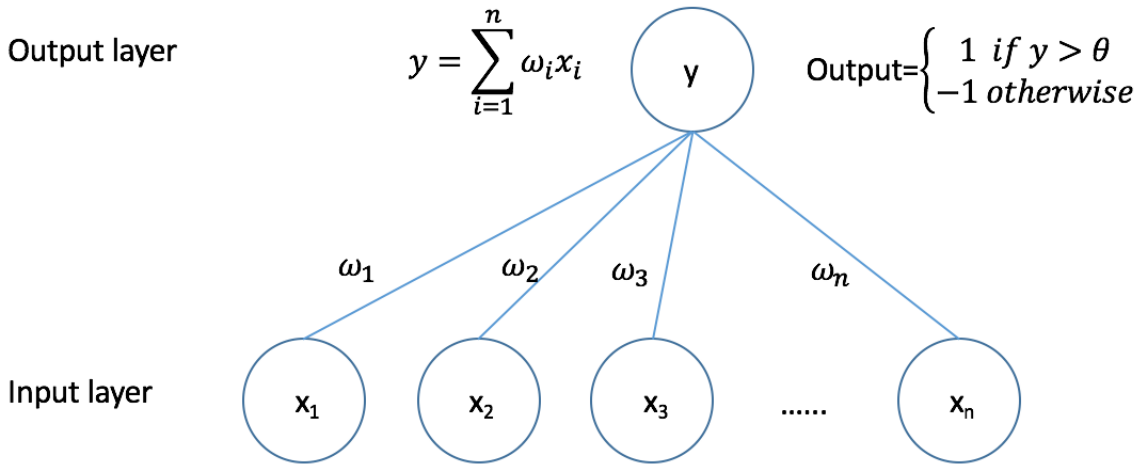
Multilayer perceptrons and backpropagation

Ye Li

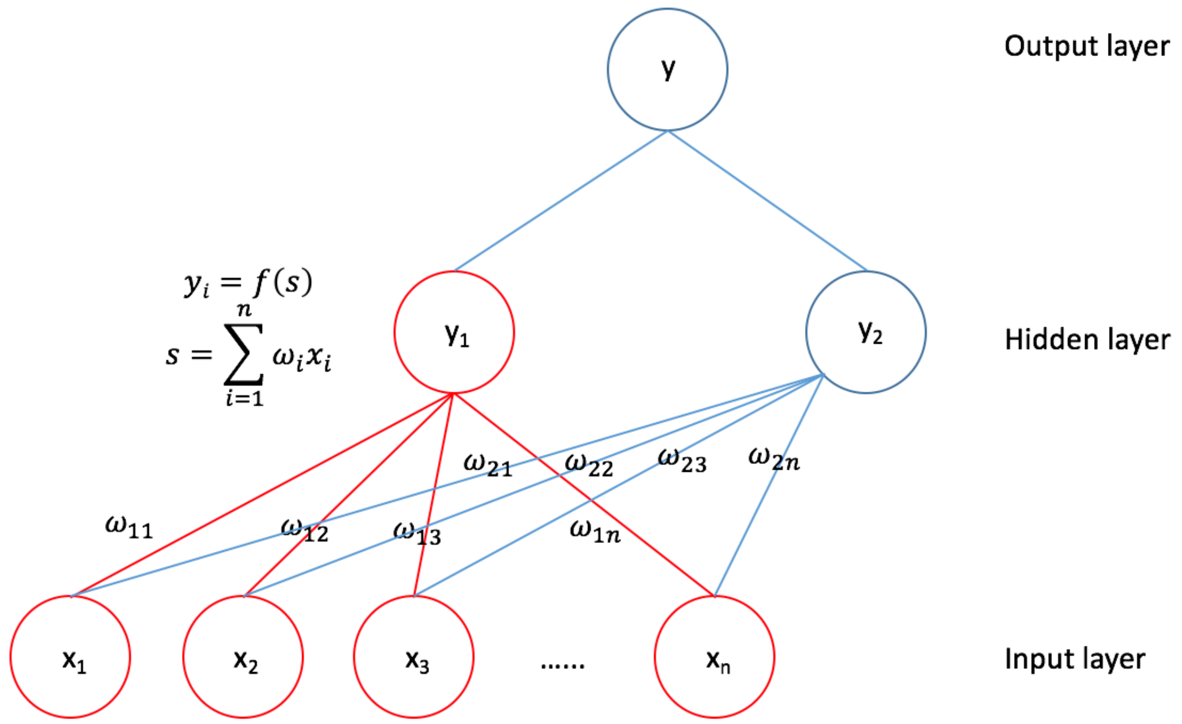
These notes give a short description of the basic ideas of multilayer perceptrons and backpropagation. It focus on answering the following questions: 1) Why are multiplayer perceptrons better than single layer perceptrons? 2) What is the main difference between multi-layer perceptrons and deep networks?

