CMP-6002B Machine Learning

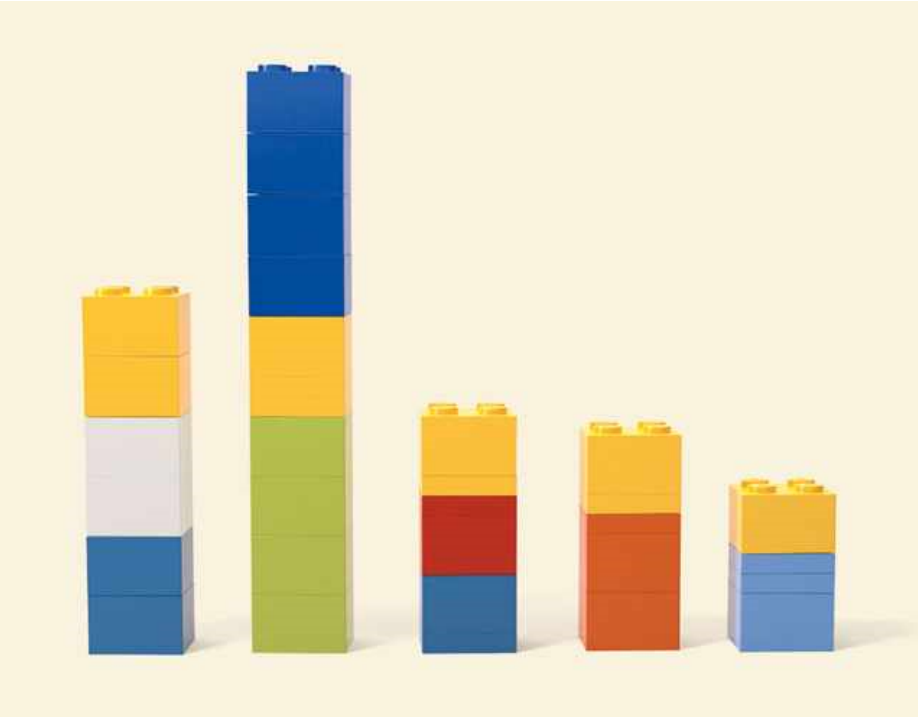
Dr Gavin Cawley

Lecture 5 - Artificial Neural Networks

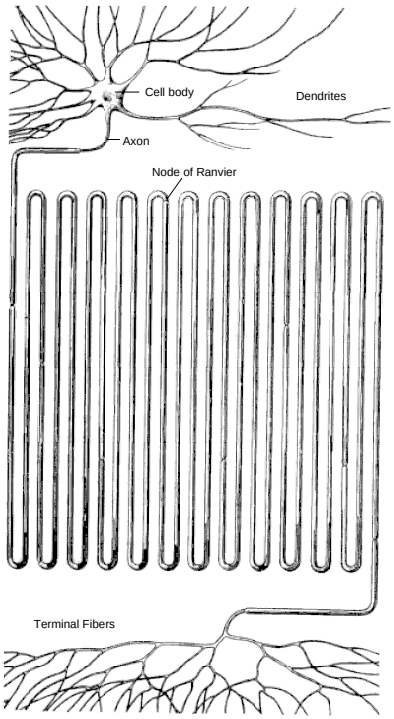
Introduction

- Historical background - the Perceptron.
- Limitations of single layer Perceptron networks.
- Capabilities of Multi-Layer Perceptron (MLP) networks.
- Training algorithms.
- Neural networks estimate Bayesian *a-posteriori* probabilities.
- Radial Basis Function (RBF) networks.
- Preprocessing.
- Avoiding overfitting.
- Recipe for neural network pattern recognition.
- Things we didn't cover.

Humans And Computers Are Good At Different Things

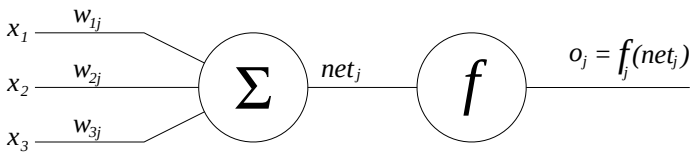


Biological Inspiration



- The human brain is composed of
  - $\approx 10^{11}$ neurons
  - $\approx 10^{15}$ connections
- The neuron "sums" inputs from *dendrites* via *synapses*.
- The synapses weight the inputs from other neurons.
- Neuron fires if sum exceeds threshold via *axon*.
- Learning involves change in the synaptic weights.

The Perceptron

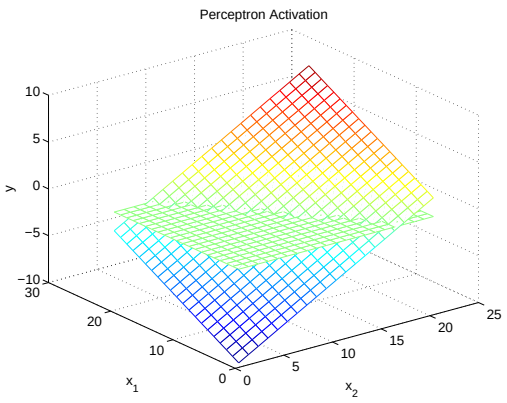


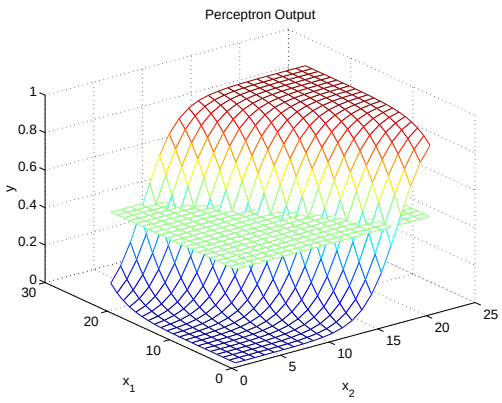
- The Perceptron [11] is a simple model of a biological neuron.
- It calculates a weighted sum of its inputs (plus the *bias*), known as the *activation* of the neuron, *net*.
- The output of the neuron is a (non-linear) function of the net activation.

$$f(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases} \qquad f(x) = \frac{1}{1 + e^{-x}}$$

Linear Classification

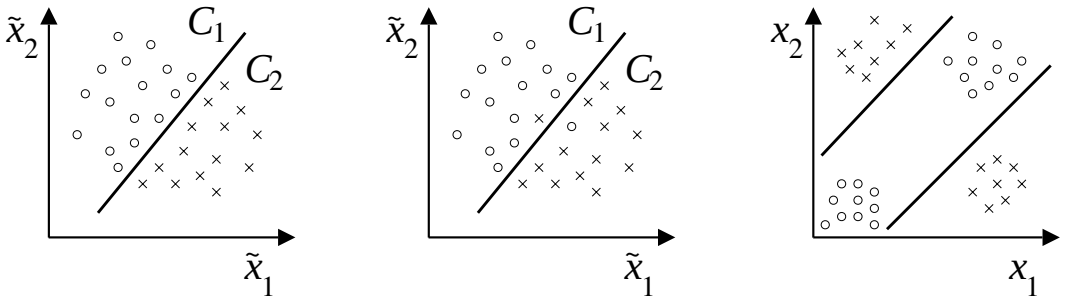
- The Perceptron implements a linear classifier.
- It can separate data into two classes using a straight line.
- For example, consider a Perceptron with two inputs x_1 and x_2, and a single output





