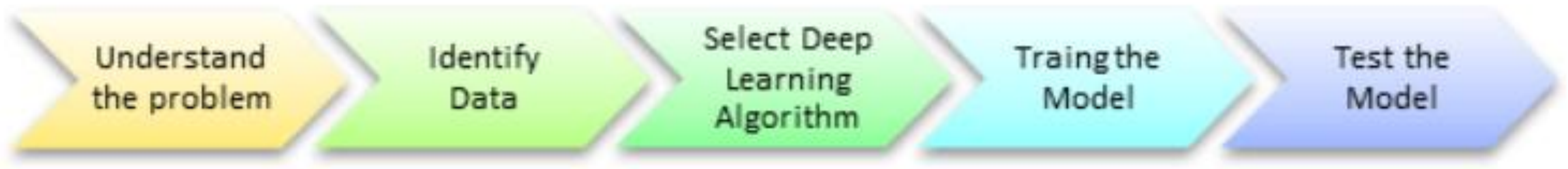
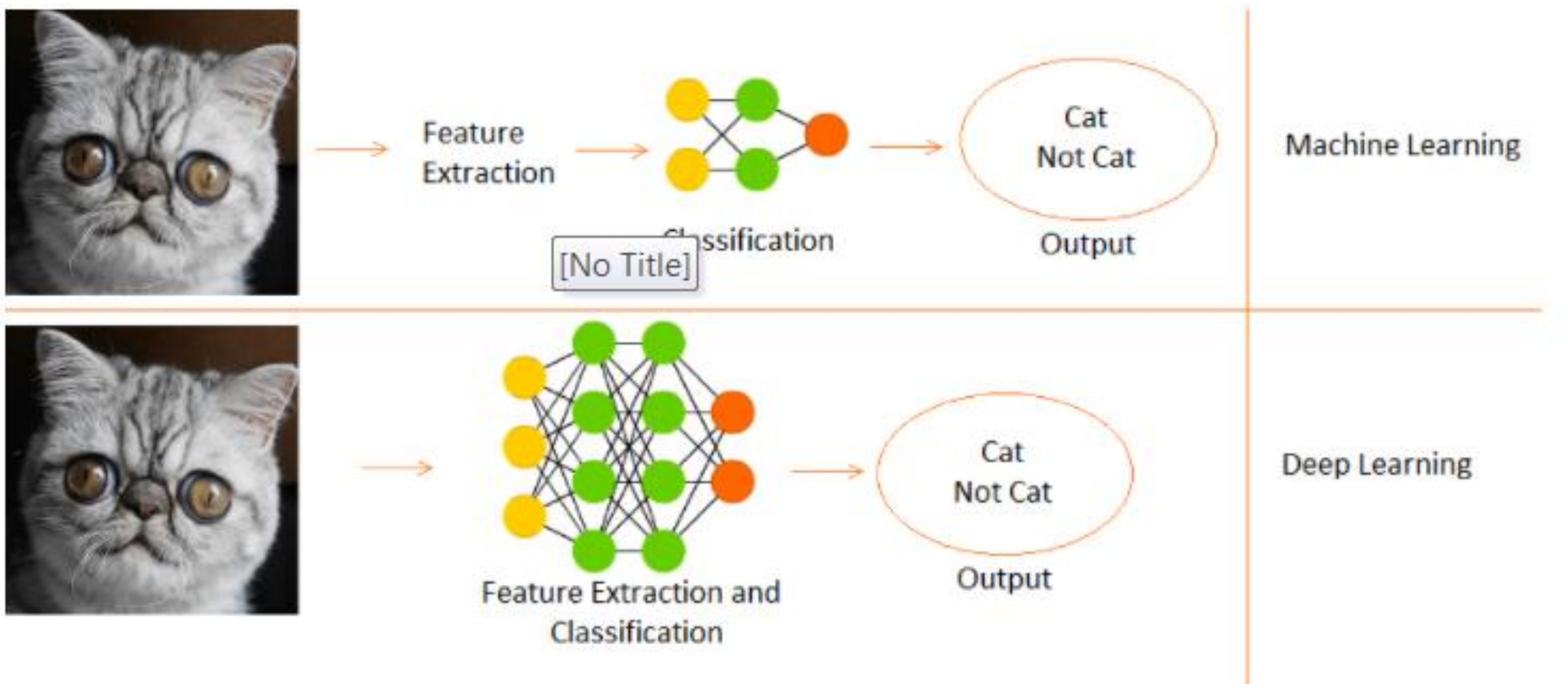
DEEP LEARNING

Deep Learning

- Deep learning is a subdivision of machine learning.
- It imitates the working of a human brain with the help of artificial neural networks which is similar to the network of neurons in a human brain.
- It is called 'deep learning' because it makes use of deep neural networks.



- As the volume of data increases, machine learning techniques become inefficient in terms of performance and accuracy.
- Deep learning solves the problem. It learns features directly from the data.



Deep learning – an analogy

Think how a child learns a language. The child points to an object and says ‘cat’. Others provides a feedback: ‘Yes’ or ‘No’. After enough feedback, the child eventually understands how a cat looks like.

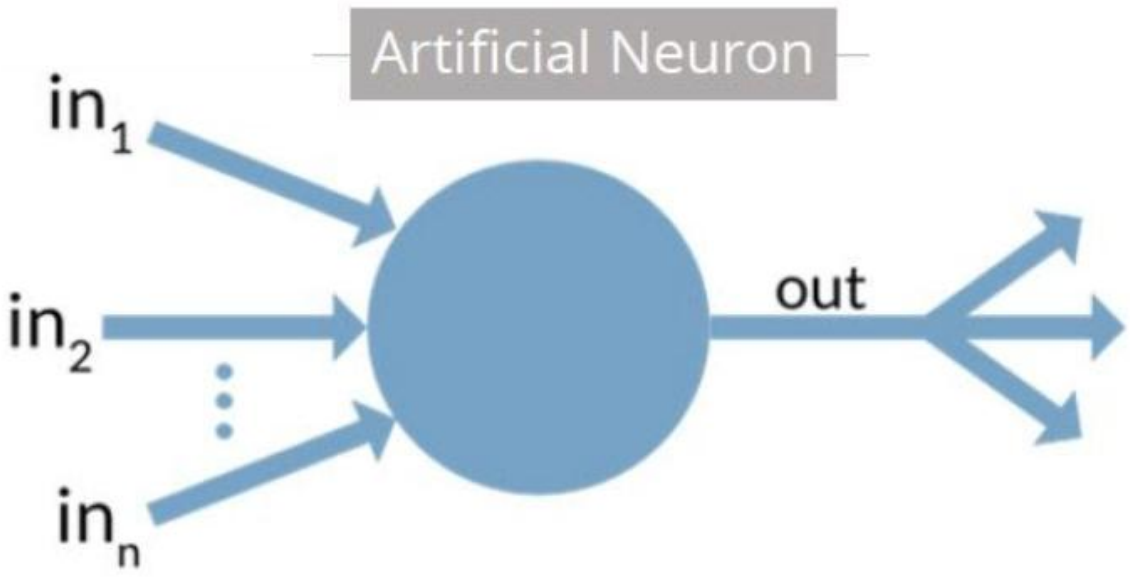
The child’s brain organizes its billions of neurons in order to deliver the right answer. Each neuron transmits signals to other neurons whose hierarchy is quiet complex.

Applications

- ***Speech Recognition (Natural Language Processing)***
- ***Image recognition***
- ***Self-driving cars***
- ***Social networking platforms***
- ***Targeted marketing***
- ***Medical diagnosis***
- ***Ecosystem evaluation***
- ***Computer vision***
- ***Chemical compound identification***

Artificial neurons

- It is an elementary unit in an artificial neural network.
- A mathematical function based on a model of biological neurons.
- One or more inputs are separately weighted.
- Inputs are summed and passed through a nonlinear function to produce output.



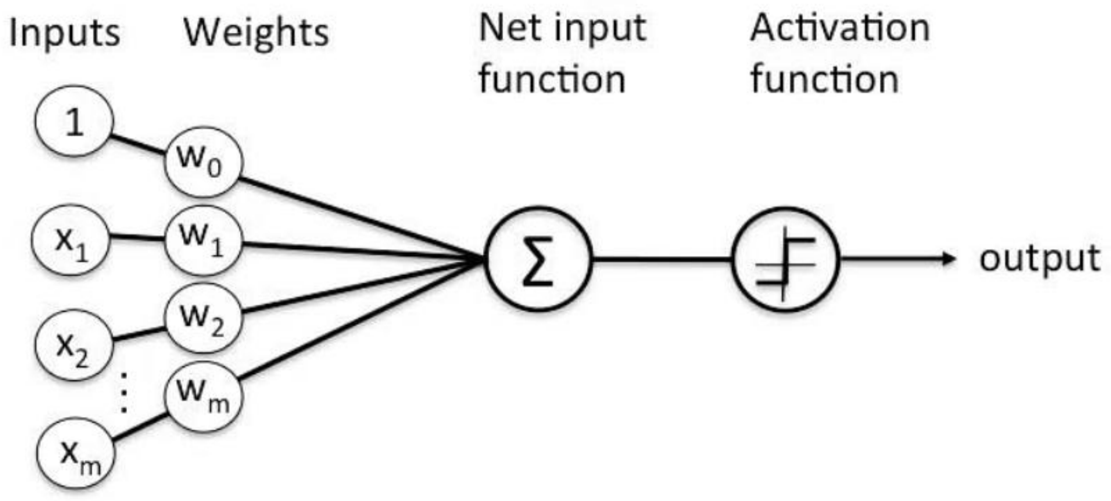
- Every neuron holds an internal state called activation signal.
- Every neuron is connected to another neuron via a connection link.
- Each connection link carries information about the input signal.

