Multi Layer Networks

Examples and applications

C.S.Kumar

Practical Considerations for Back-Propagation Learning

Most of the practical considerations necessary for general Back-Propagation learning were already covered when we talked about training single layer Perceptrons:

1. Do we need to pre-process the training data? If so, how?
2. How do we choose the initial weights from which we start the training?
3. How do we choose an appropriate learning rate η?
4. Should we change the weights after each training pattern, or after the whole set?
5. Are some activation/transfer functions better than others?
6. How can we avoid flat spots in the error function?
7. How can we avoid local minima in the error function?
8. How do we know when we should stop the training?

However, there are also two important issues that were not covered before:

9. How many hidden units do we need?
10. Should we have different learning rates for the different layers?

How Many Hidden Units?

The best number of hidden units depends in a complex way on many factors, including:

1. The number of training patterns
2. The numbers of input and output units
3. The amount of noise in the training data
4. The complexity of the function or classification to be learned
5. The type of hidden unit activation function
6. The training algorithm

Too few hidden units will generally leave high training and generalisation errors due to under-fitting. Too many hidden units will result in low training errors, but will make the training unnecessarily slow, and will result in poor generalisation unless some other technique (such as *regularisation*) is used to prevent over-fitting.

Different Learning Rates for Different Layers?

A network as a whole will usually learn most efficiently if all its neurons are learning at roughly the same speed. So maybe different parts of the network should have different learning rates η. There are a number of factors that may affect the choices:

1. The later network layers (nearer the outputs) will tend to have larger local gradients (*deltas*) than the earlier layers (nearer the inputs).
2. The activations of units with many connections feeding into or out of them tend to change faster than units with fewer connections.
3. Activations required for linear units will be different for Sigmoidal units.
4. There is empirical evidence that it helps to have different learning rates η for the thresholds/biases compared with the real connection weights.

In practice, it is often quicker to just use the same rates η for all the weights and thresholds, rather than spending time trying to work out appropriate differences. A very powerful approach is to use evolutionary strategies to determine good learning rates.

A Statistical View of the Training Data

Suppose we have a ***training data set*** D for our neural network:

$$D = \{ x_i^p, y^p : i = 1 \ldots ninputs, p = 1 \ldots npatterns \}$$

This consists of an output y^p for each input pattern x_i^p. To keep the notation simple we shall assume we only have one output unit – the extension to many outputs is obvious.

Generally, the training data will be generated by some actual function $g(x_i)$ plus random noise ε^p (which may, for example, be due to data gathering errors), so

$$y^p = g(x_i^p) + \varepsilon^p$$

We call this a ***regressive model*** of the data. We can define a statistical expectation operator $\mathcal{E}$ that averages over all possible training patterns, so

$$g(x_i) = \mathcal{E}[y \mid x_i]$$

We say that the regression function $g(x_i)$ is the *conditional mean of the model output y given the inputs* x_i.

A Statistical View of Network Training

The neural network training problem is to construct an output function $net(x_i, W, D)$ of the network weights $W = \{w_{ij}^{(n)}\}$, based on the data D, that best approximates the regression model, i.e. the underlying function $g(x_i)$.

We have seen how to train a network by minimising the sum-squared error cost function:

$$E(W) = \tfrac{1}{2} \sum_{p \in D} \left(y^p - net(x_i^p, W, D) \right)^2$$

with respect to the network weights $W = \{w_{ij}^{(n)}\}$. However, we have also observed that, to get good generalisation, we do not necessarily want to achieve that minimum. What we really want to do is minimise the difference between the network's outputs $net(x_i, W, D)$ and the underlying function $g(x_i) = \mathcal{E}[y \mid x_i]$.

The natural sum-squared error function, i.e. $\left(\mathcal{E}[y \mid x_i] - net(x_i, W, D) \right)^2$, depends on the specific training set D, and we really want our network training regime to produce good results averaged over all possible noisy training sets.

