Unit 5 - Assignment 2

In [2]:
```python
from IPython.display import Image
```

1. What is multilayer perception? Explain with help of signal flow graph.

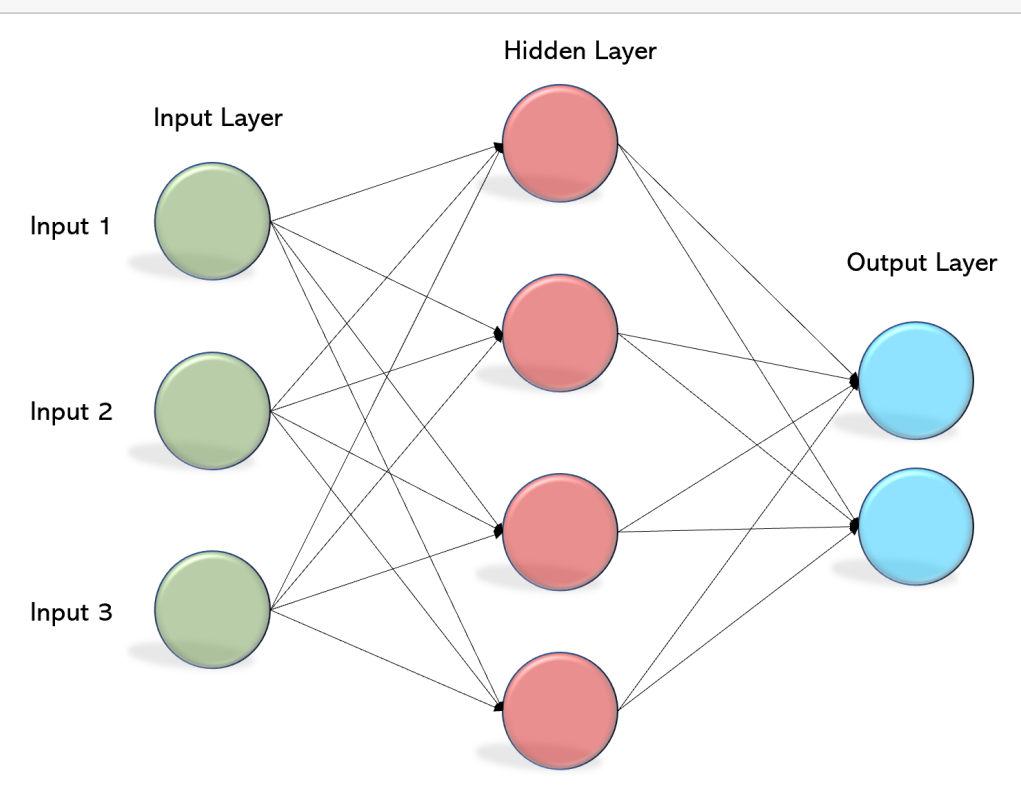
Ans:- A multilayer perceptron (MLP) is a class of feedforward artificial neural network (ANN). The term MLP is used ambiguously, sometimes loosely to refer to any feedforward ANN, sometimes strictly to refer to networks composed of multiple layers of perceptrons.

An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

Multilayer perceptrons are often applied to supervised learning problems: they train on a set of input-output pairs and learn to model the correlation (or dependencies) between those inputs and outputs. Training involves adjusting the parameters, or the weights and biases, of the model in order to minimize error. Backpropagation is used to make those weigh and bias adjustments relative to the error, and the error itself can be measured in a variety of ways, including by root mean squared error (RMSE). A typical multilayer perceptron (MLP) network consists of a set of source nodes forming the input layer, one or more hidden layers of computation nodes, and an output layer of nodes. The input signal propagates through the network layer-bylayer. The signal-flow of such a network with one hidden layer.

In [2]:
```python
Image(filename="img2/mlp.png")
```

Out[2]:



2. Explain back propagation algorithm in detail.

Ans:- Back propagation is a supervised learning method and is a generalization of the delta rule. It requires a dataset of the desired output for many inputs, making up the training set. It is a common method of training artificial neural networks so as to minimize the objective function. The back propagation learning algorithm can be divided into two phases: propagation and weight update. Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations. Backward propagation of the propagation's output activations through the neural network using the training pattern's target in order to generate the deltas of all output and hidden neurons.

Phase - 1: Propagation → each propagation involves the following steps:

- Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.
- Backward propagation of the propagation's output activations through the neural network using the training pattern's target in order to generate the deltas of all output and hidden neurons.