1 Multilayer Perceptrons

The idea of multilayer perceptron is to address the limitations of single layer perceptrons, namely, it can only classify linearly separable data into binary classes $(1, -1)$. A single layer perceptron is a feed-forward network based on a threshold transfer function and has the structure as shown in the figure below.


Output layer
$y = \sum_{i=1}^{n} \omega_i x_i$
y
Output= $\begin{cases} 1 \text{ } if\ y > \theta \\ -1 \text{ } otherwise \end{cases}$
ω_1
ω_2
ω_3
ω_n
Input layer
x_1
x_2
x_3
......
x_n

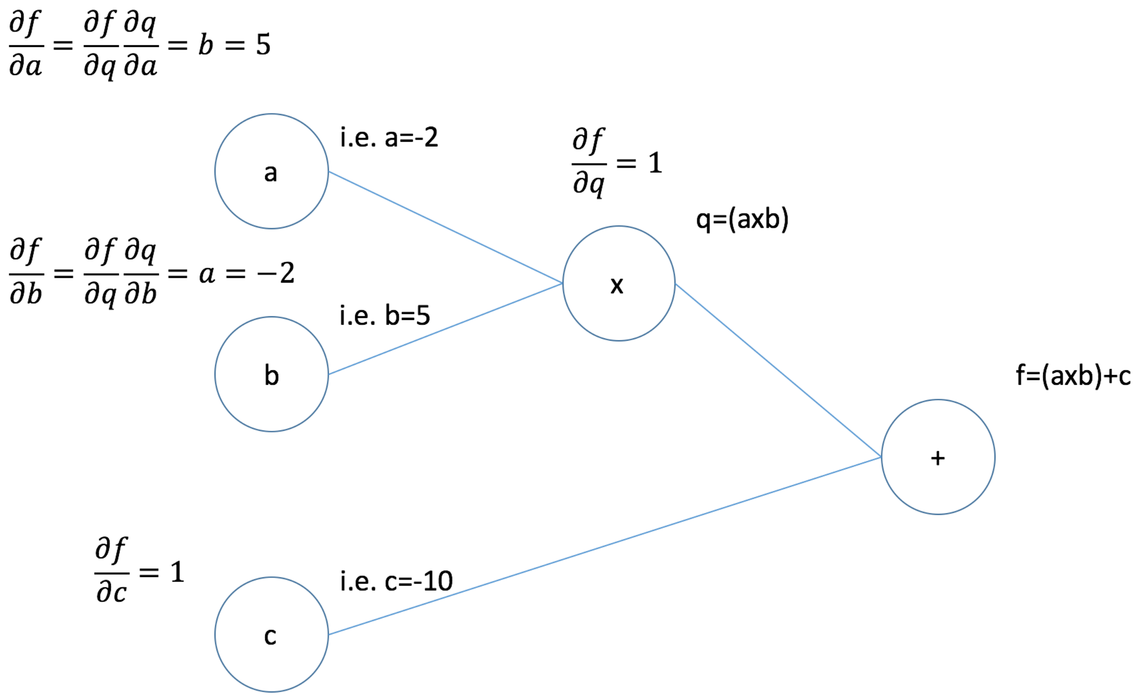
A multilayer perceptron is built on top of single layer percentrons. It uses the outputs of the perceptrons at one layer as inputs to perceptrons at its next layer. Thus, many levels can be specified and non-linear relationship between inputs and outputs can be modeled. A multilayer perceptron differs from a single layer perceptron by two main ingredients: 1) a soft thresholding function after each summation (linear combination of inputs) and 2) introduction of hidden layers. Many people have tried to come up with explanations about the hidden units but it is still unclear. However, one thing that we do know is that the number of hidden units is related to the capacity of the perceptron. A sample structure of a multilayer perceptron graph is given below.



