Building Blocks of Neural Networks

Activation Functions

1. Artificial neuron works in three steps
    1. First it multiplies the input signals with corresponding weights
    2. Second, adds the weighted signals
    3. Third, converts the result to another value using a mathematical transformation function

2. For the third step, there are multiple mathematical functions available but all together are called the activation function

3. The purpose of the activation function is to act like a switch for the neuron. Should the neuron fire or not. Also…

4. The activation function is critical to the overall functioning of the neural network. Without it, the whole neural network will mathematically become equivalent to one single neuron!

5. Activation function is one of the critical component that gives the neural networks ability to deal with complex problems

Activation Functions - why?

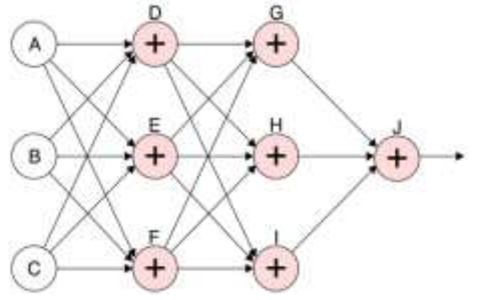
1. Let us take a fully connected neural network. Every neuron in every layer takes multiple inputs
2. The inputs are weighted and summed up at each neuron
3. The nodes in the second layer are simply scaling up the output of neurons in previous layer

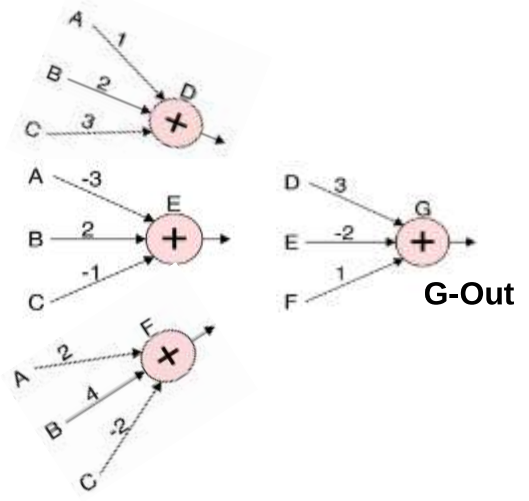
4. For e.g the neuron G takes as input weight sums from D,E and F, G's output is scaled version of output of D, E and F
5. **G_Out** = 3D -2E + 1F
6. = 3(1A + 2B + 3C) – 2(-3A + 2B -1C) + 1(2A+4B-2C)
7. = **<u>8A + 6B + 3C</u>**

8. Thus this part of the network is like a single neuron with weights of 8, 6, 3!!!

9. Same argument holds for other neurons

10. Thus the entire neural network collapses to on neuron!
11. A single neuron is not capable of doing much



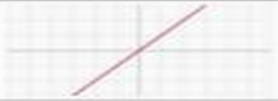
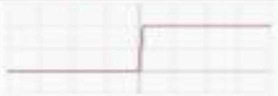
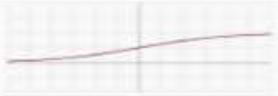
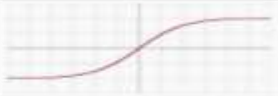
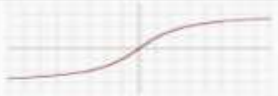
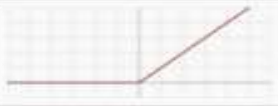
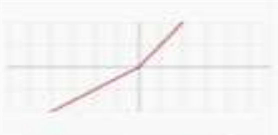
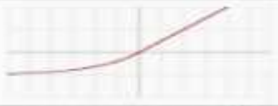
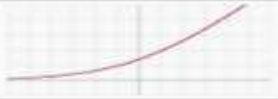


