Lecture (3)
Neural Networks

Turki Haj Mohamad

November 16, 2018

1 Single hidden layer neural network

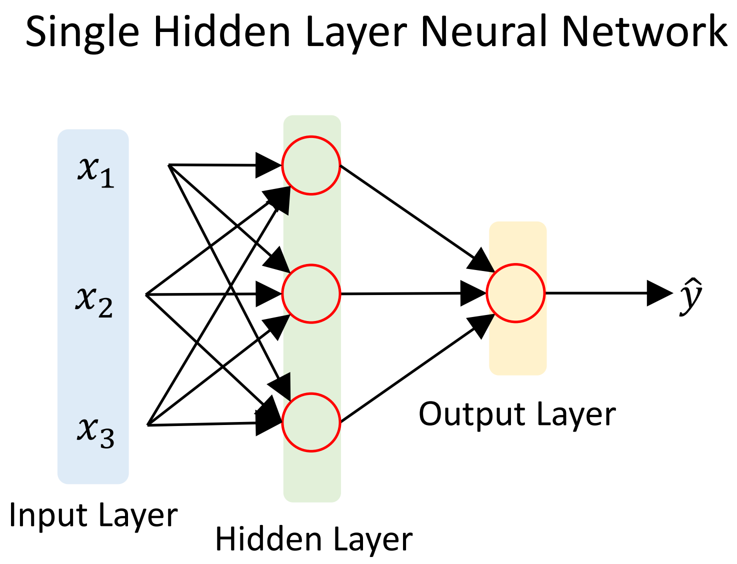

Figure 1: Neural network structure

In notation conventions in neural networks this network is called a two layer neural network "2 layer NN". the hidden layer is layer one and the output layer is layer two (we don't count the input layer).

1.1 Computing neural network's output

Figure 2 shows the logistic regression model and its corresponding computation graph. The node/unit in the logistic regression represents two steps of computation; first it computes z as shown and second it computes the activation a as a sigmoid function of $a = \sigma(z)$.

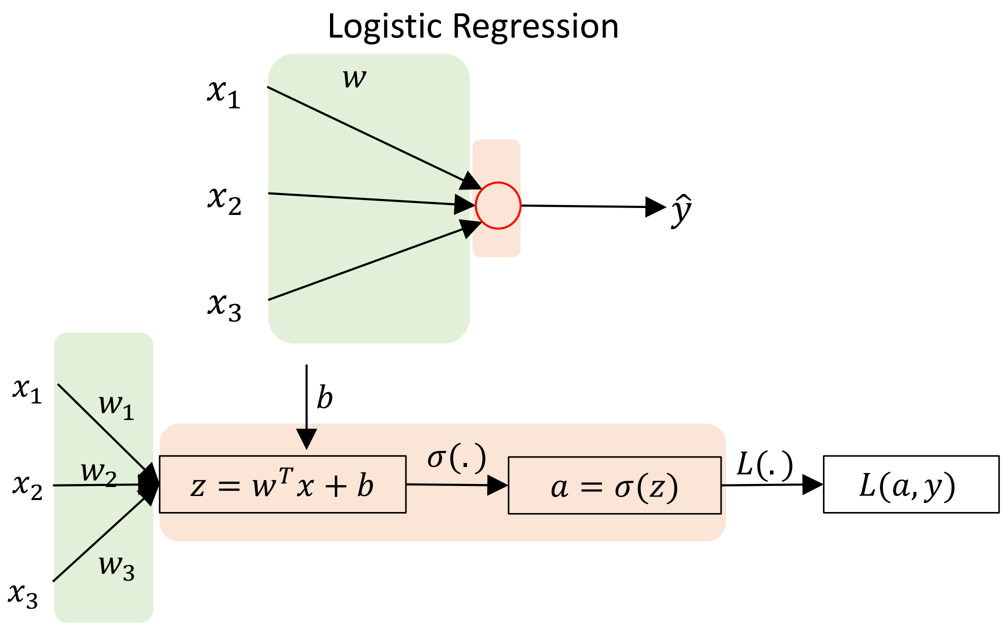

Figure 2: Logistic regression

Neural network does this repeated computation a lot more times. For example, the first node in the hidden layer computes $z_1^{[1]} = w_1^{[1]T}\mathrm{x} + b_1^{[1]}$ then it computes $a_1^{[1]} = \sigma(z_1^{[1]})$. In general, the notation convention is $a_i^{[l]}$ where i is the node number and l is the layer number.