Neural Networks

- Neural networks are computing systems with interconnected artificial neurons that work like human brain neurons.
- They are just software simulations and hence are called artificial neural networks.
- They can recognize hidden patterns and correlation in raw data, cluster and classify it.
- This helps in solving complex problems in real-life situation.

Perceptron

- A Perceptron is an algorithm for supervised learning of binary classifiers.



- 2 types of perceptrons:

Single layer and multi layer.

- Single layer perceptrons can learn only linearly separable patterns
- Multilayer perceptrons or feedforward neural networks with two or more layers have the greater processing power

Activation Functions

- The activation functions help the network use the important information and suppress the irrelevant data points.
- It will decide whether the neuron’s input to the network is relevant or not.
- Also referred to as threshold or transformation for the neurons which can converge the network in the process of prediction.
- They are used to map the input between the required values like (0, 1) or (-1, 1).
- The output can then be defined as

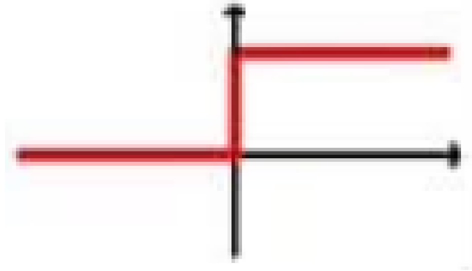
$$y = Activation \sum ((weight * input) + bias)$$

Types of activation functions

Unit step function

- Used while creating a binary classifier.
- Simplest activation function.

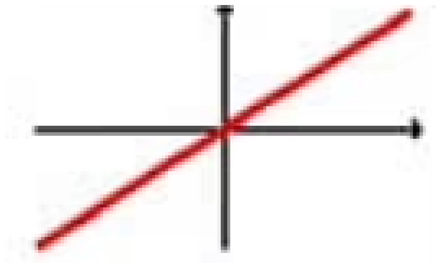
$$f(x) = \begin{cases} 1\,, & x > 0 \\ 0\,, & x < 0 \end{cases}$$



Linear function

- Here the activation is proportional to the input.

$$f(x) = ax$$

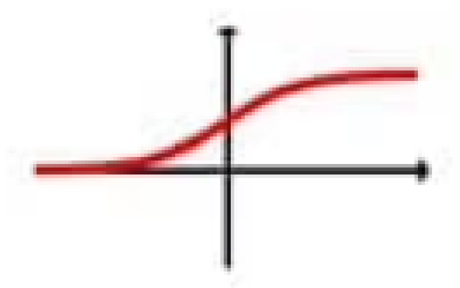


Sigmoid (Logistic)

- Most widely used non-linear activation function only for binary classification problems.
- Sigmoid transforms the values between the range 0 and 1.

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

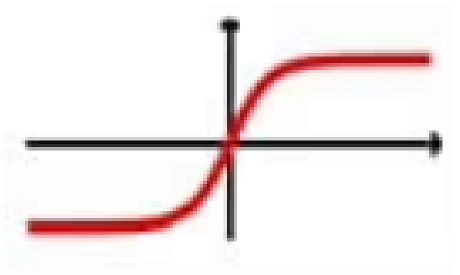
- If input is less than 0.5, output=0, else 1.



Tanh (hyperbolic tangent)

- Sigmoid is not symmetric around zero and all neurons will be of the same sign.
- Tanh scales the sigmoid function to values in the range -1 to 1.

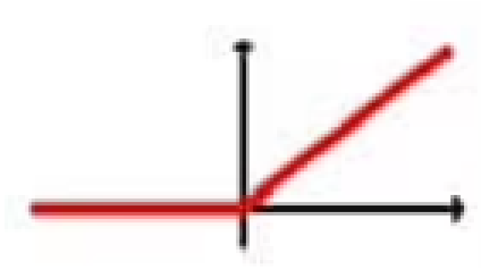
$$f(x) = \frac{2}{1+e^{-2x}} - 1$$



ReLU (Rectified Linear Unit)

- Non-linear activation function.
- For negative input values, the result is zero.
- They do not activate all the neurons at the same time.

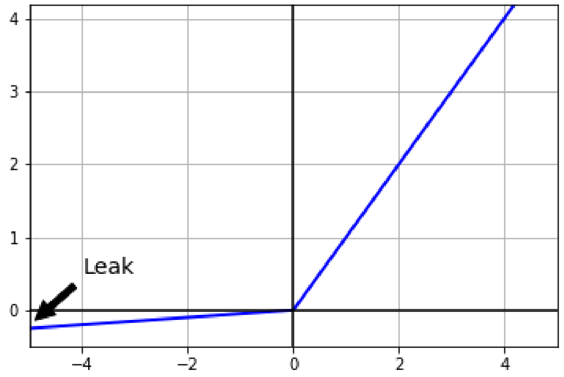
$$f(x) = \max(0, x)$$



Leaky ReLU

- Improved version of ReLU
- Instead of defining the ReLU as 0 for negative values of x, a small linear component of x is defined here.

$$f(x) = \begin{cases} 0.01x, & x < 0 \\ x, & x \geq 0 \end{cases}$$



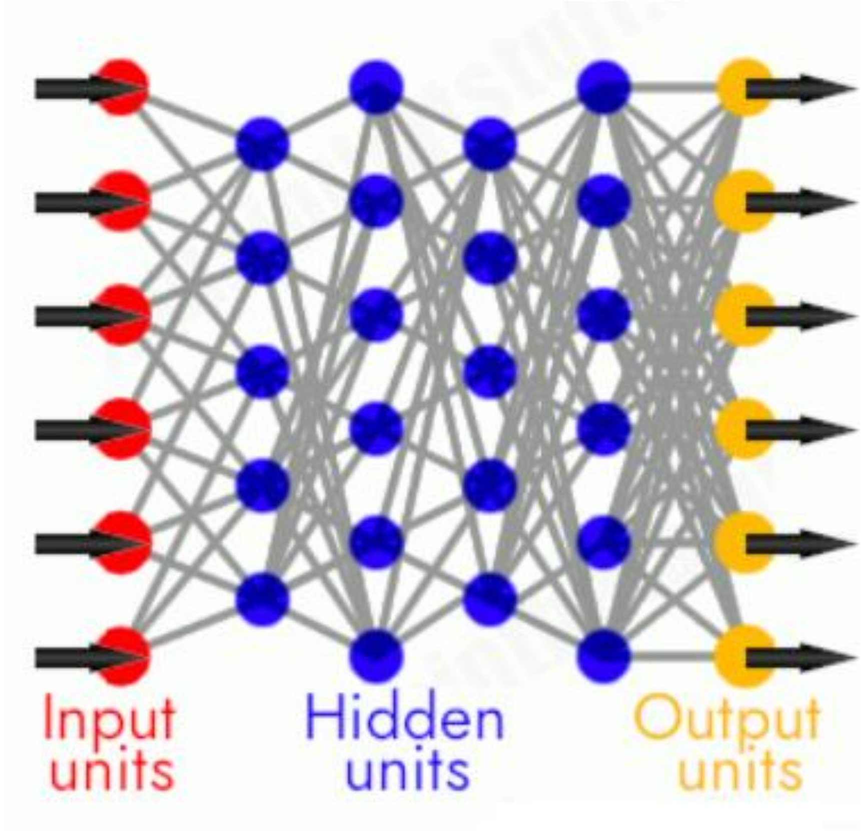
Softmax

- Combination of multiple sigmoids.
- This can be used for multi class classification problems.
- This function returns the probability for a data point belonging to each individual class.

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} x_j}, i = 0,1,2,..n$$

Neural networks-layers

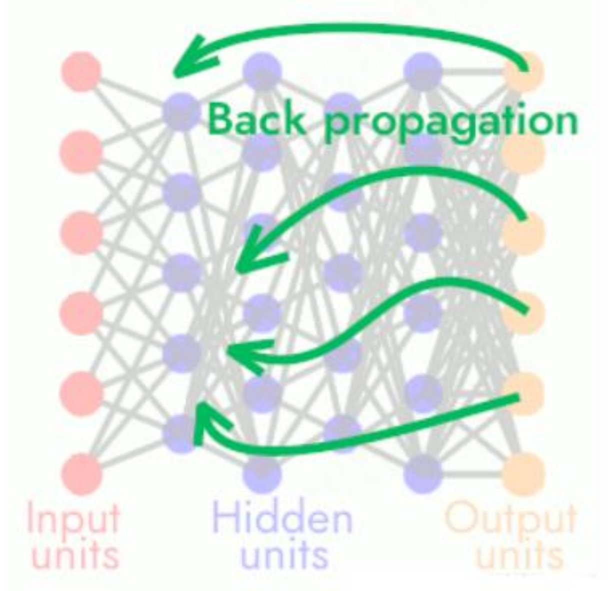
- Neural networks contain millions of thousands of artificial units called **neurons**.
- The units or nodes receiving the inputs are the **input units** forming the **input layer**.
- The nodes signaling its response out are the **output units** forming the **output layer**.
- In between the input and output units lie one or more layers of **hidden units** forming the **hidden layers**.



- Most neural networks are **fully connected (FC)**, where each node is connected to one another.
- The connections between one unit and another are represented by a number called a **weight**.
- It can be either positive (if one unit excites another) or negative (if one unit suppresses or inhibits another).
- The higher the weight, the more influence one unit has on another.
- A simple neural network consist of just 3 layers.
- When more number of hidden layers are stacked one after the other to solve complex problems, it is called a **deep neural network (DNN).**
- Each of the layers in the network contain an **activation function** which determines the firing of a neuron.

