

**DATA SCIENCE**

ASSIGNMENT - 1

NAME : NATARAJ.C

REGISTER NO : 121012012756

DEGREE : B.TECH

BRANCH : CSE –3rd YEAR

SEMESTER : VI

SPECIALIZATION : DATA SCIENCE

SUBJECT : DEEP LEARNING

SUB.CODE : XCSHD04

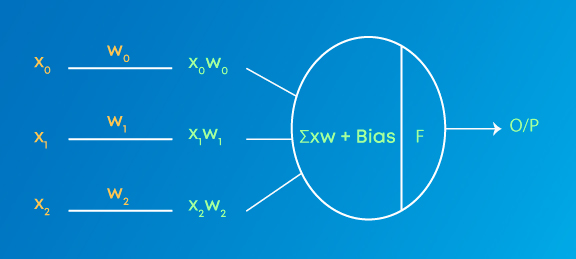
1.Types of Neural Networks

## ****An Introduction to Artificial Neural Network****

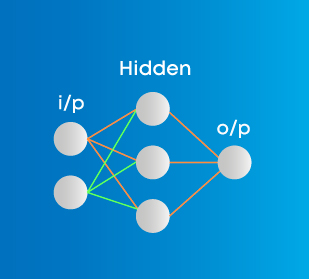
Neural networks represent [deep learning](https://www.mygreatlearning.com/blog/what-is-deep-learning/) using [artificial intelligence](https://www.mygreatlearning.com/artificial-intelligence/courses). Certain application scenarios are too heavy or out of scope for traditional machine learning algorithms to handle. As they are commonly known, Neural Network pitches in such scenarios and fills the gap.

Artificial neural networks are inspired by the biological neurons within the human body which activate under certain circumstances resulting in a related action performed by the body in response. Artificial neural nets consist of various layers of interconnected artificial neurons powered by activation functions that help in switching them ON/OFF. Like traditional [machine algorithms](https://www.mygreatlearning.com/blog/clustering-algorithms-in-machine-learning/), here too, there are certain values that neural nets learn in the training phase.

Briefly, each neuron receives a multiplied version of inputs and random weights, which is then added with a static bias value (unique to each neuron layer); this is then passed to an appropriate activation function which decides the final value to be given out of the neuron. There are various activation functions available as per the nature of input values. Once the output is generated from the final neural net layer, loss function (input vs output)is calculated, and backpropagation is performed where the weights are adjusted to make the loss minimum. Finding optimal values of weights is what the overall operation focuses around.



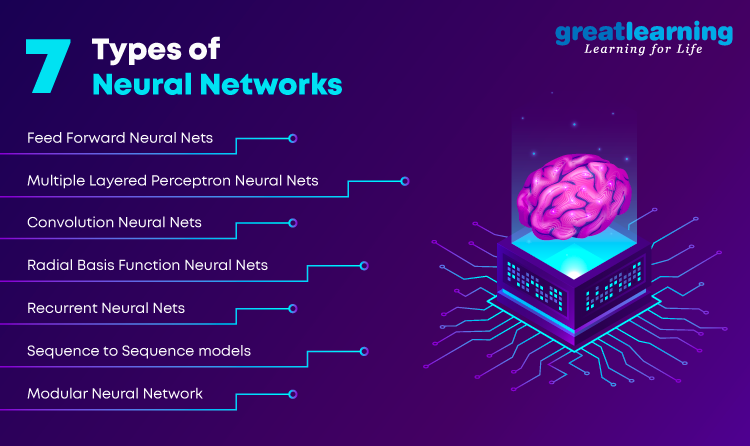
**Weights**are numeric values that are multiplied by inputs. In backpropagation, they are modified to reduce the loss. In simple words, weights are machine learned values from Neural Networks. They self-adjust depending on the difference between predicted outputs vs training inputs.  
**Activation Function**is a mathematical formula that helps the neuron to switch ON/OFF.



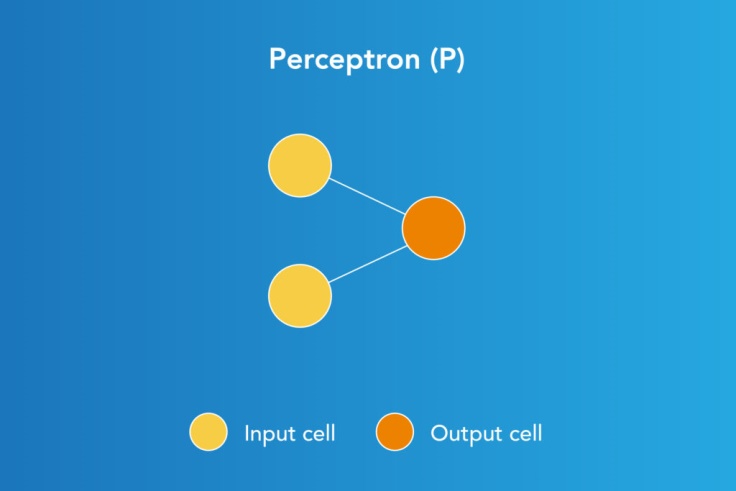
* **Input layer**represents dimensions of the input vector.
* **Hidden layer**represents the intermediary nodes that divide the input space into regions with (soft) boundaries. It takes in a set of weighted input and produces output through an activation function.
* **Output layer**represents the output of the neural network.

The nine types of neural networks are:

* Perceptron
* Feed Forward Neural Network
* Multilayer Perceptron
* Convolutional Neural Network
* Radial Basis Functional Neural Network
* Recurrent Neural Network
* LSTM – Long Short-Term Memory
* Sequence to Sequence Models
* Modular Neural Network



## ****A. Perceptron****

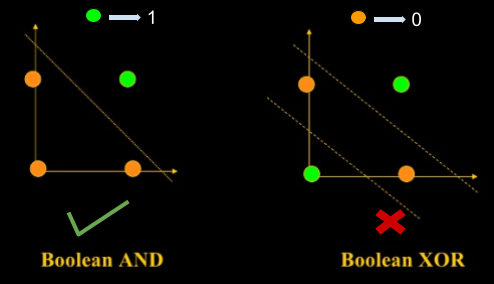


Perceptron

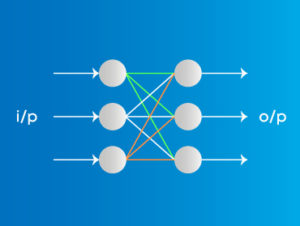
Perceptron model, proposed by Minsky-Papert is one of the simplest and oldest models of Neuron. It is the smallest unit of neural network that does certain computations to detect features or business intelligence in the input data. It accepts weighted inputs, and apply the activation function to obtain the output as the final result. Perceptron is also known as TLU(threshold logic unit)Perceptron is a supervised learning algorithm that classifies the data into two categories, thus it is a binary classifier. A perceptron separates the input space into two categories by a hyperplane represented by the following equation:

**Advantages of Perceptron**  
Perceptrons can implement Logic Gates like AND, OR, or NAND.

**Disadvantages of Perceptron**  
Perceptrons can only learn linearly separable problems such as boolean AND problem. For non-linear problems such as the boolean XOR problem, it does not work.



