## Hopfield

In a Hopfield network, the purpose of the weight matrix is to store the memory patterns. When these patterns are presented to the network, the weights act as a sort of "filter" to enhance the correct pattern and suppress others, based on the principle of energy minimization.

What this shows is that the network, when presented with the perturbed version of ξ1, didn't immediately recall ξ1. However, after 2 iterations (2 steps of synchronous updating), the network settled into the correct state. This behaviour is referred to as a limit cycle of duration 2. The limit cycle is a characteristic behaviour of some dynamic systems in which the system oscillates between certain states (in this case, neuron configurations) in a periodic manner.

So, in simplistic terms, this statement is saying: "The network didn't immediately recognize the 'noisy' input as matching ξ1. But after adjusting its state over two iterations, it was able to 'clean up the noise' and properly recall ξ1."

Alignment
Retrieval performance noisy synchronous updating
Weight matrix

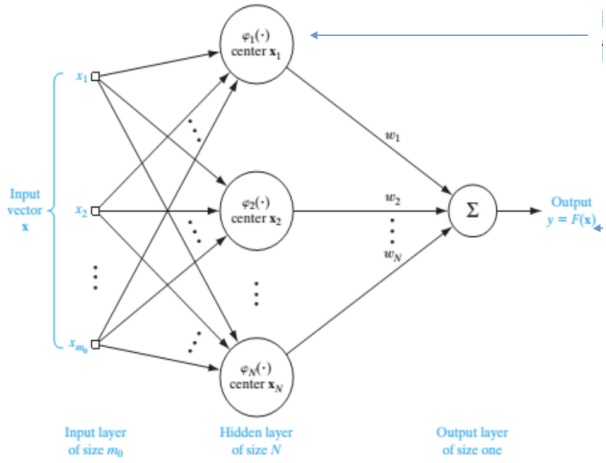
## RBF

Supervised learning alternative to BackPropagation, both pattern classification and regression
Classifies non-separable patterns using a two-stage process - non-linear in the first stage and linear in the second stage

RBF networks are based on the Separability theorem (Cover), which states that if the mapping (x) is nonlinear and hidden-unit space is high relative to input space then it is more likely to be separable

The first stage is to transform patterns into new space that is more likely to be separable
The second stage uses a linear algorithm ( LMS) to solve the problem in the new space
The hidden units are RBF which we assume are Gaussian functions, defined by mean and std σ

Each hidden unit is centred on a training pattern and patterns are learned perfectly
The problem is that if training patterns are noisy, the network learns the noise




The idea is to prevent learning noise in the training patterns also called the regularisation problem
There are fewer hidden units than the number of training patterns
Centres can be learned by unsupervised learning methods e.g. SOFM and can also change the standard deviation

MLP AND RBF COMPARED:

1. **Possibly multiple hidden layers vs. Single Layer:**
   - MLP: Multilayer Perceptron (MLP) has an input layer, one or more hidden layers, and an output layer. The multiple hidden layers enable the network to learn more complex and abstract representations of the input data.
   - RBF: Radial Basis Function (RBF) networks typically have a single hidden layer that transforms the input space into a higher dimensional space. This simplicity makes them easier to train and interpret than deep MLPs.

2. **Common computation nodes vs. different in hidden & o/p layers:**
   - MLP: The neurons (computation nodes) in all layers of an MLP are typically identical, computing a weighted sum of their inputs followed by a non-linear activation function.
   - RBF: The neurons in the hidden layer of an RBF network compute the Euclidean distance between the input and their centre and apply a radial basis function

(usually a Gaussian function). The neurons in the output layer compute a weighted sum of their inputs, i.e., a linear function.

3. **All layers are usually nonlinear vs. nonlinear hidden but linear output:**
   - MLP: In an MLP, every layer typically includes a non-linear activation function, such as a sigmoid or ReLU, which allows the network to learn non-linear mappings from input to output.
   - RBF: In an RBF network, the hidden layer applies a non-linear transformation to the input (through the radial basis function), but the output layer is typically linear. This combination of non-linear transformation followed by a linear combination allows RBF networks to approximate any function given enough neurons in the hidden layer.