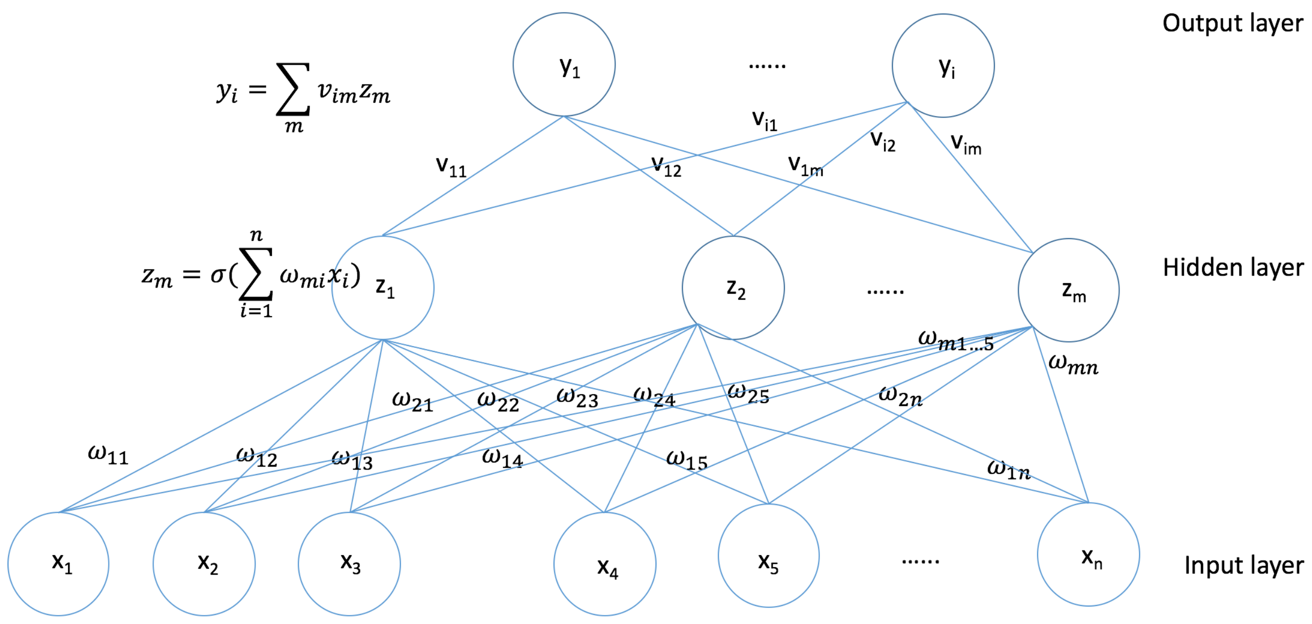
The red nodes in the graph represent the original part from the single layer perceptron that we showed in the previous graph. When drawing the graph, I intentionally left the single layer perceptron part unchanged so that we can better see what's added to the new graph. Basically, there two important points: 1) the newly introduced hidden nodes use/share the same inputs with other hidden units at the same level/layer and 2) use of non-linear function in calculation of the output at the hidden layer. With these features, almost any input-output function can be modeled by a multilayer perceptron with enough hidden layers. Thus, the multilayer perceptron is often preferred over the single layer perceptron in more sophisticated data such as linear inseparable data, due to its ability to capture non-linearity.