$$a^{[0]} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \mathrm{x}$$

where, $a^{[0]} \in \mathbb{R}^{n_x}$ is the input vector, $a^{[1]} \in \mathbb{R}^{n_h}$ is the hidden layer output (a_1^1 is the output of the first node in the hidden layer), $a^{[2]} \in \mathbb{R}^{n_o}$. Note that,

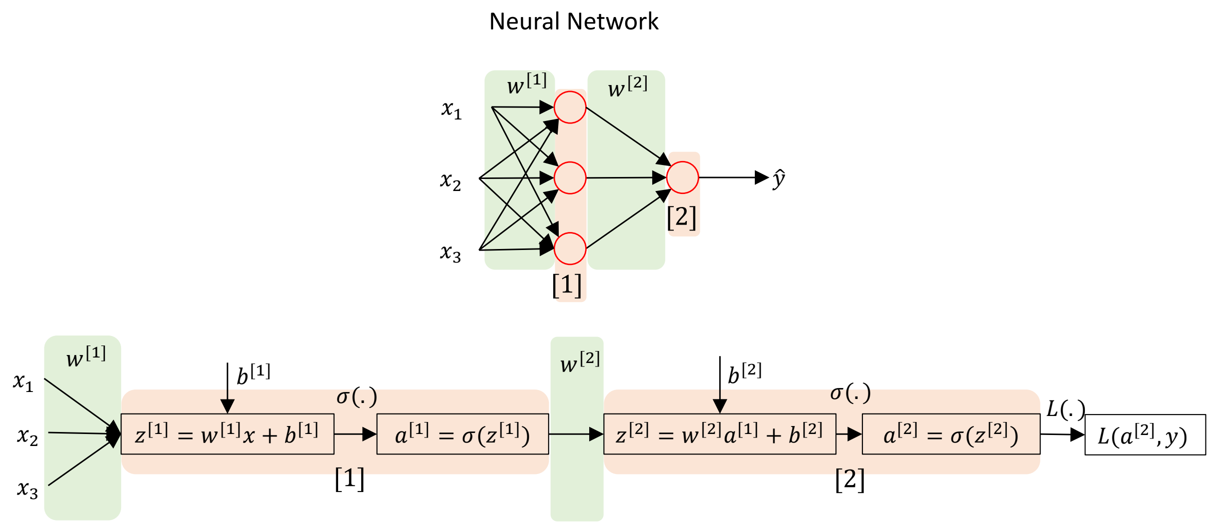
Neural Network
$w^{[1]}$
$w^{[2]}$
x_1
x_2
x_3
$\hat{y}$
[1]
[2]
$b^{[1]}$
$\sigma(.)$
$z^{[1]} = w^{[1]}x + b^{[1]}$
$a^{[1]} = \sigma(z^{[1]})$
$b^{[2]}$
$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}$
$a^{[2]} = \sigma(z^{[2]})$
$L(.)$
$L(a^{[2]}, y)$

Figure 3: Neural network

n_x is the dimension of input or the number of features, n_h is the number of hidden units/nodes and n_o is the number of outputs.

$$\begin{aligned} z_1^{[1]} &= w_1^{[1]T}\mathrm{x} + b_1^{[1]}, \quad a_1^{[1]} = \sigma(z_1^{[1]}) \\ z_2^{[1]} &= w_2^{[1]T}\mathrm{x} + b_2^{[1]}, \quad a_2^{[1]} = \sigma(z_2^{[1]}) \\ z_3^{[1]} &= w_3^{[1]T}\mathrm{x} + b_3^{[1]}, \quad a_3^{[1]} = \sigma(z_3^{[1]}) \end{aligned}$$

$$z^{[1]} = \begin{bmatrix} - & w_1^{[1]T} & - \\ - & w_2^{[1]T} & - \\ - & w_3^{[1]T} & - \end{bmatrix} \mathrm{x} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \end{bmatrix} = w^{[1]}\mathrm{x} + b^{[1]}$$

