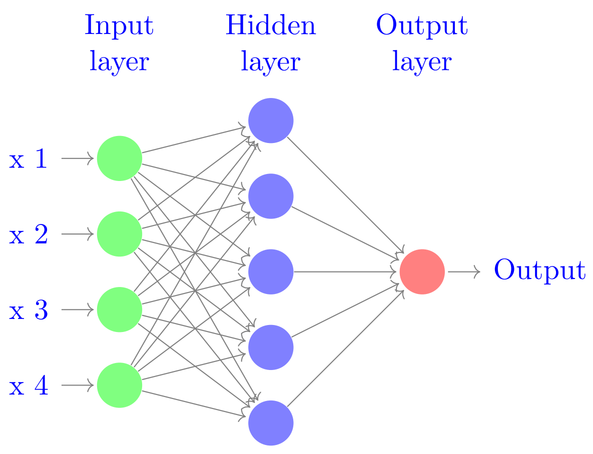
Machine Learning

Assignment 6.3

Submitted By: Ranji Raj

December 4, 2020

a) Looks of a Neural Net



- There can be exactly one input layer and the output layer but, the number of nodes/units can be in any number.
- The hidden layer, $h_l^{[layernumber]}$ however can be in any number which adds up to the complexity of neural networks.
- For N units there are N number of weights (includes bias).
- Each node contain both linear part as well as the non-linear part.
- Each of the connections among input layer and hidden layer can be represented by form a weight-vector matrix, which has the dimension *nodes at next layer X nodes*

at previous layer in the above neural net,

$$W^{[1]}_{5x4} = \begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \\ w_{13} & w_{23} & w_{33} & w_{43} \\ w_{14} & w_{24} & w_{34} & w_{44} \\ w_{15} & w_{25} & w_{35} & w_{45} \end{bmatrix}$$

- Similarly, the strength between the hidden layer and and output layer is denoted in weight-vector matrix as,

$$W^{[2]}_{1x5} = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} & w_{15} \end{bmatrix}$$

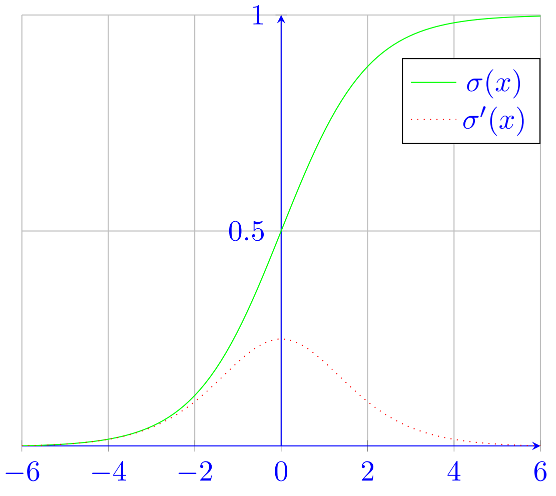
- When the weight-vectors are multiplied they are converted to row-vectors and column-vectors.

b) Classification by a Neural Net

- In neural nets, the linear and non-linear part both an indispensable part of a unit. The linear part computes the linear combination which is then forwarded to the non-linear part which does the classification.
- The way by which it does classification is by using a special activation function called as the *sigmoid function* or *squashing function* which comes from the flagship binary classifier *Logistic Regression.*

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

- The sigmoid curve takes input from the linear part in the range $(-\infty, \infty)$ and squashes in between $[0, 1]$ which serves as the basis for classification.



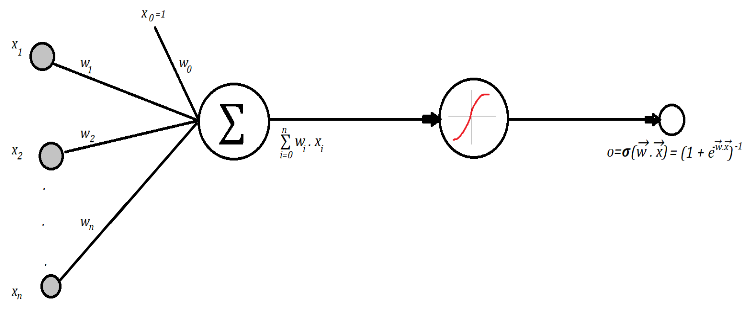

x_0=1
x_1
x_2
x_n
w_0
w_1
w_2
w_n
$\sum_{i=0}^{n} w_i . x_i$
$o=\sigma(\vec{w}.\vec{x}) = (1 + e^{-\vec{w}.\vec{x}})^{-1}$

Figure 1: Classification by sigmoid curve

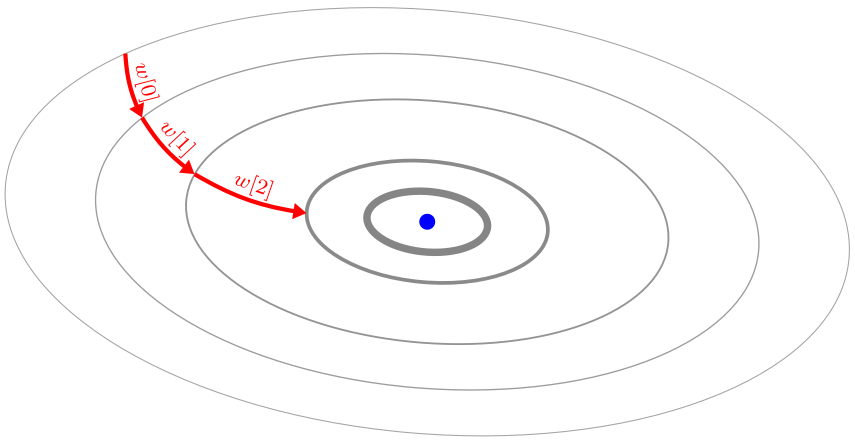
c) Learning a Neural Net

- Converges only asymptotically towards the minimum error hypothesis regardless whether the problem is linearly separable or not.
- Error defined as,
$$E(\vec{w}) = \frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$
- Uses the principle of gradient descent which searches the hypotheses space H of possible weight vectors to find the set that best fit the training examples.
- Learning rule:
$$\Delta\vec{w} = -\eta\nabla E(\vec{w})$$
i.e.,
$$\Delta w_i = -\eta\frac{\partial E}{\partial w_i}$$

