**Unit 4 SML**

**Support Vector Machines and Neural Networks**

Support vector machines and neural networks are on the higher side of computational complexity and require significant resources for calculations but do provide significantly better results compared with other machine learning methods.

A support vector machine (SVM) can be imagined as a surface that maximizes the boundaries between various types of points of data that is represent in multidimensional space (hyperplane), which creates the most homogeneous points in each subregion.

SVMs can be used on any type of data but have special extra advantages for data types with very high dimensions relative to the observations.

**Example:**

* Text classification, in which language has the very dimensions of word vectors
* For the quality control of DNA sequencing by labeling chromatograms correctly

**Support vector machines working principles**

SVMs are mainly classified into three types based on their working principles:

* Maximum margin classifiers
* Support vector classifiers
* Support vector machines

**Maximum margin classifier**

People usually generalize support vector machines with maximum margin classifiers. It is feasible to draw infinite hyperplanes to classify the same set of data upon, but the question, is which one to consider as an ideal hyperplane?

Answer: The maximum margin classifier provides the hyperplane with the maximum margin of separation width.

Hyperplanes:

* In n-dimensional space, a hyperplane is a flat affine subspace of dimension n-1.
* This means, in 2-dimensional space, the hyperplane is a straight line which separates the 2-dimensional space into two halves.
* The hyperplane is defined by the following equation:

Diagram

Description automatically generated

* Points which lay on the hyperplane have to follow the above equation.
* However, there are regions above and below as well. This means observations could fall in either of the regions, also called the region of classes:

Text

Description automatically generated

* The mathematical representation of the maximum margin classifier is as follows, which is an optimization problem:

Diagram, text

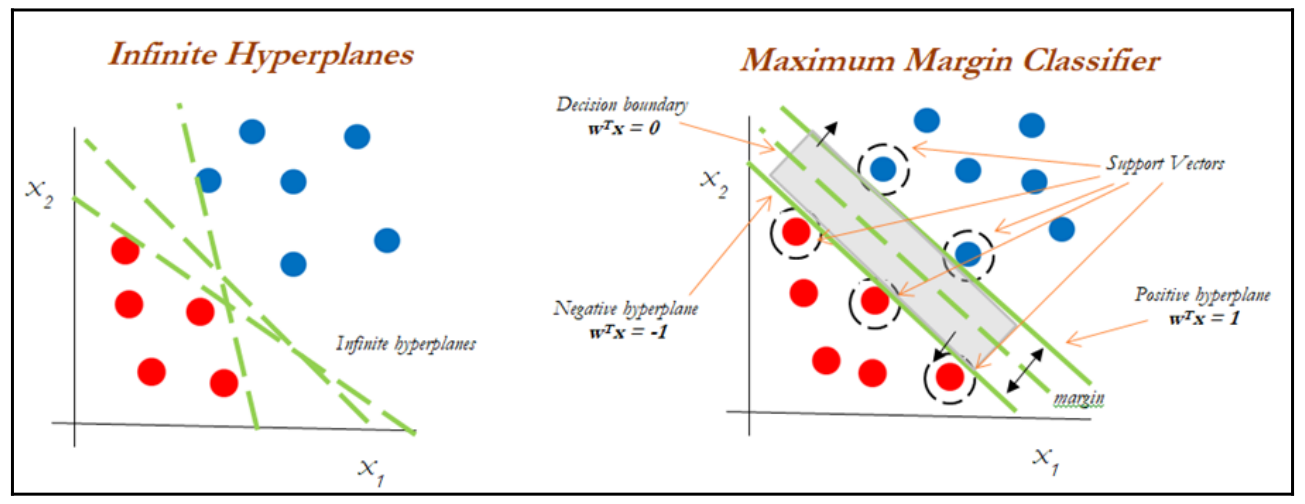
Description automatically generated

* Constraint 2 ensures that observations will be on the correct side of the hyperplane by taking the product of coefficients with x variables and finally, with a class variable indicator.

**Note:** In non-separable cases, the maximum margin classifier will not have a separating hyperplane, which is also known as no feasible solution. This issue will be solved with support vector classifiers.

In the following diagram, we can draw infinite separate hyperplanes to separate the two classes (blue and red). However, the maximum margin classifier attempts to fit the widest slab (maximize the margin between positive and negative hyperplanes) between two classes and the observations touching both the positive and negative hyperplanes called support vectors:

**Note:** Classifier performance purely depends on the support vectors and any changes to observation values which are not support vectors (or observations that do not touch hyperplanes) do not impact any change in the performance of the Maximum Margin Classifier, as only extreme points are considered in the algorithm.



**Support vector classifier**

Support vector classifiers are an extended version of maximum margin classifiers, in which some violations are tolerated for non-separable cases in order to create the best fit, even with slight errors within the threshold limit.

In real-life scenarios, we hardly find any data with purely separable classes; most classes have a few or more observations in overlapping classes.

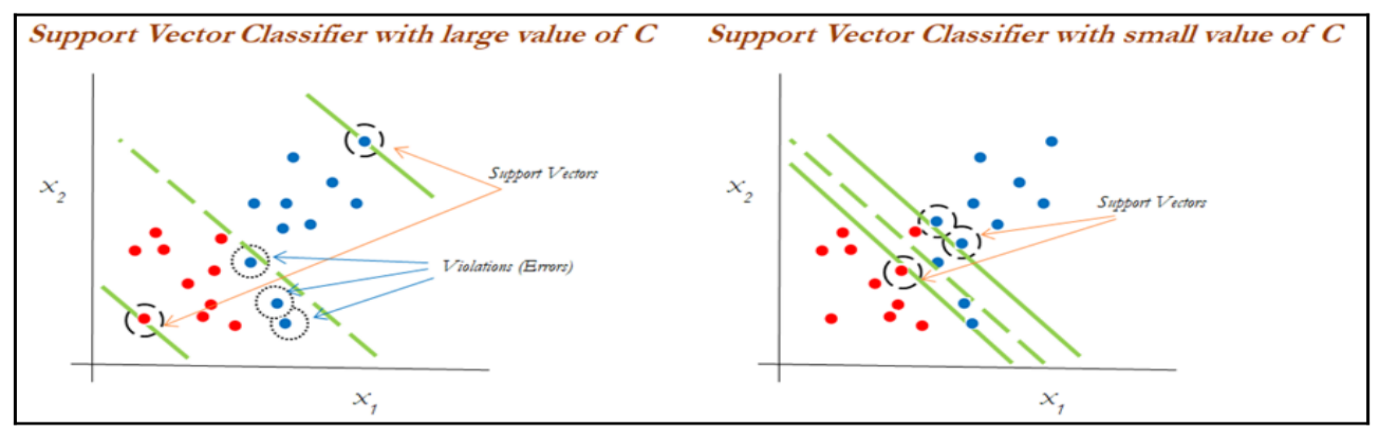
* The mathematical representation of the support vector classifier is as follows, a slight correction to the constraints to accommodate error terms:



In constraint 3, the C value is a non-negative tuning parameter to either accommodate more or fewer overall errors in the model.

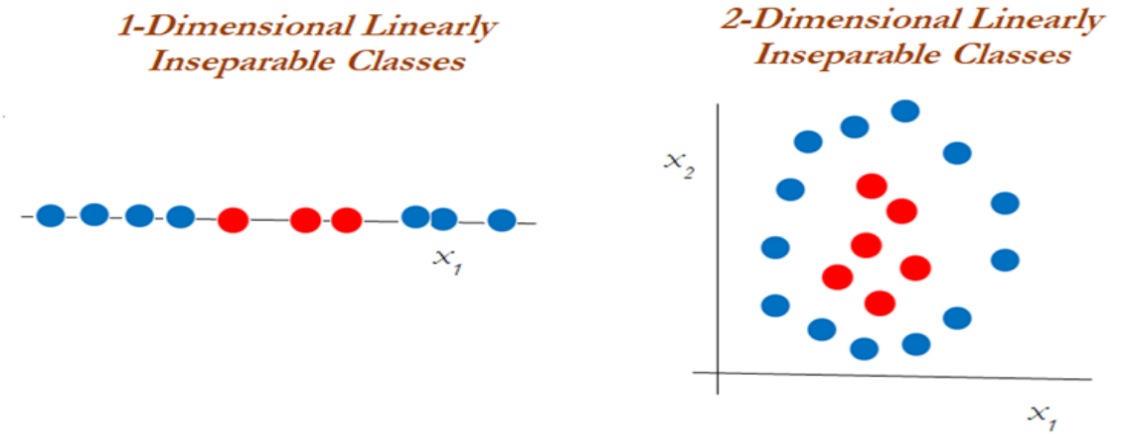
* a high value of C will lead to a more robust model, whereas a lower value creates the flexible model due to less violation of error terms.

The impact of changing the C value on margins is shown in the following diagram; with the high value of C, the model would be more tolerating and also have space for violations (errors) in the left diagram, whereas with the lower value of C, no scope for accepting violations leads to a reduction in margin width.



**Support vector machines:**

SVMs are used when the decision boundary is non-linear and would not be separable with support vector classifiers whatever the cost function is! The following diagram explains the non-linearly separable cases for both 1-dimension and 2-dimensions:

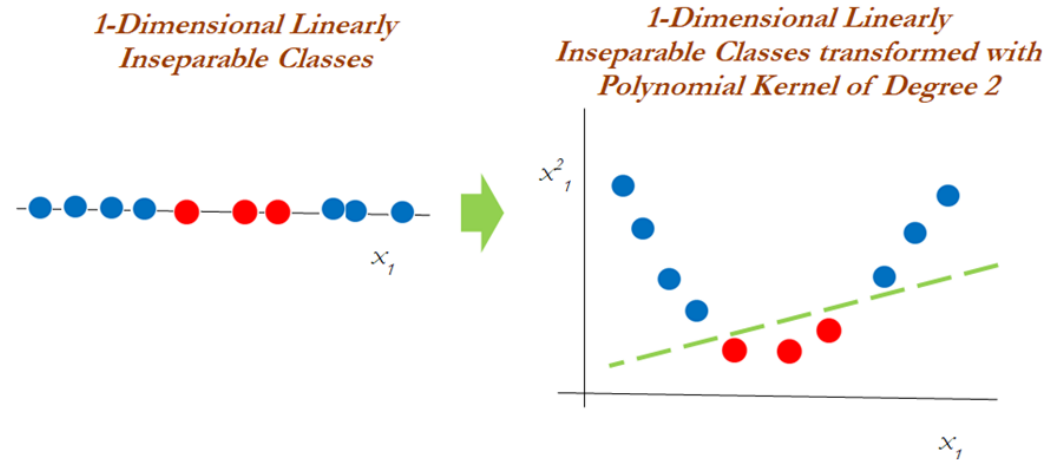


We cannot classify using support vector classifiers whatever the cost value is. We need to use another way of handling the data (kernel trick) using the kernel function to work with non-linearly separable data.

