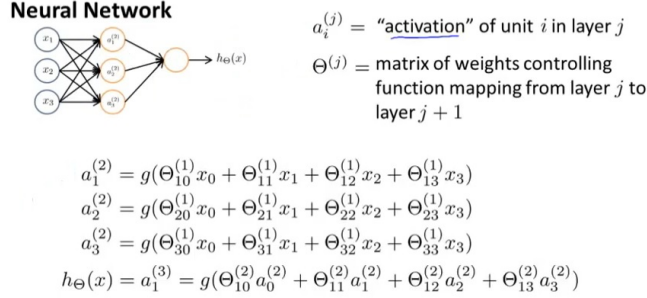


Figure 3: Neural Networks

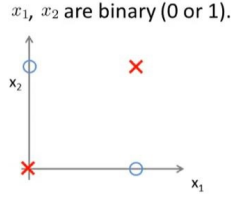


Figure 4: Neuron Model

### 3 Cost Function

Generally, neural networks outputs  $k$  hypothesis when solving a  $k$  classes classification problem. Using notations shown in Tab. 1 and regularized logistic regression cost function, the neural networks' cost function is defined as (??).

Table 1:	
Notation	Meaning
$L$	total No. of layers
$s_l$	No. of units in layer $l$
$K$	No. of output units
$(h_{\theta}(x))_i$	$i_{th}$ output hypothesis

$$J(\theta) = -\frac{1}{m} \left[ \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_{\theta}(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_{\theta}(x^{(i)}))_k) \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\theta_{ji}^{(l)})^2 \quad (1)$$

The index of  $i$  starts from 1 instead of 0 is because the bias is always ignored to regularize.

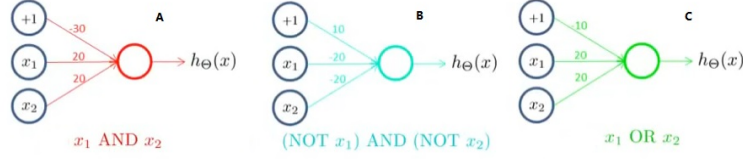


Figure 5: Neuron Model

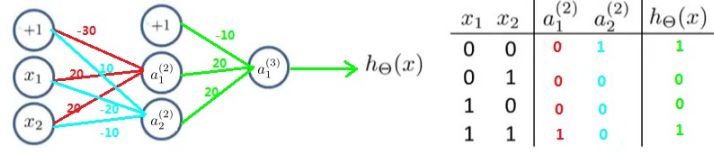


Figure 6: Neuron Model

## 4 Forward and Backward Propagation

When implementing gradient descent algorithm, we need compute  $J(\Theta)$  and the partial derivative terms of  $\frac{\partial}{\partial \Theta_{ij}^{(l)}}$  for all i, j, and l. Forward and backward propagation are used to calculate the two terms. Fig. 7 shows a four-layer neural networks. Starting from the first layer and calculate output as input of previous layer iteratively, the final hypothesis will be the last output. Concretely, we can use following procedure to compute the hypothesis for Fig. 7's neural networks.

$$\begin{aligned}
 a^{(1)} &= x \\
 z^{(2)} &= \Theta^{(1)} a^{(1)} \\
 a^{(2)} &= g(z^{(2)}) (\text{add } a_0^{(2)}) \\
 z^{(3)} &= \Theta^{(2)} a^{(2)} \\
 a^{(3)} &= g(z^{(3)}) (\text{add } a_0^{(3)}) \\
 z^{(4)} &= \Theta^{(3)} a^{(3)} \\
 a^{(4)} &= h_{\Theta}(x) = g(z^{(4)})
 \end{aligned}$$

To calculate the derivative terms, we introduce a new notation of  $\delta_j^{(l)}$  to represent the error of node i on layer j. For the output unit, it is obvious that  $\delta_j^{(L)} = a_j^{(L)} - y_j^{(L)}$ , and for layers which is prior to L, (2) is used to iteratively calculate its value. The term  $g'(z^{(l)})$  in (2) can be extended to  $a^{(l)} \cdot (1 - a^{(l)})$ .

$$\delta^{(l)} = (\theta^{(l)})^T \delta^{(l+1)} \cdot g'(z^{(l)}) \quad (2)$$

Algorithm 1 shows the back propagation process and output all layers  $\delta$  values with which we can define  $\frac{\partial}{\partial \Theta_{ij}^{(l)}}$  as (??). The prove of Algorithm 1 and Formula

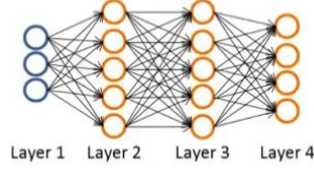


Figure 7: Neuron Model

?? is skipped, and I cannot figure it out myself. The partial derivative terms' calculation is quit obscure and different from that in logistic regression gradient descent process. I need refer other references to find more details.

---

Algorithm 1: Back Propagation

---

Training set  $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

Set  $\Delta_{ij}^{(l)} = 0$  for all  $i, j, l$

For  $i = 1$  to  $m$

$a^{(1)} = x^{(i)}$

Perform forward propagation to compute  $a^{(l)}$  for  $l=2, 3, \dots, L$

Using  $y^{(i)}$  to compute  $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute  $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$

$\Delta_{ij}^{(l)} = \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

---

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \begin{cases} \frac{1}{m} \delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} & \text{if } j \neq 0 \\ \frac{1}{m} \delta_{ij}^{(l)} & \text{if } j = 0 \end{cases} \quad (3)$$