Activation Functions - Types

4. Types of non-linear activation functions include –
   a. Piecewise linear functions
      I. Step function
      II. ReLU – Rectified Linear Units
      III. Leaky ReLU
      IV. Parametric ReLU
      V. Shifted ReLU

   b. Smooth functions
      I. Smooth ReLU / Exponential ReLU
      II. Sigmoid / Logistic functions
      III. Hyperbolic Tangent (tanh)
      IV. Swish (combination of Sigmoid and ReLU)

Activation Functions -

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for} & x < 0 \\ 1 & \text{for} & x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} & x \neq 0 \\ ? & \text{for} & x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for} & x < 0 \\ x & \text{for} & x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for} & x < 0 \\ 1 & \text{for} & x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)		$f(x) = \begin{cases} \alpha x & \text{for} & x < 0 \\ x & \text{for} & x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for} & x < 0 \\ 1 & \text{for} & x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for} & x < 0 \\ x & \text{for} & x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for} & x < 0 \\ 1 & \text{for} & x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

Neurons stretch the features pace through non-linear functions and achieve Cover's theorem

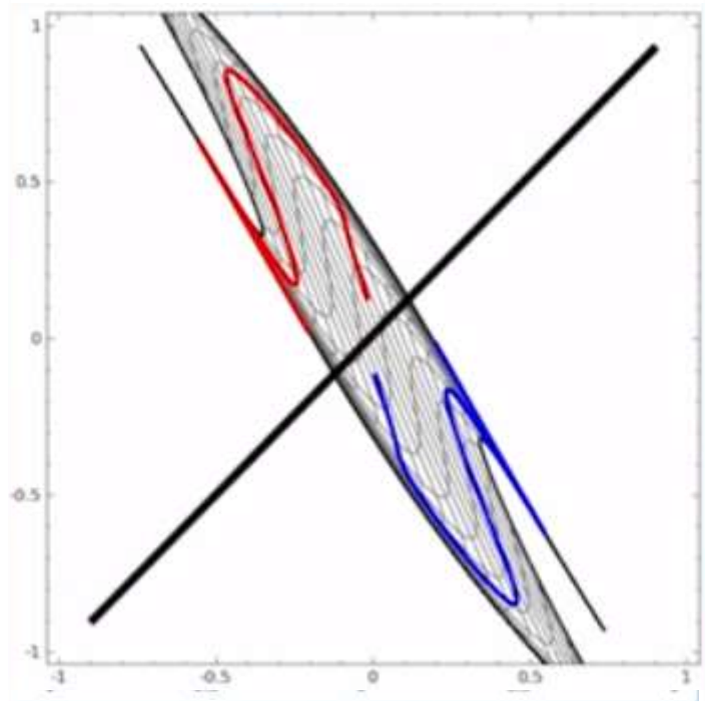
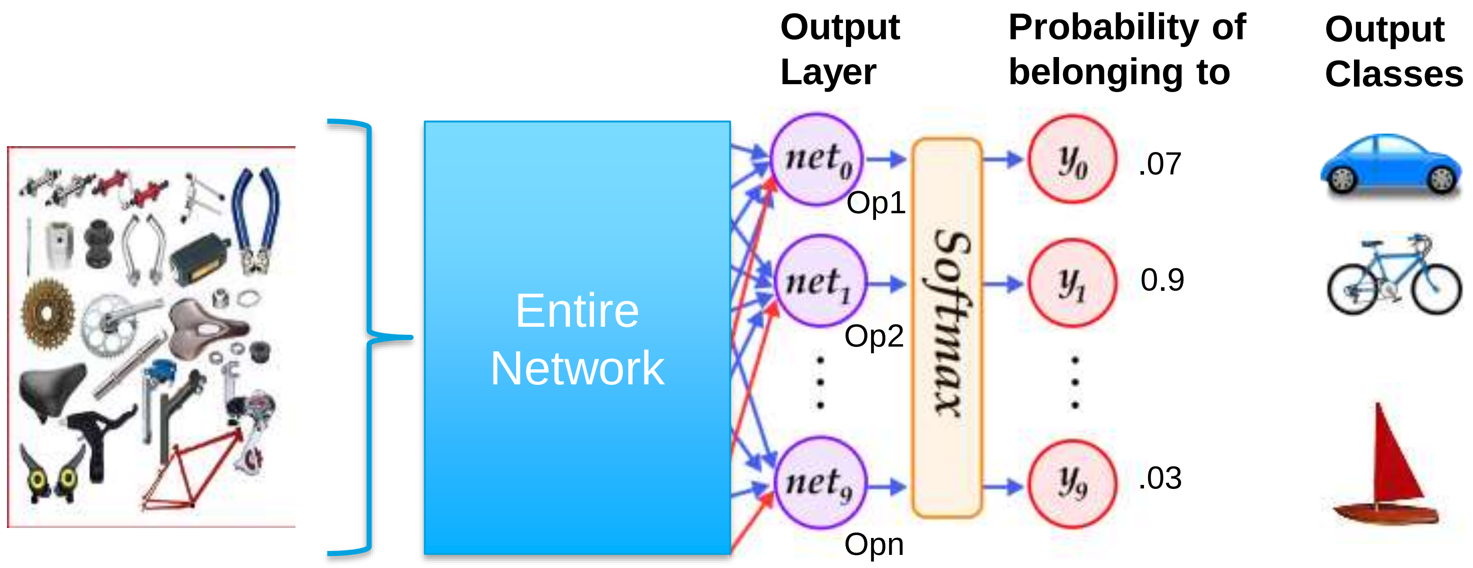