4. **Computation of inner product of i/p vector & weight vector vs. Euclidean norm between i/p vector and centre of the appropriate unit:**
   - MLP: Neurons in an MLP compute the inner product (dot product) of the input vector and the weight vector. This linear operation is followed by a non-linear activation function.
   - RBF: Neurons in the hidden layer of an RBF network compute the Euclidean distance between the input vector and the centre (mean) of the RBF. This distance is then passed through a radial basis function, such as a Gaussian function, yielding a measure of similarity between the input vector and the neuron's centre.

5. **Global approximation and therefore good at extrapolation vs. local approximation with fast learning but poor extrapolation:**
   - MLP: MLPs perform a global approximation of the function they're learning. That means they use all inputs to compute an output and learn the general trends in the data. This makes them capable of extrapolation, predicting outputs for inputs outside the range seen during training.
   - RBF: RBF networks perform a local approximation. Each RBF in the hidden layer responds to a specific region in the input space, and the output is a weighted sum of these local responses. This makes them learn quickly and fit the training data well, but they're typically poor at extrapolation because each neuron is only sensitive to its local region of the input space.

HOW RDF SOLVES THE XOR PERCEPTRON PROBLEM

The XOR problem is a classic problem in the field of machine learning that demonstrates the limitations of simple, linear classifiers like the perceptron.

The XOR function has four possible inputs: (0, 0), (0, 1), (1, 0), and (1, 1). The output is 1 only when exactly one of the inputs is 1, otherwise, it's 0. The problem here is that this

function isn't linearly separable; you can't draw a straight line (in 2D space) or a plane (in 3D space) to separate the inputs that produce an output of 1 from those that produce an output of 0.

Radial Basis Function (RBF) networks solve this problem by introducing a non-linear transformation of the input space. Essentially, the RBF acts like a lens that warps the input space in such a way that the XOR function becomes linearly separable. In other words, after going through the RBF, the XOR function can be solved by a simple linear classifier like a perceptron.

In the provided example, two Gaussian functions serve as the RBFs. These functions map the original 2D input space into another 2D space, but this time, the XOR function is linearly separable. Therefore, a perceptron (or another simple linear classifier) can be used to classify the transformed inputs accurately.

So, the key intuition here is that RBFs allow us to transform complex, non-linear problems into simpler, linear ones that can be solved using basic techniques.

PROBLEMS AND MODIFICATIONS TO RBF

1. **Noisy Data**: The theoretical formulation of RBF networks is based on interpolation theory, which assumes the data is noise-free. However, in real-world scenarios, the data often contain noise, which can lead to inaccurate and misleading results when interpolation is applied directly. Therefore, a different design approach may be required for RBF networks to handle noisy data more effectively.

2. **Size of the Hidden Layer**: In standard RBF networks, the hidden layer is typically the same size as the input layer. This could lead to the wastage of computational resources, especially for large training samples. It's mentioned that this design can introduce redundancy in the hidden layer because any correlation existing between adjacent data points in the training sample gets transplanted into adjacent units in the hidden layer. To address this, it's suggested to make the size of the hidden layer a fraction of the size of the training sample. This can help to use computational resources more efficiently while reducing redundancy.