## ****B. Feed Forward Neural Networks****



### ****Applications on Feed Forward Neural Networks:****

* **Simple classification (where traditional Machine-learning based classification algorithms have limitations**)
* **Face recognition [Simple straight forward image processing]**
* Computer vision [Where target classes are difficult to classify]
* Speech Recognition

The simplest form of neural networks where input data travels in one direction only, passing through artificial neural nodes and exiting through output nodes. Where hidden layers may or may not be present, input and output layers are present there. Based on this, they can be further classified as a single-layered or multi-layered feed-forward neural network.

Number of layers depends on the complexity of the function. It has uni-directional forward propagation but no backward propagation. Weights are static here. An activation function is fed by inputs which are multiplied by weights. To do so, classifying activation function or step activation function is used. For example: The neuron is activated if it is above threshold (usually 0) and the neuron produces 1 as an output. The neuron is not activated if it is below threshold (usually 0) which is considered as -1. They are fairly simple to maintain and are equipped with to deal with data which contains a lot of noise.

### ****Advantages of Feed Forward Neural Networks****

1. Less complex, easy to design & maintain
2. Fast and speedy [One-way propagation]
3. Highly responsive to noisy data

### ****Disadvantages of Feed Forward Neural Networks:****

1. Cannot be used for deep learning [due to absence of dense layers and back propagation]

## C. Multilayer Perceptron

### types of neural networks ****Applications on Multi-Layer Perceptron****

* **Speech Recognition**
* **Machine Translation**
* **Complex Classification**

An entry point towards complex neural nets where input data travels through various layers of artificial neurons. Every single node is connected to all neurons in the next layer which makes it a fully connected neural network. Input and output layers are present having multiple hidden Layers i.e. at least three or more layers in total. It has a bi-directional propagation i.e. forward propagation and backward propagation.

Inputs are multiplied with weights and fed to the activation function and in backpropagation, they are modified to reduce the loss. In simple words, weights are machine learnt values from Neural Networks. They self-adjust depending on the difference between predicted outputs vs training inputs. Nonlinear activation functions are used followed by softmax as an output layer activation function.

### ****Advantages on Multi-Layer Perceptron****

1. Used for deep learning [due to the presence of dense fully connected layers and back propagation]

### ****Disadvantages on Multi-Layer Perceptron:****

1. Comparatively complex to design and maintain

Comparatively slow (depends on number of hidden layers)

## ****D. Convolutional Neural Network****

### types of neural networks ****Applications on Convolution Neural Network****

* **Image processing**
* [**Computer Vision**](https://www.mygreatlearning.com/blog/deep-learning-computer-vision/)
* **Speech Recognition**
* **Machine translation**

[Convolution neural](https://www.mygreatlearning.com/academy/learn-for-free/courses/convolutional-neural-networks) network contains a three-dimensional arrangement of neurons instead of the standard two-dimensional array. The first layer is called a convolutional layer. Each neuron in the convolutional layer only processes the information from a small part of the visual field. Input features are taken in batch-wise like a filter. The network understands the images in parts and can compute these operations multiple times to complete the full image processing. Processing involves conversion of the image from RGB or HSI scale to grey-scale. Furthering the changes in the pixel value will help to detect the edges and images can be classified into different categories.

Propagation is uni-directional where CNN contains one or more convolutional layers followed by pooling and bidirectional where the output of convolution layer goes to a fully connected neural network for classifying the images as shown in the above diagram. Filters are used to extract certain parts of the image. In MLP the inputs are multiplied with weights and fed to the activation function. Convolution uses RELU and MLP uses nonlinear activation function followed by softmax. Convolution neural networks show very effective results in image and video recognition, semantic parsing and paraphrase detection.

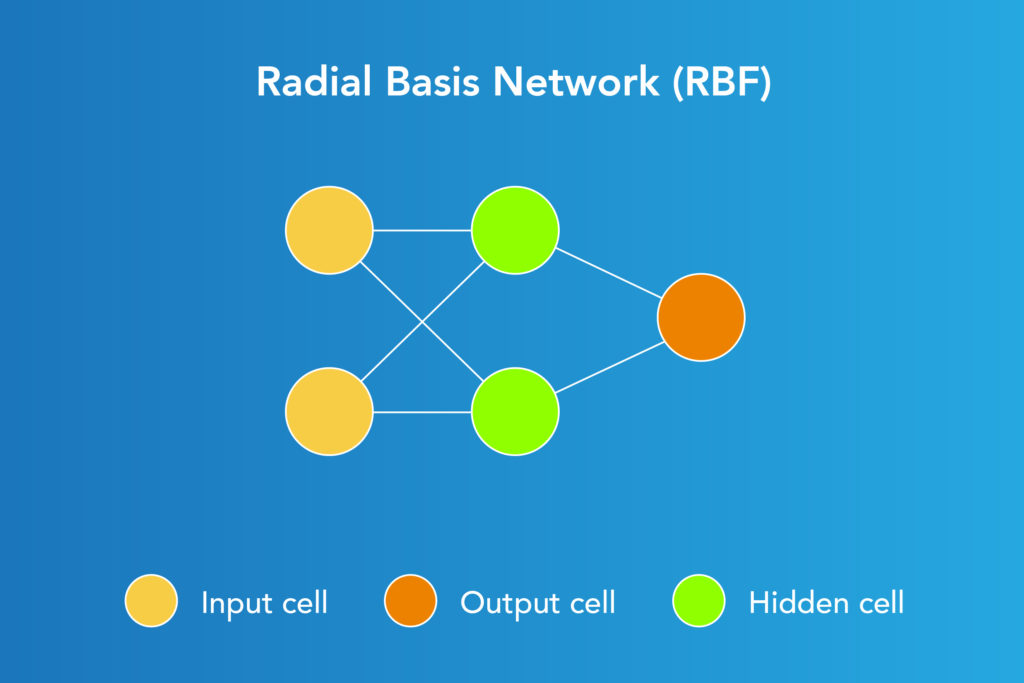
### ****Advantages of Convolution Neural Network:****