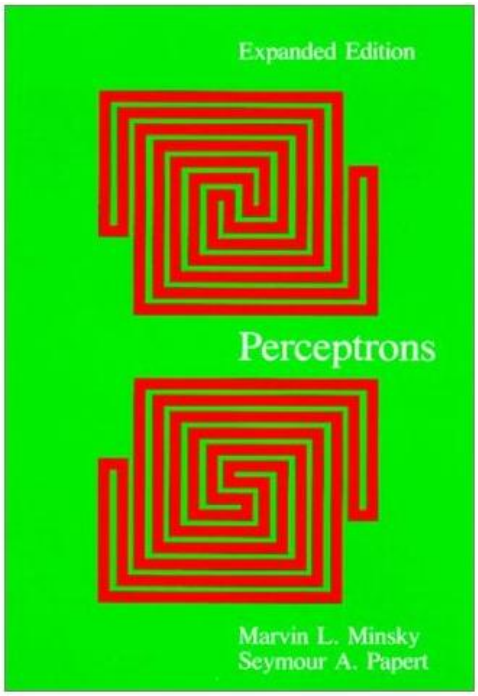
Linearly Separable and Non-Separable Problems

- A single Perceptron can solve any linearly separable two-class classification problem without error.
- Unfortunately most interesting classification problems are not linearly separable.
- The XOR problem is often cited as a "worst case".

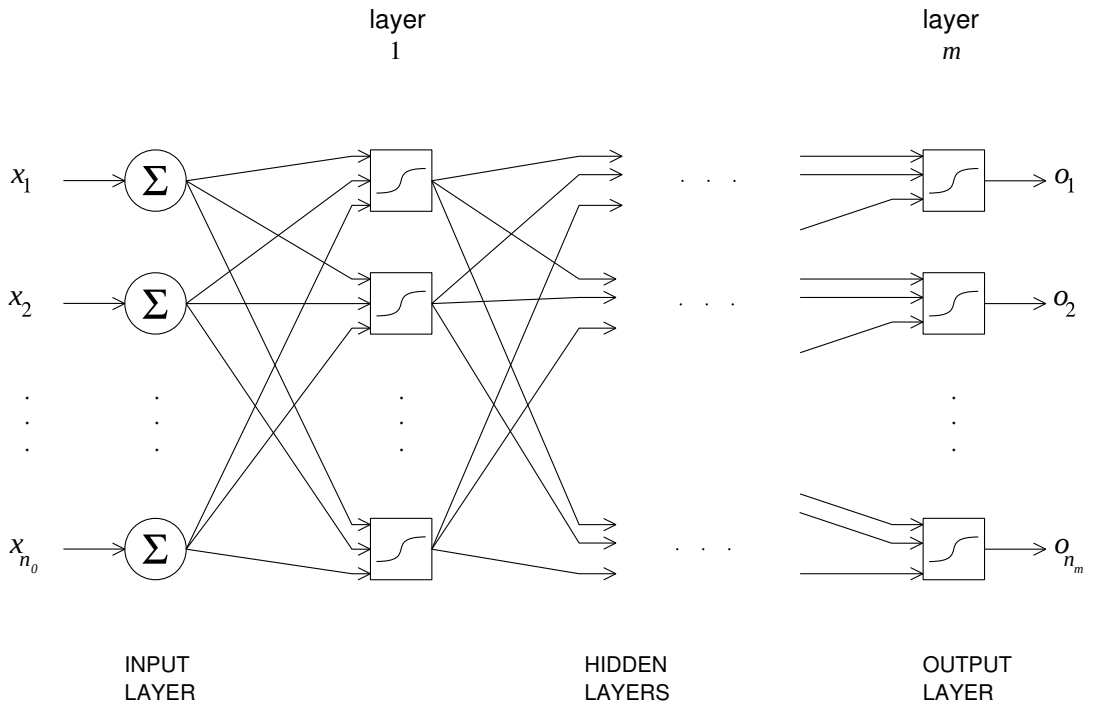


Minsky and Papert

- In 1969 Minsky and Paper published a book that halted research in neural networks for a decade or more.
- Single perceptrons have limited capabilities.
- Networks of many perceptrons are more capable...
- ...but there is no prospect of finding a training algorithm...
- .. however one had already been invented!

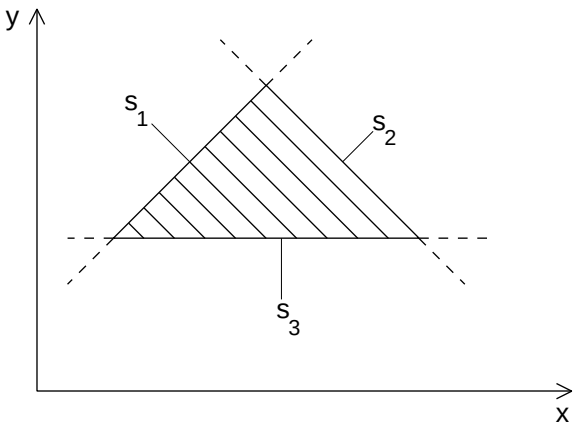


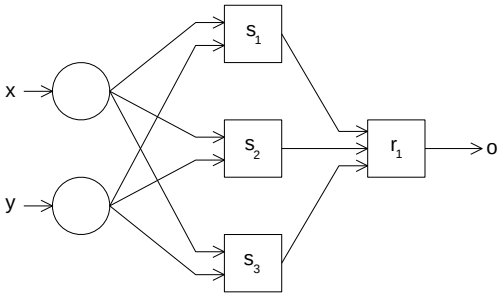
The Multi-Layer Perceptron (MLP) Network



Decision Regions for a Two Layer Network

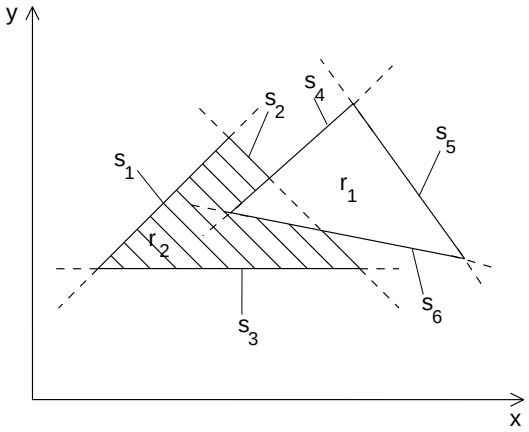
- A two layer network can form a convex decision region.

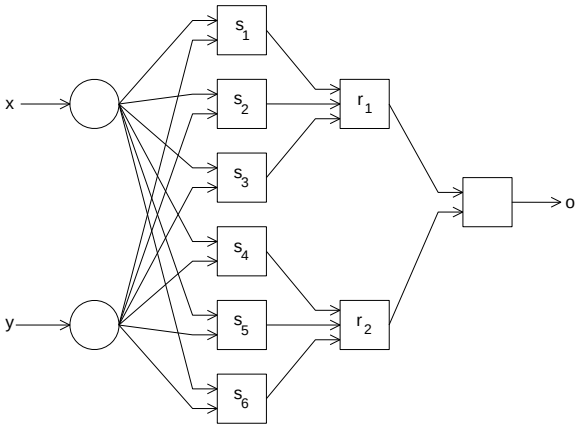




Decision Regions for a Three Layer Network

- A three layer network can form a concave decision region.
- A three layer network can also form disjoint decision regions.