Neural network working - forward and backward propagation

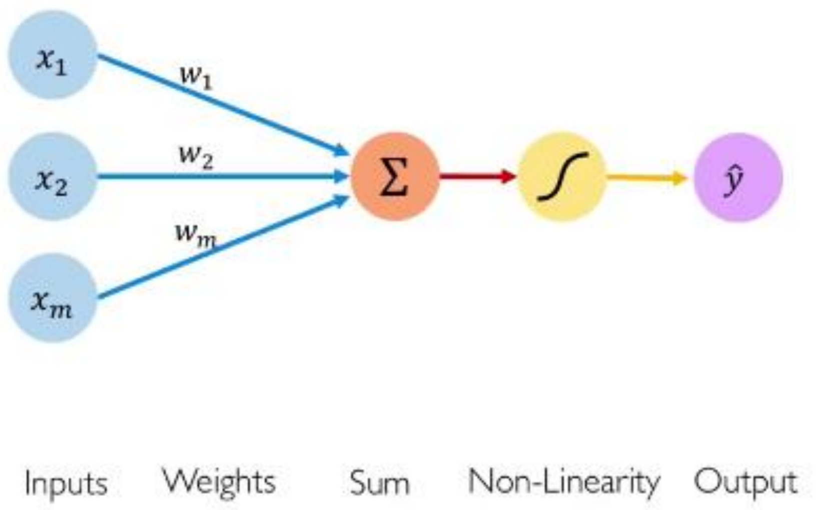
- Patterns of information fed into the network through input units triggers the hidden units and finally reach the output units.
- This design is called a **feedforward network**.
- The firing of units is determined by the weights of the connections they travel along.
- To make a neural network learn, a feed back mechanism is used called **backward propagation** or simply back propagation.



- The predicted output ($\hat{y}$) of the network is compared with the desired output (y)
- The difference between y and $\hat{y}$ is the **error (loss).**
- A simple loss function can be represented as:

$$Loss = (y - \widehat{y})^2$$

- This loss is then back propagated through the network.
- The aim is to minimize the loss which is done by updating the weights.
- This is done by using various **optimizers**.



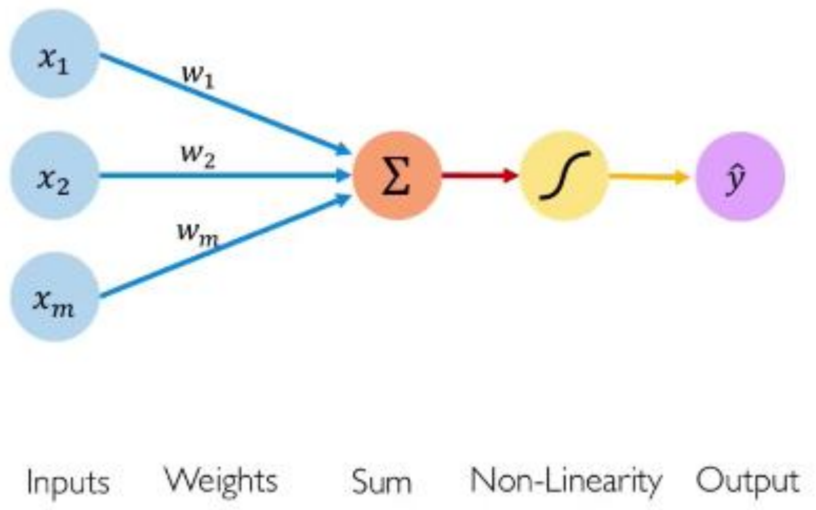
$$w_{1new} = w_1 - \eta \frac{\partial L}{\partial w_1}$$
$$w_{2new} = w_2 - \eta \frac{\partial L}{\partial w_2}$$
$$w_{3new} = w_3 - \eta \frac{\partial L}{\partial w_3}$$

η is the learning rate which shouldn’t be too small or too large.

Example

Consider a single record with the following features regarding whether a student will pass or fail an examination

Playing: 2 hrs
Studying: 4 hrs
Sleeping: 8 hrs
Output: 1

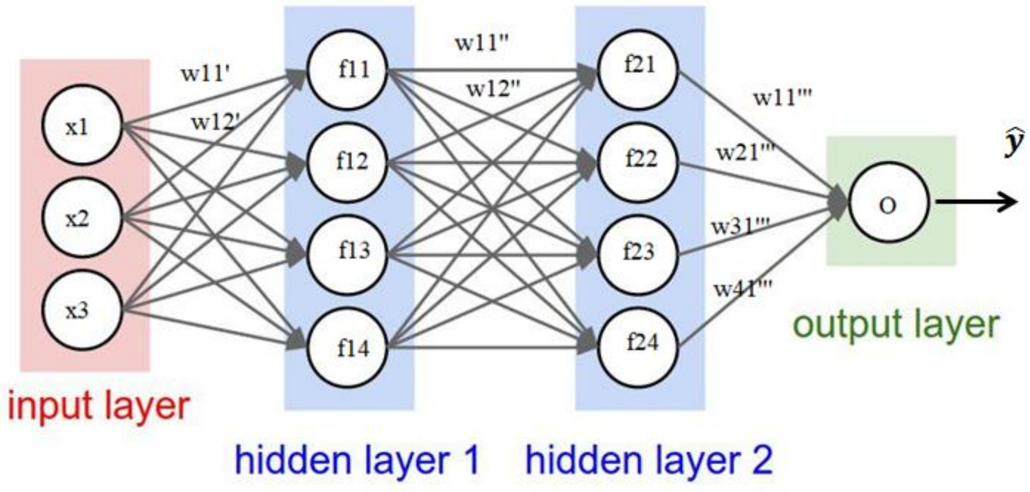


Suppose the result is predicted as 0,
Loss = 1-0 = 1 which is high.

This is then back propagated to update the weights, so that the model correctly predicts the result as 1.

Multilayer Neural Network

The loss function $(y - \widehat{y})^2$ is calculated and the network aims to minimize this using an optimizer.



When the number of records increase the loss function is called **cost function**.

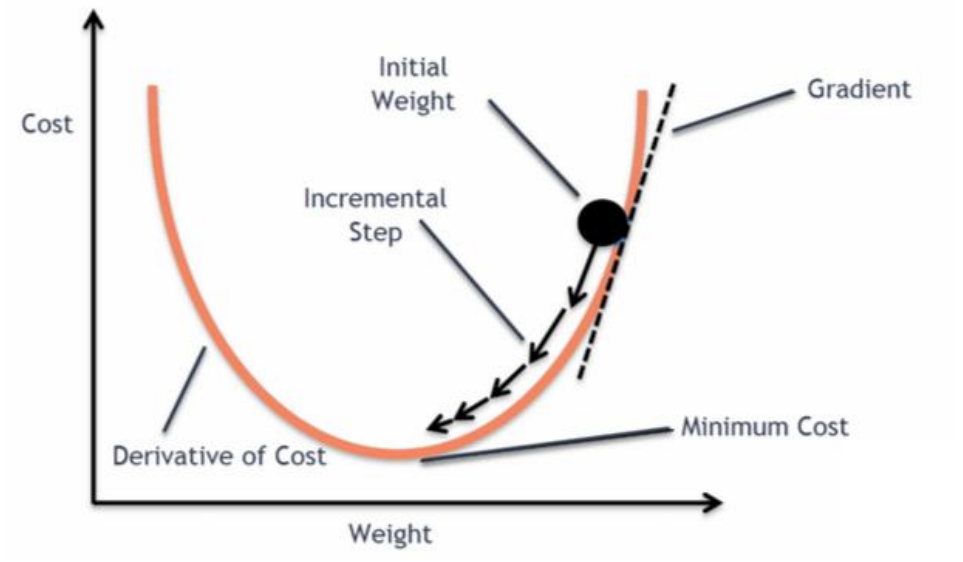
Gradient Descent

- It is an optimizer used to reduce the cost function.
- Weight updation is done using

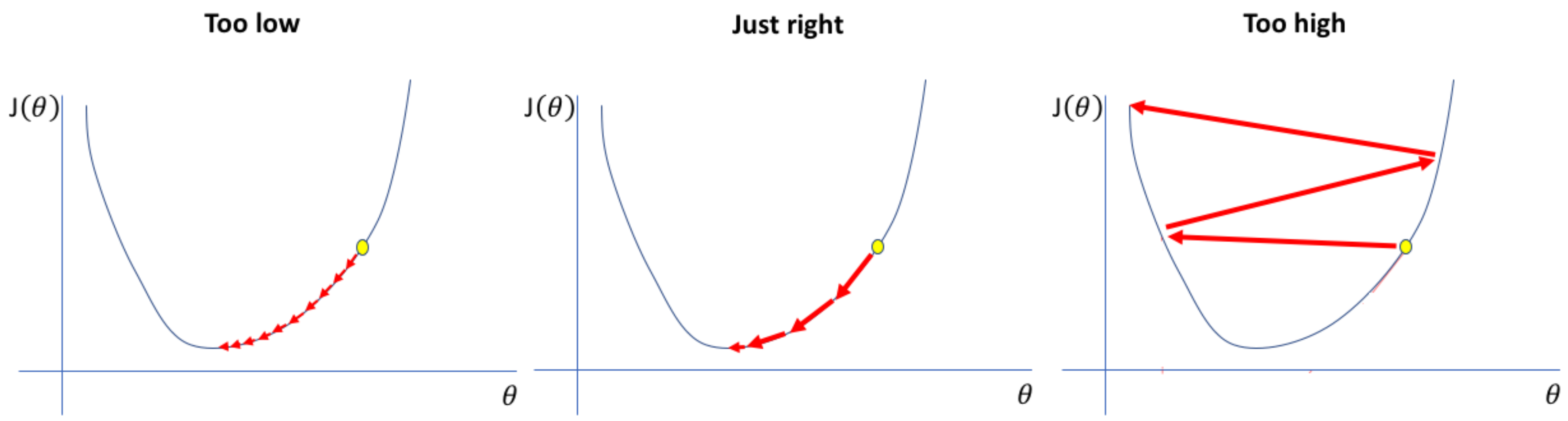
$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w}$$

where $\frac{\partial L}{\partial w_{old}}$ is the gradient.