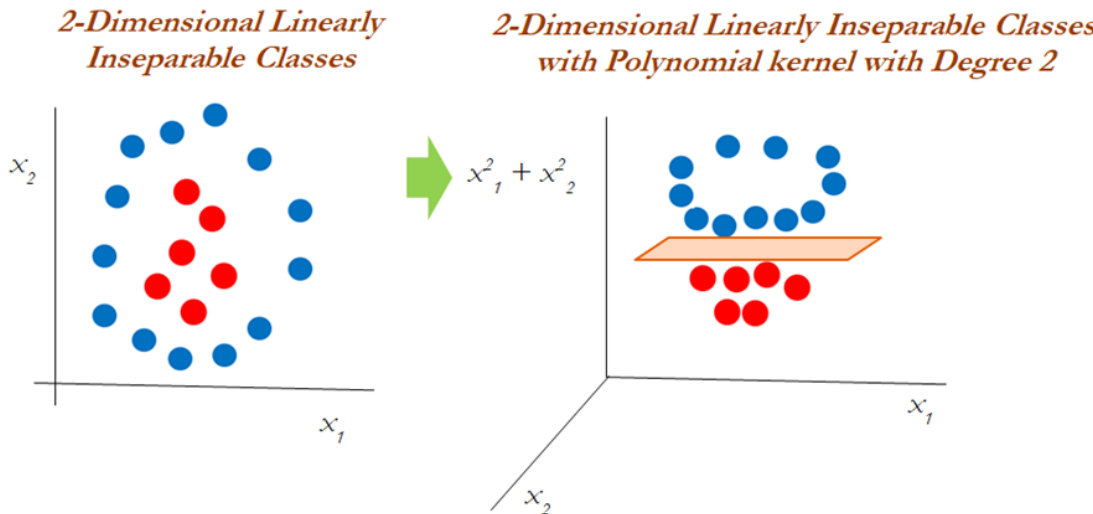
In the following diagram, a polynomial kernel with degree 2 has been applied in transforming the data from 1-dimensional to 2-dimensional data. So, the data becomes linearly separable in higher dimensions.

In the left diagram, different classes (red and blue) are plotted on X1 only, whereas after applying degree 2, we now have 2- dimensions, X1 and X12(the original and a new dimension).

The degree of the polynomial kernel is a tuning parameter; We need to tune them with various values to check where higher accuracies are possible with the model:



Whereas, in the 2-dimensional case, the kernel trick is applied as below with the polynomial kernel with degree 2. The observations have been classified successfully using a linear plane after projecting the data into higher dimensions:



**Kernel functions**

Kernel functions are the functions that, given the original feature vectors, return the same value as the dot product of its corresponding mapped feature vectors.

Kernel functions do not explicitly map the feature vectors to a higher-dimensional space or calculate the dot product of the mapped vectors.

Kernels produce the same value through a different series of operations that can often be computed more efficiently.

The main reason for using kernel functions is to eliminate the computational requirement to derive the higher-dimensional vector space from the given basic vector space, so that observations be separated linearly in higher dimensions.

Why someone needs to like this is, derived vector space will grow exponentially with the increase in dimensions, and it will become almost too difficult to continue computation, even when you have a variable size of 30 or so.

The following example shows how the size of the variables grows.

**Example**: When we have two variables such as x and y, with a polynomial degree kernel, it

needs to compute x2, y2, and xy dimensions in addition.

If we have three variables x, y, and z, then we need to calculate the x2, y2, z2, xy, yz, xz, and xyz vector spaces.

The increase of one more dimension creates so many combinations. Hence, care needs to be taken to reduce its computational complexity; this is where kernels do wonders. Kernels are defined more formally in the following equation:



**Polynomial Kernel:** These are popularly used, especially with degree 2. The inventor of support vector machines, Vladimir N Vapnik, developed using a degree 2 kernel for classifying handwritten digits. Polynomial kernels are given by the following equation:



**Radial Basis Function (RBF) / Gaussian Kernel:** RBF kernels are a good first choice for problems requiring nonlinear models.

A decision boundary that is a hyperplane in the mapped feature space is similar to a decision boundary that is a hypersphere in the original space.

The feature space produced by the Gaussian kernel can have an infinite number of dimensions, a feat that would be impossible otherwise. RBF kernels are represented by the following equation:



This is simplified as the following equation:

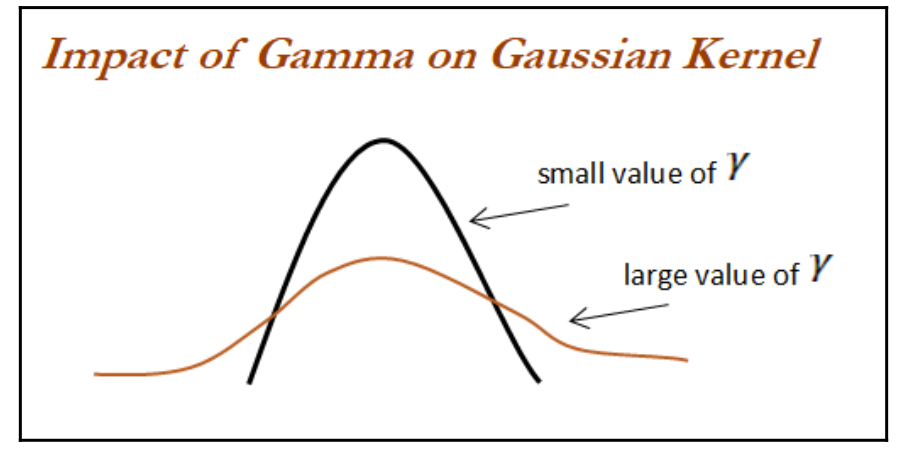


It is advisable to scale the features when using support vector machines, but it is very

important when using the RBF kernel.

When the value of the gamma value is small, it gives you a pointed bump in the higher dimensions. A small gamma will give low bias and high variance solutions.

A larger value gives you a softer, broader bump. A high gamma will give high bias and low variance solutions and so control the fit of the model using RBF kernels:



**Example code – Refer Textbook Page no. 227**

**Artificial neural networks – ANN**

**ANNs model the relationship between a set of input signals and output signals using a model derived from a replica of the biological brain, which responds to stimuli from its sensory inputs.**

The human brain consists of about 90 billion neurons, with around 1 trillion connections between them.

ANN methods try to model problems using interconnected artificial neurons (or nodes) to solve machine learning problems.

**Biological neurons work:** ANNs have taken inspiration from the biological neuron.

Incoming signals are received by the cell's dendrites through a biochemical process that allows the impulses to be weighted according to their relative importance.

As the cell body begins to accumulate the incoming signals, a threshold is reached, at which the cell fires and the output signal is then transmitted via an electrochemical process down the axon.

At the axon terminal, an electric signal is again processed as a chemical signal to be passed to its neighboring neurons, which will be dendrites to some other neuron.