Bias and Variance

If we define the expectation or average operator $\mathcal{E}_D$ which takes the ***ensemble average*** over all possible training sets D, then some rather messy algebra allows us to show that:

$$\mathcal{E}_D\left[\left(\mathcal{E}[y \mid x_i] - net(x_i, W, D)\right)^2\right]$$

$$= \left(\mathcal{E}_D\left[net(x_i, W, D)\right] - \mathcal{E}[y \mid x_i]\right)^2 + \mathcal{E}_D\left[\left(net(x_i, W, D) - \mathcal{E}_D\left[net(x_i, W, D)\right]\right)^2\right]$$

$$= \quad (\text{bias})^2 \quad + \quad (\text{variance})$$

This error function consists of two positive components:

(bias)^2 the difference between the average network output $\mathcal{E}_D[net(x_i, W, D)]$ and the regression function $g(x_i) = \mathcal{E}[y \mid x_i]$. This can be viewed as the ***approximation error***.

(variance) the variance of the approximating function $net(x_i, W, D)$ over all the training sets D. It represents the ***sensitivity*** of the results on the particular choice of data D.

In practice there will always be a trade-off between these two error components.

The Extreme Cases of Bias and Variance

We can best understand the concepts of ***bias*** and ***variance*** by considering the two extreme cases of what the network might learn.

Suppose our network is lazy and just generates the same constant output whatever training data we give it, i.e. $net(x_i, W, D) = c$. In this case the variance term will be zero, but the bias will be large, because the network has made no attempt to fit the data.

Suppose our network is very hard working and makes sure that it fits every data point:

$$\mathcal{E}_D\left[net(x_i, W, D)\right] = \mathcal{E}_D\left[y(x_i)\right] = \mathcal{E}_D\left[g(x_i) + \varepsilon\right] = \mathcal{E}[y \mid x_i]$$

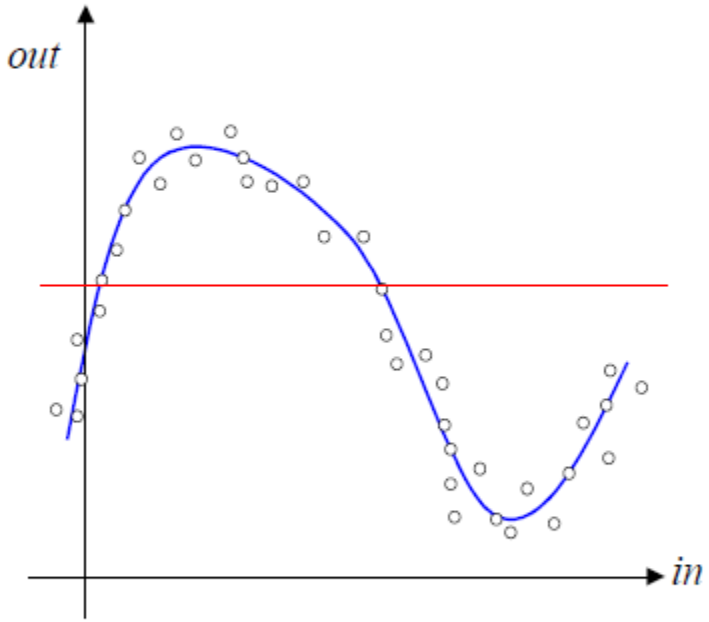
so the bias is zero, but the variance is:

$$\mathcal{E}_D\left[\left(net(x_i, W, D) - \mathcal{E}_D\left[net(x_i, W, D)\right]\right)^2\right] = \mathcal{E}_D\left[\left(g(x_i) + \varepsilon - \mathcal{E}_D\left[g(x_i) + \varepsilon\right]\right)^2\right] = \mathcal{E}_D\left[(\varepsilon)^2\right]$$

i.e. the variance of the noise on the data, which could be substantial.

Examples of the Two Extreme Cases

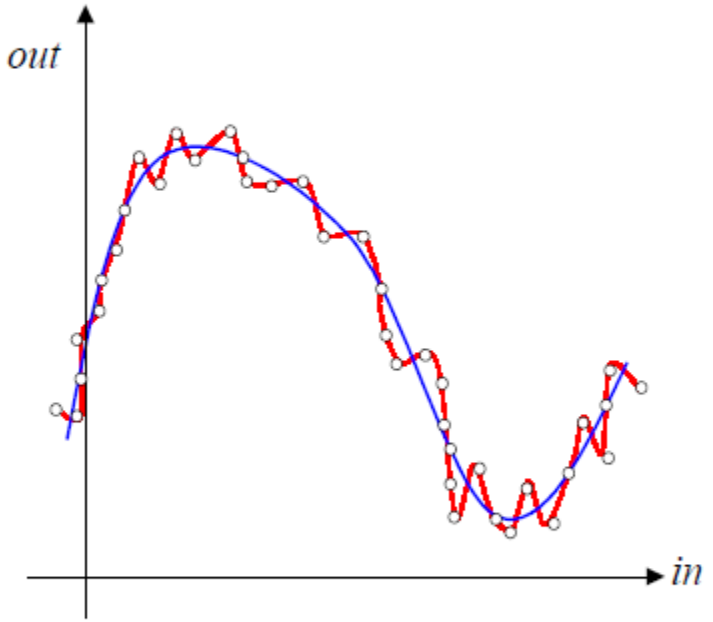
The lazy and hard-working networks approach our function approximation as follows:



Ignore the data ⇒

Big approximation errors (high bias)

No variation between data sets (no variance)



Get every data point ⇒

No approximation errors (zero bias)

Variation between data sets (high variance)

Under-fitting, Over-fitting and the Bias/Variance Trade-off

If our network is to generalize well to new data, we obviously need it to generate a good approximation to the underlying function $g(x_i) = \mathbb{E}[y \mid x_i]$, and we have seen that to do this we must minimise the sum of the bias and variance terms. There will clearly have to be a ***trade-off*** between minimising the bias and minimising the variance.

A network which is too closely fitted to the data will tend to have a large variance and hence give a large expected generalization error. We then say that ***over-fitting*** of the training data has occurred.

We can easily decrease the variance by smoothing the network outputs, but if this is taken too far, then the bias becomes large, and the expected generalization error is large again. We then say that ***under-fitting*** of the training data has occurred.

This trade-off between bias and variance plays a crucial role in the application of neural network techniques to practical applications.

Preventing Under-fitting and Over-fitting

To ***prevent under-fitting*** we need to make sure that:

1. The network has enough hidden units to represent to required mappings.
2. We train the network for long enough so that the sum squared error cost function is sufficiently minimised.

To ***prevent over-fitting*** we can:

1. Stop the training early – before it has had time to learn the training data too well.
2. Restrict the number of adjustable parameters the network has – e.g. by reducing the number of hidden units, or by forcing connections to share the same weight values.
3. Add some form of *regularization* term to the error function to encourage smoother network mappings.
4. Add noise to the training patterns to smear out the data points.

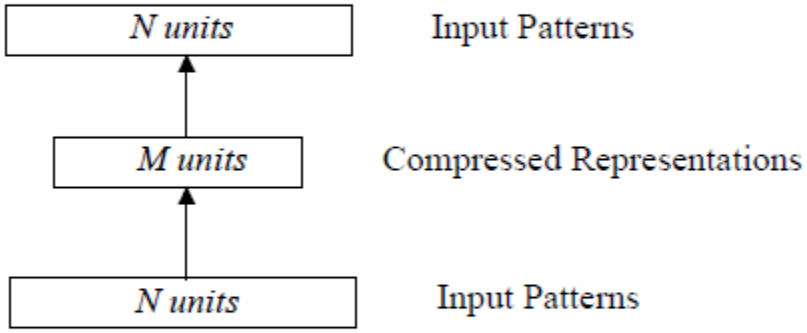
Real World Applications

The real world applications of feed-forward networks are endless. Some well known ones that get mentioned in the recommended text books are:

1. Airline Marketing Tactician (Beale & Jackson, Sect. 4.13.2)
2. Backgammon (Hertz et al., Sect. 6.3)
3. Data Compression – PCA (Hertz et al., Sect. 6.3; Bishop, Sect. 8.6) •
4. Driving – ALVINN (Hertz et al., Sect. 6.3) •
5. ECG Noise Filtering (Beale & Jackson, Sect. 4.13.3)
6. Financial Prediction (Beale & Jackson, Sect. 4.13.3; Gurney, Sect. 6.11.2) •
7. Hand-written Character Recognition (Hertz et al., Sect. 6.3; Fausett, Sect. 7.4) •
8. Pattern Recognition/Computer Vision (Beale & Jackson, Sect. 4.13.5) •
9. Protein Secondary Structure (Hertz et al., Sect. 6.3)
10. Psychiatric Patient Length of Stay (Gurney, Sect. 6.11.1)
11. Sonar Target Recognition (Hertz et al., Sect. 6.3)
12. Speech Recognition (Hertz et al., Sect. 6.3)

Data Compression - PCA

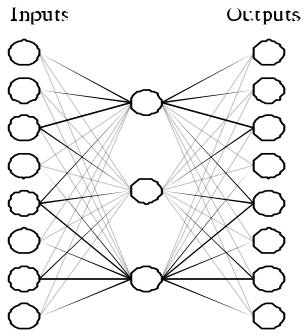
An ***auto-associator*** network is one that has the same outputs as inputs. If in this case we make the number of hidden units M smaller than the number of inputs/outputs N, we will have clearly compressed the data from N dimensions down to M dimensions.