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \end{bmatrix} = \sigma(z^{[1]})$$

$$z^{[2]} = w^{[2]}a^{[1]} + b^{[2]}, \quad a_{[2]} = \sigma(z^{[2]}) \quad a^{[2]} = \hat{y}$$

where $w^{[1]} \in \mathbb{R}^{n_h \times n_x}$, $b^{[1]} \in \mathbb{R}^{n_h}$, $w^{[2]} \in \mathbb{R}^{n_o \times n_h}$, $b^{[2]} \in \mathbb{R}^{n_o}$

1.2 Implementing vectorization across multiple examples

The above calculations demonstrates the calculation of the neural network for one training example. In case of multiple example, we need to implement a vectorized version to avoid using loops.
- for a single example

$$\begin{aligned} z^{[1](i)} &= w^{[1]}\mathrm{x}^{(\mathrm{i})} + b^{[1]} \\ a^{[1](i)} &= \sigma(z^{[1](i)}) \\ z^{[2](i)} &= w^{[2]}a^{[1](i)} + b^{[2]} \\ a^{[2](i)} &= \sigma(z^{[2](i)}) = \hat{y}^{(i)} \end{aligned}$$

- for multiplr examples

$$X = \begin{bmatrix} | & | & \cdots & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & \cdots & | \end{bmatrix} \qquad Y = \begin{bmatrix} y^{(1)} & y^{(2)} & \cdots & y^{(m)} \end{bmatrix}$$

$$\begin{aligned} Z^{[1]} &= w^{[1]}\mathrm{X} + b^{[1]} \\ A^{[1]} &= \sigma(Z^{[1]}) \\ Z^{[2]} &= w^{[2]}A^{[1]} + b^{[2]} \\ A^{[2]} &= \sigma(Z^{[2]}) = \hat{Y} \end{aligned}$$

where

$$Z^{[1]} = \begin{bmatrix} | & | & \cdots & | \\ z^{[1](1)} & z^{[1](2)} & \cdots & z^{[1](m)} \\ | & | & \cdots & | \end{bmatrix} \quad A^{[1]} = \begin{bmatrix} | & | & \cdots & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & \cdots & | \end{bmatrix}$$

This matrix notation represents different training examples horizontally and different hidden nodes/unites vertically.

1.3 Activation function

we can use different activation functions $\sigma(.) \rightarrow g(.)$ where $g(.)$ is a nonlinear function. One of the common choices is $g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$, which is usually perform better than the sigmoid function because the values are ranging between -1 and +1 and the mean of the activations are closer to zero which makes the learning for the next layer easier. activation functions can be different for different layers, for example the output layer can have sigmoid function especially when dealing with binary classification problems.

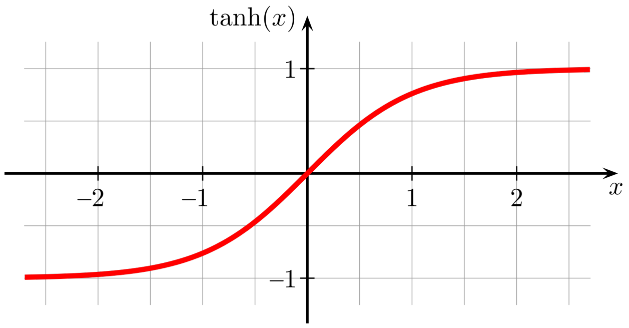

Figure 4: tanh function

One of the downsides of both the sigmoid function and the tanh function is that if z is either very large or very small, then the gradient becomes very small, which can slow down gradient descent. Therefore, one other choice that is very popular in machine learning is what's called the rectify linear unit RelU function $g(z) = max(0, z)$. The derivative is 1 as long as z is positive, which helps the neural network learn much faster than when using the tanh or the sigmoid activation function.

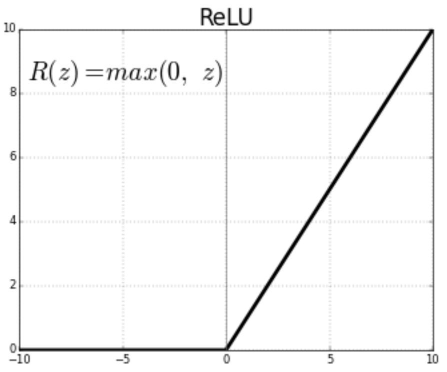

Figure 5: ReLU

For regression problems (target $y \in \mathbb{R}$, we use a linear activation function $g(z) = z$ for only the output layer.

To implement backpropagation, we need to find the derivative of activation functions.