A similar working principle is loosely used in building an artificial neural network, in

which each neuron has a set of inputs, each of which is given a specific weight.

The neuron computes a function on these weighted inputs.

A linear neuron takes a linear combination of weighted input and applies an activation function (sigmoid, tanh, relu, and so on) on the aggregated sum.

The network feeds the weighted sum of the input into the logistic function (in case of sigmoid function). The logistic function returns a value between 0 and 1 based on the set

threshold; for example, here we set the threshold as 0.7. Any accumulated signal greater

than 0.7 gives the signal of 1 and vice versa; any accumulated signal less than 0.7 returns the

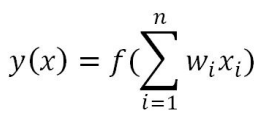
value of 0:

Diagram

Description automatically generated

**Note:** Neural network models are being considered as universal approximators, which can solve any type of problems with the fine-tuned architecture. In fact, deep learning is a branch of machine learning, where every problem is being modelled with artificial neural networks.

**Formula:** A typical artificial neuron with n input dendrites The w weights allow each of the n inputs of x to contribute a greater or lesser amount to the sum of input signals. The accumulated value is passed to the activation function, f(x), and the resulting signal, y(x), is the output axon:



**Parameters required for choosing for building neural networks:**

**Activation function:** Choosing an activation function plays a major role in aggregating signals into the output signal to be propagated to the other neurons of the network.

**Network architecture or topology:** the number of layers required and the number of neurons in each layer.

More layers and neurons will create a highly non-linear decision boundary, whereas if we reduce the architecture, the model will be less flexible and more robust.

**Training optimization algorithm:** The selection of an optimization algorithm plays a critical role as well, in order to converge quickly and accurately to the best optimal solutions.

**Applications of Neural Networks:** Now, neural networks (a branch of deep learning) has gained huge attention in terms of its application in AI, in terms of speech, text, vision, and many other areas. Some of the famous applications are the following:

* Images and videos: To identify an object in an image or to classify whether it is a dog or a cat
* Text processing (NLP): Deep-learning-based chatbot and so on
* Speech: Recognize speech
* Structured data processing: Building highly powerful models to obtain a non-linear decision boundary

**Activation functions**

Activation functions are the mechanisms by which an artificial neuron processes information and passes it throughout the network.

The activation function takes a single number and performs a certain fixed mathematical functional mapping on it.

**Different types of activation functions:**

* Sigmoid
* Tanh
* Relu
* Linear

**Sigmoid function:** Sigmoid has the mathematical form σ(x) = 1 / (1+e−x). It takes a real-valued number and squashes it into a range between 0 and 1. Sigmoid makes calculating derivatives easy and is easy to interpret.

**Tanh function:** Tanh squashes the real-valued number into the range [-1, 1]. The output is

zero-centered. In practice, tanh non-linearity is always preferred to sigmoid non-linearity.

Also, it can be proved that tanh is scaled sigmoid neuron tanh(x) = 2σ (2x) − 1.

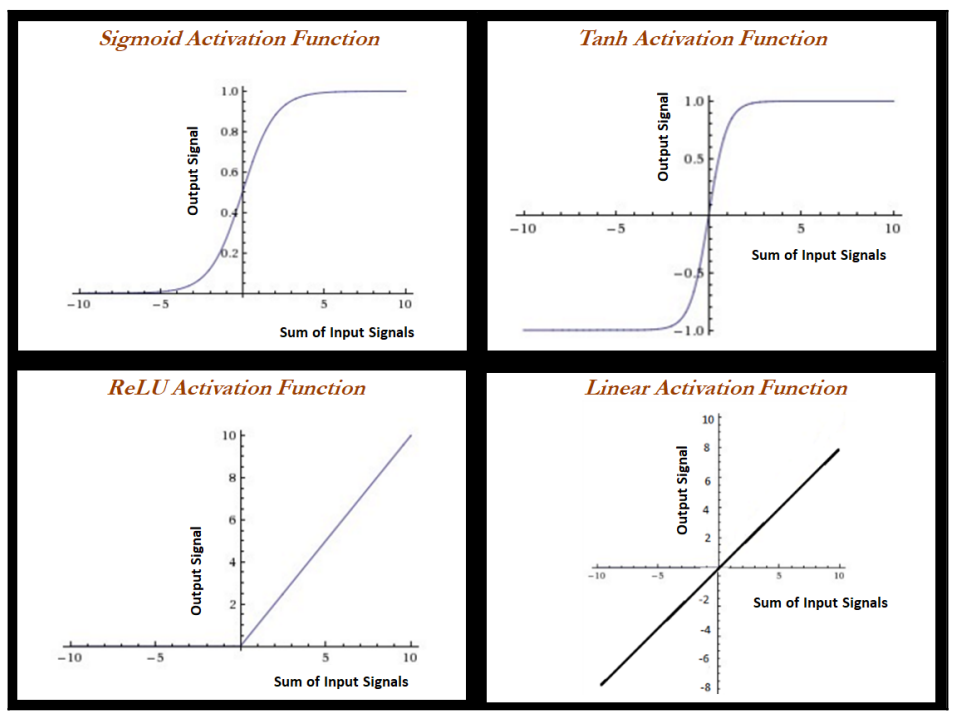
**Rectified Linear Unit (ReLU) function:** ReLU has become very popular in the last few years. It computes the function f(x) = max (0, x). Activation is simply thresholds at zero.

**Linear function:** The linear activation function is used in linear regression problems, where

it always provides a derivative as 1 due to the function used being f(x) = x.

Relu is now being used in place of Sigmoid or Tanh due to its better convergence property.