Image Source: https://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

Ref: https://cs.stanford.edu/people/karpathy/convnetjs//demo/classify2d.html

SoftMax Function -

1. A kind of operation applied at the output neurons of a classifier network
2. Used only when we have two or more output neurons and is applied simultaneously to all the output neurons
3. Turns raw numbers coming out of the pen-ultimate layer into probability values in the output layer
4. Suppose output layer neurons emit (Op1, Op2, Opn). The raw numbers may not make much sense. We convert that into probabilities using Softmax which becomes more meaningful. For e.g. input belongs to cycle is 30 times more likely than sailboat, 13 times more probable than car



Forward Propagation

1. The directed acyclic path taken by input data from input layer to get transformed using non-linear functions into final network level outputs

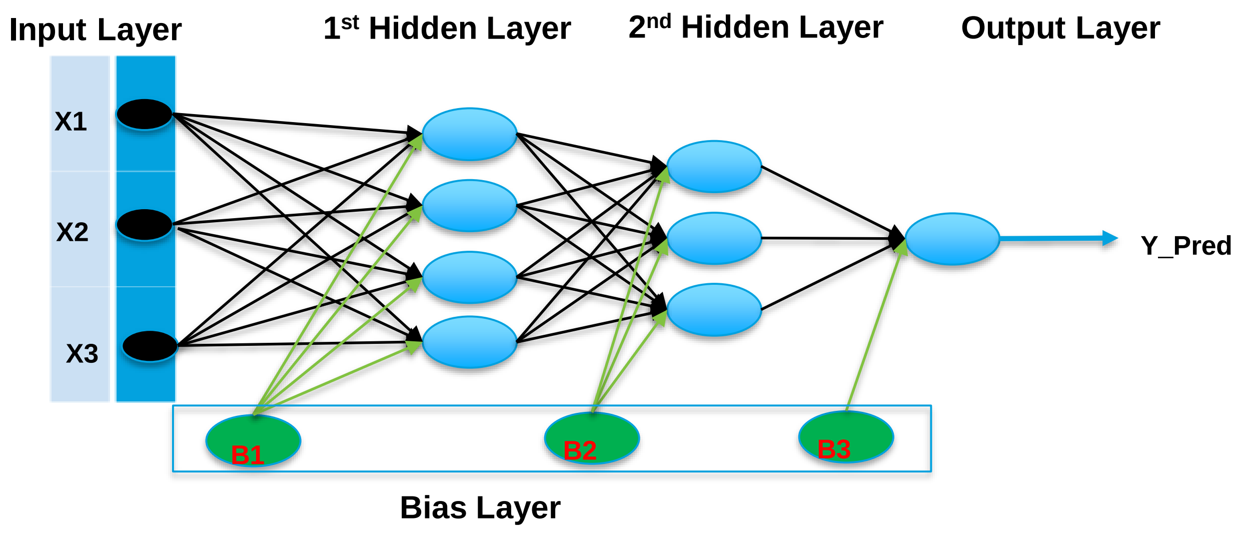
2. Input data is propagated forward from the input layer to hidden layer till it reaches final layer where predictions are emitted