Phase - 2: Weight update → For each weight-synapse follow the following steps:

- Multiply its output delta and input activation to get the gradient of the weight.
- Bring the weight in the opposite direction of the gradient by subtracting a ratio of it from the weight.

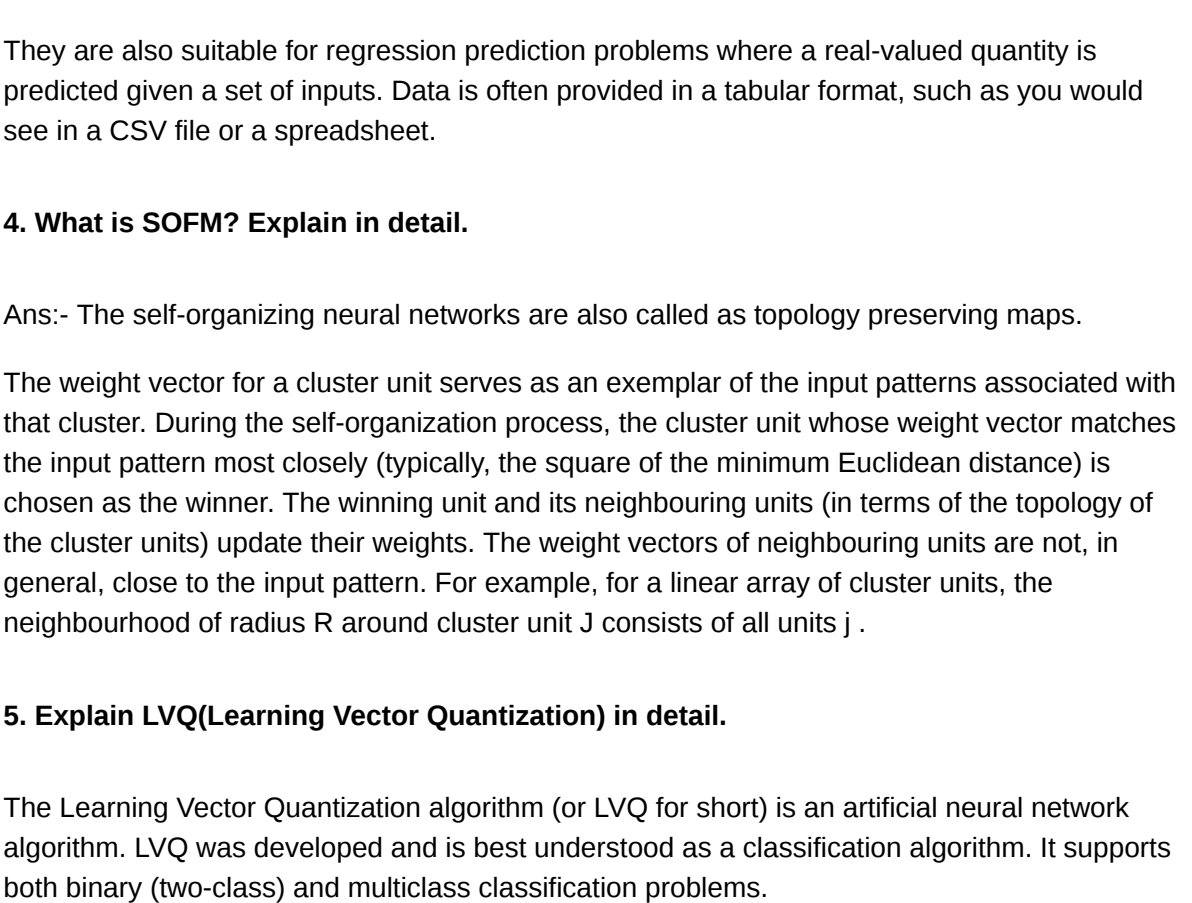
3. How the MLP used for classification? Explain.

Ans:- Multilayer Perceptrons, or MLPs for short, are the classical type of neural network.

They are comprised of one or more layers of neurons. Data is fed to the input layer, there may be one or more hidden layers providing levels of abstraction, and predictions are made on the output layer, also called the visible layer.

In [5]:
```python
Image(filename="img2/mlp02.png")
```

Out[5]:



MLPs are suitable for classification prediction problems where inputs are assigned a class or label.

They are also suitable for regression prediction problems where a real-valued quantity is predicted given a set of inputs. Data is often provided in a tabular format, such as you would see in a CSV file or a spreadsheet.

4. What is SOFM? Explain in detail.

Ans:- The self-organizing neural networks are also called as topology preserving maps.