Such data compression networks have many applications where data transmission rates or memory requirements need optimising, such as in image compression. They clearly work by removing the redundancy that exists in the data. It can be shown that the hidden unit representation spans the M ***principal components*** of the original N dimensional data, so standard PCA considerations apply (Bourland & Kamp, 1988).

Learning Hidden Layer Representations

8-3-8 Binary Encoder-Decoder



8 inputs

8 outputs

3 hidden

A target function:

Input		Output
10000000	→	10000000
01000000	→	01000000
00100000	→	00100000
00010000	→	00010000
00001000	→	00001000
00000100	→	00000100
00000010	→	00000010
00000001	→	00000001

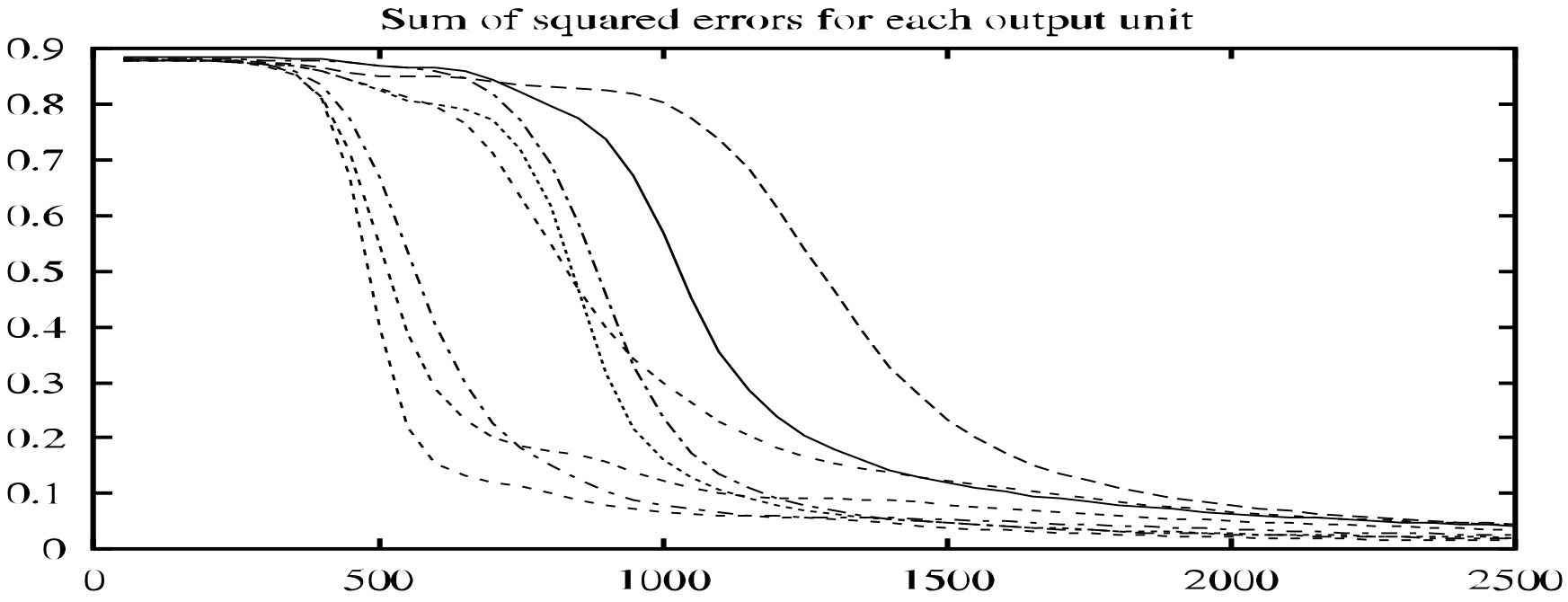
Can this be learned??