- sigmoid function $\sigma(z) = \frac{1}{1+e^{-z}} \rightarrow \frac{d\sigma(z)}{dz} = \sigma(z)(1-\sigma(z))$
- tanh function $g(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \rightarrow \frac{dg(z)}{dz} = 1 - \tanh(z)^2$
- RelU function $g(z) = \max(0, z) \rightarrow \frac{dg(z)}{dz} = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases}$

1.4 Gradient descent for neural network

We need to solve for the neural network parameters: $w^{[1]}$, $w^{[2]}$, $b^{[1]}$, and $b^{[2]}$, where, $w^{[1]} \in \mathbb{R}^{n_h \times n_x}$, $b^{[1]} \in \mathbb{R}^{n_h}$, $w^{[2]} \in \mathbb{R}^{n_o \times n_h}$, and $b^{[2]} \in \mathbb{R}^{n_o}$. To do so gradient descent will be performed to minimize the following cost function.

$$J(w^{[1]}, b^{[1]}, w^{[2]}, b^{[2]}) = \frac{1}{m} \sum_{i=1}^{m} (L(\hat{y}, y)) \tag{1}$$

Gradient descent

$$\begin{aligned} w^{[1]} &:= w^{[1]} - \alpha \frac{dJ}{dw^{[1]}} \\ b^{[1]} &:= b^{[1]} - \alpha \frac{dJ}{db^{[1]}} \\ w^{[2]} &:= w^{[2]} - \alpha \frac{dJ}{dw^{[2]}} \\ b^{[2]} &:= b^{[2]} - \alpha \frac{dJ}{db^{[2]}} \end{aligned}$$

Forward propagation:

$$\begin{aligned} Z^{[1]} &= w^{[1]}\mathrm{X} + b^{[1]} \\ A^{[1]} &= \sigma(Z^{[1]}) \\ Z^{[2]} &= w^{[2]}A^{[1]} + b^{[2]} \\ A^{[2]} &= \sigma(Z^{[2]}) = \hat{Y} \end{aligned}$$

Back propagation:

$$\begin{aligned}
dZ^{[2]} &= A^{[2]} - Y \\
dw^{[2]} &= \frac{1}{m} dZ^{[2]} A^{[1]T} \\
db^{[2]} &= \frac{1}{m} \sum dZ^{[2]} \\
dZ^{[1]} &= w^{[2]T} dZ^{[2]} * g^{[1]'}(Z^{[1]}) \\
dw^{[1]} &= \frac{1}{m} dZ^{[1]} X^T \\
db^{[1]} &= \frac{1}{m} \sum dZ^{[1]}
\end{aligned}$$

1.5 Assignment: Classification with a hidden layer

Build a neural network to classify the following data sets: 1) flower data set 2) double moon data set. Use different hidden unit activation functions, number of hidden units, learning rate.

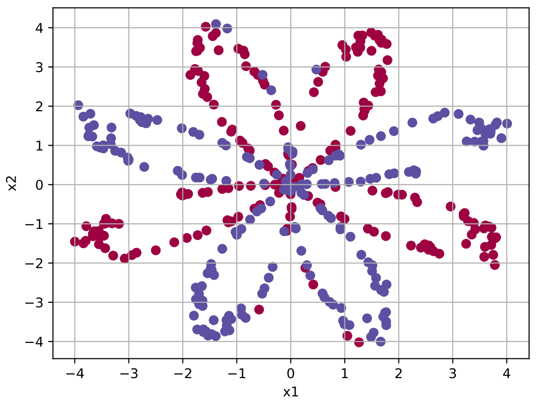

Figure 6: Data set 1

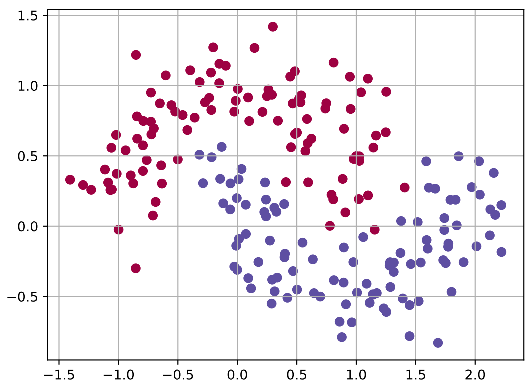

Figure 7: Data set 2