The weight vector for a cluster unit serves as an exemplar of the input patterns associated with that cluster. During the self-organization process, the cluster unit whose weight vector matches the input pattern most closely (typically, the square of the minimum Euclidean distance) is chosen as the winner. The winning unit and its neighbouring units (in terms of the topology of the cluster units) update their weights. The weight vectors of neighbouring units are not, in general, close to the input pattern. For example, for a linear array of cluster units, the neighbourhood of radius R around cluster unit J consists of all units j .

5. Explain LVQ(Learning Vector Quantization) in detail.

The Learning Vector Quantization algorithm (or LVQ for short) is an artificial neural network algorithm. LVQ was developed and is best understood as a classification algorithm. It supports both binary (two-class) and multiclass classification problems.

A codebook vector is a list of numbers that have the same input and output attributes as your training data. For example, if your problem is a binary classification with classes 0 and 1, and the inputs width, length height, then a codebook vector would be comprised of all four attributes: width, length, height and class.

The model representation is a fixed pool of codebook vectors, learned from the training data. They look like training instances, but the values of each attribute have been adapted based on the learning procedure.

In the language of neural networks, each codebook vector may be called a neuron, each attribute on a codebook vector is called a weight and the collection of codebook vectors is called a network.

Data Preparation for LVQ

Generally, it is a good idea to prepare data for LVQ in the same way as you would prepare it for K-Nearest Neighbours.

- **Classification**: LVQ is a classification algorithm that works for both binary (twoclass) and multi-class classification algorithms. The technique has been adapted for regression.
- **Multiple-Passes**: Good technique with LVQ involves performing multiple passes of the training dataset over the codebook vectors (e.g. multiple learning runs). The first with a higher learning rate to settle the pool codebook vectors and the second run with a small learning rate to fine tune the vectors.
- **Multiple Best Matches**: Extensions of LVQ select multiple best matching units to modify during learning, such as one of the same class and one of a different class which are drawn toward and away from a training sample respectively. Other extensions use a custom learning rate for each codebook vector. These extensions can improve the learning process.
- **Normalize Inputs**: Traditionally, inputs are normalized (rescaled) to values between 0 and 1. This is to avoid one attribute from dominating the distance measure. If the input data is normalized, then the initial values for the codebook vectors can be selected as random values between 0 and 1.
- **Feature Selection**: Feature selection that can reduce the dimensionality of the input variables can improve the accuracy of the method. LVQ suffers from the same curse of dimensionality in making predictions as K-Nearest Neighbours.

6. What is RBF? Explain its architecture.

Ans:- Reference:- https://www.hackerearth.com/blog/developers/radial-basis-function-network/

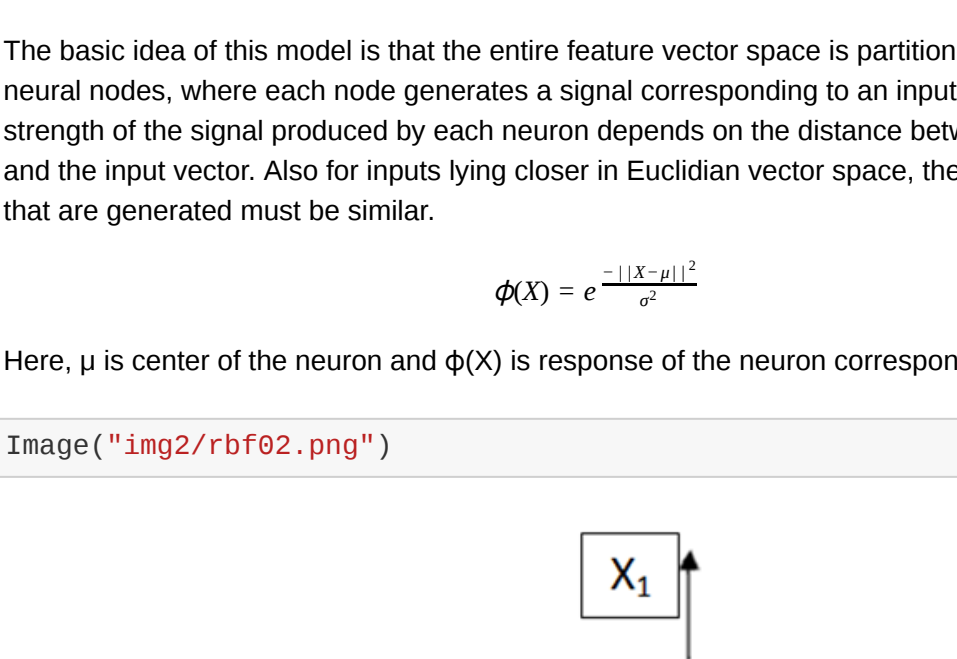
RBF network is an artificial neural network with an input layer, a hidden layer, and an output layer. The Hidden layer of RBF consists of hidden neurons, and activation function of these neurons is a Gaussian function. Hidden layer generates a signal corresponding to an input vector in the input layer, and corresponding to this signal, network generates a response.

Design and Development

To generate an output, neuron process the input signal through a function called activation function of the neuron. In RBF activation function of hidden neuron is ϕ(X) i.e. for an input vector X output produced by the hidden neuron will be ϕ(X) .

In [6]:
```python
Image("img2/rbf01.png")
```

Out[6]:



$$\phi(x) = e^{\frac{(x-y)^2}{\sigma^2}}$$

Above figure represents a Gaussian neural activation function for 1-D input x with center (mean) μ . Here ϕ(x) represents the output of Gaussian node for given value of x. It can be clearly seen that the signal strength decreases as the input (X) move away from the center. The range of the Gaussian function is determined by ?, and output beyond the range is considered to be negligible.

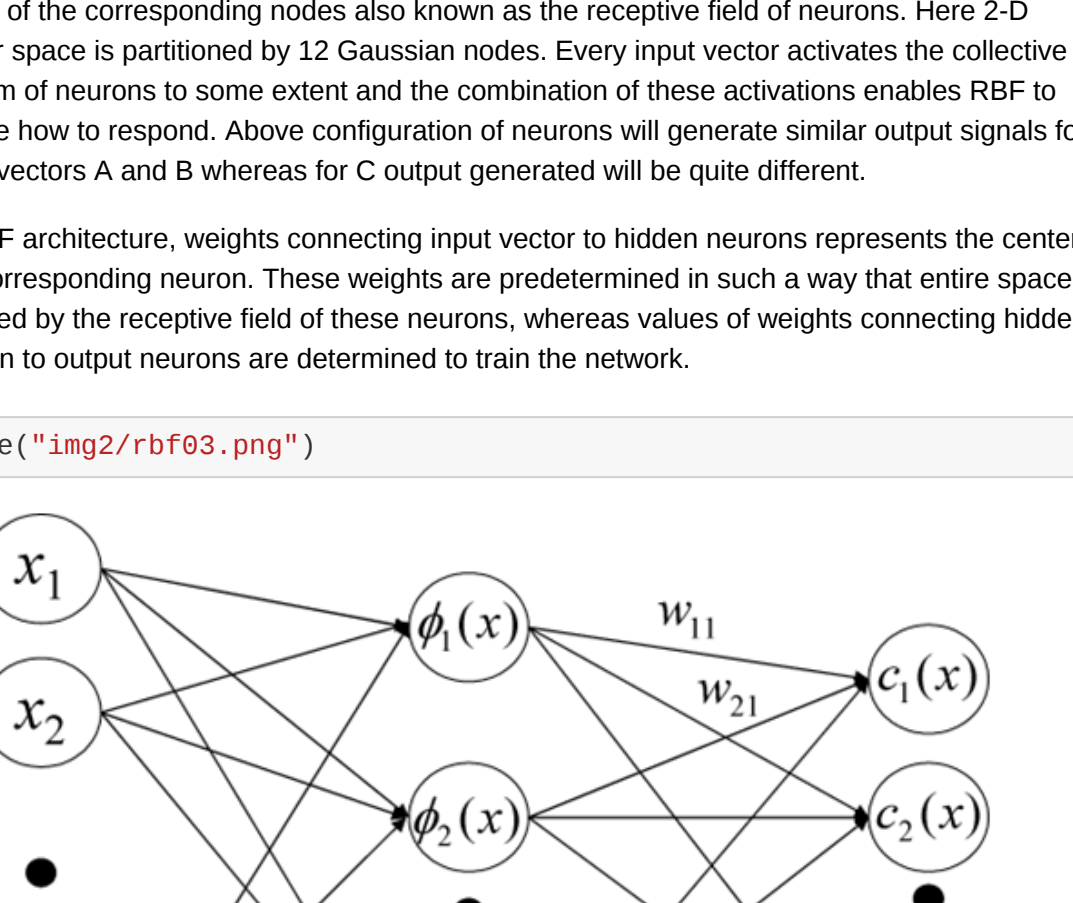
The basic idea of this model is that the entire feature vector space is partitioned by Gaussian neural nodes, where each node generates a signal corresponding to an input vector, and strength of the signal produced by each neuron depends on the distance between its center and the input vector. Also for inputs lying closer in Euclidian vector space, the output signals that are generated must be similar.

$$\phi(X) = e^{\frac{-||X-\mu||^2}{\sigma^2}}$$

Here, μ is center of the neuron and ϕ(X) is response of the neuron corresponding to input X.

In [7]:
```python
Image("img2/rbf02.png")
```

Out[7]:

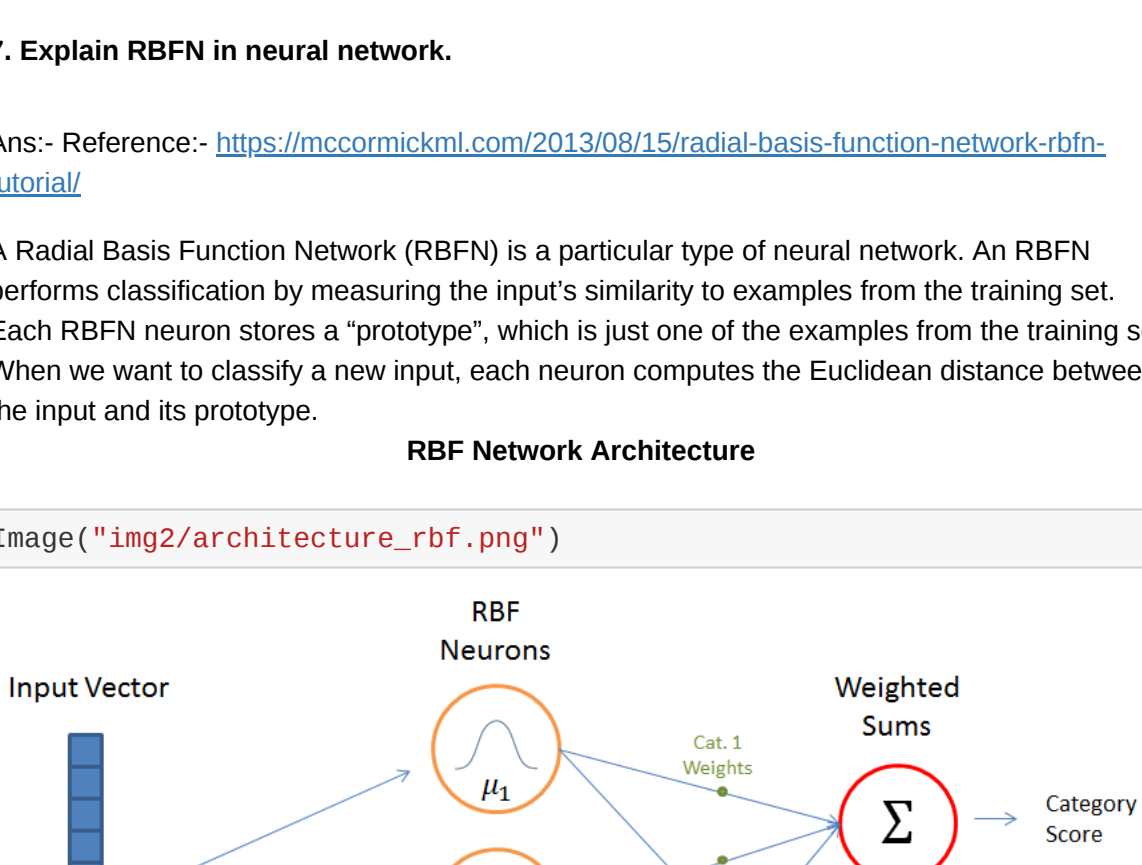


In above figure circles represent Gaussian neural nodes and boundary of circles represents the range of the corresponding nodes also known as the receptive field of neurons. Here 2-D vector space is partitioned by 12 Gaussian nodes. Every input vector activates the collective system of neurons to some extent and the combination of these activations enables RBF to decide how to respond. Above configuration of neurons will generate similar output signals for input vectors A and B whereas for C output generated will be quite different.

In RBF architecture, weights connecting input vector to hidden neurons represents the center of the corresponding neuron. These weights are predetermined in such a way that entire space is covered by the receptive field of these neurons, whereas values of weights connecting hidden neuron to output neurons are determined to train the network.

In [8]:
```python
Image("img2/rbf03.png")
```

Out[8]:



It's appropriate if vectors lying close in the Euclidian space falls under the receptive field of the same neuron; therefore centers of hidden neurons are determined using K-Means clustering.

7. Explain RBFN in neural network.

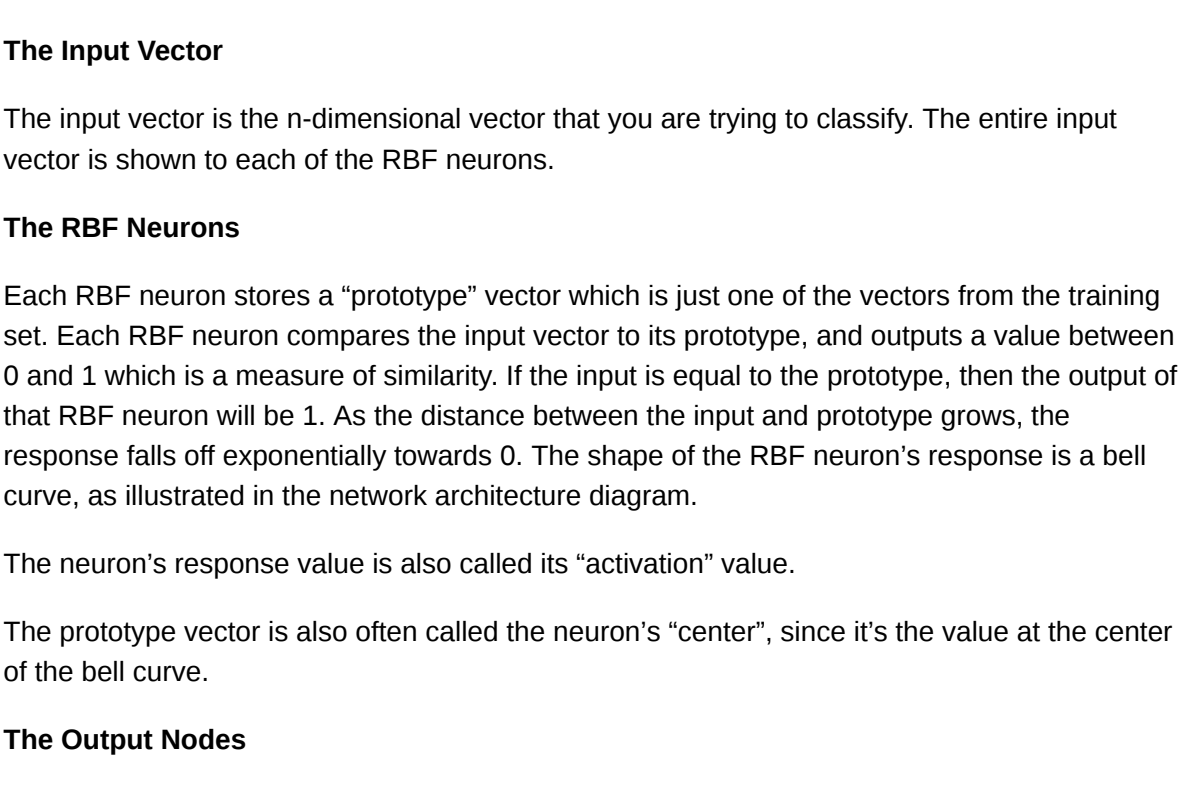
Ans:- Reference:- https://mccormickml.com/2013/08/15/radial-basis-function-network-rbfn-tutorial/

A Radial Basis Function Network (RBFN) is a particular type of neural network. An RBFN performs classification by measuring the input's similarity to examples from the training set. Each RBFN neuron stores a "prototype", which is just one of the examples from the training set. When we want to classify a new input, each neuron computes the Euclidean distance between the input and its prototype.

RBF Network Architecture

In [9]:
```python
Image("img2/architecture_rbf.png")
```

Out[9]:



The above illustration shows the typical architecture of an RBF Network. It consists of an input vector, a layer of RBF neurons, and an output layer with one node per category or class of data.

The Input Vector

The input vector is the n-dimensional vector that you are trying to classify. The entire input vector is shown to each of the RBF neurons.

The RBF Neurons

Each RBF neuron stores a "prototype" vector which is just one of the vectors from the training set. Each RBF neuron compares the input vector to its prototype, and outputs a value between 0 and 1 which is a measure of similarity. If the input is equal to the prototype, then the output of that RBF neuron will be 1. As the distance between the input and prototype grows, the response falls off exponentially towards 0. The shape of the RBF neuron's response is a bell curve, as illustrated in the network architecture diagram.