Hidden values

.89 .04 .08
.01 .11 .88
.01 .97 .27
.99 .97 .71
.03 .05 .02
.22 .99 .99
.80 .01 .98
.60 .94 .01

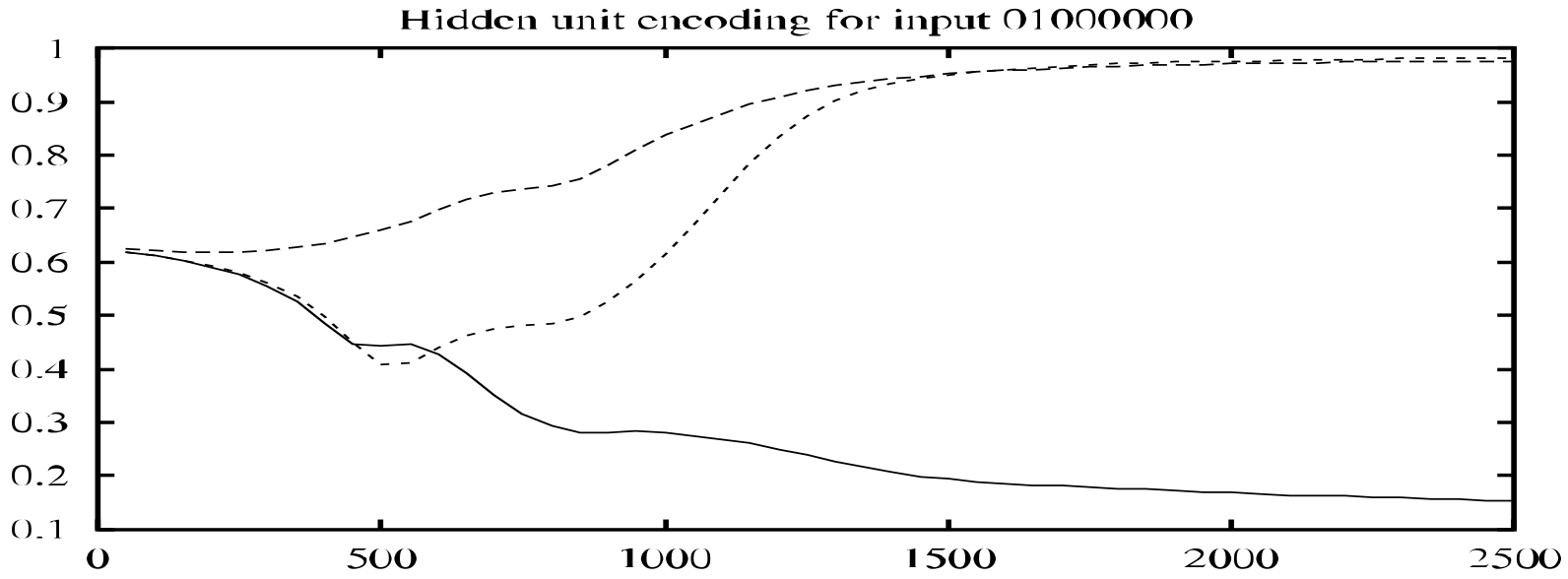
Sum of Squared Errors for the Output Units