- This happens after the back propagation for each **epoch**.



- When the slope (gradient) is positive, older weight needs to be reduced.
- When slope is negative, older weight should be increased.
- When learning rate is too small, convergence to local minima will take time.
- If it is too large, convergence will never occur and weights will just move back and forth.

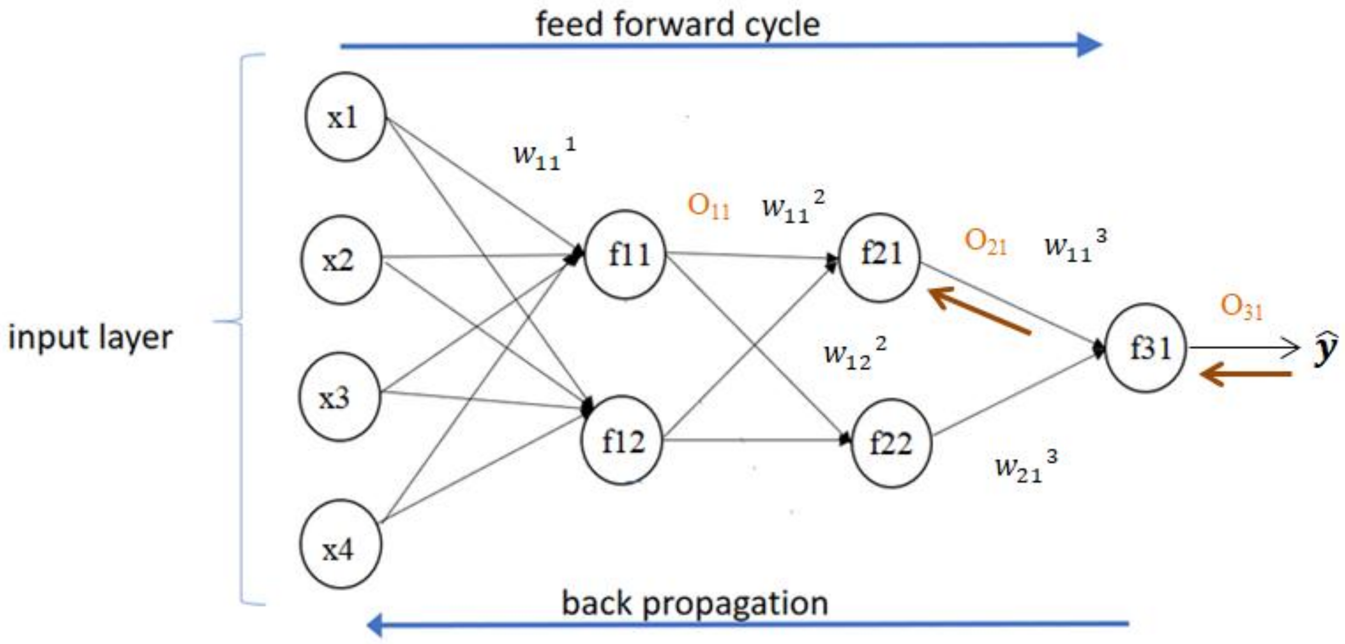


A small learning rate requires many updates before reaching the minimum point

The optimal learning rate swiftly reaches the minimum point

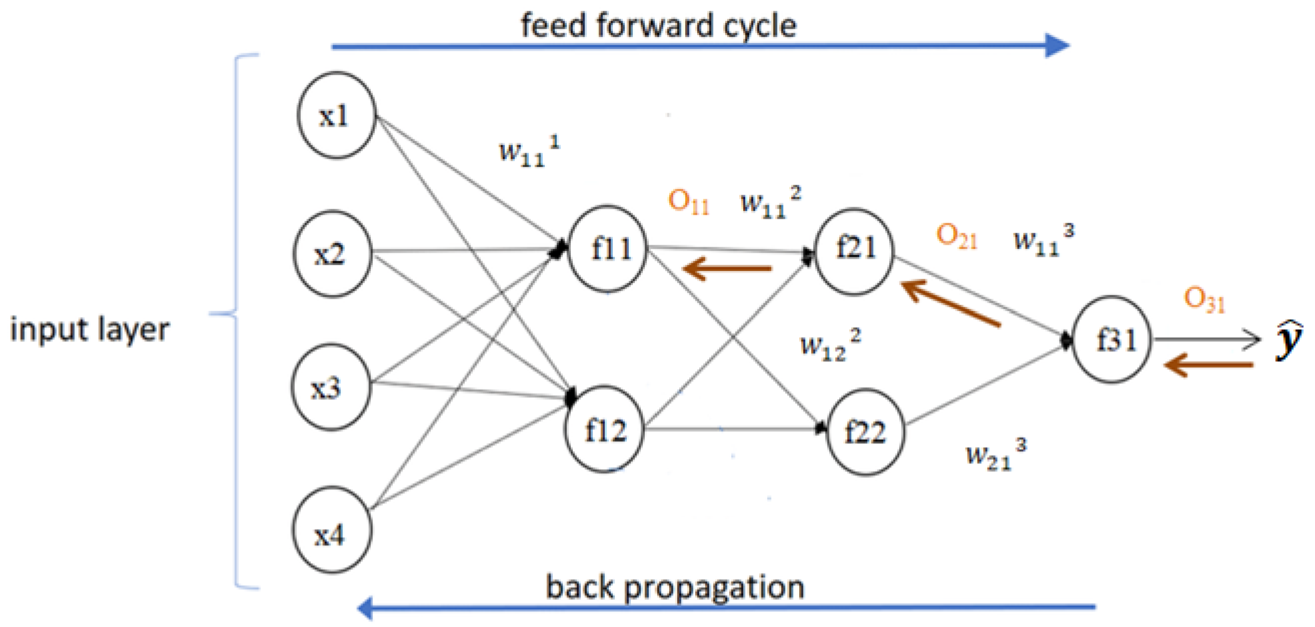
Too large of a learning rate causes drastic updates which lead to divergent behaviors

Weight updation- Chain rule



$$w_{11new}{}^{3} = w_{11old}{}^{3} - \eta \frac{\partial L}{\partial w_{11old}{}^{3}}$$

$$\frac{\partial L}{\partial w_{11old}{}^{3}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w_{11old}{}^{3}}$$



$$w_{11new}^{\;\;2} = w_{11old}^{\;\;2} - \eta \frac{\partial L}{\partial w_{11old}^{\;\;2}}$$

$$\frac{\partial L}{\partial w_{11old}^{\;\;2}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial O_{21}} \cdot \frac{\partial O_{21}}{\partial w_{11old}^{\;\;2}}$$

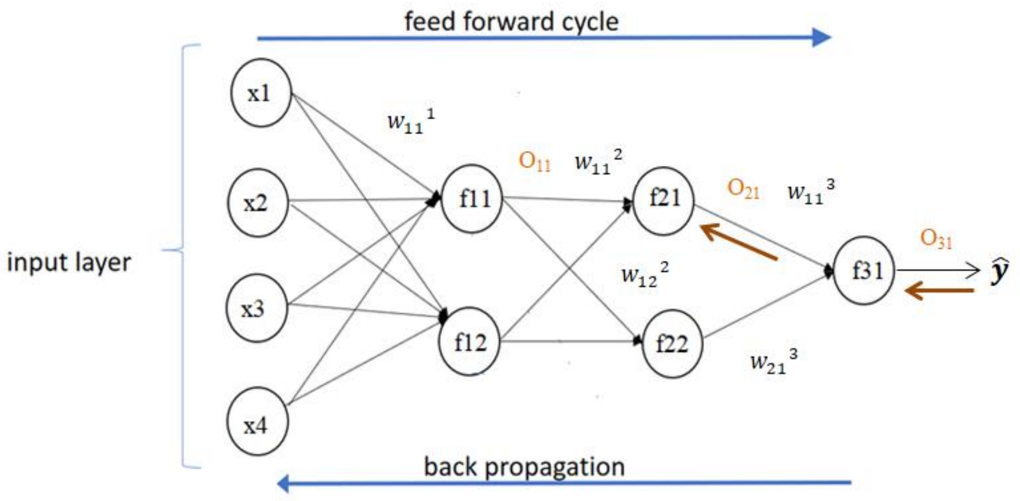
and so on..

Vanishing gradient problem

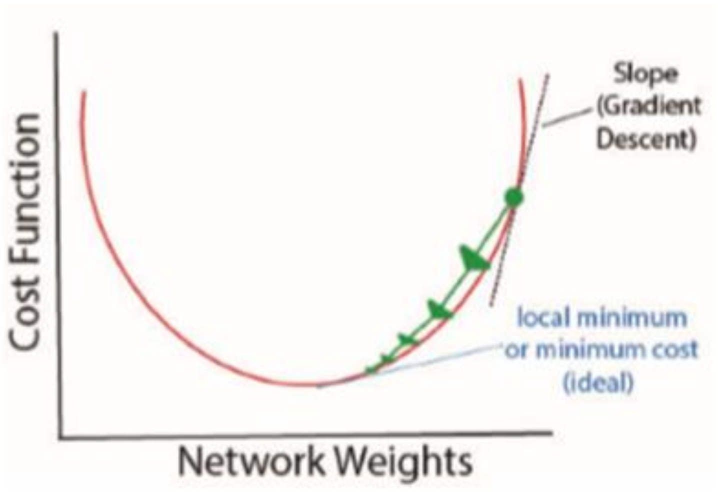
- Sigmoid was the most commonly used activation function in all the layers back in times.
- Sigmoid function ($\Phi(z)$) maps the weighted sum to values between 0 and 1.
- But its derivative always lies between 0 to 0.25.

$$w_{11new}{}^{3} = w_{11old}{}^{3} - \eta \frac{\partial L}{\partial w_{11old}{}^{3}}$$

$$\frac{\partial L}{\partial w_{11old}{}^{3}} = \frac{\partial L}{\partial O_{31}} \cdot \frac{\partial O_{31}}{\partial w_{11old}{}^{3}}$$



- Thus, the gradient will be always less than 0.25.
- As the number of layers in the deep network increases, the product of these gradients reduces further i.e., gradient vanishes.
- This causes the older weights to get hardly updated.
- This is the **vanishing gradient problem** which makes it difficult to create a network with multiple channels.
- Tanh also causes the same problem.