The linear activation function is used in linear regression cases, whereas all the other activation functions are used for classification problems:



**Forward propagation and backpropagation**

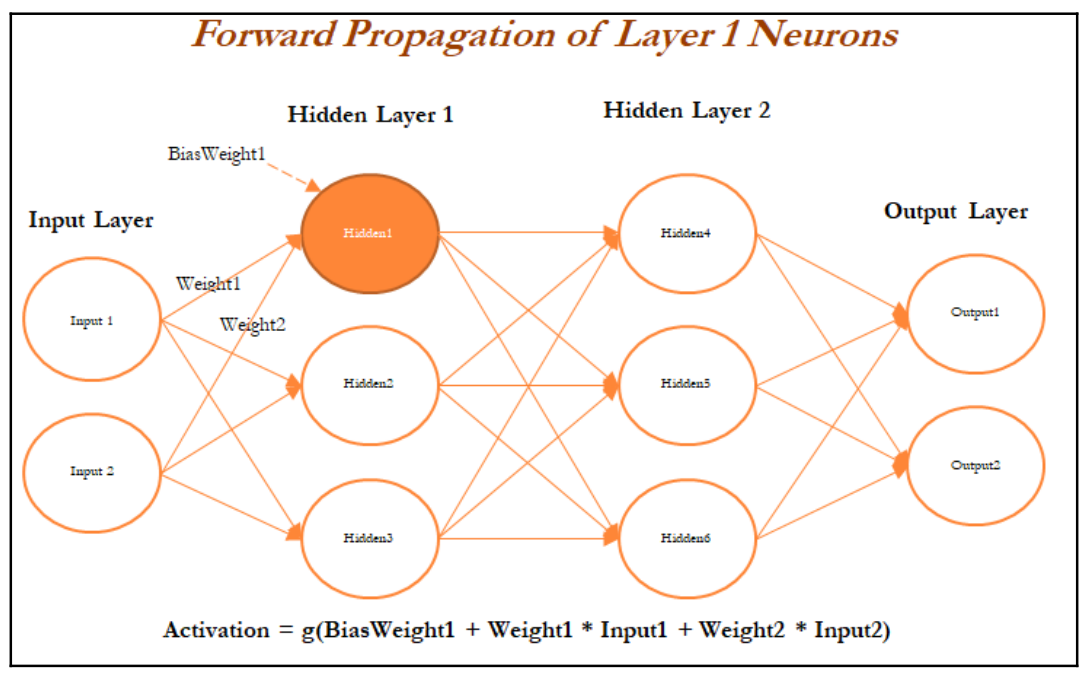
* Forward propagation and backpropagation are illustrated with the two hidden layer deep neural networks in the following example, in which both layers get three neurons each, in addition to input and output layers.
* The number of neurons in the input layer is based on the number of x (independent) variables, whereas the number of neurons in the output layer is decided by the number of classes the model needs to be predicted.
* Only one neuron in each layer in this example; however, the reader can attempt to create other neurons within the same layer.
* Weights and biases are initiated from some random numbers, so that in both forward and backward passes, these can be updated in order to minimize the errors altogether.
* During forward propagation, features are input to the network and fed through the

following layers to produce the output activation.

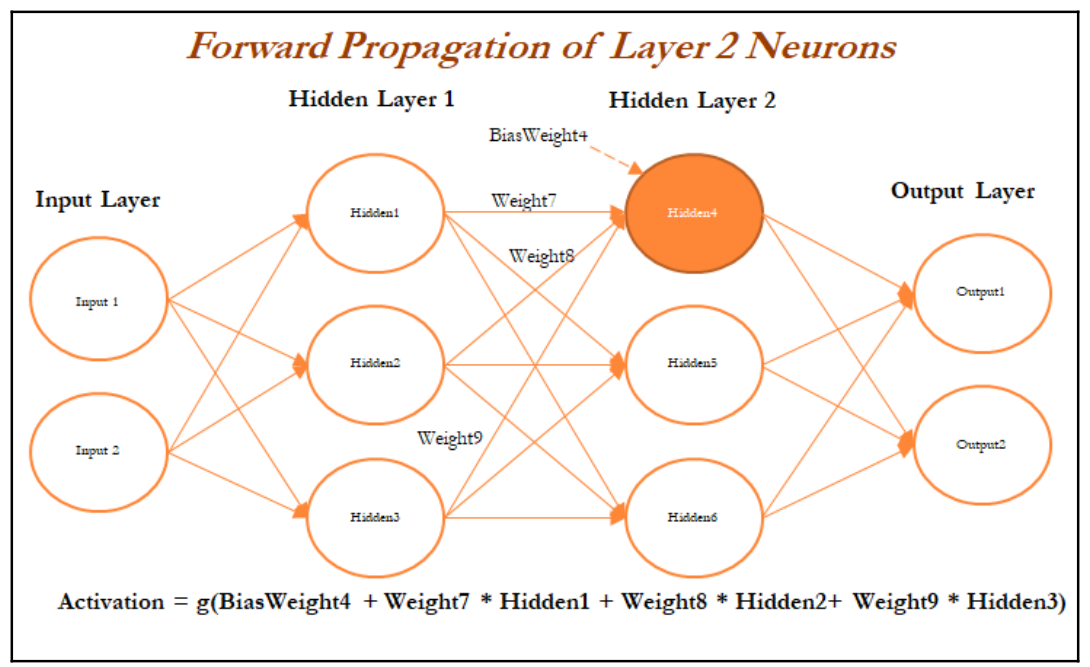
* If we see in the hidden layer 1, the activation obtained is the combination of bias weight 1 and weighted combination of input values;
* if the overall value crosses the threshold, it will trigger to the next layer, else the

signal will be 0 to the next layer values.