3. At every layer, data gets transformed in three steps in every neuron
   a. Sum weighted input at every neuron by multiplying X by the hidden weight Wh
   b. Apply the activation function on the sum
   c. Pass the result to all the neurons in next layer

4. The last layer is the output layer which may have a softmax function (if the network is a multi class classifier)

Bias Term

1. Every neuron in the hidden layers are associated with a bias term. The bias term help us to control the firing threshold in each neuron

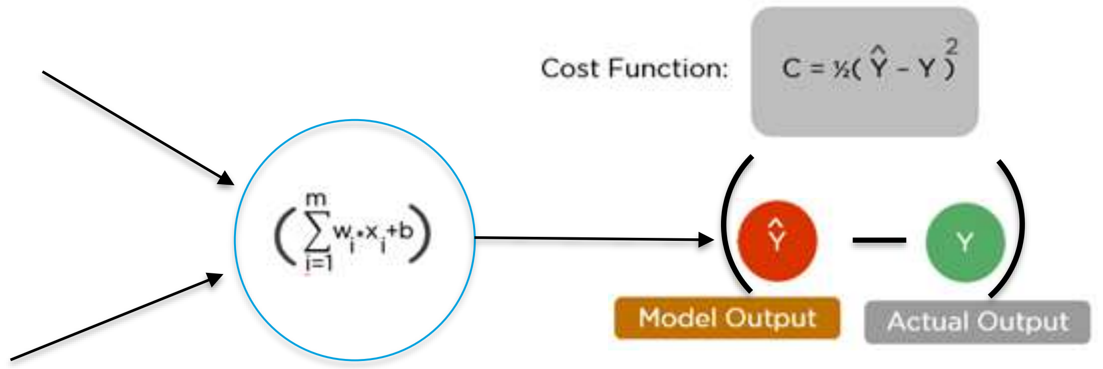


2. It acts like the intercept in a linear equation (y = sum(mx) + c). If sum(mx) is not crossing the threshold but the neuron needs to fire, bias will be adjusted to lower that neuron's threshold to make it fire! Network learns richer set of patterns using bias

3. Though all bias nodes are initialized with same value, what goes into the neuron as bias value is the initial bias value * weight of the bias node

Loss function (Mean Square Loss)

1. What is an optimization algorithm and what is its use? - Optimization algorithms helps us to ***minimize (or maximize)*** an **Objective** function (*another name for **Error** function*) **E(x)** which is simply a mathematical function dependent on the Model's internal **learnable parameters** which are used in computing the target values(**Y**) from the set of *predictors*(**X**) used in the model



2. C = ½((wi.xi + b) – y). In this expression Xi and y come from the data and are given. What the ML algorithm learns is the weight wi and bias b. Thus C = f(wi, b)

3. The optimizer algorithms try to estimate the values of wi and b which when used will give minimum or maximum C. In ML we look for minimum

Back Propagation

1. Back propagation is the process of learning that the neural network employs to re-calibrate the weights at every layer and every node to minimize the error in the output layer

2. During the first pass of forward propagation, the weights are random numbers

3. The output of the first iteration is almost always incorrect. The difference between actual value / class and predicted value / class is the error

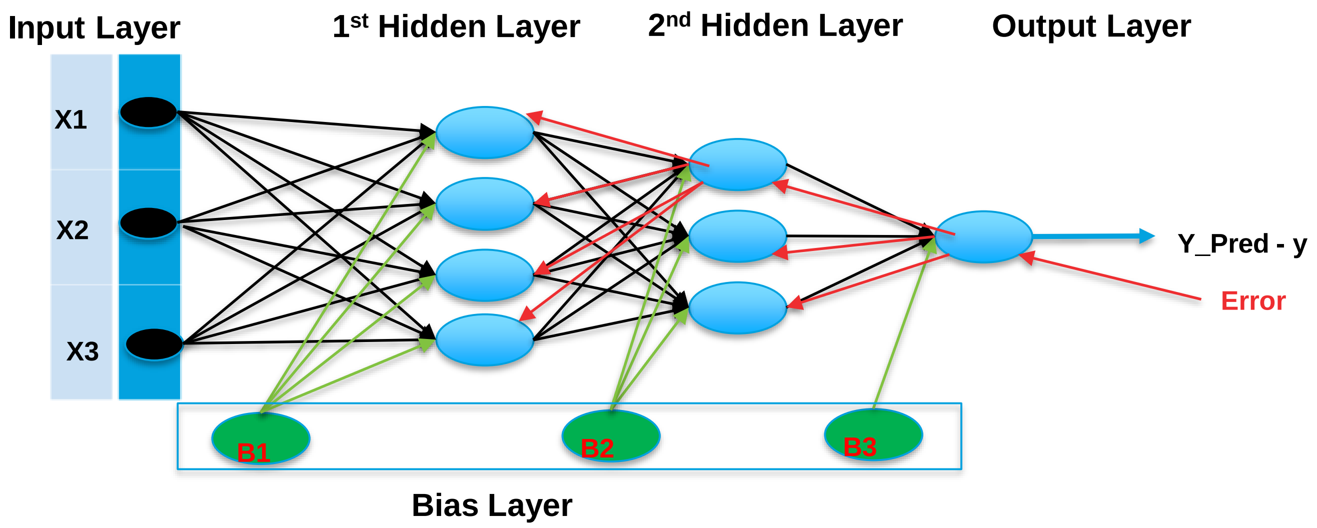
4. All the nodes in all the preceding layers contribute to error and hence need to get their share of the error and correct their weights

5. This process of allocating a proportion of the error (error gradient) to all the nodes in the previous layer is back propagation

6. The goal of back propagation is to adjust weights in proportion to the error contribution and in iterative process identify the optimal combination of weights