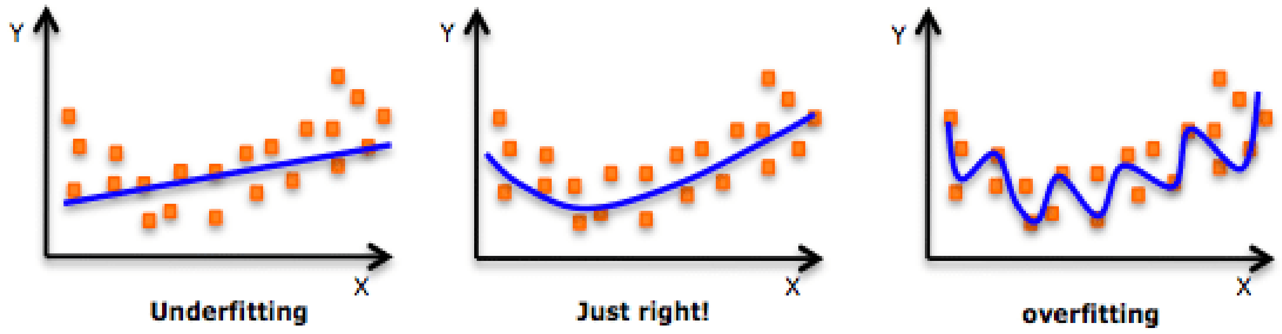


Exploding gradient problem

- Exploding gradient occurs when the derivatives will get larger and larger as we go backward with every layer during backpropagation.
- This situation is the exact opposite of the vanishing gradients.
- This problem happens because of weights, not because of the activation function.

Overfitting

- In Overfitting, the model tries to learn too many details in the training data along with the noise from the training data.
- As a result, the model performance is very poor on unseen or test datasets.
- Therefore, the network fails to generalize the features or patterns present in the training dataset.



An example of overfitting, underfitting and a model that's "just right!"

Underfitting

- Underfitting is a scenario in data science where a data model is unable to capture the relationship between the input and output variables accurately, generating a high error rate on both the training set and unseen data.

Drop Out and Regularization

- A multiple layer NN can never underfit.
- But for a fully connected NN, overfitting can occur due to the huge number of weights and biases.
- This can be handled using regularization techniques.
- One method is to use L1 and L2 regularization.
- The other is dropout.

L1 and L2 regularization

- This is done by using smaller weights leading to simpler models.
- A regularization term called penalty is added to the loss to give the final cost function.

Cost function = Loss + Regularization term

L1 regularization (Lasso regularization)

$$cost\ function = loss + \lambda\,\Sigma|w_i|$$

λ is the regularization parameter

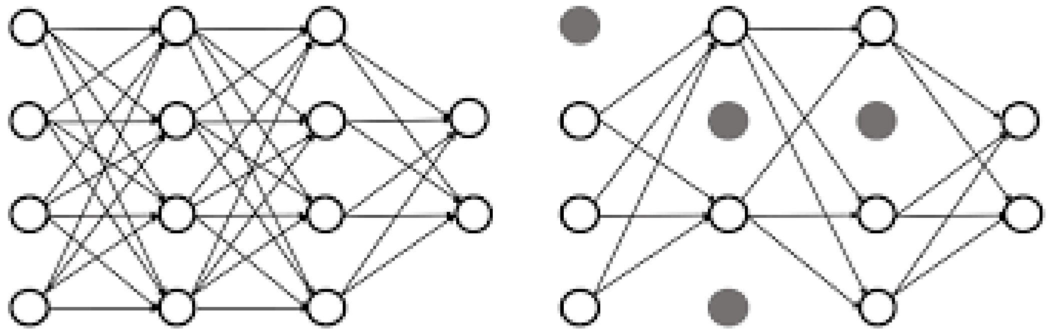
w_i are the weights.

L2 regularization (Ridge regularization)

$$cost\ function = loss + \lambda\, \Sigma {w_i}^2$$

- When regularization term is added, we are actually increasing the cost function.
- Thus, if the weights are larger, it will also make the cost to go up.
- This causes the training algorithm to bring the weights down by penalizing the weights forcing them to take smaller values thereby regularizing the network.

Dropout



(a) Standard Neural Network.　　(b) Network applying dropout

- Like a random forest classifier, only some of the input features are randomly selected during each forward propagation.
- The remaining nodes are dropped out or deactivated.
- For a particular epoch, during the back propagation only the weights of those nodes which were fired during the forward propagation gets updated.

- The amount of drop out is determined by the drop out ratio, p
- $0 \leq p \leq 1$
- For each layer, different dropouts can be fixed.
- During the testing phase, all the neurons get connected and the weights fixed during the training phase gets multiplied by the dropout ratio, p.
- The validation is done using these weights.
- p can be determined through hyper parameter tuning.
- To be specific, when an overfitting problem is happening, better to keep the p value higher (greater than 0.5).

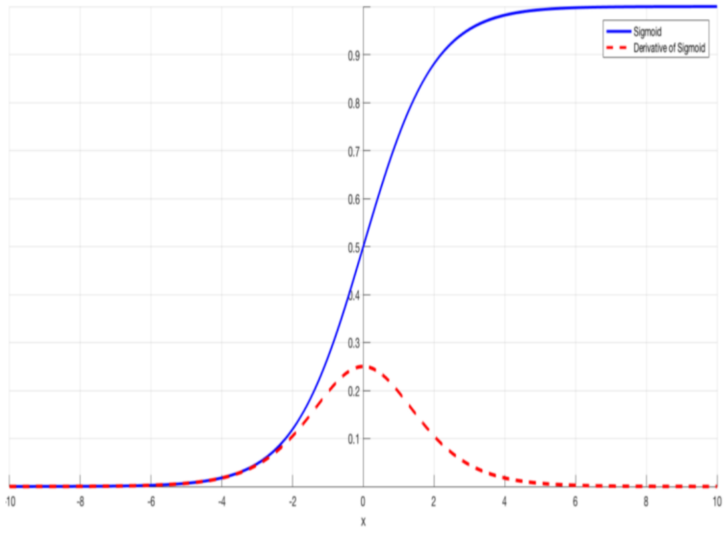
Why ReLU?