1. Used for deep learning with few parameters
2. Less parameters to learn as compared to fully connected layer

### ****Disadvantages of Convolution Neural Network:****

* Comparatively complex to design and maintain
* Comparatively slow [depends on the number of hidden layers]

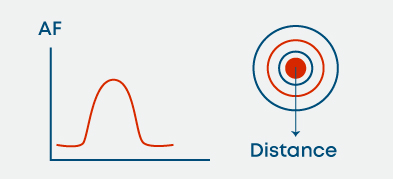
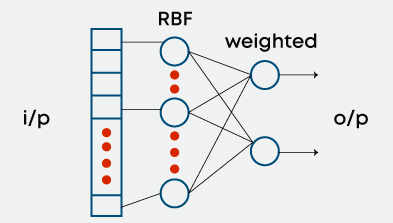
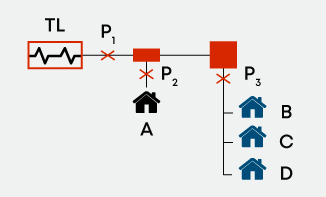
## ****E. Radial Basis Function Neural Networks****



Radial Basis Function Network consists of an input vector followed by a layer of RBF neurons and an output layer with one node per category. Classification is performed by measuring the input’s similarity to data points from the training set where each neuron stores a prototype. This will be one of the examples from the training set.

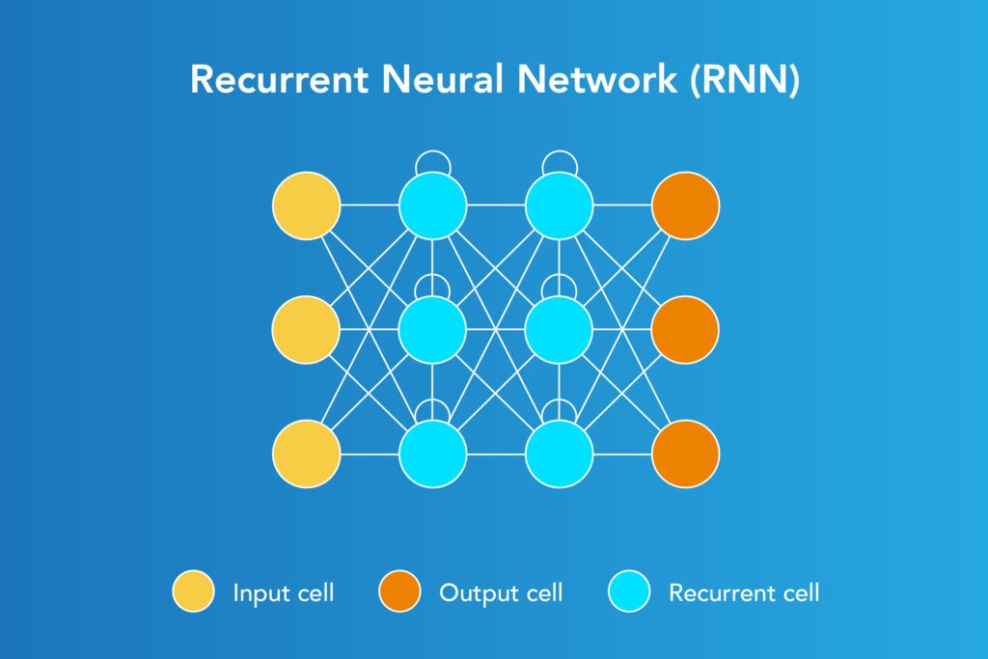
When a new input vector [the n-dimensional vector that you are trying to classify] needs to be classified, each neuron calculates the Euclidean distance between the input and its prototype. For example, if we have two classes i.e. class A and Class B, then the new input to be classified is more close to class A prototypes than the class B prototypes. Hence, it could be tagged or classified as class A.

Each RBF neuron compares the input vector to its prototype and outputs a value ranging which is a measure of similarity from 0 to 1. As the input equals to the prototype, the output of that RBF neuron will be 1 and with the distance grows between the input and prototype the response falls off exponentially towards 0. The curve generated out of neuron’s response tends towards a typical bell curve. The output layer consists of a set of neurons [one per category].

Application: Power Restoration***a. Powercut P1*** needs to be restored first***b. Powercut P3*** needs to be restored next, as it impacts more houses***c. Powercut P2*** should be fixed last as it impacts only one house

## F. Recurrent Neural Networks



### ****Applications of Recurrent Neural Networks****

* **Text processing like auto suggest, grammar checks, etc.**
* **Text to speech processing**
* **Image tagger**
* **Sentiment Analysis**
* **Translation**  
  Designed to save the output of a layer, Recurrent Neural Network is fed back to the input to help in predicting the outcome of the layer. The first layer is typically a feed forward neural network followed by recurrent neural network layer where some information it had in the previous time-step is remembered by a memory function. Forward propagation is implemented in this case. It stores information required for it’s future use. If the prediction is wrong, the learning rate is employed to make small changes. Hence, making it gradually increase towards making the right prediction during the backpropagation.