Vanishing Gradient problem

- In principle the gradient descent strategy is used to reach the minimum such as for the cost or the error function.
- In neural nets, when we have a dense pool of hidden layers the process of gradient is usually depicted in form of a **chain rule**.
- Say, for instance we have a dense network of 100 layers the gradient will be calculated as follows:
$$\frac{\partial E}{\partial w^{[1]}} = (\frac{\partial E}{\partial a^{[100]}}\frac{\partial a^{[100]}}{\partial z^{[100]}})...(\frac{\partial z^{[4]}}{\partial a^{[3]}}\frac{\partial a^{[3]}}{\partial z^{[3]}})(\frac{\partial z^{[3]}}{\partial a^{[2]}}\frac{\partial a^{[2]}}{\partial z^{[2]}})(\frac{\partial z^{[2]}}{\partial a^{[1]}}\frac{\partial a^{[1]}}{\partial z^{[1]}})\frac{\partial z^{[1]}}{\partial w^{[1]}}$$
- These are very miniature gradient values like 0.002, 0.04, 0.005... and when we multiply these small quantities they may turn to a very small value like $10^{-10} \approx 0$.

- So before attaining the global minimum, the gradient could be lost or it may have very little effect and the learning stops because the new weight will be equal to old weight.
- Vanishing gradient pose a huge problem in determining as in which direction the weights must move so as to improve the error function.
- The root cause of vanishing gradients could be the random weight-vector initialization.
- Perhaps, the most common way out to this problem is to employ a **Rectified Linear(ReLU)** activation function.



d) Weights not be set to same value - Symmetry breaking

- If all weights start out with equal values and if the solution requires that unequal weights to be developed, the neural net can never learn.
- Because, error is propagated back through the weights in proportion to the values of the weights.
- That means, all the hidden units connected directly to the output units will get **identical** error signals, and, since the weight changes depend on these error signals, the weights from those units to the output units will always be the same.
- The network is starting out at a kind of **unstable equilibrium** point that keeps the weights equal, but it is higher than some neighboring points on the error surface, and once it moves away to one of these points, it will never return.
- To counteract this, the network is kicked-off with **small random** weights under these conditions the symmetry problem do not arise.

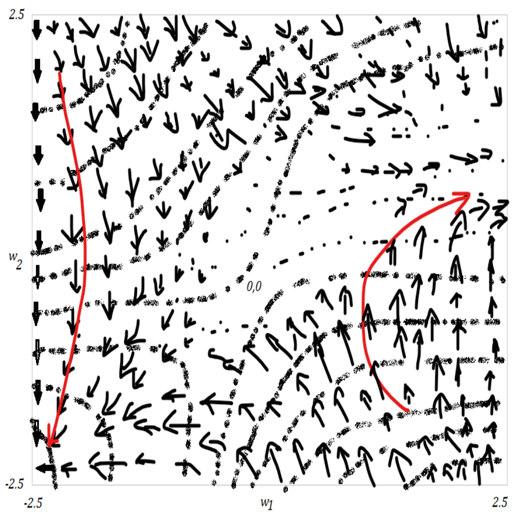


Figure 2: Saddle-shaped contour map

- If the network starts at exactly (0,0), there is no pressure for it to move at all and the network will not learn, but if it starts anywhere off the anti-diagonal, it will eventually end up in one minimum or the other.
- The map show a local minimum in the positive quadrant and a global minimum in the lower left-hand negative quadrant.

Non-linearity of neural nets

- The architecture of a neural network can be thought of as a function which takes an input X and generates some output Y. More formally written as,

$$f(X) \longrightarrow Y$$

- If only a linear activation functions are used, then combination of these functions is another linear activation function. Ex. Let the input at first layer be as follows:

$$Z^{[1]} = W^{[1]} * X + b^{[1]} \tag{1}$$

- After passing through an activation function:

$$a^{[1]} = g(Z^{[1]}) \tag{2}$$

- Now the net input at second layer:

$$Z^{[2]} = W^{[2]} * a^{[1]} + b^{[2]} \tag{3}$$

- If the activation function is linear then,

$$a^{[1]} = c * Z^{[1]}$$

  which is some multiple (c) of the net input would be the resultant output after passing through a linear activation.

- For simplicity, let c=1 and substitute the value of $a^{[1]}$ in (3). We obtain,

$$Z^{[2]} = W^{[2]} * Z^{[1]} + b^{[2]}$$

  which is,

$$Z^{[2]} = W^{[2]} * W^{[1]} * X + b^{[1]} + b^{[2]}$$

  from (1)

- This result can be also written as:

$$Z^{[2]} = W^{'} * X + B^{'}$$

- The result is just another linear function and this process will continue even in further layers. Hence, simple linear functions **cannot** approximate every function.

- Non-linearity is ensured that output at any unit should be **differentiable** of its linear combination of inputs, so that we can back propagate to the layers and reach at that weight where it can be more learned.

- Having so, narrow downs the ability of network to fit non-linear boundaries because in this case it will simply behave as a linear least-squares model.

- Non-linearity can be achieved by changing the activation functions to Sigmoid, tanh, ReLU (and its variants), Swish, Softmax like others.