MODULE 4

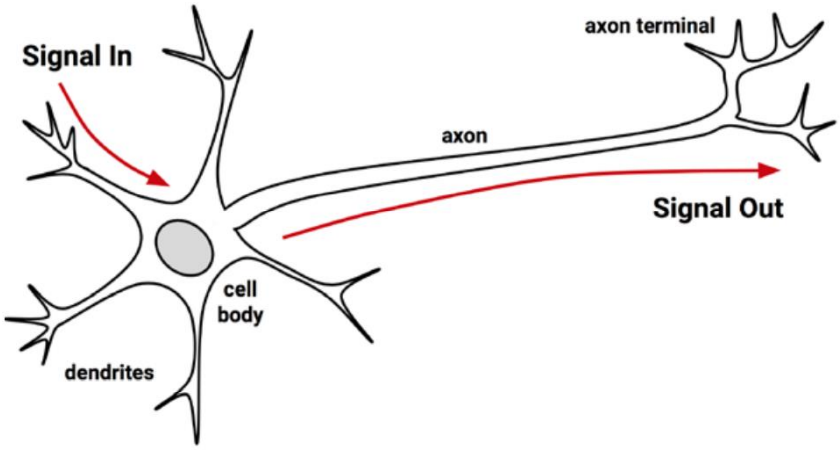
Lecture Notes

APARNA S BALAN
DEPARTMENT OF COMPUTER APPLICATIONS
Vidya Academy of Science & Technology

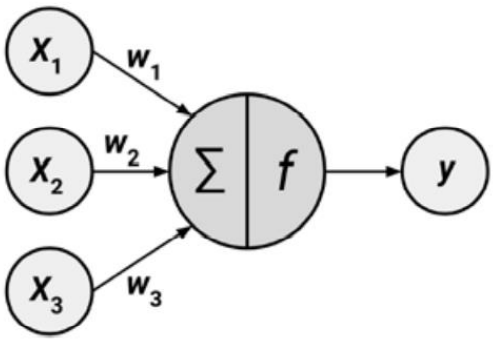
Neural Network Learning

An **Artificial Neural Network (ANN)** models the relationship between a set of input signals and an output signal using a model derived from our understanding of how a biological brain responds to stimuli from sensory inputs. ANN uses a network of **artificial neurons or nodes** to solve learning problems just as a brain uses a network of interconnected cells called neurons to create a massive parallel processor. ANNs are versatile learners that can be applied to nearly any learning task: classification, numeric prediction, and even unsupervised pattern recognition.

Artificial Neurons



In biological neurons incoming signals are received by the cell's **dendrites** through a biochemical process. The process allows the impulse to be weighted according to its relative importance or frequency. As the cell body begins accumulating the incoming signals, a threshold is reached at which the cell fires and the output signal is transmitted via an electrochemical process down the **axon**. At the axon's terminals, the electric signal is again processed as a chemical signal to be passed to the neighbouring neurons across a tiny gap known as a **synapse**.



A directed network diagram can be used to model a single artificial neuron. It defines a relationship between the input signals received by the dendrites (***x*** **variables**), and the output signal (***y*** **variable**). Just as with the biological neuron, each dendrite's signal is weighted (***w*** **values**) according to its importance. The input signals are summed by the cell body and the signal is passed on according to an activation function denoted by f.

A typical artificial neuron with n input dendrites can be represented by the following formula. The w weights allow each of the n inputs (denoted by x_i) to contribute a greater or lesser amount to the sum of input signals. The net total is used by the **activation function** $\boldsymbol{f(x)}$, and the resulting signal, $y(x)$, is the output axon.

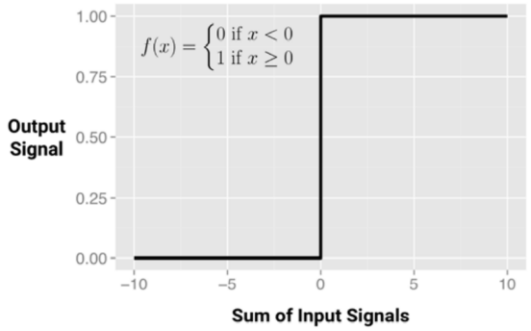
$$y(x) = f\left(\sum_{i=1}^{n} w_i x_i\right)$$

Neural networks use neurons as building blocks to construct complex models of data. Each can be defined in terms of: An **activation function**, which transforms a neuron's combined input signals into a single output signal to be broadcasted further in the network. A **network topology (or architecture),** which describes the number of neurons in the model as well as the number of layers and manner in which they are connected. The **training algorithm** that specifies how connection weights are set in order to inhibit or excite neurons in proportion to the input signal.