It Can Be Shown That...

Following the approach taken in [3] consider, without loss of generality, a network with two inputs x_1 and x_2, and a single output y. For any given value of x_1, the desired function can be approximated arbitrarily closely by a Fourier decomposition in terms of x_2, of the form

$$y(x_1, x_2) \approx \sum_s A_s(x_1) \cos(s x_2),$$

where the coefficients A_s are functions of x_1, which can in turn be represented by Fourier decomposition

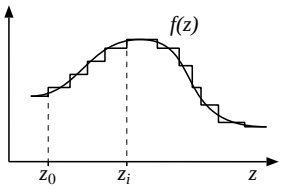
$$y(x_1, x_2) \approx \sum_s \sum_l A_{sl} \cos(l x_1) \cos(s x_2). \qquad (1)$$

Using the identity

$$\cos\alpha \cos\beta = \frac{1}{2}\cos(\alpha+\beta) + \frac{1}{2}\cos(\alpha-\beta)$$

we can write equation (1) as

$$y(x_1, x_2) \approx \textstyle\sum_s \sum_l A_{sl}\{\cos(l x_1 + s x_2) + \cos(l x_1 - s x_2)\}.$$



The cosine function can be approximated arbitrarily closely by a piecewise constant function:

$$f(z) = f_0 \sum_{i=0}^{N} \{f_{i+1} - f_i\} H(z - z_i) \qquad (2)$$

where $H(z)$ is the Heaviside function

$$f(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$

The sigmoidal activation function can approximate the step function arbitrarily well simply by scaling the weights and biases. Any functional mapping can therefore be approximated arbitrarily closely by a weighted sum of scaled and translated sigmoids, i.e. a two-layer perceptron network [10]. See also [5, 9].

Neural Nets are *Universal Approximators*

- Two layer perceptron networks implement *universal approximators*.
- They can approximate any continuous functional (one-to-one or one-to-many) mappings with arbitrary accuracy.
- In general the accuracy improves with increasing numbers of hidden layer neurons.
- A two layer network can form arbitrarily complex decision surfaces.
- It is still worthwhile trying a three-layer network as this might require fewer hidden layer units in total.

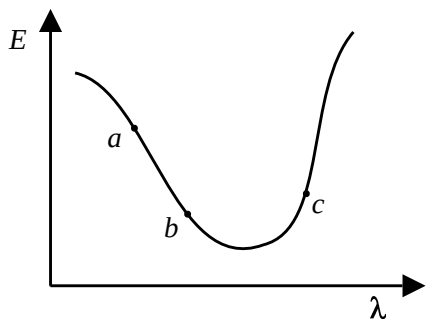
Training Algorithms

- Define an error metric, e.g. the sum-of squares error

$$E = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{c} \{y_k(\boldsymbol{x}^n; \boldsymbol{w}) - t_k^n\}^2 ,$$

where y_k is the k^{th} output of the network, t_k^n is the desired (*target*) output for the n^{th} training pattern and $\boldsymbol{x}^n$ is the corresponding input pattern.

- Adjust the weights, $\boldsymbol{w}$, to minimise the error, E via *gradient descent*.

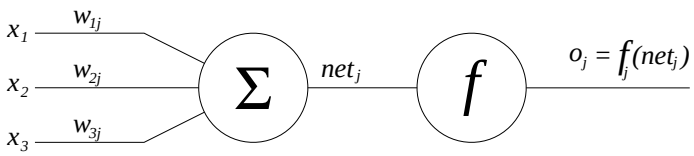


The Perceptron Learning Rule [14]

1. Choose a training pattern (at random or cycle through available patterns).
2. Calculate the output of the Perceptron for this pattern.
3. Update weights according to the rule

$$w_{ji}(t+1) = w_{ji}(t) + \eta(t_j - o_j)f'_k(net_j)x_i.$$

4. Repeat steps 1-3 until convergence.



The Back-Propagation Algorithm

The weights are iteratively updated after each pattern

$$w_{ji}(t+1) = w_{ji}(t) + \Delta w_{ji}.$$

The change made to weight w_{ji} is

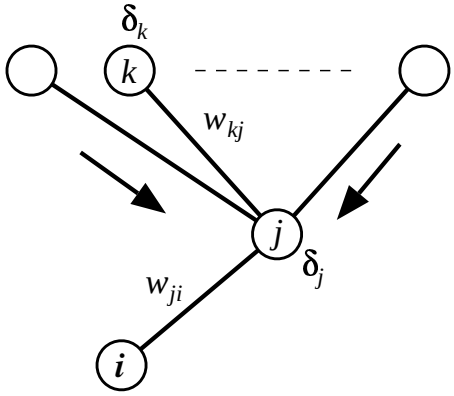
$$\Delta w_{ji} = \eta \delta_j o_i.$$

For output unit u_k,

$$\delta_k = (t_k - o_k)f'_k(net_k).$$

For a hidden unit u_j,

$$\delta_j = f'_j(net_j) \sum_k \delta_k w_{kj}.$$



The Gory Details Are Here...

Consider a feedforward network where the output of neuron i does not depend on neuron j, for all $i > j$. The activation of neuron j for training pattern p is

$$net_{pj} = \sum_i w_{ji} o_{pi}, \tag{3}$$

where w_{ji} is the weight from unit i to unit j and o_{pi} is the output of unit i, for training pattern p. The o_{pj} is a non-decreasing, differentiable function f_j of its activation

$$o_{pj} = f_j(\text{net}_{pj}). \tag{4}$$

We aim to minimise the overall output error of the network, by updating the weights according to a gradient descent procedure. Let E_p be the error of the network for pattern p, e.g.

$$E_p = \frac{1}{2} \sum_j (t_{pj} - o_{pj})^2, \tag{5}$$

where t_{pj} is the target for neuron j for pattern p. Let E be the error of the network, over all training patterns,

$$E = \sum_p E_p.$$

For gradient descent, the update for each weight, $\Delta_{pw_{ji}}$, should be proportional to the partial derivative of E_p with respect to w_{ji}:

$$\Delta_{pw_{ji}} \propto -\frac{\partial E_p}{\partial w_{ji}}.$$

Applying the chain rule

$$\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial net_{pj}} \cdot \frac{\partial net_{pj}}{\partial w_{ji}},$$

and from equation (3)

$$\begin{aligned} \frac{\partial net_{pj}}{\partial w_{ji}} &= \frac{\partial}{\partial w_{ji}} \sum_k w_{jk} o_{pk} = o_{pi}. \\ \delta_{pj} &= -\frac{\partial E_p}{\partial net_{pj}}, \\ \frac{\partial E_p}{\partial w_{ji}} &= \delta_{pj} o_{pi}, \end{aligned} \tag{6}$$

so for gradient descent, the change made to each weight is given by

$$\Delta_{pw_{ji}} = \eta \delta_{pj} o_{pi},$$

And Here...

where η is known as the *learning rate*. Applying the chain rule to equation (6) gives

$$\delta_{pj} = -\frac{\partial E_p}{\partial o_{pj}} \cdot \frac{\partial o_{pj}}{\partial net_{pj}}, \tag{7}$$

and from equation (4)

$$\frac{\partial o_{pj}}{\partial net_{pj}} = f'_j(net_{pj}).$$

For output unit j, from equation (5)

$$-\frac{\partial E_p}{\partial o_{pj}} = -\frac{\partial}{\partial o_{pj}} \frac{1}{2} \sum_k (t_{pk} - o_{pk})^2 = (t_{pj} - o_{pj}),$$

so for an output unit j

$$\delta_{pj} = (t_{pj} - o_{pj}) f'_j(\text{net}_{pj}).$$

For a hidden unit, once again using the chain rule

$$\frac{\partial E_p}{\partial o_{pj}} = \sum_k \frac{\partial E_p}{\partial net_{pk}} \cdot \frac{\partial net_{pk}}{\partial o_{pj}}. \tag{8}$$