2 Backpropagation

To put simple, backpropagation is to apply the chain rule many many times to calculate the gradient of a loss function with respect to all the inputs (weights, input data) in the network. Below is a simple network containing only 3 inputs and a single output and the question to ask is: how much change there is on the final result if I change an input by Δ? In other words, how would change in a affect f, which is the final result in the network shown below. To answer this question, we need the partial derivative w.r.t that particular input, which is $\frac{\partial f}{\partial a} = 5$. And, this simply means that an increase in a would increase f by an amount equal to 5Δ (Δ here denotes the change in a itself). In general, positive gradient would positively influence the loss (final result) and negative gradient would negatively influence the loss, by the amount that's equal to the gradient multiplied with Δ. In a real neural network or a large computational circuit (imagine many many operations and inputs), we can think a as a weight, ω_0, and b as an input, x_0. In the update equation, if we want to decrease the loss (always want to minimize the loss), all we need to do is simply update the weight by a tiny bit in the opposite direction of its local partial gradient, i.e., decrease ω_0 from -2 to -3 and we would get a smaller f.


$\frac{\partial f}{\partial a} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial a} = b = 5$
i.e. a=-2
a
$\frac{\partial f}{\partial q} = 1$
q=(axb)
$\frac{\partial f}{\partial b} = \frac{\partial f}{\partial q}\frac{\partial q}{\partial b} = a = -2$
x
i.e. b=5
b
f=(axb)+c
+
$\frac{\partial f}{\partial c} = 1$
i.e. c=-10
c

For training a multilayer perceptron, we have to estimate the weights of the percenptron. First we need the loss, namely an error function.



For the multilayer perceptron shown above, the loss can be defined as:

$$E[\omega, \nu] = \sum_i \{y_i - \sum_m \nu_{im} \sigma(\sum_n \omega_{mn} x_n)\}^2$$

As explained in the first section, the update terms are the negative derivatives of the loss with respect to the local parameters(weights) times a small change δ, specified by learning rate:

$$\Delta \omega_{mn} = -\frac{\partial E}{\partial \omega_{mn}} \times \delta$$

which is computed by the chain rule, and

$$\Delta \nu_{im} = -\frac{\partial E}{\partial \nu_{im}} \times \delta$$

which is computed directly as they are weights of the last layer.
By defining $z_m = \sigma(\sum_n \omega_{mn} x_n)$, $E = \sum_i (y_i - \sum_m \nu_{im} z_m)^2$, we can write:

$$\frac{\partial E}{\partial \omega_{mn}} = \frac{\partial E}{\partial z_m} \frac{\partial z_m}{\partial \omega_{mn}}$$

where $\frac{\partial E}{\partial z_m} = -2 \sum_i (y_i - \sum_m (\nu_{im} z_m)) \nu_{im}$ and assume σ is a sigmoid function whose derivative is $\sigma(t)(1 - \sigma(t))$, then we have $\frac{\partial z_m}{\partial \omega_{mn}} = x_n \sigma(\sum_n \omega_{mn} x_n)\{1 - \sigma(\sum_n \omega_{mn} x_n)\}$. So we have:

$$\frac{\partial E}{\partial \omega_{mn}} = 2 \sum_i (y_i - \sum_m (\nu_{im} z_m)) \nu_{im} x_n \sigma(\sum_n \omega_{mn} x_n)\{1 - \sigma(\sum_n \omega_{mn} x_n)\}$$

and $\sum_i (y_i - \sum_m (\nu_{im} z_m))$ is the error at the output layer.