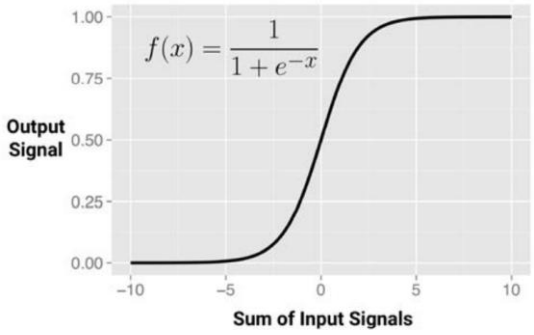
The number of input nodes is predetermined by the number of features in the input data. The number of output nodes is predetermined by the number of outcomes to be modelled or the number of class levels in the outcome.

Activation Functions

The **activation function** is the mechanism by which the artificial neuron processes incoming information and passes it throughout the network. In biological neuron, the activation function process involves summing the total input signal and determining whether it meets the firing threshold. If the threshold is met, the neuron passes on the signal; otherwise, it does nothing. A **threshold activation function** works in the similar way, it results in an output signal only once a specified input threshold has been attained.

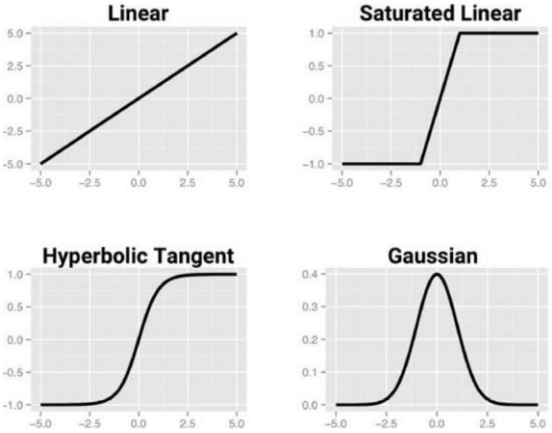


This figure shows a typical threshold function; the neuron fires when the sum of input signals is at least zero. Because its shape resembles a stair, it is sometimes called a **unit step activation function**. But it is rarely used in artificial neural networks.



The most commonly used alternative is the **sigmoid activation function** (logistic sigmoid), which uses the equation $f(x) = \frac{1}{1+e^{-x}}$, where e is the base of the natural logarithm. Now the threshold activation function generates "S" shape graph. Output values can fall anywhere in the range from 0 to 1. The sigmoid is **differentiable**, which means that

it is possible to calculate the derivative across the entire range of inputs.



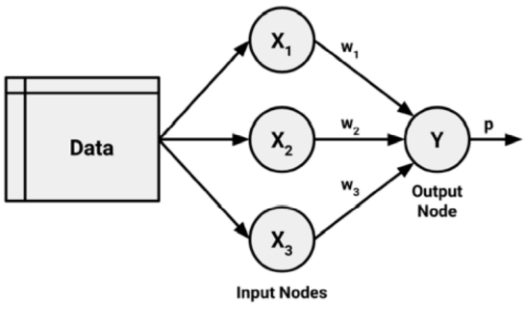
Other available activation functions are: linear, saturated linear, hyperbolic tangent, gaussian etc. The output signal range for these activation functions are different. The possible signal ranges are (0, 1), (-1, +1), or (-inf, +inf). Appropriate activation function can be selected to fit specific types of data during the construction of neural networks.

For many of the activation functions, the range of input values that affect the output signal is relatively narrow. For example, in the case of sigmoid, the output signal is always nearly 0 or 1 for an input signal below -5 or above +5, respectively. Activation functions like the sigmoid are sometimes called **squashing functions** because it squeezes the input values into a smaller range of outputs. The solution to the squashing problem is to transform all neural network inputs such that the features' values fall within a small range around 0, which can be achieved by standardizing or normalizing the features.