### ****Advantages of Recurrent Neural Networks****

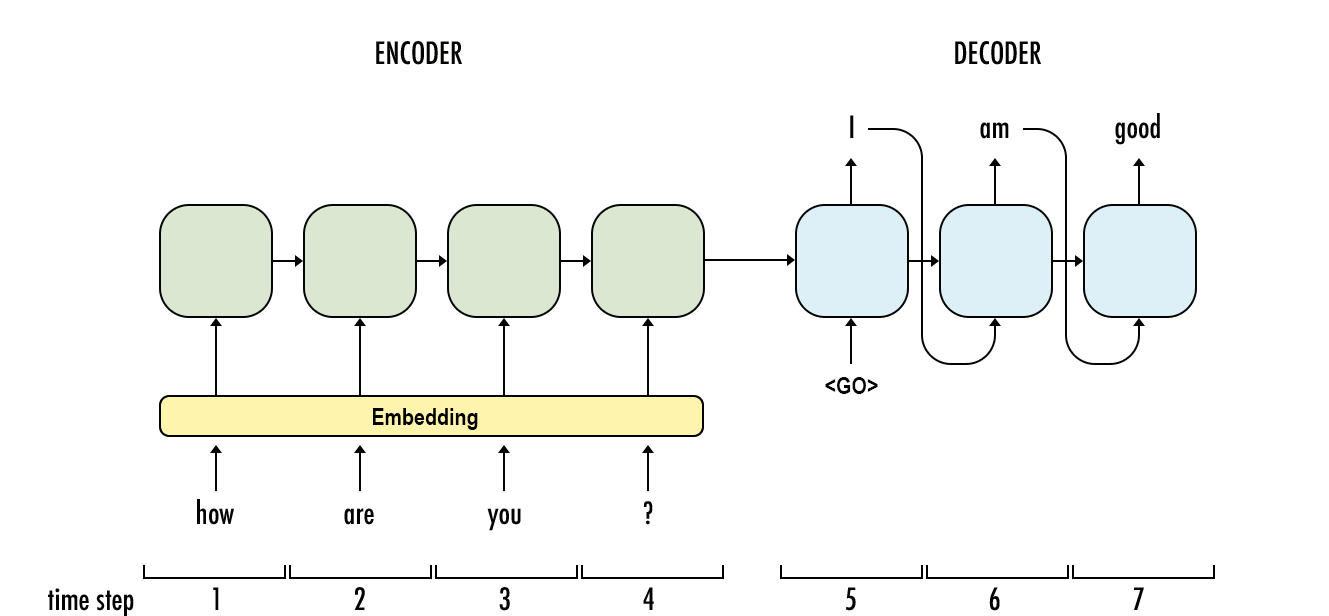
1. Model sequential data where each sample can be assumed to be dependent on historical ones is one of the advantage.
2. Used with convolution layers to extend the pixel effectiveness.

### ****Disadvantages of Recurrent Neural Networks****

1. Gradient vanishing and exploding problems
2. Training recurrent neural nets could be a difficult task
3. Difficult to process long sequential data using ReLU as an activation function.

## ****G. Sequence to sequence models****

### 



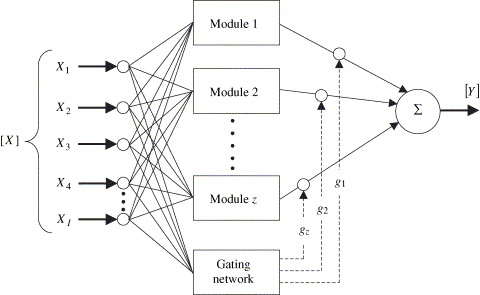
A sequence to sequence model consists of two Recurrent Neural Networks. Here, there exists an encoder that processes the input and a decoder that processes the output. The encoder and decoder work simultaneously – either using the same parameter or different ones. This model, on contrary to the actual RNN, is particularly applicable in those cases where the length of the input data is equal to the length of the output data. While they possess similar benefits and limitations of the RNN, these models are usually applied mainly in [chatbots](https://www.mygreatlearning.com/blog/basics-of-building-an-artificial-intelligence-chatbot/" \t "_blank), machine translations, and question answering systems.

## ****H. Modular Neural Network****

### ****Applications of Modular Neural Network****

1. **Stock market prediction systems**
2. **Adaptive MNN for character recognitions**
3. **Compression of high level input data**

A modular neural network has a number of different networks that function independently and perform sub-tasks. The different networks do not really interact with or signal each other during the computation process. They work independently towards achieving the output.



As a result, a large and complex computational process are done significantly faster by breaking it down into independent components. The computation speed increases because the networks are not interacting with or even connected to each other.

### ****Advantages of Modular Neural Network****

1. Efficient
2. Independent training
3. Robustness

### ****Disadvantages of Modular Neural Network****

1. Moving target Problems

2.Activation Functions in Neural Networks

## ****Introduction :****

Activation functions are mathematical equations that determine the output of a neural network model. Activation functions also have a major effect on the [neural network’s](https://www.mygreatlearning.com/blog/types-of-neural-networks/) ability to converge and the convergence speed, or in some cases, activation functions might prevent neural networks from converging in the first place. Activation function also helps to normalize the output of any input in the range between 1 to -1 or 0 to 1. Activation function must be efficient and it should reduce the computation time because the neural network sometimes trained on millions of data points.

Let’s consider the simple neural network model without any hidden layers.

Here is the output-

**Y =  ∑ (weights\*input + bias)**

and it can range from -infinity to +infinity. So it is necessary to bound the output to get the desired prediction or generalized results.

**Y = Activation function(∑ (weights\*input + bias))**

So the activation function is an important part of an artificial neural network. They decide whether a neuron should be activated or not and it is a non-linear transformation that can be done on the input before sending it to the next layer of neurons or finalizing the output.

**Properties of activation functions**

1. Non Linearity
2. Continuously differentiable
3. Range
4. Monotonic
5. Approximates identity near the origin

## ****Types of Activation Functions****

The activation function can be broadly classified into 2 categories.

1. Binary Step Function
2. Linear Activation Function

## ****Binary Step Function****

A binary step function is generally used in the Perceptron linear classifier. It thresholds the input values to 1 and 0, if they are greater or less than zero, respectively.

The step function is mainly used in binary classification problems and works well for linearly severable pr. It can’t classify the multi-class problems.

## ****Linear Activation Function****

The equation for Linear activation function is:

**f(x) = a.x**

When a = 1 then f(x) = x and this is a special case known as identity.

#### ****Properties:****

1. Range is -infinity to +infinity
2. Provides a convex error surface so optimisation can be achieved faster
3. df(x)/dx = a which is constant. So cannot be optimised with gradient descent

#### ****Limitations:****

1. Since the derivative is constant, the gradient has no relation with input
2. Back propagation is constant as the change is delta x

## ****Non-Linear Activation Functions****

Modern neural network models use non-linear activation functions. They allow the model to create complex mappings between the network’s inputs and outputs, such as images, video, audio, and data sets that are non-linear or have high dimensionality.