Training



Hidden Unit Encoding for Input 0100000

Training

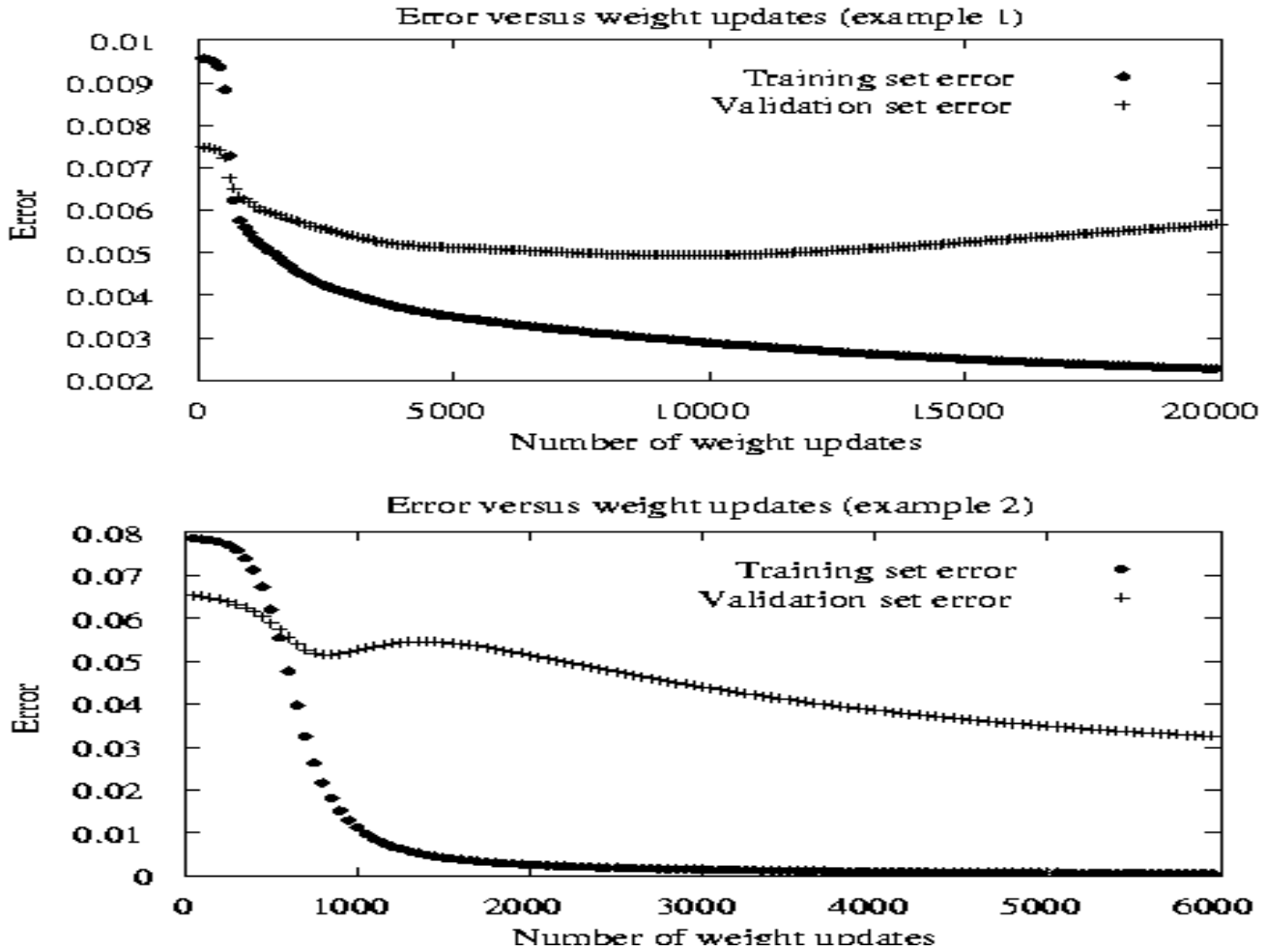


Generalization and Overfitting

- Continuing training until the training error falls below some predetermined threshold is a poor strategy since BP is susceptible to overfitting.
  - Need to measure the generalization accuracy over a validation set (distinct from the training set).

- Two different types of overffiting
  - Generalization error first decreases, then increases, even the training error continues to decrease.
  - Generalization error decreases, then increases, then decreases again, while the training error continues to decreases.

Two Kinds of Overfitting Phenomena



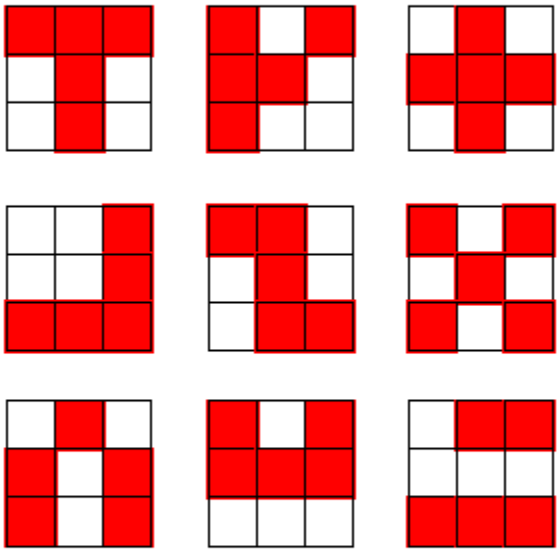
Techniques for Overcoming the Overfitting Problem

- Weight decay
  - Decrease each weight by some small factor during each iteration.
  - This is equivalent to modifying the definition of E to include a penalty term corresponding to the total magnitude of the network weights.
  - The motivation for the approach is to keep weight values small, to bias learning against complex decision surfaces.