The neuron's response value is also called its "activation" value.

The prototype vector is also often called the neuron's "center", since it's the value at the center of the bell curve.

The Output Nodes

The output of the network consists of a set of nodes, one per category that we are trying to classify. Each output node computes a sort of score for the associated category. Typically, a classification decision is made by assigning the input to the category with the highest score.

The score is computed by taking a weighted sum of the activation values from every RBF neuron. By weighted sum we mean that an output node associates a weight value with each of the RBF neurons, and multiplies the neuron's activation by this weight before adding it to the total response.

Because each output node is computing the score for a different category, every output node has its own set of weights. The output node will typically give a positive weight to the RBF neurons that belong to its category, and a negative weight to the others.

8. Draw and explain bell shaped curve in RBF.

Ans:- **RBF Neuron Activation Function**

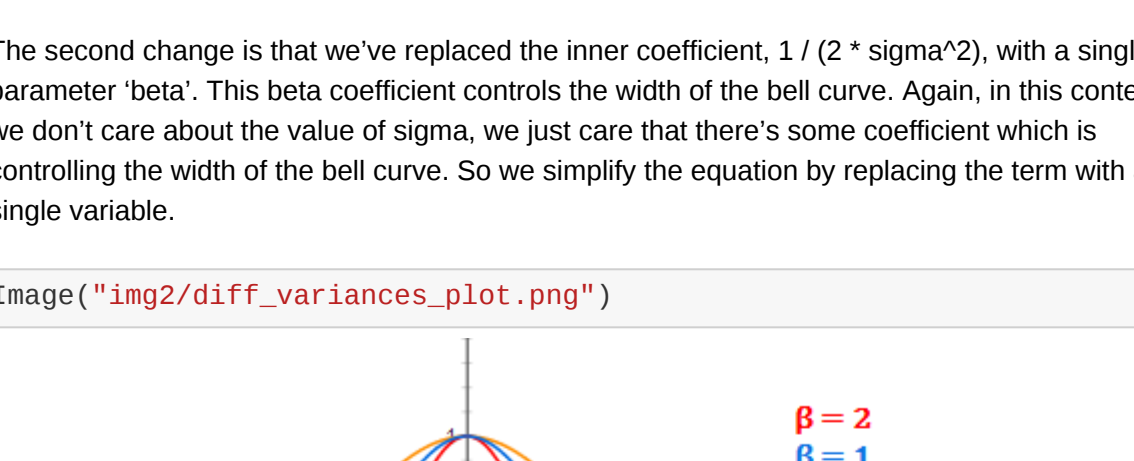
Each RBF neuron computes a measure of the similarity between the input and its prototype vector (taken from the training set). Input vectors which are more similar to the prototype return a result closer to 1. There are different possible choices of similarity functions, but the most popular is based on the Gaussian. Below is the equation for a Gaussian with a one-dimensional input.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Where x is the input, mu is the mean, and sigma is the standard deviation. This produces the familiar bell curve shown below, which is centered at the mean, mu (in the below plot the mean is 5 and sigma is 1).

In [13]:
```python
Image("img2/bell_curve.png")
```

Out[13]:



The RBF neuron activation function is slightly different, and is typically written as:

$$\Phi(x) = e^{-\beta||x-\mu||^2}$$

In the Gaussian distribution, mu refers to the mean of the distribution. Here, it is the prototype vector which is at the center of the bell curve.

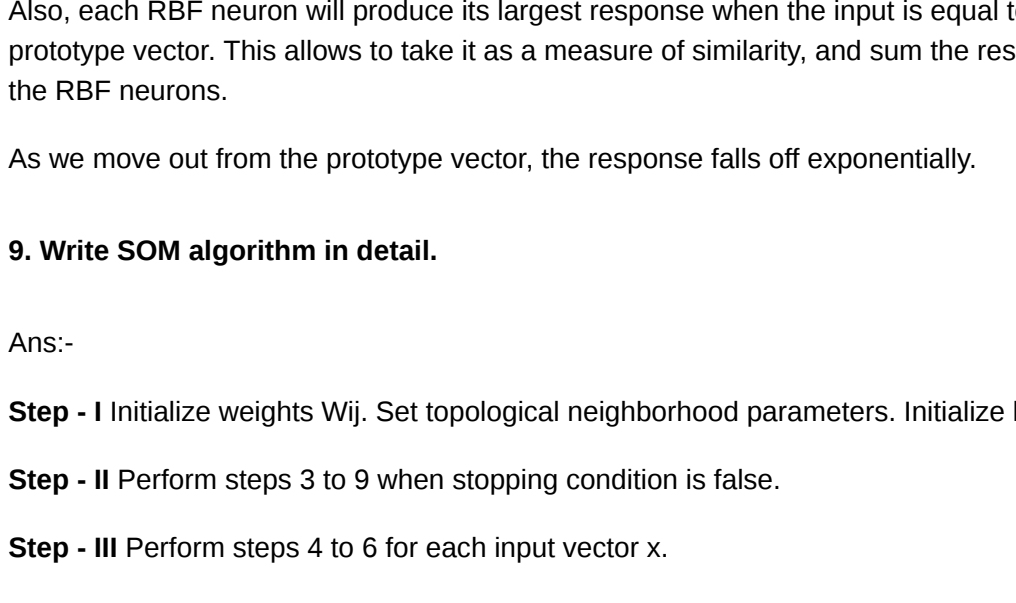
For the activation function, phi, we aren't directly interested in the value of the standard deviation, sigma, so we make a couple simplifying modifications.

The first change is that we've removed the outer coefficient, 1 / (sigma *sqrt(2* pi)). This term normally controls the height of the Gaussian. Here, though, it is redundant with the weights applied by the output nodes. During training, the output nodes will learn the correct coefficient or "weight" to apply to the neuron's response.

The second change is that we've replaced the inner coefficient, 1 / (2 * sigma^2), with a single parameter 'beta'. This beta coefficient controls the width of the bell curve. Again, in this context, we don't care about the value of sigma, we just care that there's some coefficient which is controlling the width of the bell curve. So we simplify the equation by replacing the term with a single variable.

In [14]:
```python
Image("img2/diff_variances_plot.png")
```

Out[14]:



RBF Neuron activation for different values of beta

There is also a slight change in notation here when we apply the equation to n-dimensional vectors. The double bar notation in the activation equation indicates that we are taking the Euclidean distance between x and mu, and squaring the result. For the 1-dimensional Gaussian, this simplifies to just (x - mu)^2.

It's important to note that the underlying metric here for evaluating the similarity between an input vector and a prototype is the Euclidean distance between the two vectors.

Also, each RBF neuron will produce its largest response when the input is equal to the prototype vector. This allows to take it as a measure of similarity, and sum the results from all of the RBF neurons.

As we move out from the prototype vector, the response falls off exponentially.

9. Write SOM algorithm in detail.

Ans:-

Step - I Initialize weights Wij. Set topological neighborhood parameters. Initialize learning rate.

Step - II Perform steps 3 to 9 when stopping condition is false.

Step - III Perform steps 4 to 6 for each input vector x.

Step - IV Compute square of Euclidean distance that is for j=1 to m,

$$\propto (t+1) = 0.5 \propto (t)$$

Step - V Find the winning unit index J, so that D(j) is minimum.

Step - VI For all unit j within a specific neighborhood of J and for all i, calculate new weights,

$$w_{ij}(new) = w_{ij}(old) + \propto [x_i - w_{ij}(old)]$$

$$or$$

$$w_{ij}(new) = (1 - \propto)w_{ij}(old) + \propto x_i$$

Step - VII Update learning rate using formula,

$$D(j) = \sum_{i=1}^{n}\sum_{i=1}^{m}(x_i - w_{ij})^2$$

Step - VIII Reduce radius of topological neighborhood at specific time intervals.

Step - IX Test for stopping condition of network.

10. How the loss function is calculated in back propagation algorithm?

Ans:- The loss function is a function that maps values of one or more variables onto a real number intuitively representing some "cost" associated with those values. For backpropagation, the loss function calculates the difference between the network output and its expected output, after a training example has propagated through the network.

Loss function. Basically it is a performance metric on how well the NN manages to reach its goal of generating outputs as close as possible to the desired values.

The most intuitive loss function is simply loss = (Desired output — actual output). However this loss function returns positive values when the network undershoot (prediction < desired output), and negative values when the network overshoot (prediction > desired output). If we want the loss function to reflect an absolute error on the performance regardless if it's overshooting or undershooting we can define it as: loss = Absolute value of (desired — actual).

Author Name:- Hemant Ghuge
LinkedIn:- https://www.linkedin.com/in/hemantghuge/
GitHub:- https://github.com/HemantGorakshGhuge