Majorly there are 3 types of Non-Linear Activation functions.

1. Sigmoid Activation Functions
2. Rectified Linear Units or ReLU
3. Complex Nonlinear Activation Functions

## ****Sigmoid Activation Functions****

Sigmoid functions are bounded, differentiable, real functions that are defined for all real input values, and have a non-negative derivative at each point.

#### ****Sigmoid or Logistic Activation Function****

The sigmoid function is a logistic function and the output is ranging between 0 and 1.

The output of the activation function is always going to be in range (0,1) compared to (-inf, inf) of linear function. It is non-linear, continuously differentiable, monotonic, and has a fixed output range. But it is not zero centred.

## ****Hyperbolic Tangent****

The function produces outputs in scale of [-1, 1] and it is a continuous function. In other words, function produces output for every x value.

**Y = tanh(x)  
tanh(x) = (ex – e-x) / (ex + e-x)**

## ****Inverse Hyperbolic Tangent (arctanh)****

It is similar to sigmoid and tanh but the output ranges from [-pi/2,pi/2]

## ****Softmax****

The softmax function is sometimes called the soft argmax function, or multi-class logistic regression. This is because the softmax is a generalization of logistic regression that can be used for multi-class classification, and its formula is very similar to the sigmoid function which is used for logistic regression. The softmax function can be used in a classifier only when the classes are mutually exclusive.

## ****Gudermannian****

The Gudermannian function relates circular functions and hyperbolic functions without explicitly using complex numbers.

The below is the mathematical equation for Gudermannian function:

## ****GELU (Gaussian Error Linear Units)****

An activation function used in the most recent Transformers such as Google’s BERT and OpenAI’s GPT-2. This activation function takes the form of this equation:

**GELU(x)=0.5x(1+tanh(√2/π(x+0.044715×3)))**

So it’s just a combination of some functions (e.g. hyperbolic tangent tanh) and approximated numbers.

It has a negative coefficient, which shifts to a positive coefficient. So when x is greater than zero, the output will be x, except from when x=0 to x=1, where it slightly leans to a smaller y-value.

## ****Problems with Sigmoid Activation Functions****

#### ****1. Vanishing Gradients Problem****

The main problem with deep neural networks is that the gradient diminishes dramatically as it is propagated backward through the network. The error may be so small by the time it reaches layers close to the input of the model that it may have very little effect. As such, this problem is referred to as the “vanishing gradients” problem.

A small gradient means that the weights and biases of the initial layers will not be updated effectively with each training session. Since these initial layers are often crucial to recognizing the core elements of the input data, it can lead to overall inaccuracy of the whole network.

#### ****2. Exploding Gradients****

Exploding gradients are a problem where large error gradients accumulate and result in very large updates to neural network model weights during training. These large updates in turn results in an unstable network. At an extreme, the values of weights can become so large as to overflow and result in NaN values.

## ****Rectified Linear Units or ReLU****

The sigmoid and hyperbolic tangent activation functions cannot be used in networks with many layers due to the vanishing gradient problem. The rectified linear activation function overcomes the vanishing gradient problem, allowing models to learn faster and perform better. The rectified linear activation is the default activation when developing multilayer Perceptron and convolutional neural networks.

#### ****Rectified Linear Units(ReLU)****

ReLU is the most commonly used activation function in neural networks and The mathematical equation for ReLU is:

**ReLU(x) = max(0,x)**

So if the input is negative, the output of ReLU is 0 and for positive values, it is x.

Though it looks like a linear function, it’s not. ReLU has a derivative function and allows for backpropagation.

There is one problem with ReLU. Let’s suppose most of the input values are negative or 0, the ReLU produces the output as 0 and the neural network can’t perform the back propagation. This is called the Dying ReLU problem. Also, ReLU is an unbounded function which means there is no maximum value.

Pros:

1. Less time and space complexity
2. Avoids the vanishing gradient problem.

Cons:

1. Introduces the dead relu problem.
2. Does not avoid the exploding gradient problem.

#### ****Leaky ReLU****

The dying ReLU problem is likely to occur when:

1. Learning rate is too high
2. There is a large negative bias

Leaky ReLU is the most common and effective method to solve a dying ReLU problem. It adds a slight slope in the negative range to prevent the dying ReLU issue.

Again this doesn’t solve the exploding gradient problem.

#### ****Parametric ReLU****

PReLU is actually not so different from Leaky ReLU.

So for negative values of x, the output of PReLU is alpha times x and for positive values, it is x.

Parametric ReLU is the most common and effective method to solve a dying ReLU problem but again it doesn’t solve exploding gradient problem.

## ****Exponential Linear Unit (ELU)****

ELU speeds up the learning in neural networks and leads to higher classification accuracies, and it solves the vanishing gradient problem. ELUs have improved learning characteristics compared to the other activation functions. ELUs have negative values that allow them to push mean unit activations closer to zero like batch normalization but with lower computational complexity.

The mathematical expression for ELU is:

ELU is designed to combine the good parts of ReLU and leaky ReLU and it doesn’t have the dying ReLU problem. it saturates for large negative values, allowing them to be essentially inactive.

## ****Scaled Exponential Linear Unit (SELU)****

SELU incorporates normalization based on the central limit theorem. SELU is a monotonically increasing function, where it has an approximately constant negative output for large negative input. SELU’s are mostly commonly used in Self Normalizing Networks (SNN).

The output of a SELU is normalized, which could be called internal normalization, hence the fact that all the outputs are with a mean of zero and standard deviation of one. The main advantage of SELU is that the Vanishing and exploding gradient problem is impossible and since it is a new activation function, it requires more testing before usage.

## ****Softplus or SmoothReLU****

The derivative of the softplus function is the logistic function.

The mathematical expression is:

And the derivative of softplus is:

**Swish function**

The Swish function was developed by Google, and it has superior performance with the same level of computational efficiency as the ReLU function. ReLU still plays an important role in deep learning studies even for today. But experiments show that this new activation function overperforms ReLU for deeper networks

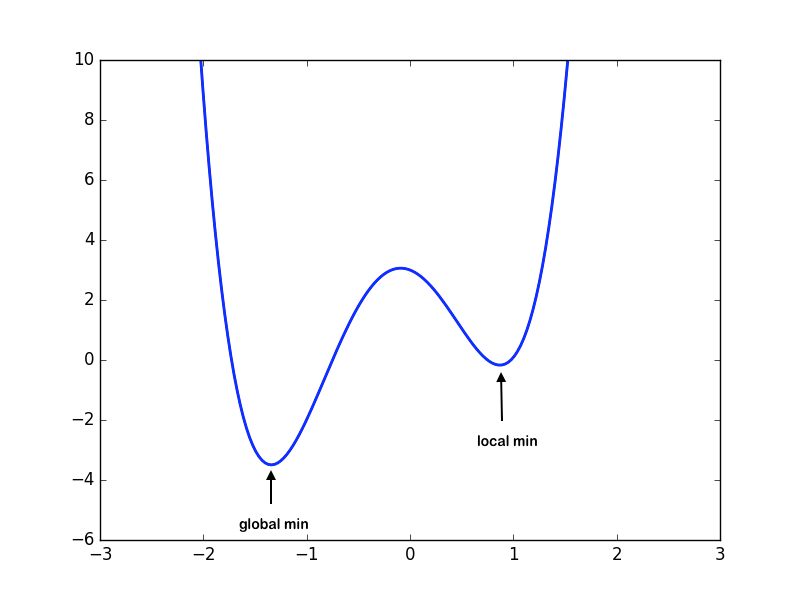
The mathematical expression for Swish Function is:

The modified version of swish function is:

Here, β is a parameter that must be tuned. If β gets closer to ∞, then the function looks like ReLU. Authors of the Swish function proposed to assign β as 1 for reinforcement learning tasks.

# ****Optimizers in Deep Learning****

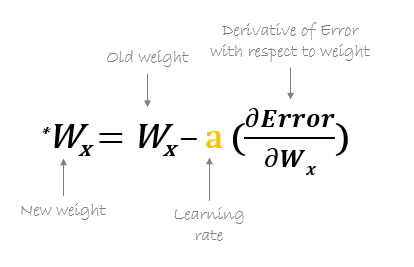
.The optimizer comes into the picture. It tries to lower the loss function by updating the model parameters in response to the output of the loss function. Thereby helping to reach the Global Minima with the lowest loss and most accurate output.



This shows the global and local minima with the loss on Y-axis and weights on X-axis

**Updating Weights:**

Before going deep into various types of optimizers, it is very essential to know that the most important function of the optimizer is to update the weights of the learning algorithm to reach the least cost function. Here is the formula used by all the optimizers for updating the weights with a certain value of the [learning rate](https://en.wikipedia.org/wiki/Learning_rate).



The formula for updating the weights

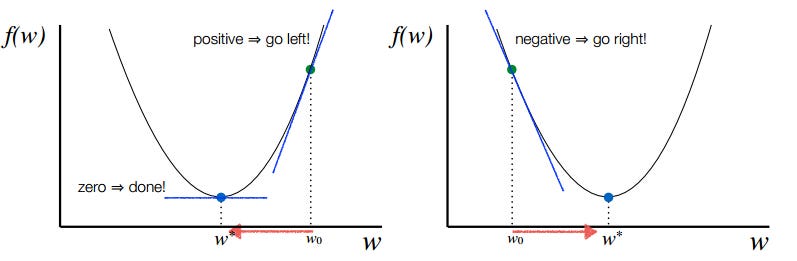
***TYPES OF OPTIMIZERS :***

1. Gradient Descent
2. Stochastic Gradient Descent
3. Adagrad
4. Adadelta
5. RMSprop
6. Adam

G**radient Descent :**

This is one of the oldest and the most common optimizer used in neural networks, best for the cases where the data is arranged in a way that it possesses a convex optimization problem. It will try to find the least cost function value by updating the weights of your learning algorithm and will come up with the best-suited parameter values corresponding to the Global Minima.

This is done by moving down the hill with a negative slope, increasing the older weight, and positive slope reducing the older weight.

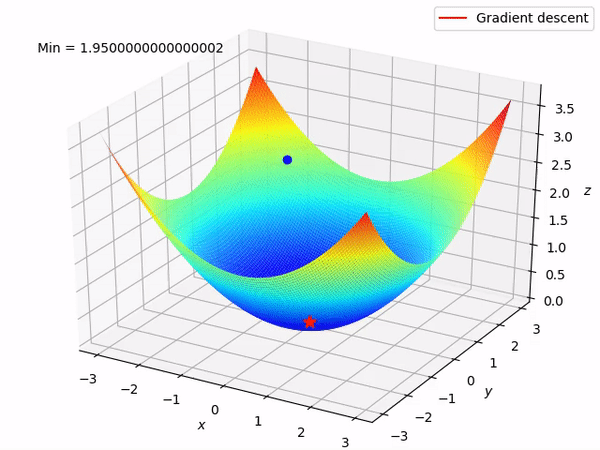


Deciding the direction of descent

Although there are challenges while using this optimizer, suppose the data is arranged in a way that it possesses a non-convex optimization problem then it can possibly land on the Local Minima instead of the Global Minima thereby providing the parameter values with a higher cost function.

There is also a saddle point problem. This is a point where the gradient is zero but is not an optimal point. This is still an active area of research.

For certain cases, problems like Vanishing Gradient or Exploding Gradient may also occur due to incorrect parameter initialization. These problems occur due to a very small or very large gradient, which makes it difficult for the algorithm to converge.



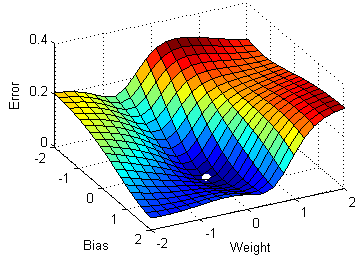
Finding the global minima for a convex optimization problem using Gradient Descent

S**tochastic Gradient Descent :**

This is another variant of the Gradient Descent optimizer with an additional capability of working with the data with a non-convex optimization problem. The problem with such data is that the cost function results to rest at the local minima which are not suitable for your learning algorithm.

Rather than going for batch processing, this optimizer focuses on performing one update at a time. It is therefore usually much faster, also the cost function minimizes after each iteration (EPOCH). It performs frequent updates with a high variance that causes the objective function(cost function) to fluctuate heavily. Due to which it makes the gradient to jump to a potential Global Minima.

However, if we choose a learning rate that is too small, it may lead to very slow convergence, while a larger learning rate can make it difficult to converge and cause the cost function to fluctuate around the minimum or even to diverge away from the global minima.



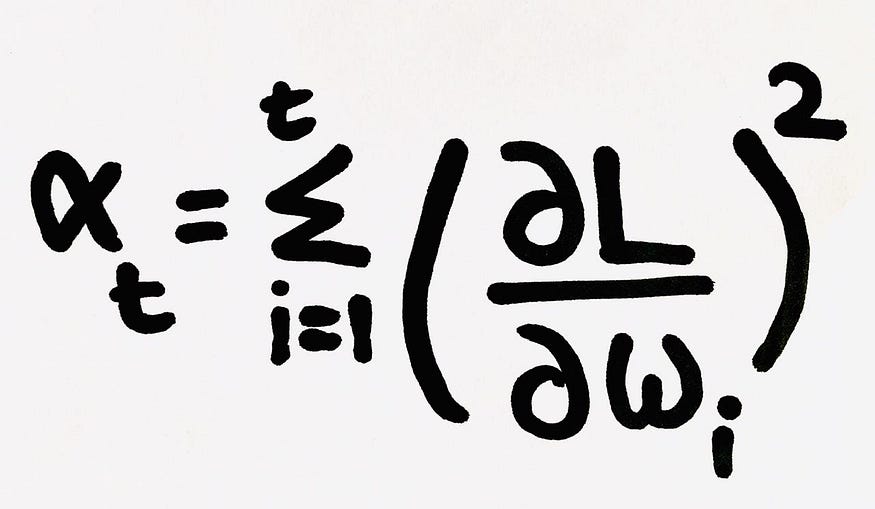
Converging at the global minima using SGD for non-convex data

A**dagrad :**

This is the Adaptive Gradient optimization algorithm, where the learning rate plays an important role in determining the updated parameter values. Unlike Stochastic Gradient descent, this optimizer uses a different learning rate for each iteration(EPOCH) rather than using the same learning rate for determining all the parameters.

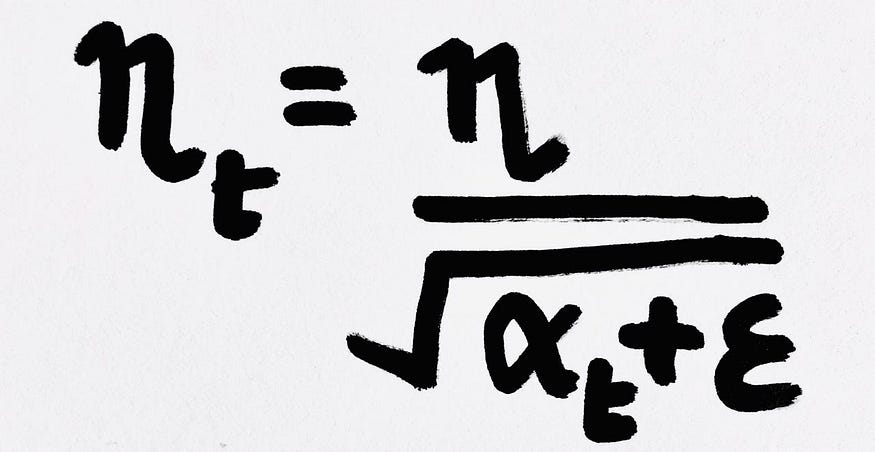
Thus it performs smaller updates(lower learning rates) for the weights corresponding to the high-frequency features and bigger updates(higher learning rates) for the weights corresponding to the low-frequency features, which in turn helps in better performance with higher accuracy. Adagrad is well-suited for dealing with sparse data.

So at each iteration, first the alpha at time t will be calculated and as the iterations increase the value of t increases, and thus alpha t will start increasing.



The alpha term increases because of squaring the derivative of loss with respect to the weight at time step t

Now the learning rate is calculated at each time step. Given by,

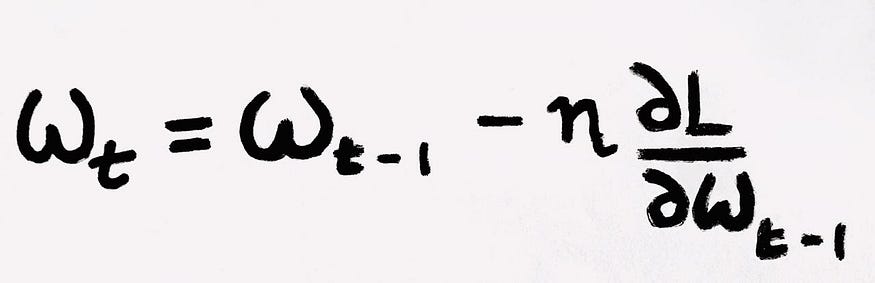


Here epsilon is a small positive number added to alpha to avoid the error if at any instance alpha becomes zero

Therefore as the alpha at time step t increases, it makes the learning rate to decrease gradually.



As the learning rate changes for each iteration, the formula for updating the weight also changes. Given by,



This determines the updated weight for each iteration

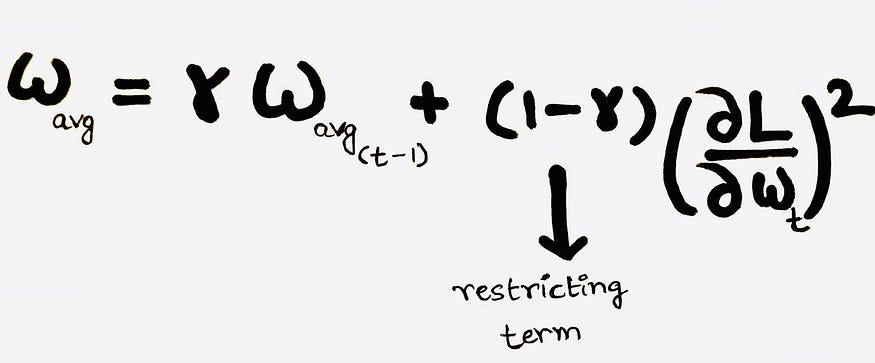
However, there is a disadvantage of getting into the problem of [Vanishing Gradient](https://en.wikipedia.org/wiki/Vanishing_gradient_problem) because after a lot of iterations the alpha value becomes very large making the learning rate very small leading to no change between the new and the old weight. This in turn causes the learning rate to shrink and eventually become very small, where the algorithm is not able to acquire any further knowledge.