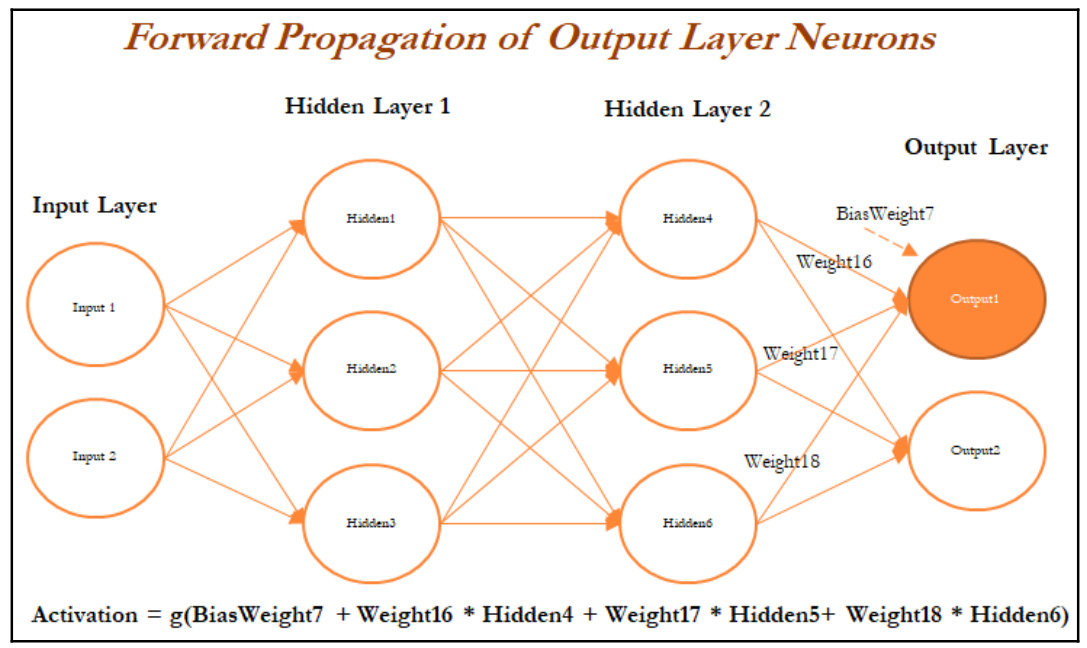
* Bias values are necessary to control the trigger points. If the weighted combination signal is low; bias will compensate the extra amount for adjusting the aggregated value, which can trigger for the next level. The complete equation can be seen in the following diagram:



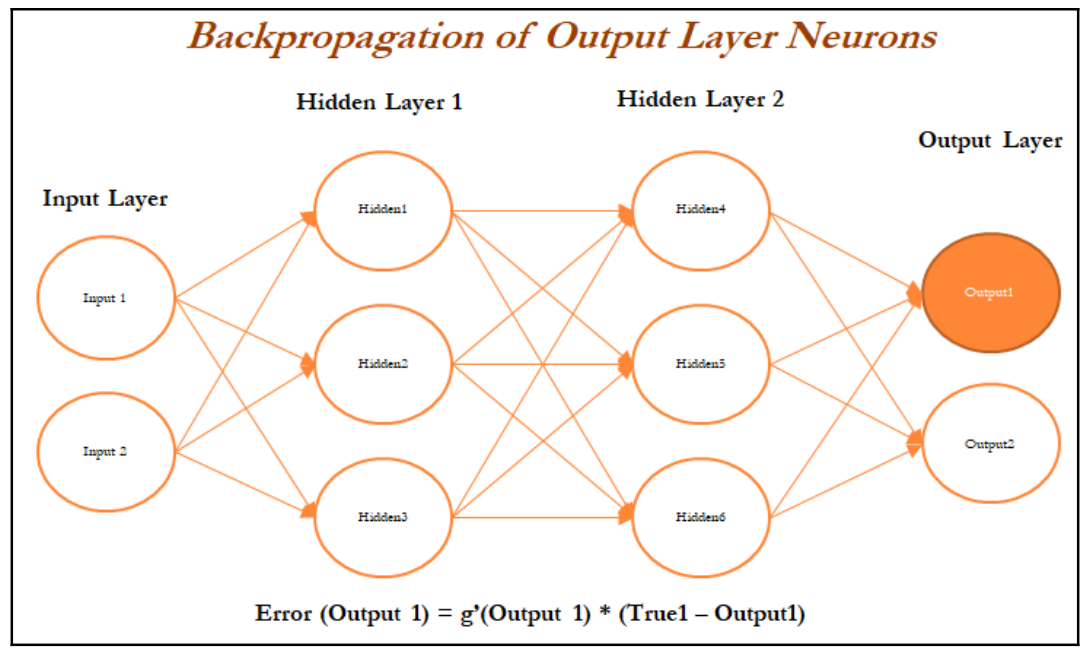
Once all the neurons are calculated in Hidden Layer 1 (Hidden1, Hidden2, and Hidden neurons), the next layer of neurons needs to be calculated in a similar way from the output of the hidden neurons from the first layer with the addition of bias (bias weight 4). The following figure describes the hidden neuron 4 shown in layer 2:



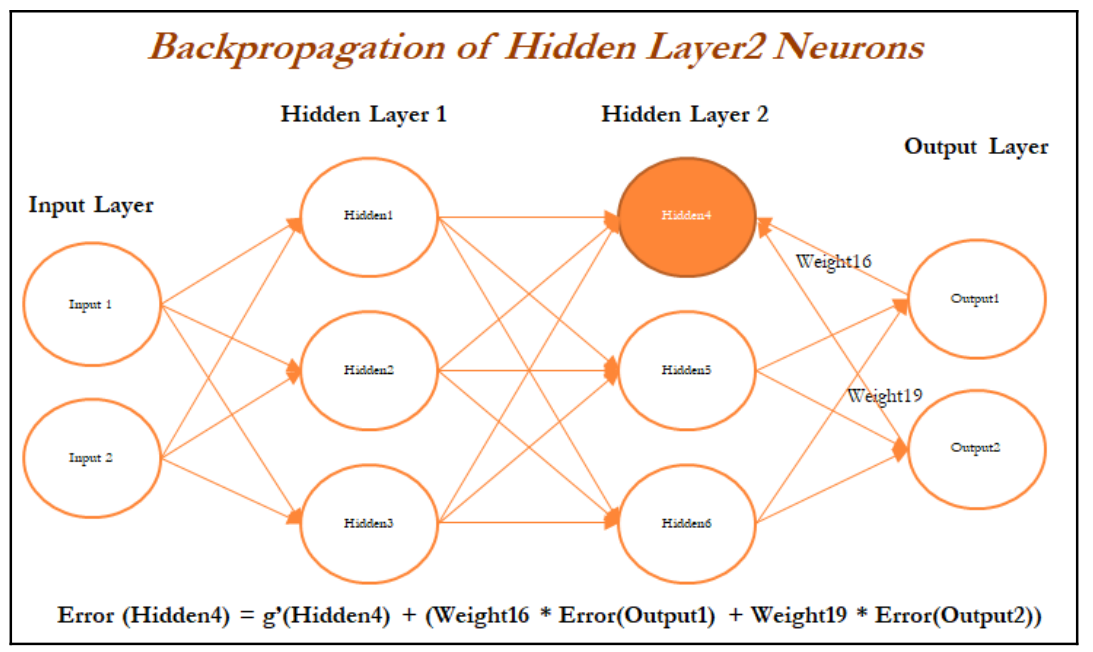
In the last layer (also known as the output layer), outputs are calculated in the same way from the outputs obtained from hidden layer 2 by taking the weighted combination of weights and outputs obtained from hidden layer 2. Once we obtain the output from the model, a comparison needs to be made with the actual value and we need to backpropagate the errors across the net backward in order to correct the weights of the entire neural network:



In the following diagram, we have taken the derivative of the output value and multiplied by that much amount to the error component, which was obtained from differencing the actual value with the model output:



In a similar way, we will backpropagate the error from the second hidden layer as well. In the following diagram, errors are computed from the Hidden 4 neuron in the second hidden layer:



In the following diagram, errors are calculated for the Hidden 1 neuron in layer 1 based on errors obtained from all the neurons in layer 2:



Once all the neurons in hidden layer 1 are updated, weights between inputs and the hidden layer also need to be updated, as we cannot update anything on input variables. In the following diagram, we will be updating the weights of both the inputs and also, at the same time, the neurons in hidden layer 1, as neurons in layer 1 utilize the weights from input only:

Diagram, schematic

Description automatically generated

Finally, in the following figure, layer 2 neurons are being updated in the forward propagation pass:

Diagram, schematic

Description automatically generated

We have not shown the next iteration, in which neurons in the output layer are updated with errors and backpropagation started again. In a similar way, all the weights get updated until a solution converges or the number of iterations is reached.

**Optimization of neural networks**

Various techniques have been used for optimizing the weights of neural networks:

* Stochastic gradient descent (SGD)
* Momentum
* Nesterov accelerated gradient (NAG)
* Adaptive gradient (Adagrad)
* Adadelta
* RMSprop
* Adaptive moment estimation (Adam)
* Limited memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS)

**Stochastic gradient descent - SGD**

Gradient descent is a way to minimize an objective function J(θ) parameterized by a

model's parameter θ ε Rd  by updating the parameters in the opposite direction of the

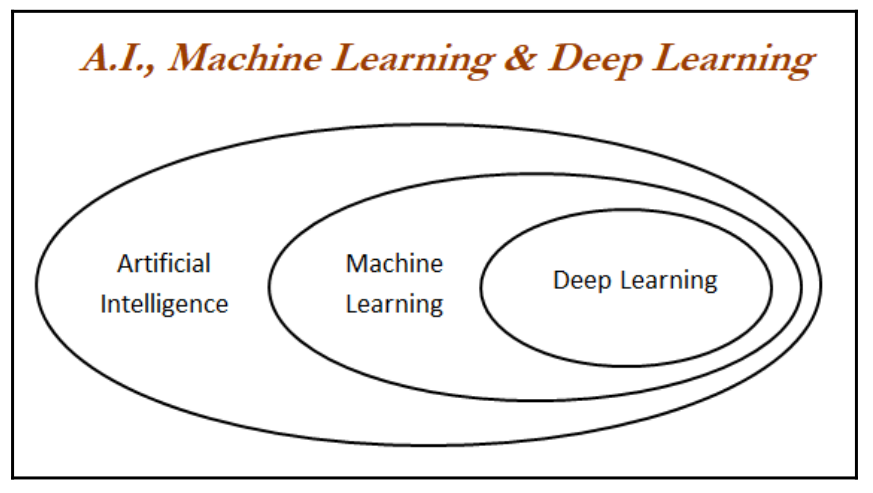
gradient of the objective function with regard to the parameters.

The learning rate determines the size of the steps taken to reach the minimum:

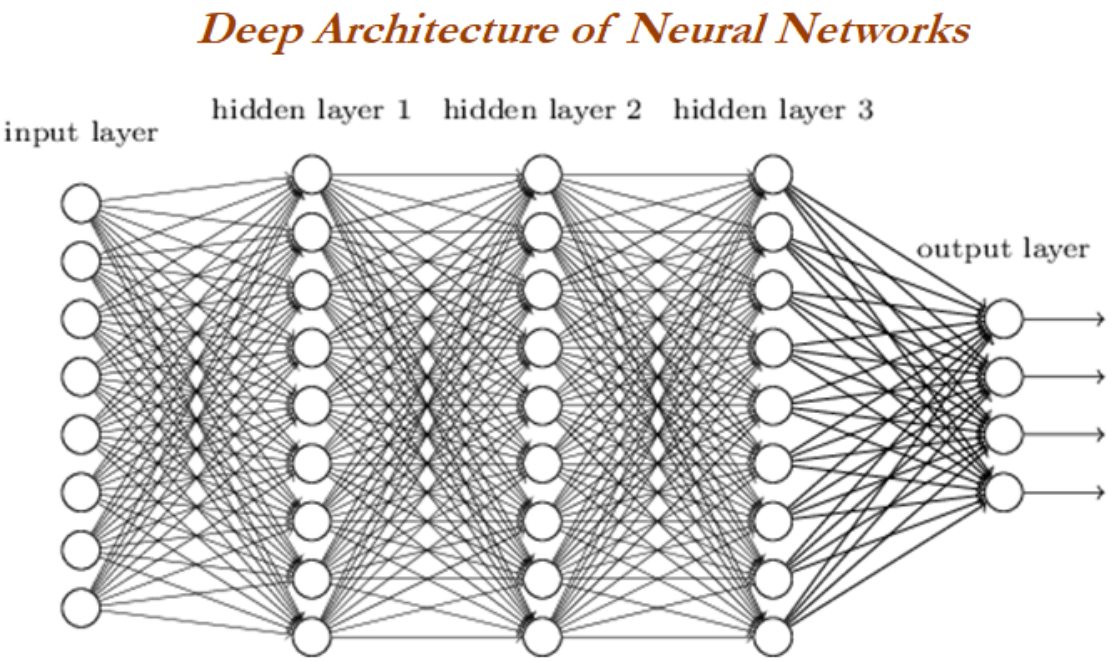
* Batch gradient descent (all training observations utilized in each iteration)
* SGD (one observation per iteration)
* Mini batch gradient descent (size of about 50 training observations for each iteration)