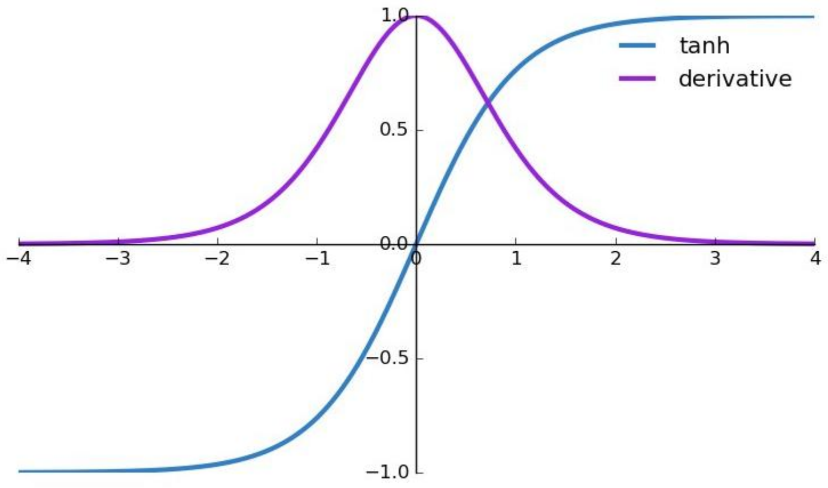
Sigmoid activation function

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

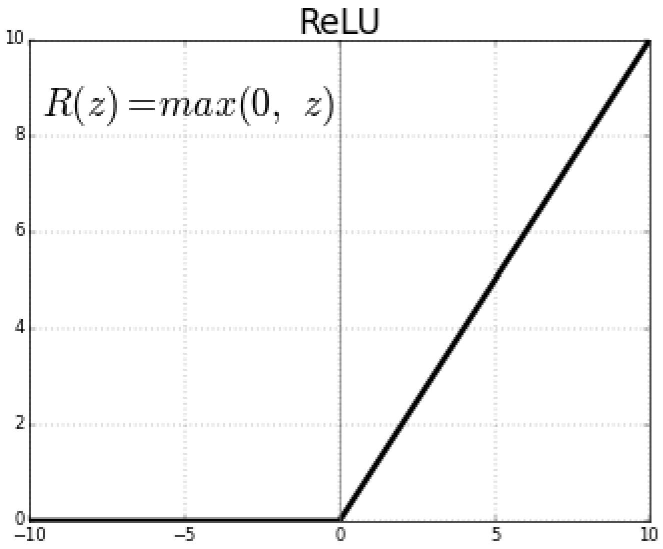
Tanh activation function

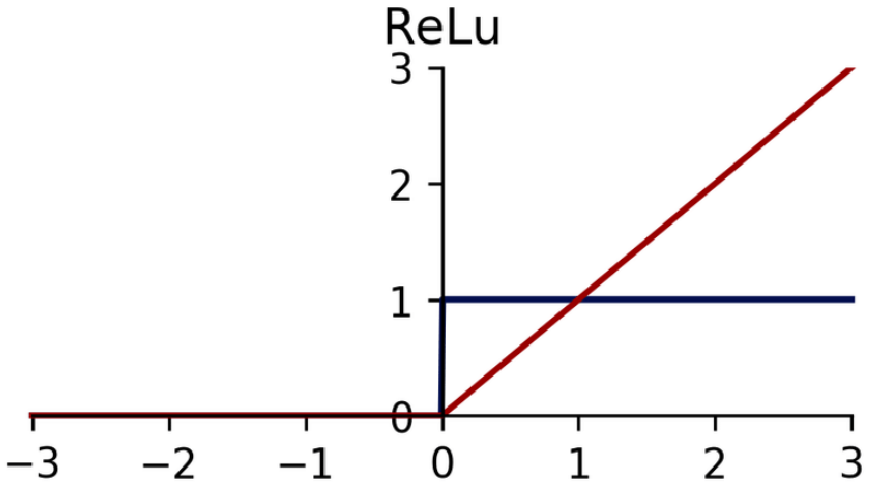
$$f(x) = \frac{2}{1+e^{-2x}} - 1$$





- For sigmoid, $0 \le \frac{\partial f}{\partial z} \le 0.25$
- For tanh, $0 \le \frac{\partial f}{\partial z} \le 1$
- They create vanishing gradient problem.
- Relu can solve this problem
- The graph of Relu is a 45° line whose derivative is 1.

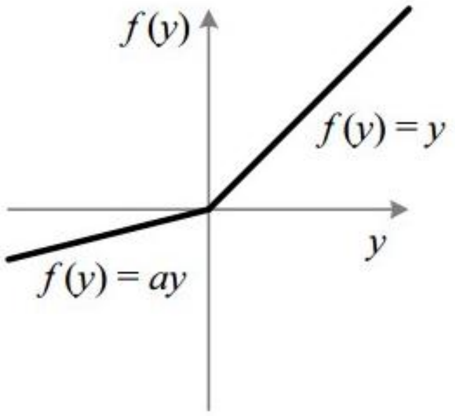




- So, while updating the weights, only 2 values are possible for the derivative terms in the updation formula.

$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w}$$

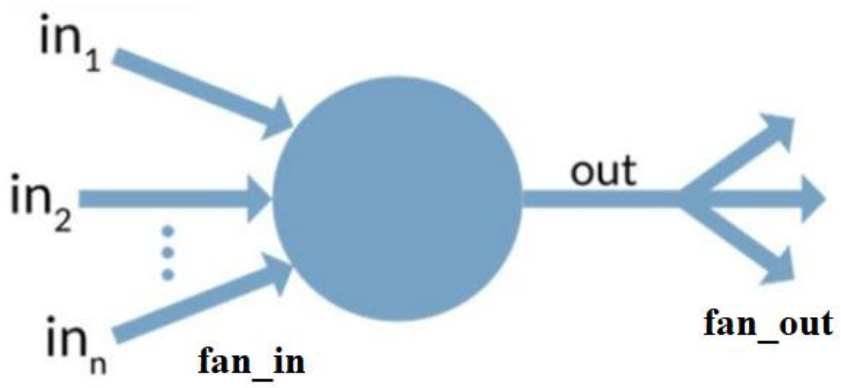
- Here, the term $\frac{\partial L}{\partial w}$ will be either 0 or 1.
- When it is 1, depending on the learning rate, a small updation to the older weight will occur thus causing the gradient descent to converge.
- But, if the value is 0, no updation occurs which create a **dead neuron** or dead activation function.
- A solution to thus is the **leaky relu**.
- The derivative of this function will be a.
- Removes dead activation function.



Weight initialization techniques

Keys:

- Weights should be small
- Weights should not be same
- Weights should have good variance



Uniform distribution

Here weights will be sampled from a uniform distribution with a and b as follows:

$$w_{ij} \sim Uniform\left[\frac{-1}{\sqrt{fan\_in}}, \frac{1}{\sqrt{fan\_in}}\right]$$

Xavier/Glorat distribution

<u>Xavier Normal (Glorat Normal)</u>

$$w_{ij} \sim N(0, \sigma);\ \sigma = \sqrt{\frac{2}{fan\_in + fan\_out}}$$

Xavier Uniform (Glorat Uniform)

$$w_{ij} \sim U\left[\frac{-\sqrt{6}}{\sqrt{fan\_in + fan\_out}}, \frac{\sqrt{6}}{\sqrt{fan_{in} + fan\_out}}\right]$$

He init

He normal

$$w_{ij} \sim N(0, \sigma);\ \sigma = \sqrt{\frac{2}{fan\_in}}$$

He uniform

$$w_{ij} \sim U\left[-\sqrt{\frac{6}{fan\_in}}, \sqrt{\frac{6}{fan\_in}}\right]$$

- Different techniques work differently for different datasets and problems.
- Uniform and Xavier Glorat distribution works well with sigmoid activation function.
- He init method works for Relu activation function.

GD, SGD, Minibatch SGD

- When all the n data points of the dataset is considered at one epoch for the updation of weights, the optimizer is called **gradient descent (GD)**.

  The loss function is given by:

$$\sum_{i=1}^{n}(y-\widehat{y})^2$$

- When only a single data point is considered at one epoch for updating the weights, it is called **stochastic gradient descent (SGD)**.

$$(y-\widehat{y})^2$$

- When only a batch of the original dataset is considered, where batch size k<n, then it is called **mini batch SGD**.

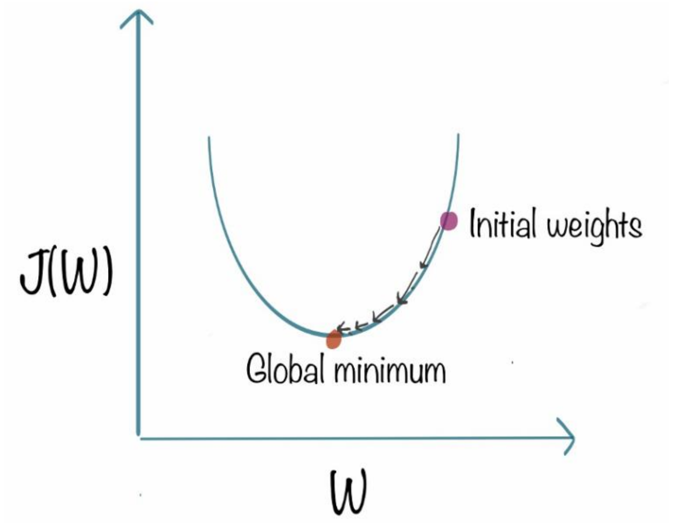
$$\sum_{i=1}^{k}(y-\widehat{y})^2$$

- When the dataset contains millions of records, using SGD is not practical as it takes more time to converge.
- For GD, the amount of memory required for large datasets is large.
- Then we go for mini batch SGD where the batch size needs to be specified.
- Mini batch SGD is used now in almost all neural networks.

Global minima and Local minima

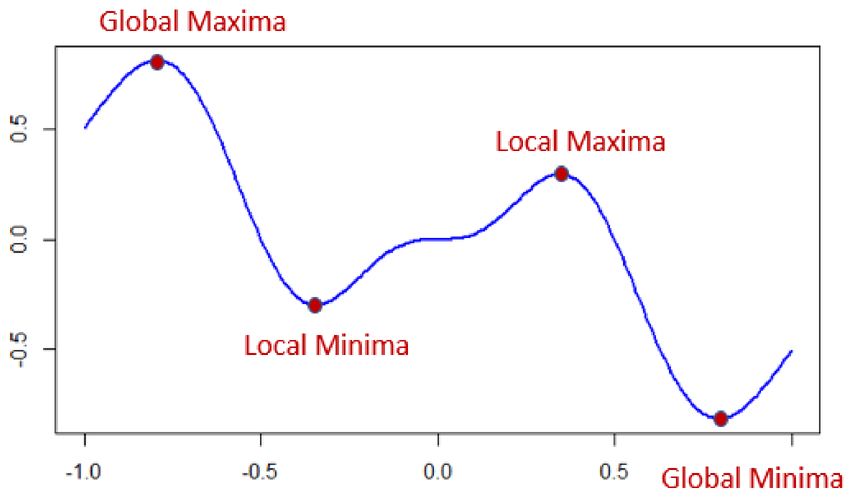
- $L(w) = \sum_{i=1}^{k}(y - \hat{y})^2$ is a loss function called **mean squared error (MSE)**