A**dadelta :**

This is an extension of the Adaptive Gradient optimizer, taking care of its aggressive nature of reducing the learning rate infinitesimally. Here instead of using the previous squared gradients, the sum of gradients is defined as a reducing weighted average of all past squared gradients(weighted averages) this restricts the learning rate to reduce to a very small value.

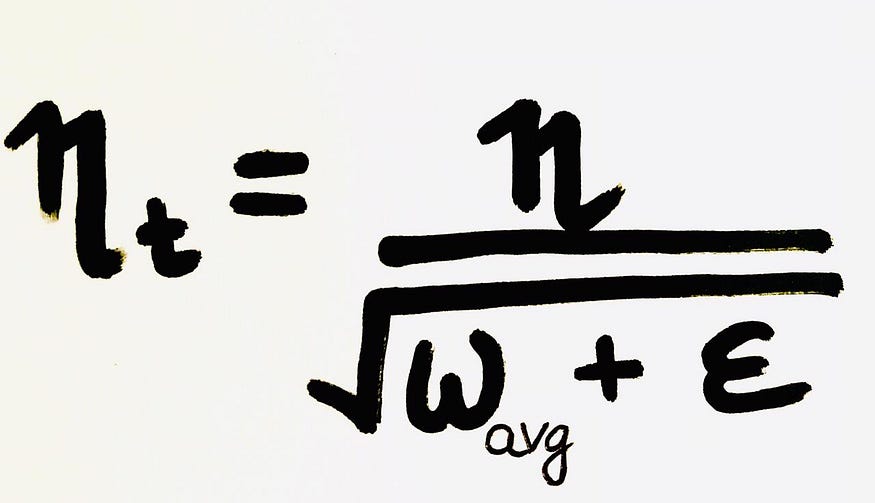
The formula for the new weight remains the same as in Adagrad. However, there are some changes in determining the learning rate at time step t for each iteration.

At each iteration, first the weighted average is calculated. Where we have the restricting term(gamma = 0.95) which helps in avoiding the problem of [Vanishing Gradient](https://en.wikipedia.org/wiki/Vanishing_gradient_problem).



Formula for calculating the Weighted average

After which the Learning rate is calculated using the formula,



Formula for calculating the learning rate at time step t

Thus because of the restricting term, the weighted average will increase at a slower rate, making the learning rate to reduce slowly to reach the global minima.

R**MSprop :**

Both the optimizing algorithms, RMSprop(Root Mean Square Propagation) and Adadelta were developed around the same time, for the same purpose to resolve Adagrad’s problem of destructive learning rates. However, both use the same method which utilizes an Exponential Weighted Average to determine the learning rate at time t for each iteration.

RMSprop is an adaptive learning rate method proposed by Geoffrey Hinton, which appropriately divides the learning rate by an exponentially weighted average of squared gradients. It is suggested to set gamma at 0.95, as it has been showing good results for most of the cases.

A**dam :**

This is the Adaptive Moment Estimation algorithm which also works on the method of computing adaptive learning rates for each parameter at every iteration. It uses a combination of [Gradient Descent with Momentum](https://engmrk.com/gradient-descent-with-momentum/) and RMSprop to determine the parameter values.

When introducing the algorithm, there was a list of attractive benefits of using Adam on non-convex optimization problems which made it the most commonly used optimizer.

It comes with several advantages combining the benefits of both Gradient with Momentum and RMSProp like low memory requirements, appropriate for non-stationary objectives, works best with large data and parameters with efficient computation. This works using the same methodology of adaptive learning rate in addition to storing an exponential weighted average of the past squared derivative of loss with respect to the weight at time t-1.

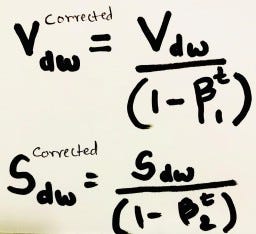
It comes with several parameters, which are β1, β2, and ε (epsilon). Where β1 and β2 are the initial restricting parameters for Momentum and RMSprop respectively. Here, β1 corresponds to the first moment and β2 corresponds to the second moment.

For updating the weights with an adaptive learning rate at iteration t, first, we need to calculate the first and second moment given by the following formulae,

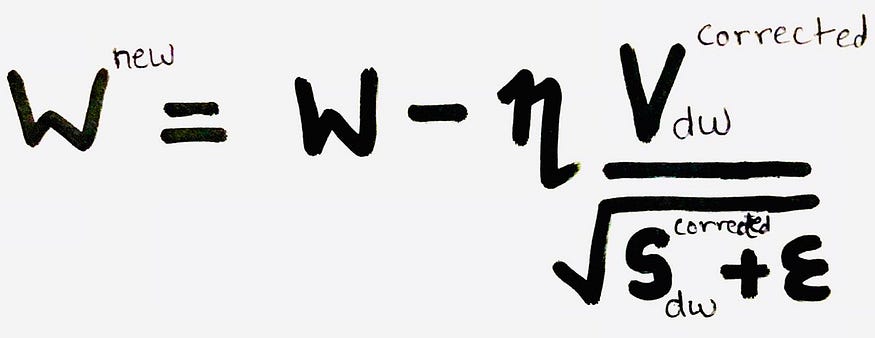
VdW = β1 x VdW + (1-β1) x dW — — GD with Momentum (1st)

SdW = β2 x SdW + (1-β2) x dW² — — RMSprop (2nd)

The corrected VdW and SdW is given by,



Therefore the new weight will be updated using the formula,



The initial value of n is to be tuned for better results.

Adam is relatively easy to configure where the default configuration parameters do well on most problems. It is proposed to have default values of β1=0.9 ,β2 = 0.999 and ε =10E-8. Studies show that Adam works well in practice, in comparison to other adaptive learning algorithms.

References :

* 1. <https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/>
  2. <https://www.mygreatlearning.com/blog/types-of-neural-networks/>
  3. <https://www.upgrad.com/blog/types-of-activation-function-in-neural-networks/>