7. At each layer, at each node, gradient descent algorithm is applied to adjust the weights

Back Propagation



1. Error in output node shown as e1, is contributed by node 1 ,2 and 3 of layer 2 through weights w(3,1), w(3,2), w(3,3)

2. Proportionate error is assigned back to node 1 of hidden layer 2 is (w(3,1) / w(3,1) + w(3,2) + w(3,3)) * e1

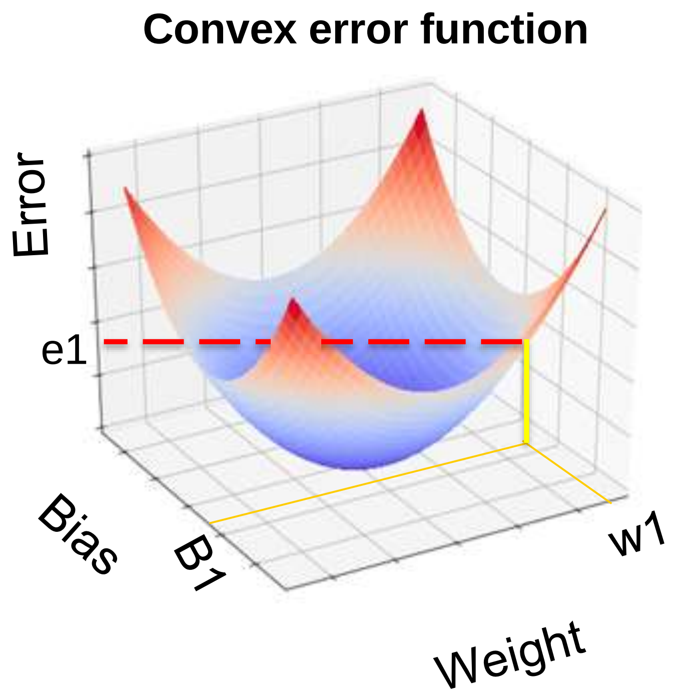
3. The error assigned to node 1 of hidden layer 2 is proportionately sent back to hidden layer 1 neurons

4. All the nodes in all the layers re-adjust the input weights and bias weights to address the assigned error (for this they use gradient descent)

5. The input layer is not neurons, they are like input parameters to a function and hence have no errors

Gradient Descent

The challenge is, all the weights in all the inputs of all the neurons need to be adjusted. It is not manually possible to find the right combination of weights using brute force. Instead, the neural network algorithm uses a learning function called gradient descent



1. A random combination of bias B1 and input weights W1 (showing only one as more than one is not possible to visualize)

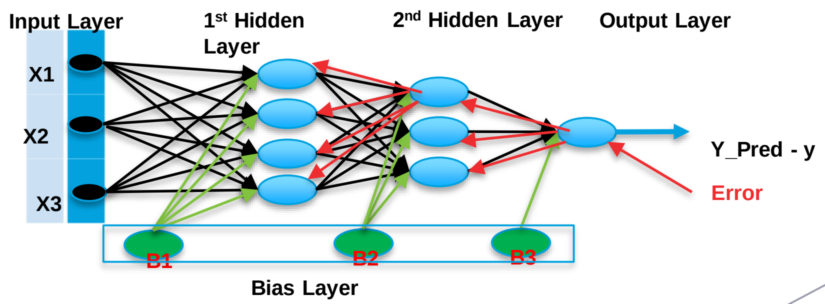
2. Each combination of W1 and B1 is one particular linear model in a neuron. That model is associated with proportionate error e1 (red dashed line).

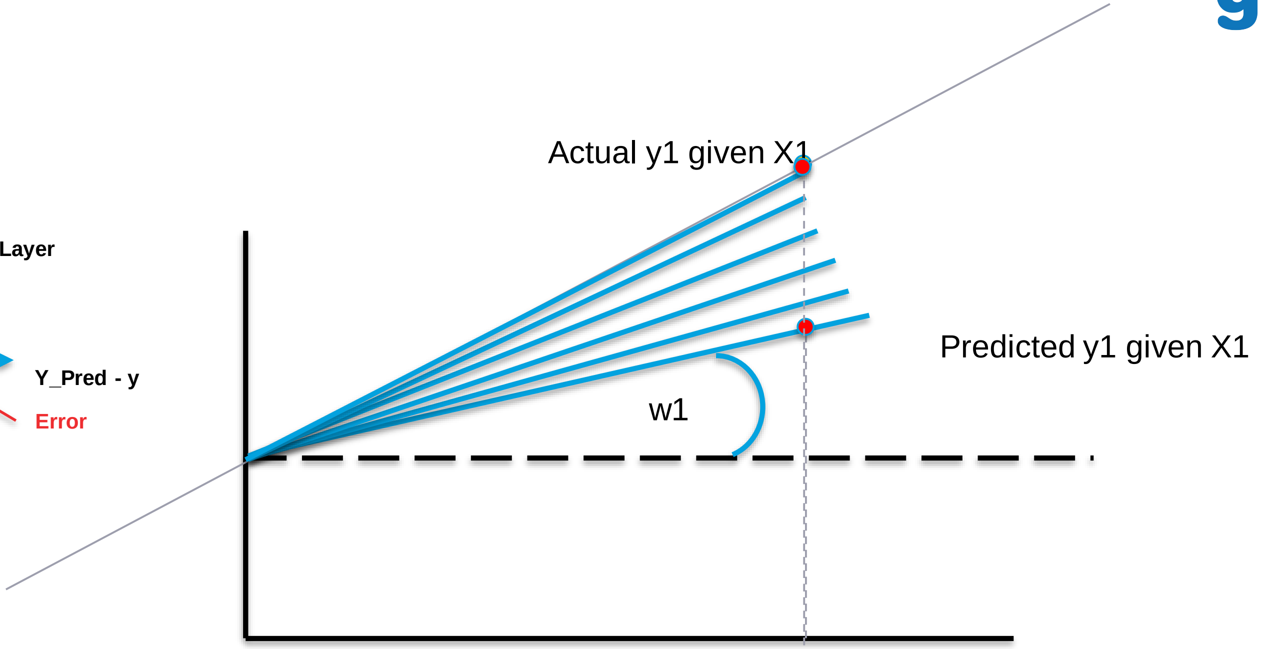
3. Objective is to drive e1 towards 0. For which we need to find the optimal weight (Woptimal) and bias (Boptimal)

4. The algorithm uses gradient descent algorithm to change bias and weight form starting values of B1 and W1 towards the Boptimal, Woptimal.

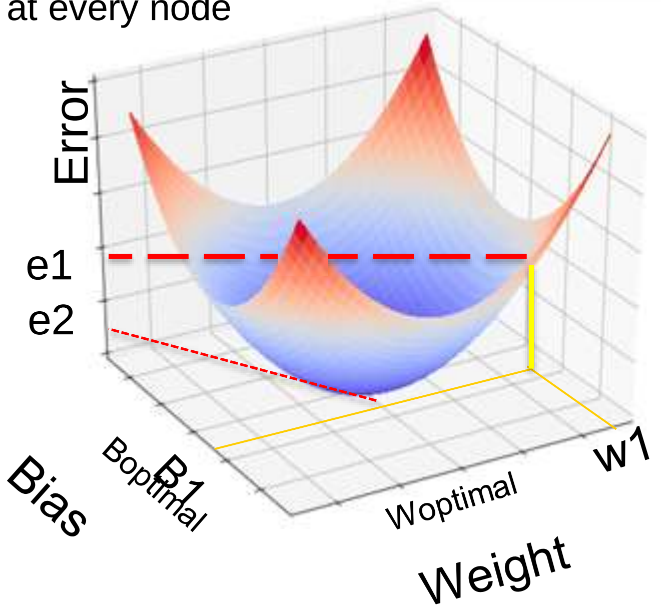
Note: in 3D error surface can be visualized as shown but not in more than 3 dimensions

Gradient Descent





Happens at every node



Least error E2 is at the global minima of the convex function which only one unique combination of weight (woptimal) and bias (boptimal) will fetch us.

Gradient Descent

1. Gradient descent is a way to minimize an objective function / cost function such as Sum of Squared Errors (SSE) that is dependent on model parameters of weight / slope and bias

2. The parameters are updated in such a way that the overall error in the neural network is minimized

3. This movement from starting weight and bias to optimal weight and bias may not happen in one shot. It is likely to happen in multiple iterations. The values change in steps

4. The step size can be influenced using a parameter called Learning Rate. It decides the size of the steps i.e. the amount by which the parameters are updated. Too small learning step will slow down the entire process while too large may lead to an infinite loop

5. The mathematical expression of gradient descent

learning step

$$\theta = \theta - \boxed{\eta \cdot}\nabla_\theta J(\theta)$$

Update Model parameter at e2

Old Model parameter at e1

Gradient descent with learning step

Thank You