  This creates the following graph



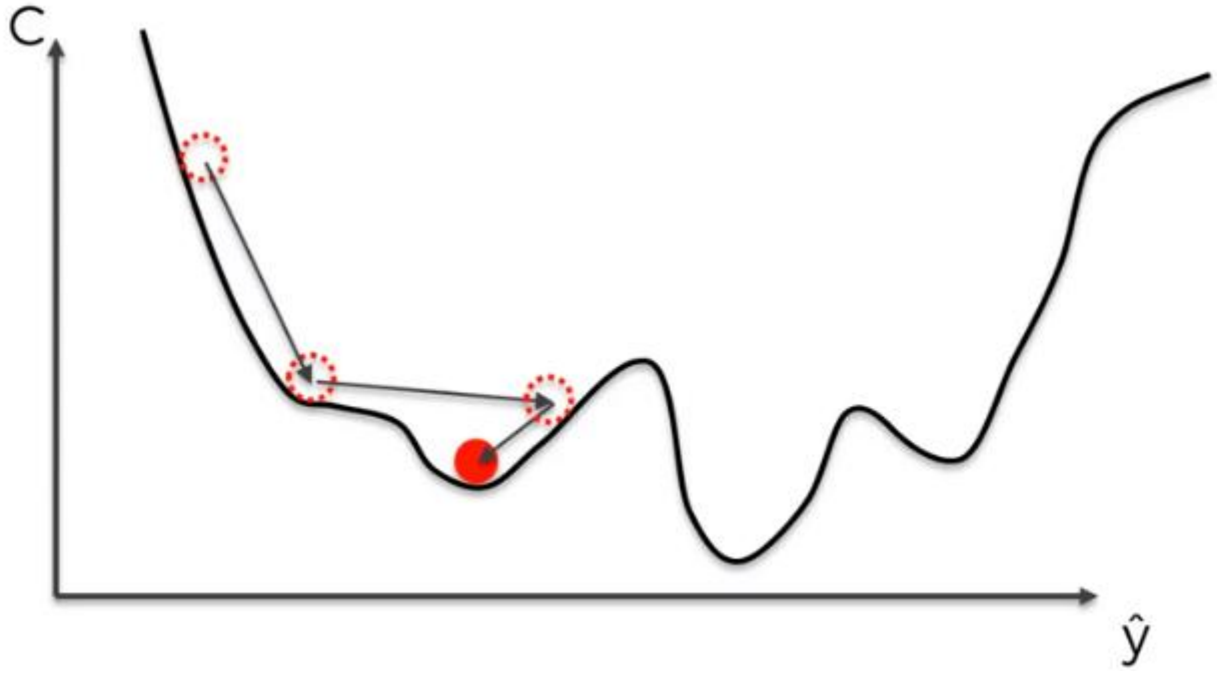
- This has only one global minima but no maxima.

- For other types of loss functions, the shape of the curve will vary.



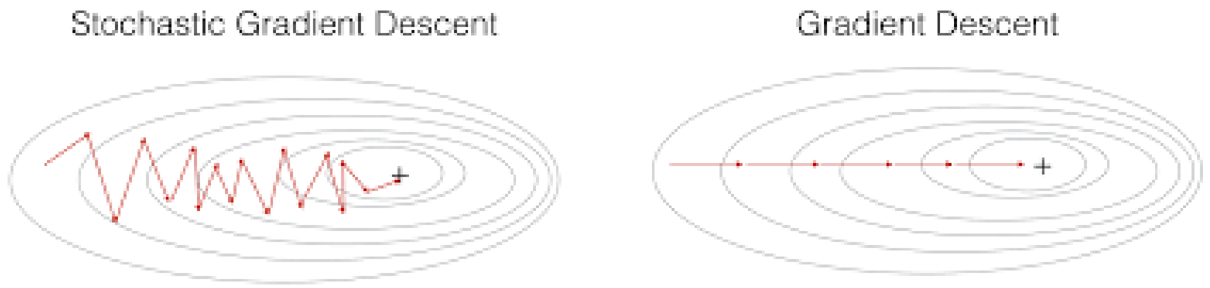
- Here, global minima as well as local minima will be present.
- Local minima will be the lowest error point in that locality.
- Similarly the maxima.
- The derivatives of the loss at these points will be zero.
- This causes the network to misinterpret the weight at that point to be the perfect one.

- The loss function may get converged to the wrong minimum point where the loss is still not zero or closer to zero.



- This is prevented by using advanced optimizers which are modifications of the older ones.

SGD with momentum



- GD converges smoothly to the global minima but SGD (mini batch SGD) will have a lot of noisy data on its convergence curve.
- SGD with momentum helps accelerate gradient vectors in the right directions, leading to faster convergence.
- This is done using exponentially moving average.

Let at $t_1$the gradient generated be b_1, at t_2 it be b_2 and so on… in a normal gradient descent.

Based on the concept of moving average, let the gradient at time t_1

$V_1 = b_1$

At t_2, $V_2 = \beta V_1 + (1-\beta)b_2\ where, 0 \leq \beta \leq 1$

Let$\beta = 0.5$

Then, $V_2 = 0.5b_1 + 0.5b_2$

At t_3, $V_3 = \beta . V_2 + (1-\beta)b_3 = \beta(\beta V_1 + (1-\beta)b_2) + (1-\beta)b_3$

$= \beta^2 b_1 + \beta(1-\beta)b_2 + (1-\beta)b_3 = 0.25b_1 + 0.25b_2 + 0.5b_3$

This is the moving average.

This try to remove the noise in the convergence curve i.e., smoothens the curve.

SGD with momentum can be defined as follows:

$$w_{new} = w_{old} - \eta \frac{\partial L}{\partial w_{old}}$$

$$\boldsymbol{w_t = w_{t-1} - \eta V_{dwt}}$$

$$V_{dwt} = \beta V_{dwt-1} + (1 - \beta) \frac{\partial L}{\partial w_{t-1}}$$

$\boldsymbol{\beta}$ is the momentum. Commonly used $\boldsymbol{\beta}$ value is 0.9 or 0.95

Adaptive Gradient Descent optimizer (Adagrad)

$$w_t = w_{t-1} - \eta \frac{\partial L}{\partial w_{t-1}}$$

where w_t is the new weight and w_{t-1} is the old weight

- In GD, SGD and mini batch SGD, the learning rate was the same for all neurons and all epochs.
- The idea behind Adagrad is to **use different learning rates for different neurons in different epochs**.
- Datasets contain sparse and dense features; sparse means most of the values will be zeros whereas dense means most of the values will be non-zeros.
- Adagrad applies different learning rates for the sparse and dense data, for different iterations.

In case of Adagrad, the weight updation equation becomes

$$w_t = w_{t-1} - \eta_t' \frac{\partial L}{\partial w_{t-1}}$$

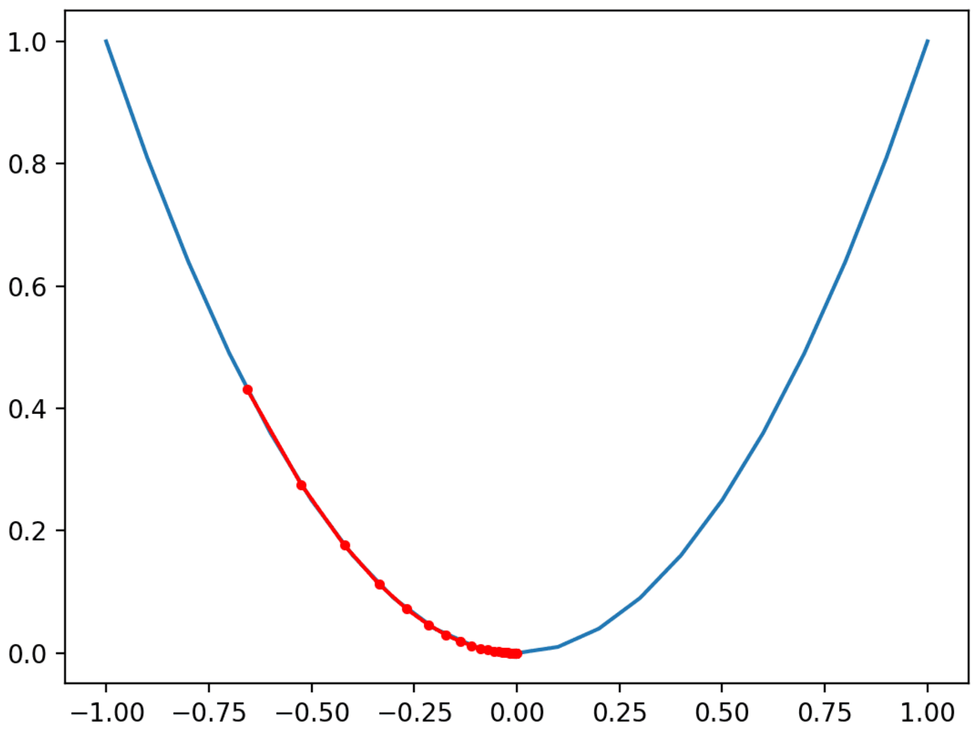
t represents each epoch (iteration)

$$\eta_t' = \frac{\eta}{\sqrt{\alpha_t + \varepsilon}}$$

ε is a small positive value to prevent division by zero.

$$\alpha_t = \sum_{i=1}^{t} \left(\frac{\partial L}{\partial w_i}\right)^2$$

α_t will be a larger value and this can cause the η_t' to decrease.
As a result the gradient of the weights will be decreasing slowly.



Limitation of Adagrad

- The radically diminishing learning rate is a problem.
- As the number of iterations increases for a more deeper network, the learning rates can sometimes decrease aggressively.
- Solution is AdaDelta and RMSprop.

AdaDelta and RMSprop

- Adadelta is an extension of Adagrad meant to reduce its aggressive, monotonically decreasing learning rate.
- Both AdaDelta and RMSprop works in a similar manner.
- Instead of accumulating all past squared gradients, Adadelta restricts the accumulation of past gradients to some fixed size.
- Adadelta implements the accumulation as an exponentially decaying average of the squared gradients.

$$\eta_t{}' = \frac{\eta}{\sqrt{w_{avg_t} + \varepsilon}}$$

$$w_{avg_t} = \beta w_{avg_{t-1}} + (1-\beta)\left(\frac{\partial L}{\partial w_t}\right)^2$$

$w_{avg_t}$is the weighted average (exponential weighted average/exponential decay average/moving average)

- Here, the α_t value is restricted by a factor $(1-\beta)$
- β is chosen as 0.95 in most cases.

Adam- Adaptive Moment Estimation

- Combines mini batch GD with momentum and RMSprop.
- Momentum enables smoothening and RMSprop enables to select a perfect learning rate.

$$v_{dw} = \beta_1 v_{dw} + (1 - \beta_1) \frac{\partial L}{\partial w}$$

$$v_{db} = \beta_1 v_{db} + (1 - \beta_1) \frac{\partial L}{\partial b}$$

Both these equations corresponds to momentum.

$$S_{dw} = \beta_2 S_{dw} + (1 - \beta_2) \left(\frac{\partial L}{\partial w}\right)^2$$

$$S_{db} = \beta_2 S_{dB} + (1 - \beta_2) \left(\frac{\partial L}{\partial b}\right)^2$$

These corresponds to RMSprop.

$$w_t = w_{t-1} - \eta \frac{v_{dw}}{\sqrt{S_{dw} + \varepsilon}}$$

$$b_t = b_{t-1} - \eta \frac{v_{db}}{\sqrt{S_{db} + \varepsilon}}$$

This is the weight updation formula for Adam optimizer.

Loss/cost/error function

- Loss function: loss for a single record.

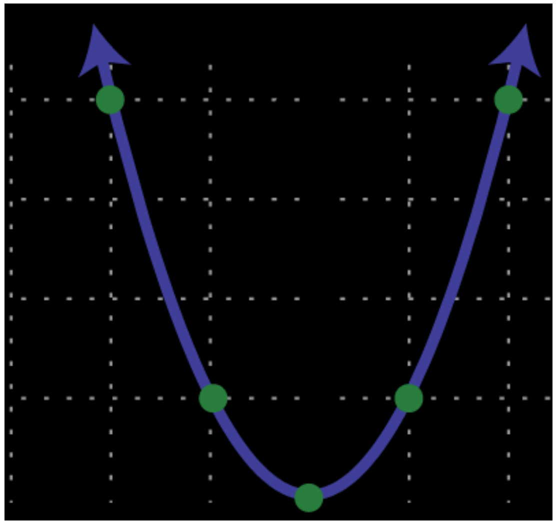
$$\frac{1}{2}(y - \hat{y})^2$$

- Cost function: loss for a batch of records.

$$\sum_{i=1}^{n} \frac{1}{n}(y - \hat{y})^2$$

- There are 2 types of problems: classification and regression.
- Based on the problem different types of loss functions are used.

Loss functions in regression problems



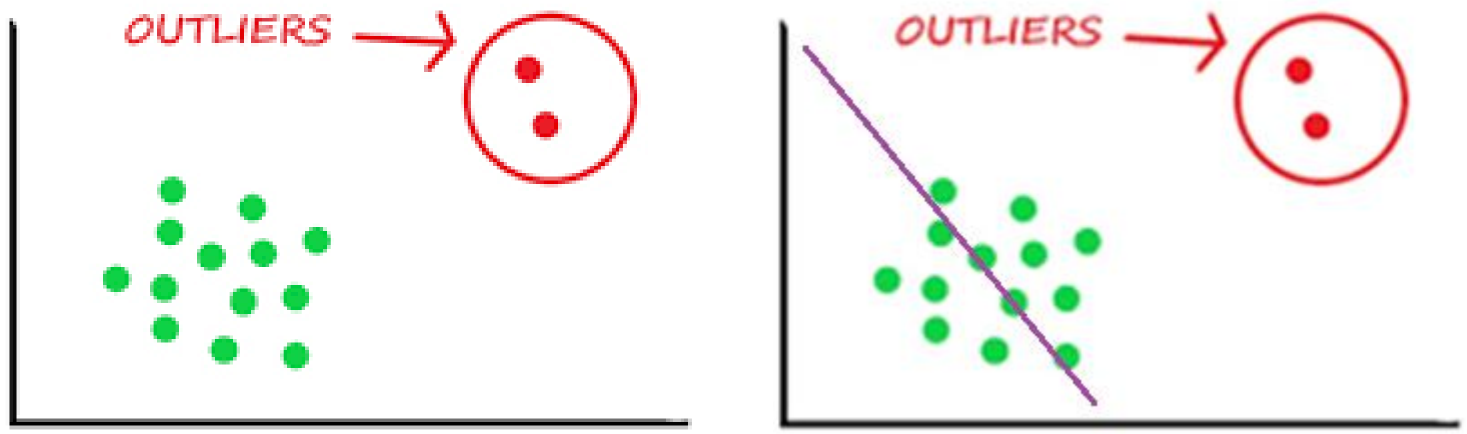
<u>1. Squared error loss</u>

$$L = \frac{1}{2}(y - \hat{y})^2$$

$$J = \sum_{i=1}^{t} \frac{1}{t}(y - \hat{y})^2$$

- J (cost function) is also called mean squared error (MSE).
- It is a quadratic equation and when it is plotted, we will get a plot with a global minima.
- No local minimas will be there.

- MSE losses penalizes the model for making larger errors by squaring them.
- The only disadvantage is that it is not robust to outliers.
- Outliers are points that are noticeably different from others.
- The outliers can cause the errors to be large and unnecessarily penalize the model.



2. Absolute error loss (Mean Absolute error loss-MAE)

$$L = \frac{1}{2}|y - \hat{y}|$$

$$J = \sum_{i=1}^{t} \frac{1}{t}|y - \hat{y}|$$

- It is robust to outliers as compared to MSE.
- But its computation is difficult.
- MAE may have local minima.

3. Huber loss

- Combines MSE and MAE.

$$Loss = \begin{cases} \frac{1}{2}(y-\hat{y})^2 \;; if\ |y-\hat{y}| \le \delta \\ \delta|y-\hat{y}| - \frac{1}{2}\delta^2 \;; otherwise \end{cases}$$

- δ is a hyper parameter defined by the network based on the outliers.
- Here both a quadratic and linear part is present.
- When the distance between the predicted and true value of a data point is less than δ, the model gets penalized as it is not treated as an outlier.
- Otherwise, the data point will be an outlier and no penalization occurs.

Loss functions in classification problems

<u>1. Cross entropy</u>

$$\mathrm{Loss} = -\mathrm{y} * \log(\hat{y}) - (1 - \mathrm{y}) * \log(1 - \hat{y})$$

$$Loss = \begin{cases} -\log(1 - \hat{y}), if\ y = 0 \\ -\log(\hat{y})\,, if\ y = 1 \end{cases}$$

- This is called **binary cross entropy** since it is used solely in binary classification problems.
- $\hat{y}$ is computed by using a **sigmoid activation function** in the output layer for a network handling binary classification problems.

2. Multiclass cross entropy loss

$$Loss(x_i, y_j) = -\sum_{j=1}^{c} y_{ij}\log(\hat{y}_{ij})$$

- This is **categorical cross entropy**.
- c is the number of categories (classes) in the problem.
- y_{ij} is the output vector for each of the input records represented using one-hot encoding.

$$y_{ij} = \begin{cases} 1, & if\ it\ belongs\ to\ class\ j \\ 0, & otherwise \end{cases}$$

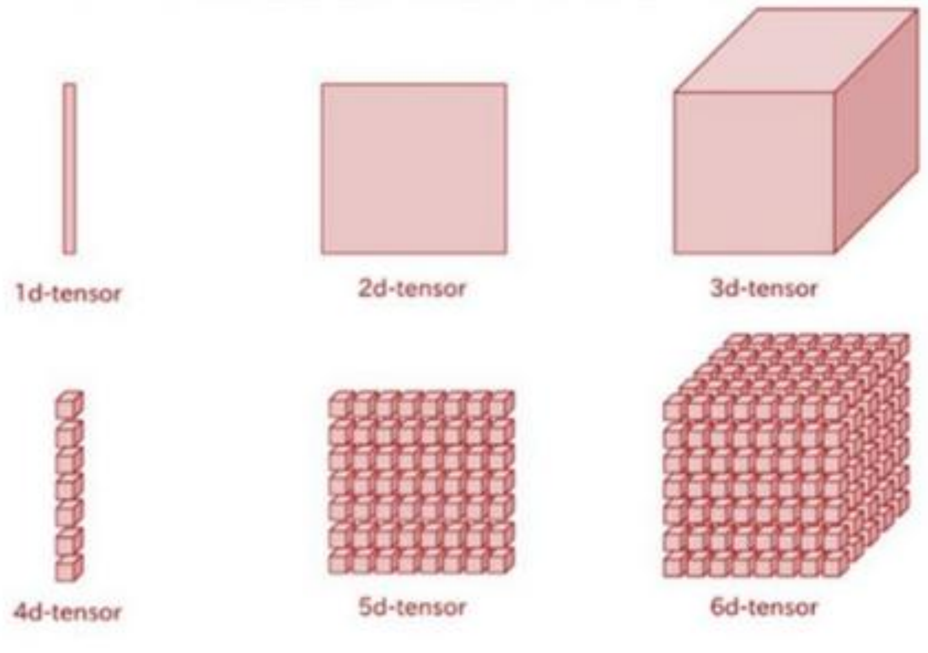
- $\hat{y}$ is computed by using a **softmax activation function** in the output layer for a multiclass classification problem.

$$\sigma(z) = \frac{e^{zi}}{\sum_{j=1}^{k} e^{zj}}$$

- This outputs a probability of an particular output belonging to each class.
- In **sparse categorical cross entropy**, the maximum probability index will be returned during the prediction when softmax is applied.

Tensorflow

- An open source library developed by Google for deep learning applications.
- Also supports traditional machine learning.
- Originally developed for large numerical computations.
- Tensors are data structures used by machine learning systems.
- A tensor is a container for numerical data.



- Three primary attributes of a tensor:

Rank : Number of axes of the tensor

Shape : Number of dimensions along each axis.

Data type : Type of data contained in it.

Keras

- A powerful open source python library for developing and evaluating deep learning models.
- It wraps numerical computation library like Tensorflow.
- In Tensorflow version >2.0, Keras is integrated with it.
- For versions < 2.0, Keras and Tensorflow need to be installed separately.
- More user friendly.
- It provides many pre-trained models like VGG16, VGG19, Xception etc.