Substituting from equation (3) into equation (8)

$$\frac{\partial E_p}{\partial o_{pj}} = \sum_k \frac{\partial E_p}{\partial net_{pk}} \cdot \frac{\partial}{\partial o_{pj}} \sum_i w_{ki} o_{pi}$$

$$\frac{\partial E_p}{\partial o_{pj}} = \sum_k \frac{\partial E_p}{\partial net_{pk}} \cdot w_{kj}, \tag{9}$$

and substituting equation (7) into equation (9) yields

$$\frac{\partial E_p}{\partial o_{pj}} = -\sum_k \delta_{pk} w_{jk}.$$

Therefore, for a hidden unit u_j,

$$\delta_{pj} = -\frac{\partial E_p}{\partial o_{pj}} \cdot \frac{\partial o_{pj}}{\partial net_{pj}} = f'_j(net_{pj}) \sum_k \delta_{pk} w_{jk}.$$

To summarise (see also [15]), the update rule for every weight in the network is given by

$$\Delta_p w_{ji} = \eta \delta_{pj} o_{pi}$$

where for output unit u_j,

$$\delta_{pj} = (t_{pj} - o_{pj}) f'_j(net_{pj})$$

and for a hidden unit u_j,

$$\delta_{pj} = f'_j(net_{pj}) \sum_k \delta_{pk} w_{kj}.$$

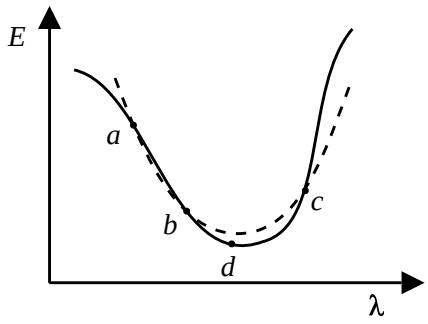
More Practical Training Algorithms

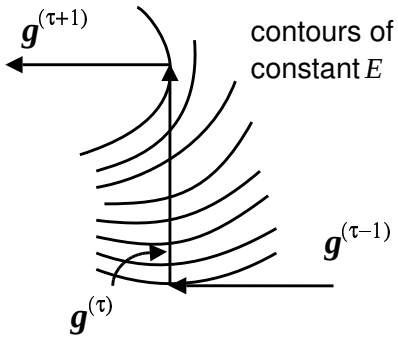
Newton's Method

- Good for training medium sized networks.

Conjugate Gradients

- Good for training very large networks.





- **Levenberg-Marquardt** is also a good method.

Target Coding Schemes

- For two class problems:
  - For a pattern belonging to $\mathcal{C}_1$ the target is 1, for a pattern belonging to $\mathcal{C}_2$ the target is 0.
  - Use a logistic activation function in output layer unit to ensure the output is constrained to lie in the range $[0, 1]$.
- For three or more classes:
  - Use a 1-of-c coding scheme, the target for output unit k is one for a pattern belonging to class $\mathcal{C}_k$ and zero otherwise.
  - Use a softmax activation function in output layer units to ensure the individual outputs are constrained to lie in the range $[0, 1]$ and that they sum to one.

$$y_k = \frac{\exp(a_k)}{\sum_i \exp(a_i)}$$

It Can Be Shown That. . .

The Sum of Squares Error (SSE) metric is given by

$$E = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{c} \left\{ y_k(\boldsymbol{x}^n; \boldsymbol{w}) - t_k^n \right\}^2 .$$

Following the approach taken in [3], in the limit of an infinite number of training patterns

$$E' = \lim_{N \to \infty} \frac{1}{2N} \sum_{n=1}^{N} \sum_{k=1}^{c} \{ y_k(\boldsymbol{x}^n; \boldsymbol{w}) - t_k^n \}^2 .$$

we can replace the summation over training patterns by integration over both t_k and $\boldsymbol{x}$ and abbreviating $y_k(\boldsymbol{x}; \boldsymbol{w})$ to y_k.

$$= \frac{1}{2} \sum_{k=1}^{c} \int \int \{ y_k - t_k \}^2 \, p(t_k | \boldsymbol{x}) dt_k \, p(\boldsymbol{x}) d\boldsymbol{x} .$$

Noting that

$$\{ y_k - t_k \}^2 = \{ y_k - \langle t_k | \boldsymbol{x} \rangle + \langle t_k | \boldsymbol{x} \rangle - t_k \}^2$$

where $\langle t_k | \boldsymbol{x} \rangle$ is the conditional expectation of t_k. then the innermost integral above is

$$\begin{aligned} I = & \int \{ y_k - \langle t_k | \boldsymbol{x} \rangle \}^2 \, p(t_k | \boldsymbol{x}) dt_k \\ & + 2 \int \{ y_k - \langle t_k | \boldsymbol{x} \rangle \} \{ \langle t_k | \boldsymbol{x} \rangle - t_k \} \, p(t_k | \boldsymbol{x}) dt_k \\ & + \int \{ \langle t_k | \boldsymbol{x} \rangle - t_k \}^2 \, p(t_k | \boldsymbol{x}) dt_k \end{aligned}$$

Following considerable algebra . . .

$$\begin{aligned} E' = & \frac{1}{2} \sum_{k=1}^{c} \int \{ y_k - \langle t_k | \boldsymbol{x} \rangle \}^2 \, p(\boldsymbol{x}) d\boldsymbol{x} \\ & + \frac{1}{2} \sum_{k=1}^{c} \int \left\{ \left\langle t_k^2 | \boldsymbol{x} \right\rangle - \langle t_k | \boldsymbol{x} \rangle^2 \right\} p(\boldsymbol{x}) d\boldsymbol{x} \end{aligned}$$

Since the second summation is independent of the output of the network, $y_k(\boldsymbol{x}; \boldsymbol{w})$, the minimum of E' with respect to the parameters of the network, $\boldsymbol{w}$, depends only on the first. The location of the minimum of E' is then given by

$$y_k(\boldsymbol{x}; \boldsymbol{w}) = \langle t_k | \boldsymbol{x} \rangle$$

As we are concerned with pattern classification using a 1-of-c coding scheme, for a pattern belonging to class $\mathcal{C}_l$, $t_k = \delta_{kl}$, and therefore $p(\mathcal{C}_k | \boldsymbol{x}) = \langle t_k | \boldsymbol{x} \rangle$

$$y_k(\boldsymbol{x}; \boldsymbol{w}) = p(\mathcal{C}_k | \boldsymbol{x})$$

This demonstrates that the outputs of the network are valid estimates of *a-posteriori* probabilities of class membership. See also [13, 20].

Neural Networks and *A-Posteriori* Probabilities

Given a sum-of-squares error metric

$$E = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{c} \left\{ y_k(\boldsymbol{x}^n; \boldsymbol{w}) - t_k^n \right\}^2 ,$$

and a 1-of-c coding scheme, such that the network has c output units, one for each class, and the target for the k^{th} output unit, for a pattern belonging to class $\mathcal{C}_l$, is

$$\delta_{kl} = \begin{cases} 1 & k = l \\ 0 & k \neq l \end{cases} ,$$

the outputs of trained classification network are estimates of Bayesian *a-posteriori* probabilities,

$$y_k(\boldsymbol{x}; \boldsymbol{w}) = p(\mathcal{C}_k | \boldsymbol{x}).$$

Setting Rejection Thresholds

- The margin between between the highest and next highest *a-posteriori* probabilities gives a measure of the confidence of the classification.
- If the margin is small we might choose not to classify the pattern at all, but instead reject the pattern.
- In a medical screening test if a confident classification cannot be made it might be better to refer the decision to a human expert rather than to make an unsafe automatic classification.
- Examples:

$p(\mathcal{C}_1)$	$p(\mathcal{C}_2)$	$p(\mathcal{C}_3)$	Result
0.15	0.8	0.05	clearly $\mathcal{C}_2$
0.29	0.35	0.36	reject

Compensating For Prior Probabilities

- Sometimes the prior probabilities in the training set are different from those used in operation.
  - Operational priors may vary.
  - Practical difficulties in collecting data.
  - Computational convenience.
- Compensate using Bayes' rule:

$$p(\mathcal{C}_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\boldsymbol{x})}$$

- Divide outputs by training set priors and multiply by operational priors and then renormalised to sum to one.
- No need to retrain the network!

Minimum Risk Classification

- Some types of misclassification are worse than others.
- Medical screening tests - false negatives are much more serious than false positives.
- Combine the outputs of the network with a *loss matrix*, L that states the cost of each type of classification error.
- This is a called a *minimum risk* classifier as it minimises the expected loss.
- Again, no need to retrain the network!
- However, it is better to incorporate the loss matrix into the error function.

Maximum Likelihood Approaches

- If the network output y^n estimates $p(\mathcal{C}_k|\boldsymbol{x^n})$, the probability of observing the target value for the n^{th} training pattern is,

$$p(t^n|\boldsymbol{x^n}) = (y^n)^{t^n}(1-y^n)^{1-t^n}.$$

- The *likelihood* of observing the entire data set is then

$$\mathcal{L} = \prod_n (y^n)^{t^n}(1-y^n)^{1-t^n}$$

- Train the network so that the outputs maximise $\mathcal{L}$.
- The best classifier the one that is most likely to generate the data observed in the training set.

A Better Error Metric

- Taking the negative logarithm of the likelihood function:

$$E = -\sum_n \{t^n \log y^n - (1 - t^n) \log(1 - y^n)\}$$

- This is called the *cross-entropy* error metric [1].
- Again use 1-of-c coding and logistic or softmax activation functions in the output layer.
- Network outputs remain valid estimates of Bayesian *a-posteriori* probabilities.
- This is an *information theoretic* measure.
- Use in preference to sum-of-squares error as the underlying statistical assumptions are more realistic. Sum-of-squares error is better for regression tasks.

Summary

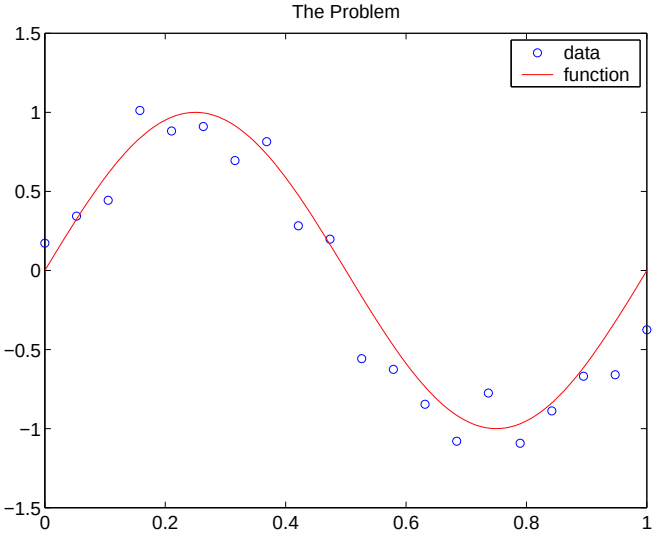
- Neural network pattern classification has strong links with Bayesian statistics:
  - Neural networks are universal approximators.
  - Neural network classifiers approximate *a-posteriori* probabilities.
  - Neural networks implement a form of non-parametric density estimation method.
- Neural networks provide a simple, but theoretically sound framework for finding answers difficult statistical questions.
- Efficient optimization methods exist, used to train networks *reasonably* quickly.

The Bias-Variance Tradeoff

- Training set error is not a good indicator of operational performance (*generalisation*).
- We can divide the generalisation error into two components [7]:
  - **Bias** - error resulting from a failure to reproduce the desired function.
  - **Variance** - error due to modelling noise in the training data.
- Bias and variance are complementary quantities.
- Optimal generalisation results from a compromise between low bias and low variance.
- There are two approaches *regularisation* and *structural stabilisation*

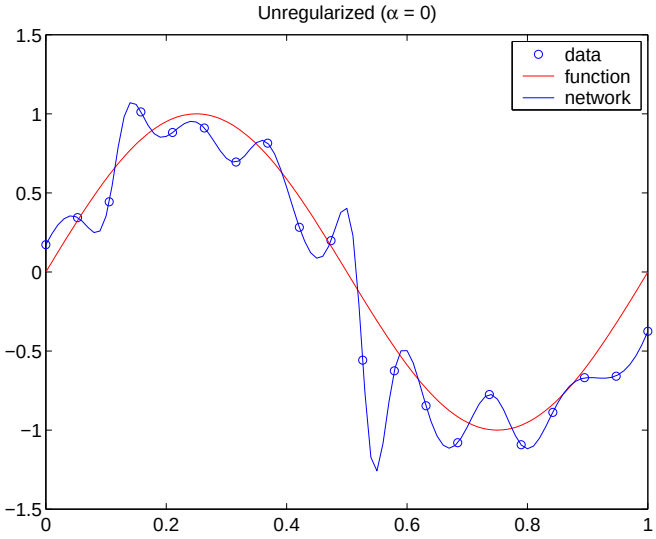
An Illustrative Problem

- Consider the problem of learning a sinusoid corrupted by noise.
- We want to learn the underlying function and ignore the noise.



Unregularised/Unstabilised Solution

- The network has "overfitted" the data.
- Bias is low, but the variance is high so generalisation is poor.



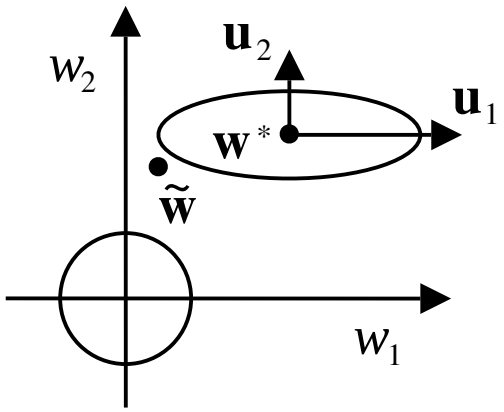
Generalisation

- Neural networks can be too powerful!
- Variance depends on two factors:
  - Network size - a complex mapping requires large numbers of hidden units/weights.
  - Magnitude of the weights - a mapping with a high degree of curvature requires weights with a large magnitude.
- The two approaches to the bias/variance tradeoff are:
  - *Regularisation* - add a term to the error function to penalise complex mappings.
  - *Structural stabilisation* - control the complexity of the network (*growing/pruning* algorithms).

Regularisation

- Add term to the error function to penalise complex mappings [18]

$$E' = E + \alpha\Omega.$$



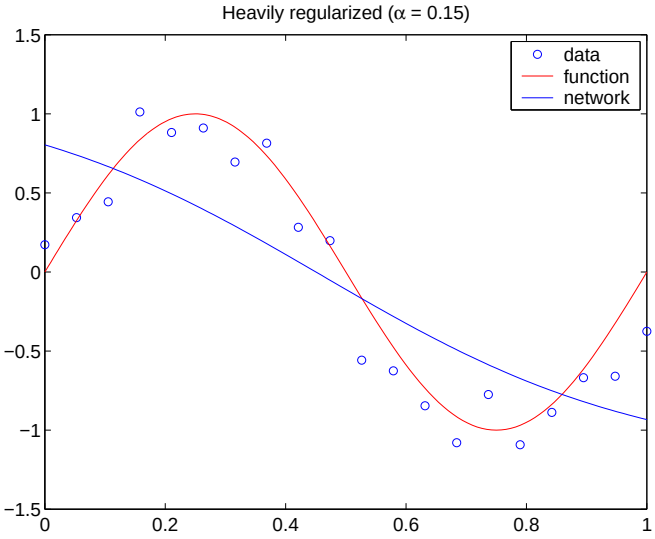
- Example : the *weight decay* regulariser,

$$\Omega = \frac{1}{2}\sum_i w_i^2,$$

  penalises large weights.
- Search for the value of α minimising the error on test data.

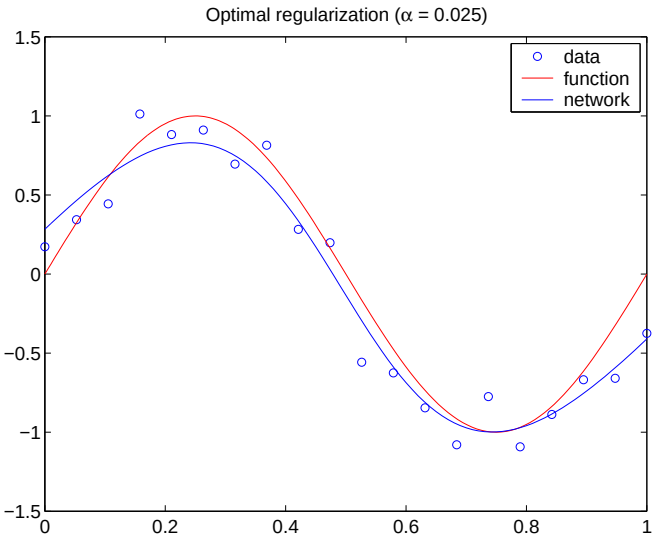
Heavily Regularised Solution

- The network has "underfitted" the data.
- Variance is low, but the bias is high so generalisation is poor.



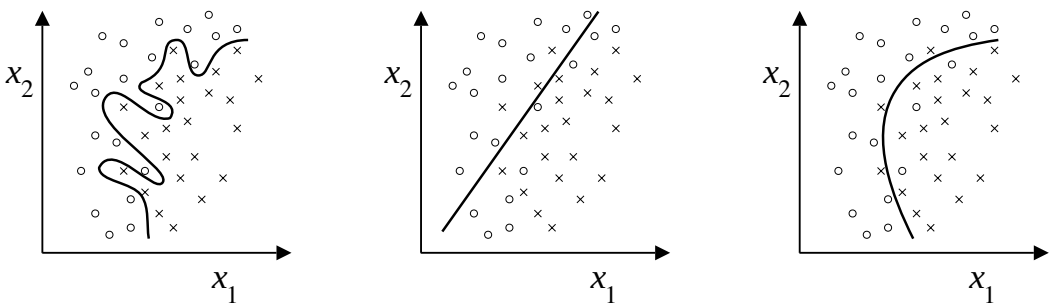
Optimally Regularised Solution

- Network learns underlying function, but ignores noise.
- Compromise between bias and variance optimises generalisation.



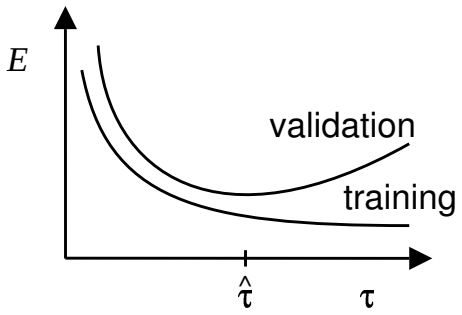
Regularisation in Classification Tasks

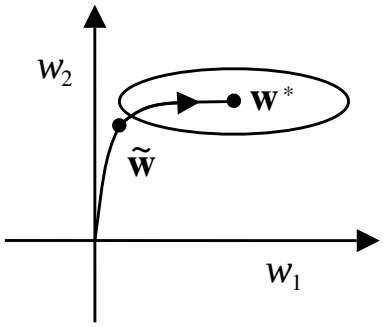
- Classification tasks also require regularisation.
- A small sample from a smooth distribution looks like a small sample from an irregular distribution.
- Smoothing the decision boundaries often improves generalisation (but makes training set error worse!).



Early Stopping

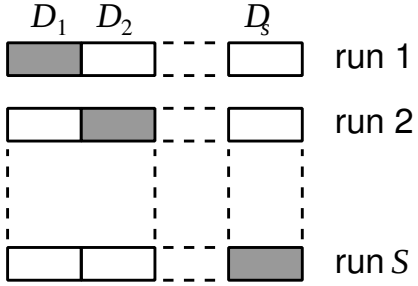
- Divide data into *training*, *validation* and *test* sets, ensuring each is large enough to be representative.
- Training stopped early when validation set error stops improving.
- Report error on statistically pure test set.
- Sometimes mistakenly called crossvalidation [16].





Crossvalidation

- Useful if only a limited amount of data is available [17].
- Divide available data into S groups of approximately equal size.
- Train S networks, network i is tested on D_i and trained on the remainder of the data.
- Average test error over the S networks.



- All data is used as training data and as test data.
- Use an ensemble of S networks [12].

Other Regularisation Methods

- Training with noise [4]:
  - Add a small amount of random noise to the input patterns during training.
  - The noise added to a particular pattern is different each time it is recycled.
  - This blurs the training data, reducing variance.
- Curvature driven smoothing [2]:
  - Define a regulariser that penalises mappings with a high degree of curvature.

$$\Omega = \frac{1}{P} \sum_i \sum_j \sum_p \left[\frac{\partial^2 y_{ip}}{\partial x_{jp}^2} \right]^2$$

  - Mathematically complex.

Structural Stabilisation

- Pruning algorithms (e.g. [8])
  - start with a large network and prune weights, hidden units or even inputs units.
  - Define a *salience* measure that is used to estimate the importance of a weight or a neuron.
- Growing algorithms (e.g. [6])
  - Start with a small network.
  - Add hidden units while generalisation improves.
- Committee methods (e.g. [12])
  - Average outputs of a committee of different classifiers.
  - Averaging process reduces variance.

Philosophical Diversion: Non-Falsifiablity

- We are interested in inductive learning, inferring a general model from specific data.
- I. Kant asks "What is the difference between a case where an inductive step is justified and one where it is not?".
- K. Popper answers "a necessary condition for the justifiability of a theorem is the feasibility of its falsification"!
- What is the difference between astrology and meteorology?
- Can you trust a model generated by a method that can generate an explanation for any phenomenon?
- Regularisation is needed to moderate the power of *all* data driven machine learning algorithms.

Radial Basis Function Neural Networks

- Alternative architecture for artificial neural networks.
  - Very little biological relevance.
  - Very efficient training algorithms.
- The hidden layer consists of *local* nodes
  - Gaussian radial basis function

$$g(\boldsymbol{x}, \boldsymbol{\omega}) = \exp\left\{-\gamma \|\boldsymbol{x} - \boldsymbol{\omega}\|^2\right\}$$

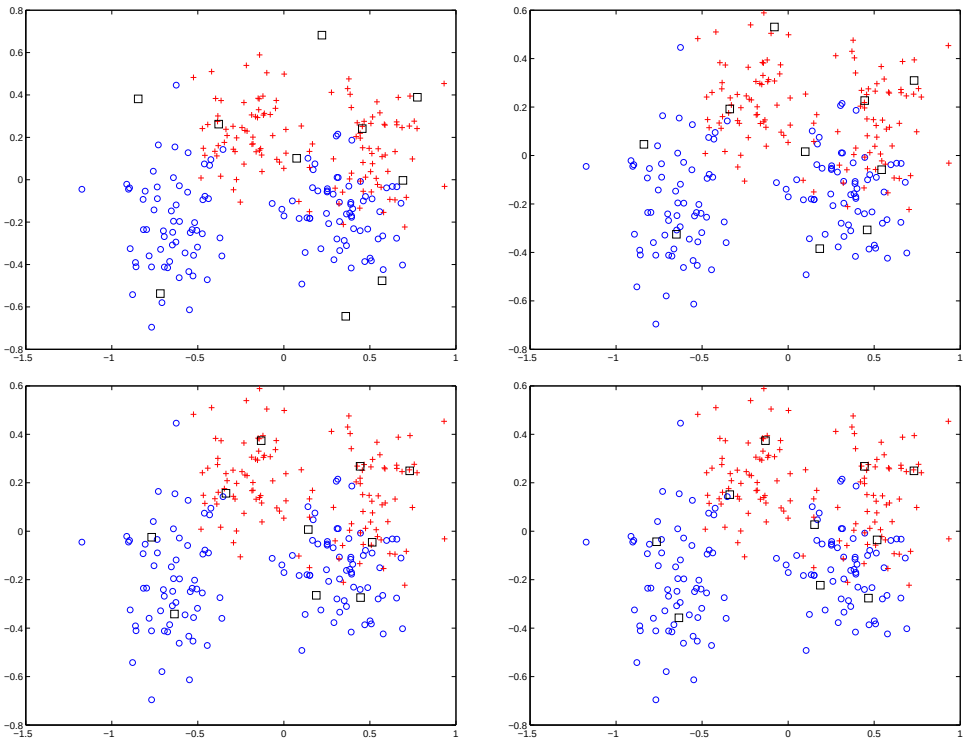
  - Input layer weights, $\boldsymbol{\omega}$ govern positioning of the centers.
  - γ controls the sensitivity of the basis functions.
- Output layer weights determined via linear regression.

$$f(\boldsymbol{x}) = \sum_{i=1}^{k} w_i g(\boldsymbol{x}, \boldsymbol{\omega}_i) + w_0$$

Positioning of Basis Functions

- Select a small set of basis functions that together are representative of the distribution of the data.
- Generally unsupervised
  - The observed responses are not used.
- The k-means clustering algorithm is a simple approach
  1. Position k basis functions at random.
  2. Assign each training pattern to the closest basis function.
  3. Move each basis function to the mean of the the corresponding training data.
  4. If not converged, go to step 2.
- But more complex approaches are also plausible
  - Gaussian mixture model.

Example - Synthetic Dataset



Output Layer Parameters

- Set output layer weights via least-squares regression
  - Use output of basis function as the new input features.
- Let the new *design matrix* be

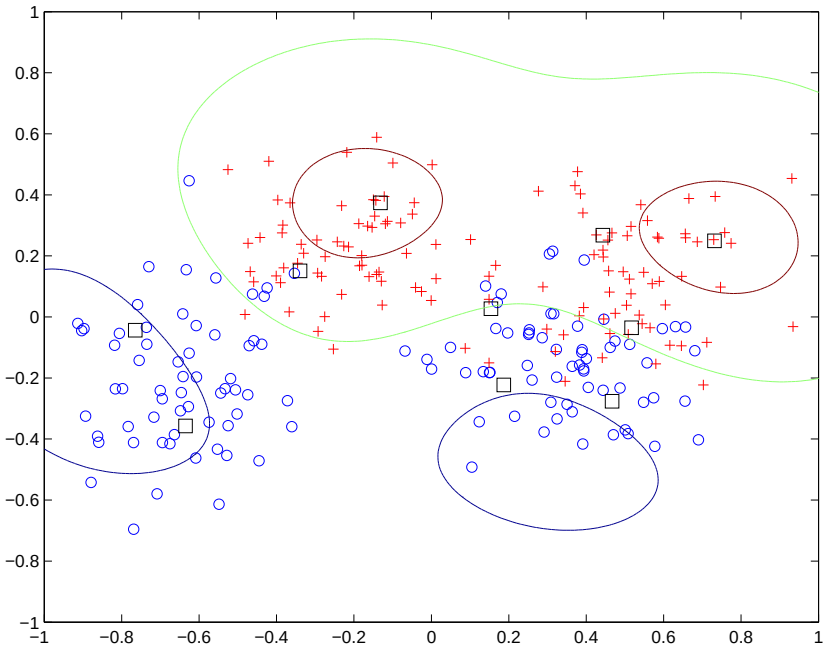
$$\mathbf{\Phi} = \begin{bmatrix} 1 & g(\boldsymbol{x}_1, \boldsymbol{\omega}_1) & \dots & g(\boldsymbol{x}_1, \boldsymbol{\omega}_k) \\ 1 & g(\boldsymbol{x}_2, \boldsymbol{\omega}_1) & \dots & g(\boldsymbol{x}_2, \boldsymbol{\omega}_k) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & g(\boldsymbol{x}_n, \boldsymbol{\omega}_1) & \dots & g(\boldsymbol{x}_n, \boldsymbol{\omega}_k) \end{bmatrix}$$

- Then the output layer parameters are given by

$$\boldsymbol{w} = \left[\mathbf{\Phi}^T \mathbf{\Phi}\right]^{-1} \mathbf{\Phi}^T \boldsymbol{t}$$

- Very efficient, much faster than training an MLP.

Example: Synthetic Benchmark

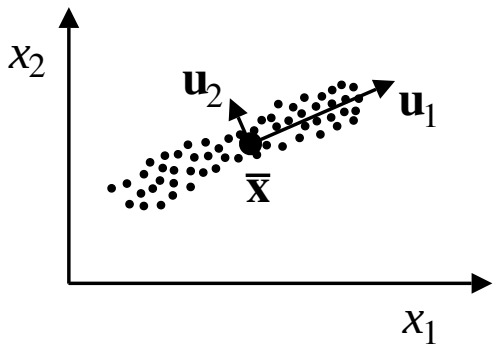


Data Preprocessing

- Standardise all continuous input variables
  - subtract mean and divide by the variance.
  - simplifies weight initialisation.
  - inputs with a high variance are "shouting"!
- The "curse of dimensionality"
  - The number of training patterns required to adequately characterise the distribution of data increases *exponentially* with the dimension of the input vector.
  - It is a very good idea to minimise the number of input variables.
  - Principal component analysis is a useful method here.

Principal Component Analysis

- Principal component analysis rotates the axes (x_1 and x_2) to give new orthogonal axes (u_1 and u_2) that maximise the variance.
- The new inputs to the network are the projections of the data onto the new axes.



- PCA eliminates the correlation between input variables.
- If the variance of a transformed input is low, it can be omitted, reducing the number of input variables.

Applications : Classification/Regression

- Common applications
  - Optical character recognition.
  - Biometrics (face identification, fingerprints etc.)
  - Many medical diagnosis/prognosis applications.
  - Credit scoring / detecting credit card fraud.
  - Stock market prediction.
  - Protein structure prediction.
- Data mining uses
  - Benchmarking more transparent methods.
  - Data exploration.

Applications : Data Cleansing

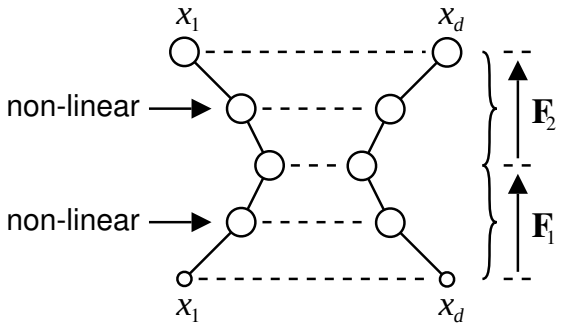
- Data cleansing
  - Create an *auto-associative* network, trained so that the output pattern is the same as the input pattern.
  - Apply suitable regularisation.
  - Filter dataset through network to remove noise.
- Repairing missing data
  - Train a network to estimate the value of one of the variables given the values of the others.
  - Use this estimate to patch up missing data in the database.

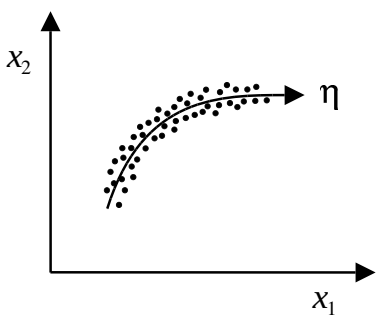
Applications : Rule Extraction

- Analysis of network dynamics:
  - Analysis of weights.
  - Analysis of hidden layer activations.
  - Sensitivity analysis.
- Extraction of rules
  1. Train a neural network.
  2. Pass training data through network.
  3. Use a rule-induction system to reproduce network function.
- First approach time consuming and difficult, the second approach does not really extract the same rules the neural network uses!

Applications : Principal Component Analysis

- A four layer *autoassciative* network is trained to output a pattern identical to the input pattern.
- The activations of the middle hidden layer perform a non-linear form of principal component analysis.
- This is a powerful data reduction technique.





Applications : Visualisation

- A network can be trained to perform a mapping between a high dimensional input space to a low dimension space for visualisation.
- Algorithm (greatly simplified):
  1. Construct a multilayer perceptron (normally with two outputs).
  2. Pick two patterns at random from the training set.
  3. Adjust the weights so that the output of the network are more similar if the two patterns belong to the same class and more dissimilar if they are from different classes.
  4. Repeat steps 2-3 until convergence is reached.

Things We Haven't Covered

We have only discussed on network structure, the MLP & RBF, there are many others.

- Neural networks for combinatorial optimisation (Hopfield).
- Self-organizing maps for unsupervised classification/visualisation.
- Neural network clustering algorithms.
- Support Vector Machines [19].
  - **NN** - minimised training error whilst keeping confidence interval constant.
  - **SVM** - minimising lower bound on generalisation whilst keeping training error constant (e.g. zero).
- Bayesian approaches.

The Bayesian Approach

- Use a weighted average over all possible models!

$$p(t|\boldsymbol{x}, \mathcal{D}) = \int p(t|\boldsymbol{x}, \boldsymbol{w}) p(\boldsymbol{w}|\mathcal{D}) d\boldsymbol{w}$$

- Two approaches:
  - Gaussian approximation to $p(\boldsymbol{w}|\mathcal{D})$.
  - Markov Chain Monte Carlo (MCMC) methods.
- The Bayesian approach can be used to:
  - determine the importance of inputs (ARD).
  - generate confidence intervals.
  - automatically select regularisation parameters.
  - implement active learning.

Guidelines Neural Network Pattern Recognition

- Don't:
  - Assume training set error is a good indicator of operational performance.
  - Assume that the data is non-linearly separable.
  - Just throw raw data at a neural net and hope!
- Do:
  - Get a good book (e.g. Bishop [3]).
  - Investigate the data first!
  - Use some form of regularisation and pre-process data.
  - Evaluate performance on unseen data (e.g. crossvalidation).
  - Try other simple techniques first.

Conclusions

- Neural networks have strong foundation in statistical theory.
- Modern optimisation procedures make network training practical.
- Regularisation or structural stabilisation is required to optimize generalisation.
- Preprocessing the data is also important in improving generalisation.
- Very good for classification, regression, data processing and visualisation application.
- Not so useful where an explanation of the networks' function is required.

References


E. B. Baum and F. Wilczek.
Supervised learning of probability distributions by neural networks.
In Anderson D. Z., editor, *Neural Information Processing Systems*, pages 52–61. American Institute of Physics, New York, 1988.


C. M. Bishop.
Curvature-driven smoothing: a learning algorithm for feedforward networks.
I.E.E.E. Transactions on Neural Networks, 4(5):882–884, 1993.


C. M. Bishop.
Neural Networks for Pattern Recognition.
Oxford University Press, 1995.


C. M. Bishop.
Training with noise is equivalent to Tikhonov regularization.
Neural Computation, 7(1):108–116, 1995.


G. Cybenko.
Approximation by superposition of a sigmoidal function.
Mathematics of Control, Signals and Systems, 2:304–314, 1989.


S. E. Fahlman and C. Lebiere.
The cascade-correlation learning architecture.
In D. S. Touretzky, editor, *Advances in Neural Information Processing*, volume 2, pages 524–532. Morgan Kaufmann, San Mateo, CA, 1990.

Proceedings of the IEEE, 78(10):1586–1589, 1990.


W. S. McCulloch and W. Pitts.
The logical calculus of the ideas immanant in nervous activity.
Bulletin of Mathematical Biophysics, 5:115–133, 1943.


M. P. Perrone and L. N. Cooper.
When networks disagree: ensemble methods for hybrid neural networks, 1995.


R. Rojas.
A short proof of the posterior probability property of classifier neural networks.
Neural Computation, 8(1):41–43, 1996.


F. Rosenblatt.
Principles of neurodynamics: Perceptrons and the theory of brain brain mechanisms.
Spartan, Washington DC, 1962.


D. E. Rummelhart, G. E. Hinton, and R. J. Williams.
Learning internal representations by error propagation.
In D. E. Rummelhart, J. L. McClelland, and the PDP Reseach Group, editors, *Parallel distributed processing: Explorations in the microstructure of cognition*, volume 1: Foundations, pages 318–362. 1986.


W. S. Sarle.
Stopped training and other remedies for