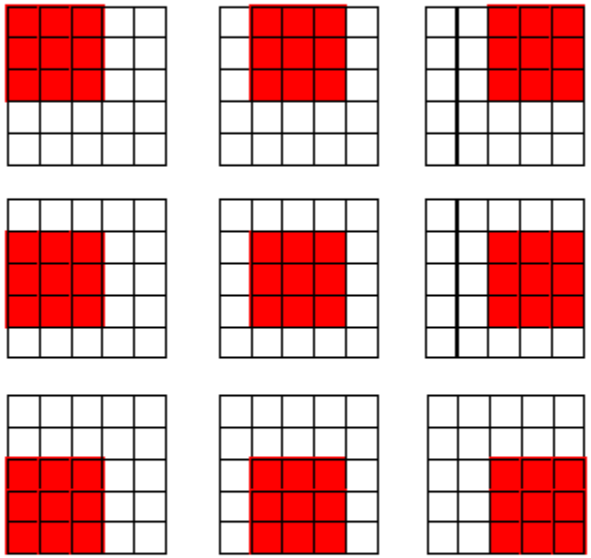
- *k*-fold cross-validation
  - Cross validation is performed *k* different times, each time using a different partitioning of the data into training and validation sets
  - The result are averaged after *k* times cross validation.

Simplified "What-Where" Training Data

"What" = Nine 3 × 3 Patterns



"Where" = Nine Positions in 5 × 5 Retina



$9 \times 9 = 81$ Training Patterns in total

e.g. 01110 00100 00100 00000 00000 — Input ($5 \times 5 = 25$ units)

100000000 — What (9 units)

010000000 — Where (9 units)

Driving – ALVINN

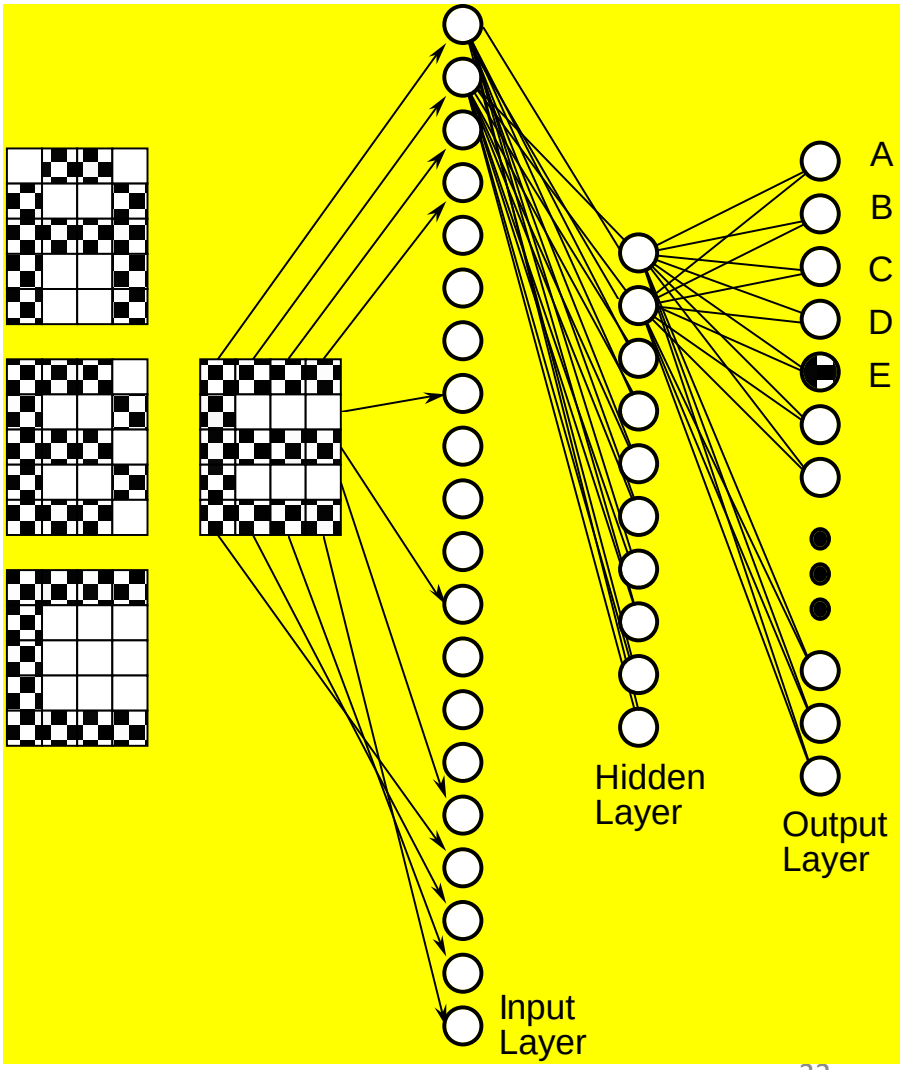
Pomerleau (1989) constructed a neural network controller ALVINN for driving a car on a winding road. The inputs were a 30×32 pixel image from a video camera, and an 8×32 image from a range finder. These were fed into a hidden layer of 29 units, and from there to a line of 45 output units corresponding to direction to drive.