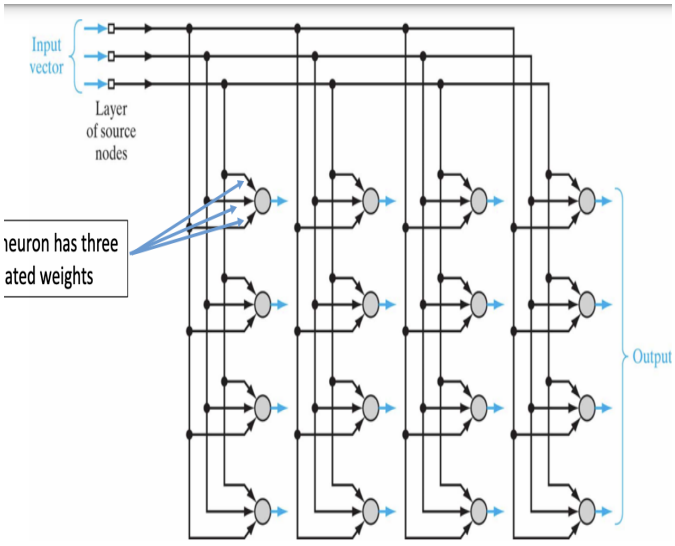
3. **No Backpropagation**: Unlike Multilayer Perceptrons (MLPs), the training of RBF networks does not involve the backpropagation of error signals. This characteristic is shared by the RBF networks depicted in both figures (5.3 and 5.4), highlighting a key difference in the learning mechanism of RBF networks compared to other types of neural networks.

4. **Same Mathematical Form**: Despite the differences in their structure, both RBF networks (the original and the one with a smaller hidden layer) realize the same type of approximating function. This is an important feature to retain as it is the core strength of the RBF networks - their ability to perform non-linear transformations and handle complex pattern recognition tasks.

**SOFM**

Self-Organizing Feature Map (SOFM) is a type of Artificial Neural Network (ANN) that uses unsupervised learning to produce a low-dimensional representation of the input space of the training samples. This is often referred to as a topographic map. This approach is used to understand high-dimensional data by reducing the dimensions and creating a map that preserves the significant relationships in the data.

The term "self-organizing" comes from the fact that the network learns to organize the data without any explicit instructions. Instead, it identifies common patterns and characteristics in the input data.




SOFM is structured with neurons arranged usually in a 1- or 2- dimensional lattice (motivated by the mappings in the brain). When a training example is presented, the map's neurons examine it, and the neuron whose weights are closest to the input

becomes the "winning" neuron. This neuron is then adjusted along with its neighbouring neurons to reflect the properties of the input vector, a process called competitive learning. The term "competitive" refers to the process where the neurons are competing to respond to the input data.

SOFM Algorithm

1. Initialisation: random values for initial weight vectors wj (0)
2. Sampling: draw a sample x from the input distribution
3. Similarity Matching

The principle of "density matching" in a Self-Organizing Feature Map (SOFM) is essential to make the weights of the neurons reflect the statistical distribution of the input vectors. It's an approach that makes use of the topology-preserving properties of the SOFM, allowing it to represent the density of the input space.

Here's how it works in more detail:

- The main idea is to adjust the network so that the weights resemble the inputs. This is done through the competitive learning rule, where for each input, the neuron with the closest weights (the "winning" neuron) is found, and its weights, as well as those of its neighbours, are adjusted to be closer to the input vector.

- The SOFM reflects variations in the statistics of the input distribution, as neurons that win more often (because they are closer to more input vectors) will have their weights adjusted more frequently and hence will represent denser regions of the input space.

- For a perfect density match, vectors that occur more frequently in the input data (high probability vectors) should map to larger regions of the neuron grid than less frequent (low probability) vectors. This is because each neuron represents a region of the input space, and the size of that region is essentially the probability of the vectors it represents.

- However, a common issue with SOFMs is that they tend to over-represent low-probability regions and under-represent high-probability regions. This happens because of the competitive nature of the learning rule: once a neuron wins a competition, it moves closer to the winning vector, leaving behind the region of space it used to cover.

- One common way to mitigate this issue is to use a Gaussian (instead of rectangular) neighbourhood function. A Gaussian neighbourhood function allows the impact of

the learning to decrease smoothly with distance from the winning neuron, rather than having a hard cut-off as with a rectangular function. This allows the network to maintain a better representation of the density of the input space, as it allows more overlap between the regions represented by different neurons.

In a nutshell, the SOFM as a density matching technique is about aligning the network's representation with the statistical properties of the input data, while also dealing with the challenges posed by the competitive learning rule.

N/B:

- Centres computed by RDF can be learned by SOFM
- SOFM can learn better global context than RDF which is better at local context
- SOFM is unsupervised and RDF is supervised