Network Topology

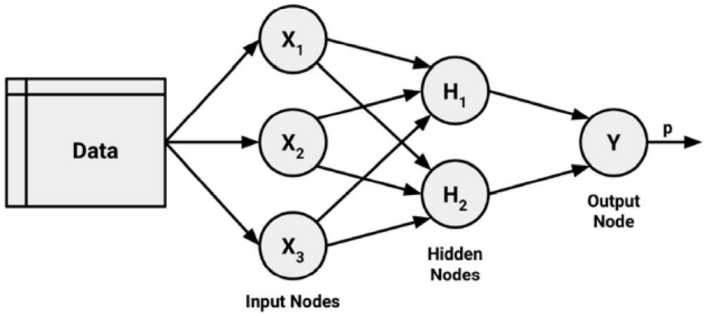
Neural network topology defines the patterns and structures of interconnected neurons. The topology determines the complexity of tasks that can be learned by the network. Different neural network topologies can be differentiated by three key characteristics: *number of layers, the direction of information travel* and *the number of nodes within each layer of the network*.

The number of layers



The given figure illustrates the topology of a very simple network. A set of neurons called **input nodes** receives unprocessed signals directly from the input data. Each input node is responsible for processing a single feature in the dataset; the feature's value will be transformed by the corresponding node's activation function. The signals sent by the input nodes are received by the output node, which uses its own activation function to generate a final prediction (denoted here as p). The input and output nodes are arranged in groups known as **layers**. Because the input nodes process the incoming data exactly as it is received, the network has only one set of connection weights (labeled here as w_1, w_2, and w_3).

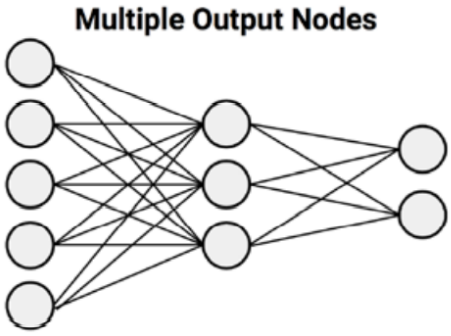
It is therefore termed a **single-layer network**. Single-layer networks can be used for basic pattern classification, particularly for patterns that are linearly separable.

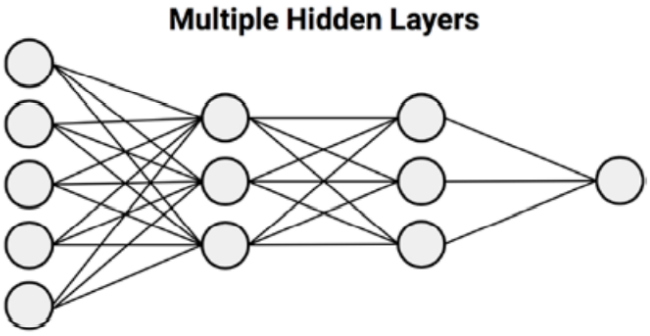


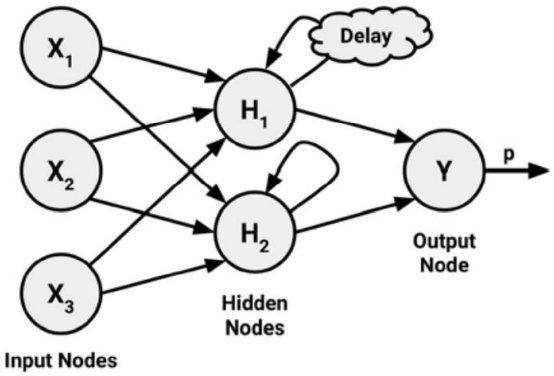
A **multilayer network** adds one or more **hidden layers** that process the signals from the input nodes prior to it reaching the output node. Most multilayer networks are fully connected, which means that every node in one layer is connected to every node in the next layer, but this is not required.

The direction of information travel

Networks in which the input signal is fed continuously in one direction from connection to connection until it reaches the output layer are called **feedforward networks**. A neural network with multiple hidden layers is called a **Deep Neural Network (DNN)** and the practice of training such network is sometimes referred to as **deep learning**. A **recurrent network (or feedback network)** allows signals to travel in both directions using loops (given in the following figure).







The addition of a short-term memory, or **delay**, increases the power of recurrent networks immensely. For example, it can be used to understand the sequences of events over a period of time. This could be used for stock market prediction, speech comprehension, or weather forecasting. Feedforward networks have been extensively applied to real-world problems. The multilayer feedforward network, sometimes called the **Multilayer Perceptron (MLP)**, is the de facto standard ANN topology.

The number of nodes in each layer

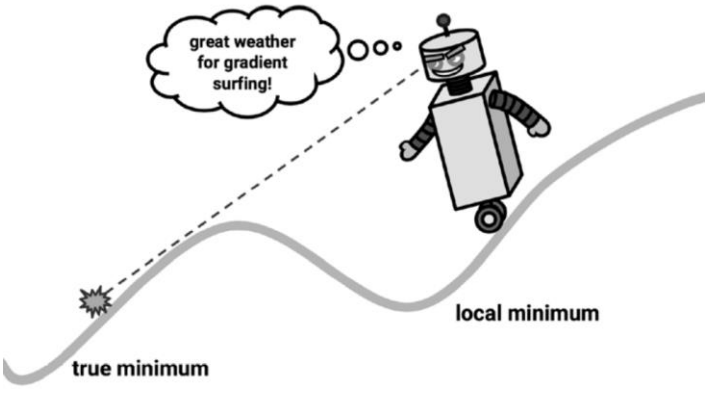
There is no reliable rule to determine the number of neurons(nodes) in the hidden layer. The appropriate number depends on the number of input nodes, the amount of training data, the amount of noisy data, and the complexity of the learning task etc. A greater number of neurons will result in a model that more closely mirrors the training data (risk of overfitting). Large neural networks can also be computationally expensive and slow to train. The best practice is to use the fewest nodes that result in adequate performance in a validation dataset. In most cases, even with only a small number of hidden nodes the neural network can offer a tremendous amount of learning ability.

Training Neural Networks with Backpropagation

During the **training**, connections between the neurons are strengthened or weakened as the neural network processes the input data. The network's connection weights are adjusted to reflect the patterns observed over time. It is very computationally intensive operation. **Backpropagation** is an efficient method of training an ANN. It uses the strategy of back-propagating errors.

The backpropagation algorithm iterates through many cycles of two processes known as an **epoch**. During the training the starting weights are set at random. Then, the algorithm iterates through the processes, until a stopping criterion is reached. Each epoch in the backpropagation algorithm includes: forward phase and backward phase.

In **forward phase**, the neurons are activated in sequence from the input layer to the output layer, applying each neuron's weights and activation function along the way. Upon reaching the final layer, an output signal is produced. In **backward phase**, the network's output signal resulting from the forward phase is compared to the true target value in the training data. The difference between the network's output signal and the true value results in an error that is propagated backwards in the network to modify the connection weights between neurons and reduce future errors.



The network uses the information sent backward to reduce the total error of the network. **Gradient descent technique** determines how much a weight should be changed. In this technique, the backpropagation algorithm uses the derivative of each neuron's activation function to identify

the gradient in the direction of each of the incoming weights. The gradient suggests how steeply the error will be reduced or increased for a change in the weight. The algorithm will attempt to change the weights that result in the greatest reduction in error by an amount known as the **learning rate**. The greater the learning rate, the faster the algorithm will attempt to descend down the gradients, which could reduce the training time at the risk of overshooting the valley.

Strengths

- Can be adapted to classification or numeric prediction problems
- Capable of modeling more complex patterns than nearly any algorithm
- Makes few assumptions about the data's underlying relationships

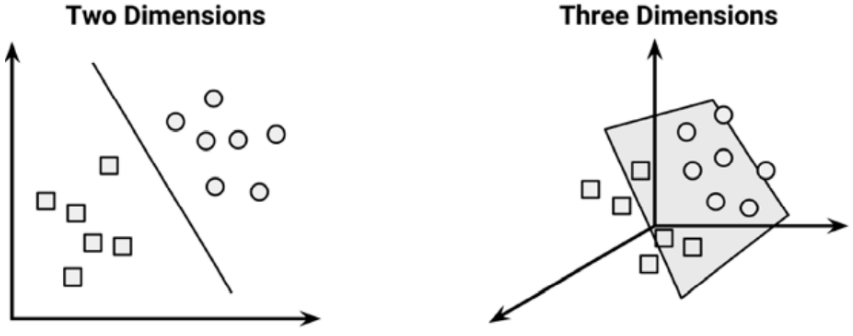
Weaknesses

- Extremely computationally intensive and slow to train, particularly if the network topology is complex
- Very prone to overfitting training data
- Results in a complex black box model that is difficult, if not impossible, to interpret

Support Vector Machines

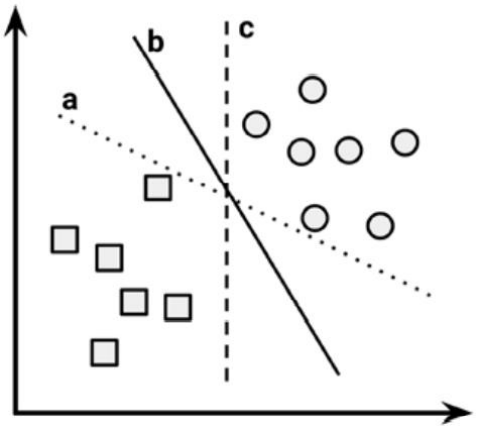
A **Support Vector Machine (SVM)** can be imagined as a surface that creates a boundary between points of data, which is plotted in multidimensional space that represent examples and their feature values. The goal of a SVM is to create a flat boundary called a **hyperplane**, which divides the space to create fairly homogeneous partitions on either side. SVMs are used to model highly complex relationships. SVMs can be used for both classification and numeric prediction.

Classification with Hyperplanes



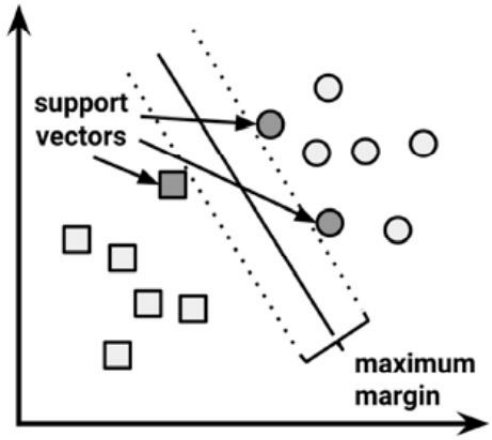
SVMs use a boundary called a **hyperplane** to partition data into groups of similar class values. The data points are said to be linearly separable if they can be separated perfectly by the straight line or flat surface. SVMs can also be extended to problems where the points are not linearly separable. The given figure depicts hyperplanes that separate groups of circles and squares in two and three

dimensions. The hyperplane is traditionally depicted as a line in 2D space, because it is difficult to illustrate space in greater than two dimensions. In reality, the hyperplane is a flat surface in a high-dimensional space.

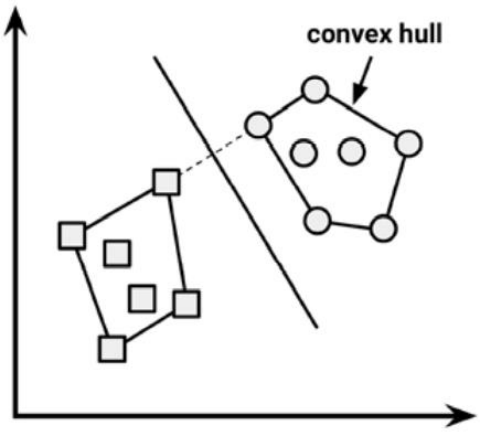


In two dimensions, the task of the SVM algorithm is to identify a line that separates the two classes. As shown in this figure, there is more than one choice of dividing line between the groups of circles and squares, which are labeled a, b, and c. But any of the three lines separating the circles and squares would correctly classify all the data points. In such a situation, **Maximum Margin Hyperplane (MMH)** that creates the greatest separation between the two classes is selected. The maximum margin will improve the chance that, in spite of random noise, the points will remain on the correct side of the boundary.

Maximum Margin Hyperplanes in Linearly Separable Data



The **support vectors** (indicated by arrows in the figure) are the points from each class that are the closest to the MMH; each class must have at least one support vector, but it is possible to have more than one. Using the support vectors alone, it is possible to define the MMH. The support vectors provide a very compact way to store a classification model, even if the number of features is extremely large.



Consider the classes are linearly separable. Then the MMH is as far away as possible from the outer boundaries of the two groups of data points. These outer boundaries are known as the **convex hull**. The MMH is then the perpendicular bisector of the shortest line between the two convex hulls. Algorithms that use a technique known as *quadratic optimization* are capable of finding the maximum margin in this way.

An alternative approach involves a search through the space of every possible hyperplane in order to find a

set of two parallel planes that divide the points into homogeneous groups yet themselves are as far apart as possible. In n-dimensional space a hyperplane can be defined using the equation: $\vec{\boldsymbol{w}}.\vec{\boldsymbol{x}} + \boldsymbol{b} = \boldsymbol{0}$. w is a vector of n weights, that is, $\{w_1, w_2, \ldots, w_n\}$, and b is a single number known as the bias (equivalent to the intercept term). Using this formula, the find a set of weights that specify two hyperplanes, as follows:

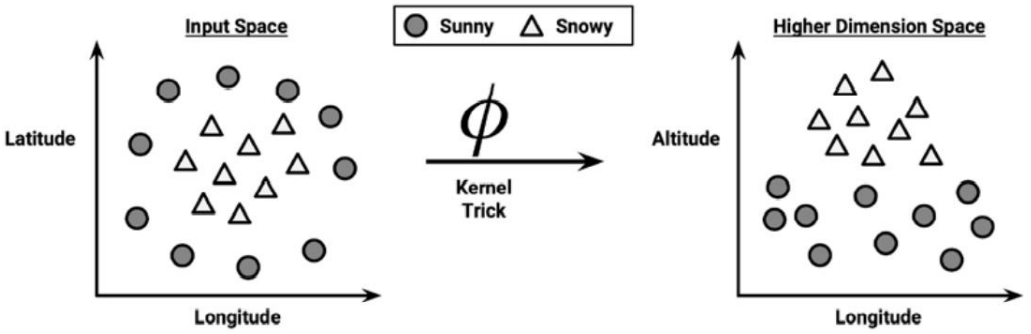
$$\vec{w}.\vec{x} + b \geq +1$$
$$\vec{w}.\vec{x} + b \leq -1$$

Since the data are linearly separable, all the points of one class fall above the first hyperplane and all the points of the other class fall beneath the second hyperplane. The *distance* between these two planes can defines in vector geometry as $\frac{2}{\|\vec{w}\|}$. Here, $||w||$ indicates the *Euclidean norm* (the distance from the origin to vector w). The above equation can be rewritten as:

$$min\frac{1}{2}\|\vec{w}\|^2$$
$$s.t.\ \ y_i(\vec{w}.\vec{x_i} - b \geq 1), \forall\vec{x_i}$$

Here the first line implies that the Euclidean norm need to minimize. The second line notes that this is subject to the condition that each of the y_i data points are correctly classified.

Using Kernels for Non-Linear Spaces



Kernel trick is a key feature of SVMs that allows to map the problem into a higher dimension space. In the scatterplot on the left depicts a nonlinear relationship between a weather class (sunny or snowy) and two features: latitude and longitude. The points at the center of the plot are members of the snowy class, while the points at the margins are all sunny. On the right side of the figure, after the kernel trick has been applied, data is viewed through a new dimension: altitude. With the addition of this feature, the classes are now perfectly linearly separable. Here, the trend is clear: snowy weather is found at higher altitudes.

SVMs with nonlinear kernels add additional dimensions to the data in order to create separation. The **kernel trick** involves a process of constructing new features that express mathematical relationships between measured characteristics.

The general form of kernel function is given below. The function denoted by φ(x), is a mapping of the data into another space. That is the general kernel function applies some transformation to the feature vectors x_i and x_j and combines them using the dot product, which takes two vectors and returns a single number.

$$K(\vec{x}_i, \vec{x}_j) = \varphi(\vec{x}_i).\varphi(\vec{x}_j)$$

The most commonly used kernel functions are *linear kernel, polynomial kernel, sigmoid kernel and Gaussian RBF kernel*. The **linear kernel** does not transform the data at all. Therefore, it can be expressed simply as the dot product of the features: $K(\vec{x}_i, \vec{x}_j) = \vec{x}_i.\vec{x}_j$. The **polynomial kernel** of degree d adds a simple nonlinear transformation of the data: $K(\vec{x}_i, \vec{x}_j) = (\vec{x}_i.\vec{x}_j + 1)^d$. The **sigmoid kernel** uses a sigmoid activation function. The Greek letters kappa and delta are used as kernel parameters: $K(\vec{x}_i, \vec{x}_j) = tanh(\kappa\, \vec{x}_i.\vec{x}_j - \delta)$. The **Gaussian RBF kernel** performs well on many types of data and is thought to be a reasonable starting point for many learning tasks: $K(\vec{x}_i, \vec{x}_j) = e^{\frac{-\left\|\vec{x}_i - \vec{x}_j\right\|^2}{2\sigma^2}}$.

Strengths

- Can be used for classification or numeric prediction problems
- Not overly influenced by noisy data and not very prone to overfitting
- May be easier to use than neural networks, particularly due to the existence of several well-supported SVM algorithms
- Gaining popularity due to its high accuracy and high-profile wins in data mining competitions

Weaknesses

- Finding the best model requires testing of various combinations of kernels and model parameters
- Can be slow to train, particularly if the input dataset has a large number of features or examples
- Results in a complex black box model that is difficult, if not impossible, to interpret.