**Introduction to deep learning**

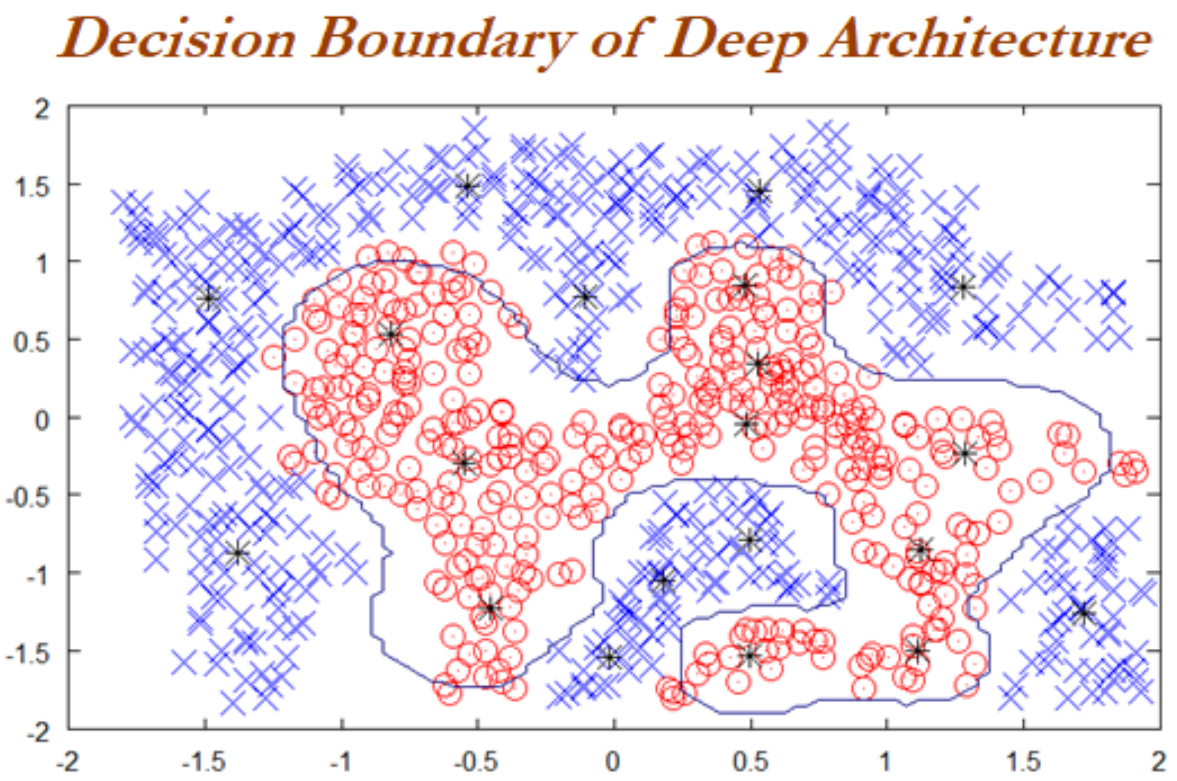
Deep learning is a class of machine learning algorithms which utilizes neural networks for building models to solve both supervised and unsupervised problems on structured and unstructured datasets such as images, videos, NLP, voice processing, and so on:



Deep neural network/deep architecture consists of multiple hidden layers of units between input and output layers. Each layer is fully connected with the subsequent layer. The output of each artificial neuron in a layer is an input to every artificial neuron in the next layer towards the output:



With the more number of hidden layers are being added to the neural network, more complex decision boundaries are being created to classify different categories. Example of complex decision boundary can be seen in the following graph:



**Solving methodology** Backpropagation is used to solve deep layers by calculating the error of the network at output units and propagate back through layers to update the weights to reduce error terms.

Thumb rules in designing deep neural networks: The following rules will provide some guidelines:

* All hidden layers should have the same number of neurons per layer
* Typically, two hidden layers are good enough to solve the majority of problems
* Using scaling/batch normalization (mean 0, variance 1) for all input variables

after each layer improves convergence effectiveness

* Reduction in step size after each iteration improves convergence, in addition to

the use of momentum and dropout

**Deep learning software**

Deep learning software has evolved multi-fold in recent times.

Here, we are using Keras to develop a model, as Keras models are easy to understand and prototype new concepts for newbies.

other software used:

* Theano: Python-based deep learning library developed by the University of

Montreal

* TensorFlow: Google's deep learning library runs on top of Python/C++
* Keras / Lasagne: Lightweight wrapper which sits on top of Theano/TensorFlow

and enables faster model prototyping

* Torch: Lua-based deep learning library with wide support for machine learning

Algorithms

* Caffe: deep learning library primarily used for processing pictures

TensorFlow is recently picking up momentum among the deep learning community, as it is being backed up by Google and also has good visualization capabilities using TensorBoard:

