# Overview of More Advanced Topics

Introduction to Neural Networks : Lecture 19

1. Optimal Network Architectures
2. Recurrent Networks
3. The Hopfield Model
4. The Boltzmann Machine
5. Bayesian Techniques
6. Adaptive Resonance Theory
7. Support Vector Machines
8. Evolving Neural Networks

# Optimal Network Architectures

We have seen that finding a good neural network architecture (e.g. setting the number of hidden units) is very important, and also very problem dependent.

There are various different criteria for what is ***optimal***, usually it is generalization ability, but learning time, memory requirements, and so on, may also be important.

In the same way that we can have our networks learn their weights incrementally, we can modify our network architecture, to suit its given task, in an incremental fashion.

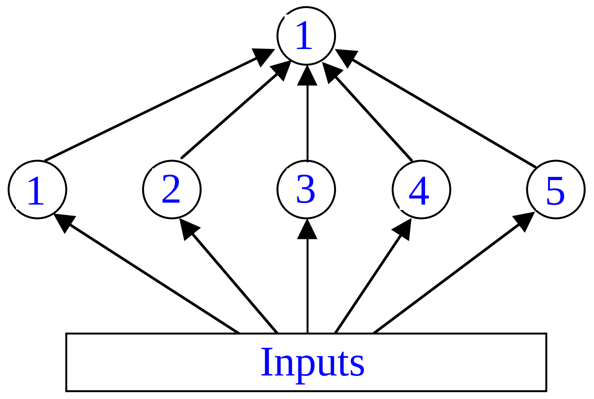
There are two natural ways to proceed:

1. Start with too few hidden units, and add some more ⇒ ***Constructive algorithms***
2. Start with too many hidden units, and take some away ⇒ ***Pruning algorithms***

Many algorithms exist for each approach – we shall just consider a couple of popular examples for each.

## Constructive Algorithms

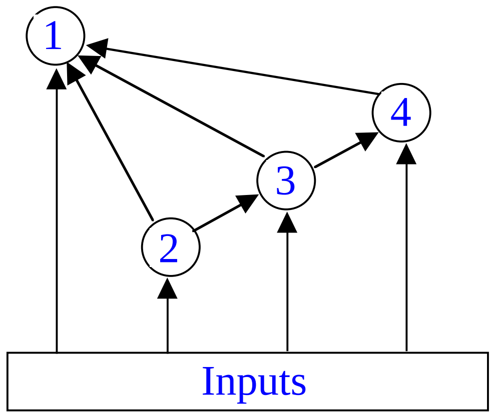
The simplest "constructive" approach is to train a series of networks, each with an increasing number of hidden units, until we are convinced that we have found the number that gives the best generalization (or whatever our criteria is). However, this is very wasteful in terms of having to start the training from scratch for each iteration. A better approach is the keep each existing network and only train the new portions of it.



For example, we can add hidden units in the sequence shown, with each additional unit trained to deal with the remaining incorrect training patterns. The hidden representation is then linearly separable, and the output weights can be trained very easily.

## Cascade Correlation

Another efficient approach, that is applicable to problems with continuous variables, is known as ***Cascade Correlation***. For each new hidden unit, we first learn the weights into the new unit by maximising the correlation (in effect the covariance) between the network's residual error and the new unit's output, and then freeze all the weights into the hidden units and train the output weights to minimise the output errors.



There are two particularly important advantages to this algorithm: (i) each training stage is just a single layer network which converges rapidly, (ii) once trained, the hidden unit activation for each training pattern is fixed, and not re-computed at each stage.

## Pruning Algorithms

We have already seen how adding a regularization term into the gradient descent cost function can be used to encourage the decay of unnecessary weights and effectively prune out unwanted connections.

We can also remove connections explicitly. If our gradient descent (e.g. sum squared) error function is $E(\{w_{ij}\})$, then we can define the ***saliency*** of each weight $w_{kl}$ as

$$s_{kl} = E(\{w_{ij} : w_{kl} = 0\}) - E(\{w_{ij}\})$$

i.e. how much the error increases when we remove that weight from the network. We can then define an iterative procedure whereby we train the network, compute all the saliencies, remove the weight(s) with the lowest saliency, and repeat the process until even the lowest saliencies are 'large', or until the error on some validation set starts rising again. This can be computationally expensive, but we can compute derivatives to estimate the saliencies and avoid multiple runs to determine the $E(\{w_{ij} : w_{kl} = 0\})$.

# Optimal Brain Damage

We can always expand our error function as a Taylor series

$$E(\{w_{ij} + \delta w_{ij}\}) = E(\{w_{ij}\}) + \sum_{i,j} \frac{\partial E(\{w_{ij}\})}{\partial w_{ij}} \delta w_{ij} + \tfrac{1}{2} \sum_{i,j} \sum_{k,l} \frac{\partial^2 E(\{w_{ij}\})}{\partial w_{ij} \partial w_{kl}} \delta w_{ij} \delta w_{kl} + O(\delta w^3)$$

in which the second order derivative is known as the ***Hessian*** matrix

$$H_{ijkl} = \frac{\partial^2 E(\{w_{ij}\})}{\partial w_{ij} \partial w_{kl}}$$

If we assume individual weights are small, and use the fact that the first derivatives are zero for a fully trained network, then we can set $\delta w_{ij} = - w_{ij}$ and estimate the saliency as

$$s_{ij} = \tfrac{1}{2} H_{ijij} w_{ij} w_{ij}$$

Iteratively training and pruning the network weights using the lowest values of this quantity is known as ***optimal brain damage***. A related, more efficient, approach is known as ***optimal brain surgeon***.

## Recurrent Networks

The fundamental feature of a recurrent network is that the network contains at least one ***feed-back connection***, so activation can flow around in a loop.

The networks can then do ***temporal processing***. In doing so they become dynamical systems and we have to worry about their *stability*, *controllability* and *observability*.

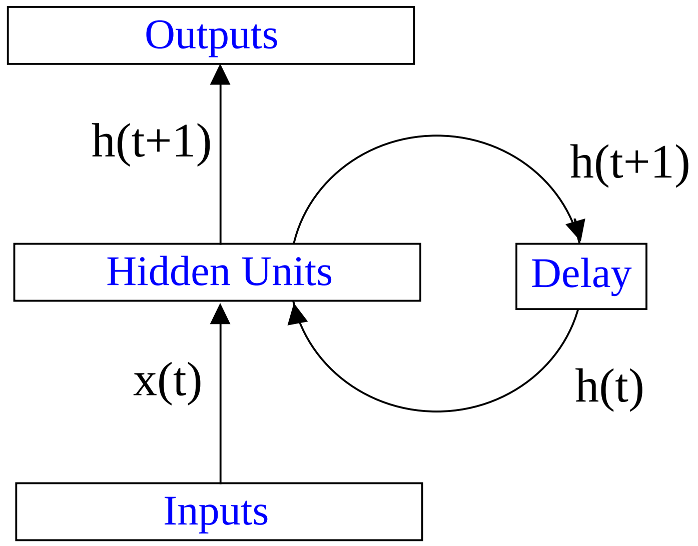
The architectures of recurrent networks can take many different forms. However, they all share the following common features:

1. They incorporate some form of static multi-layer perceptron as a sub-system.
2. They exploit the powerful non-linear mapping capabilities of the multi-layer perceptron, plus some form of memory.

Learning can be achieved by similar gradient descent procedures to those we used to derive the back-propagation algorithm. Turing machines may be simulated by fully connected recurrent networks based on neurons with sigmoidal activation functions.

# The Fully Recurrent Network

The basic ***fully recurrent network*** (***state space model***) has the hidden unit activations feeding back into the network along with the inputs of the next time step:



Note that we have to discretise time and update the activations one time step at a time. This might correspond to the time scale at which real neurons operate, or for artificial systems it can be any time step size appropriate to the problem in hand. A ***delay unit*** is introduced which simply delays the signal/activation until the next time step.

## The Hopfield Model

The ***Hopfield Model/Network*** is a fully connected network of $N$ McCulloch-Pitts neurons that deals with the basic associative memory problem:

> Store a set of $P$ binary valued patterns $\{\mathbf{t}^p\} = \{t_i^p\}$ in such a way that when presented with a new pattern $\mathbf{s} = \{s_i\}$ the network responds by producing whichever stored pattern most closely resembles **s**.

We use activation values ±1 rather than 0 and 1, so the neuron activation equation is

$$x_i = H\left(\sum_j w_{ij} x_j - \theta_i\right) \qquad \text{where} \qquad H(x) = \begin{cases} +1 & \text{if } \ x \geq 0 \\ -1 & \text{if } \ x < 0 \end{cases}$$

which can be updated ***synchronously*** or ***asynchronously***. The weights are given by

$$w_{ij} = \frac{1}{N} \sum_{p=1}^{P} t_i^p t_j^p \qquad , \qquad \theta_i = 0$$

A pattern $\mathbf{t}^q$ will be stable if the neuron update equation doesn't change anything, i.e.

$$t_i^q = H\left( \sum_j w_{ij} t_j^q - \theta_i \right) = H\left( \sum_j \frac{1}{N} \sum_p t_i^p t_j^p t_j^q \right)$$

We can see what is happening by separating out the $q$ term from the $p$ sum to give

$$t_i^q = H\left( t_i^q + \frac{1}{N} \sum_j \sum_{p \neq q} t_i^p t_j^p t_j^q \right)$$

If the second term in this were zero we could immediately conclude that pattern number $q$ was stable. We still have stability for non-zero values as long as the magnitude of the second term is smaller than 1, because that will not be enough to move us over the step of the step function $H$. This happens in most cases of interest as long as the number of stored patterns $P$ is ***small enough***. Not only are the stored patterns stable, but they are ***attractors*** of patterns close by. Estimates of what constitutes a small enough number $P$ leads to the idea of the ***storage capacity*** of a Hopfield network. A full discussion of Hopfield Networks can be found in most introductory books on neural networks.

## Boltzmann Machine

A ***Boltzmann Machine*** is a Hopfield Network composed of $N$ units with states $\{x_i\}$. The state of unit $i$ is updated asynchronously according to the rule

$$x_i = \begin{cases} +1 & \text{with probability } p_i \\ -1 & \text{with probability } 1 - p_i \end{cases}$$

where

$$p_i = \frac{1}{1 + \exp\left(-(\sum_{j=1}^{N} w_{ij} x_j - \theta_j)/T\right)}$$

with positive temperature constant $T$, and $w_{ij}, \theta_j$ are the networks weights and biases.

The fundamental difference between the Boltzmann Machine and a standard Hopfield Network is the ***stochastic activation*** of the units. If $T$ is very small, the dynamics of the Boltzmann Machine approximates the dynamics of the discrete Hopfield Network, but when $T$ is large the network visits the whole state space. Another difference is that the nodes of a Boltzmann Machine are split between visible input and output nodes, and hidden nodes, and the aim is to have the machine learn input-output mappings.

For both Hopfield Networks and Boltzmann Machines we can define an ***energy function***

$$E = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} w_{ij} x_i x_j + \sum_{i=1}^{N} \theta_i x_i$$

and the network activation updates cause the network to settle into a local minima of this energy. This implies that the stored patterns will be local minima of the energy. If a Boltzmann Machine starts off with a high temperature and is gradually cooled (known as ***simulated annealing***), it will tend to stay longer in the basins of attraction of the deepest minima, and have a good change of ending up in a global minimum at the end.

To train the network we can use the Boltzmann learning algorithm

$$\Delta w_{ij} = -\frac{\eta}{T}\left( \langle x_i x_j \rangle_{fixed} - \langle x_i x_j \rangle_{free} \right)$$

where $\langle x_i x_j \rangle_{fixed}$ is the expected/average product of $x_i$ and $x_j$ during training with the input and output nodes fixed at a training pattern and the hidden nodes free to update, and $\langle x_i x_j \rangle_{free}$ is the corresponding quantity if the output nodes are also free. Further details about Boltzmann Machines can be found in most introductory textbooks.

## Bayesian Techniques

Suppose we have input vectors $\{\mathbf{x}^p\}$ and corresponding output target vectors $T = \{\mathbf{t}^p\}$. Now consider a set $\{\mathsf{M}_i\}$ of different models $\mathsf{M}_i$ which we label to have increasing flexibility, e.g. networks with increasing numbers of hidden units. We can compute the ***posterior probability*** of each model, given the data $T$ using ***Bayes rule***

$$p(\mathsf{M}_i \mid T) = p(T \mid \mathsf{M}_i)\, p(\mathsf{M}_i) / p(T)$$

The ***prior probabilities*** $p(\mathsf{M}_i)$ can be assigned equal values, and the denominator $p(T)$ does not depend on the model, so the models can be compared using only the ***evidence*** $p(T \mid \mathsf{M}_i)$. We can use a similar expression for the network weights $W = \{w_{ij}{}^p\}$

$$p(W \mid T) = p(T \mid W)\, p(W) / p(T)$$

A good prior weight distribution $p(W)$ is a Gaussian around zero weights which will keep the network mapping smooth and hence give good generalization. In the Bayesian framework the posterior weight distribution $p(W \mid T)$ defines the "trained" network.

Bayesian inference techniques for neural networks offer many important advantages:

1. The conventional training method of error minimization arises as a particular approximation to the Bayesian approach.
2. Regularization can be given a natural interpretation, and values of large numbers of regularization parameters can be selected using only the training data, without the need for separate training and validation data.
3. Different models (e.g. networks with different numbers of hidden units, or even different network types) can be compared using only the training data.
4. Choices can be made about where in input space new data should be collected in order that it be the most informative – this is known as ***active learning***.
5. The relative importance of different inputs can be determined using the Bayesian technique of ***automatic relevance determination***. This is based on the use of a separate regularization coefficient for each input.
6. For regression problems, error bars, or ***confidence intervals***, can be assigned to the predictions generated by the network.

For further details of this powerful approach to neural networks, see Bishop Chapter 10.

## Adaptive Resonance Theory (ART)

This is actually a whole range of unsupervised ***incremental*** clustering neural networks designed for ***dynamically self-organizing data***. Most systems are either stable but not capable of forming new clusters, or incremental but unstable. The *ART-1* network manages both with binary input vectors, and *ART-2* with continuous input vectors.

All *ART* systems implement a process of matching the inputs with cluster templates that leads to a ***resonant state*** and weights updating. If no sufficiently well matched template exists, a new template is created and set equal to the current input pattern.

*ARTMAP* is a supervised version that learns input-output mappings by linking together two *ART-1* modules by an intermediate layer or *map field*. Fuzzy versions of *ART-1* give rise to the *Fuzzy ARTMAP* architecture. *ART-3* allows a series of *ART-2* modules to be linked into a processing hierarchy.

The whole *ART* family is covered well in Gurney Chapter 9.

## Support Vector Machines (SVM)

This is another category of universal feed-forward networks. It is a linear machine that constructs hyper-plane decision boundaries in such a way that the margin of separation between positive and negative examples is maximized. It does this by following a principled approach based on ***statistical learning theory***.

Crucial to the *SVM* learning algorithm is a set of ***support vectors*** consisting of a small sub-set of the training data selected by the algorithm. These are the data points that lie closest to the decision/boundary surfaces, and are therefore the most difficult to classify. They have the most direct bearing on the optimum location of the decision surfaces as the hyper-planes furthest from the support vectors.

The approach is very general and different variations can lead to many different learning machine architectures (including *RBF* and *MLP* networks).

Probably the best description of SVMs is in Haykin Chapter 6.

## Evolving Neural Networks

Most neural network systems have many parameters and other details that need setting appropriately for good performance (e.g. their architecture, learning rates, regularization parameters, and so on). In biological systems, these details have evolved so that the systems perform well. The idea of using simulated ***evolution by natural selection*** to generate high performance neural networks is becoming increasingly popular.

The general idea is to take a whole population of neural networks each specified by some ***genotypic encoding*** of its architecture and other parameters. Each individual network is then tested on its chosen task (e.g. learning a given set of training data) and its fitness determined (e.g. its speed of learning, or its final generalization error). The best/fittest individuals are then selected as the basis of the next generation. Simulated ***cross-over*** and ***mutation*** at the genotypic level results in a new population, and the process is then repeated. By selecting the best individuals of each generation to survive and 'breed', the average individual performance levels tend to improve from one generation to the next. We naturally eventually end up with a population of very fit neural networks.

## Overview and Reading

1. We started with some natural extensions of what we have seen before: constructive and pruning algorithms, and recurrent networks.
2. Then we looked at a selection of more advanced neural network types: Hopfield Networks, Boltzmann Machines, Bayesian Networks, Adaptive Resonance Theory (ART), and Support Vector Machines (SVM).
3. Finally, we looked at the idea of Evolving Neural Networks.

### Reading

1. Haykin: Sections 4.15, 6.1-6.9, 11.7, 14.7, 15.1-15.8
2. Hertz, Krogh & Palmer: Sections 2.1, 2.2, 6.6, 7.1-7.3, 9.3
3. Bishop: Sections 9.5, 10.0-10.10
4. Gurney: Sections 6.10, 7.1-7.10, 9.1-9.7
5. Beale & Jackson: Sections 6.1-6.5, 7.1-7.8