The network was originally trained using back-propagation on 1200 simulated road images. After about 40 epochs the network could drive at about 5mph – the speed being limited by the speed of the computer that the neural network was running on.

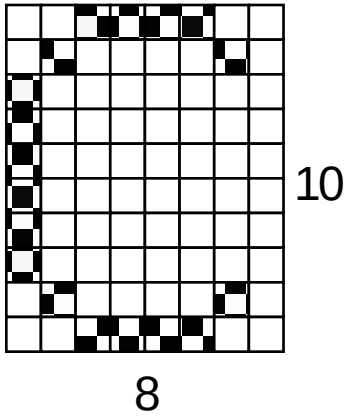
In a later study the network learnt by watching how a human steered, and by using additional views of what the road would look like at positions slightly off course. After about three minutes of training, ALVINN was able to take over and continue to drive. ALVINN has successfully driven at speeds up to 70mph and for distances of over 90 miles on a public highway north of Pittsburgh. (Apparently, actually being inside the vehicle during the test drive was a big incentive for the researchers to develop a good neural network!)

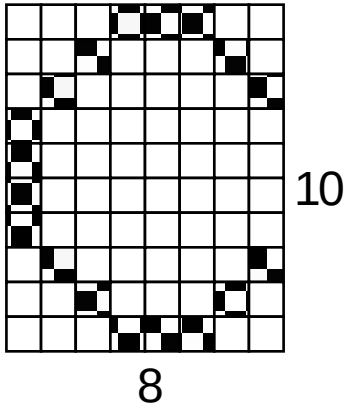
Neural network for OCR

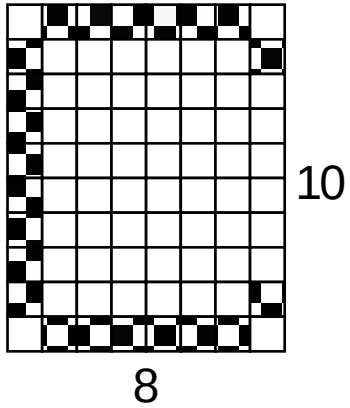
- feedforward network
- trained using Back-propagation



OCR for 8x10 characters







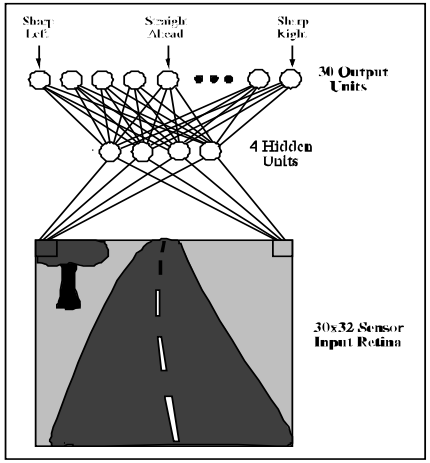
- NN are able to generalise
- learning involves generating a partitioning of the input space
- for single layer network input space must be linearly separable
- what is the dimension of this input space?
- how many points in the input space?
- this network is binary(uses binary values)
- networks may also be continuous

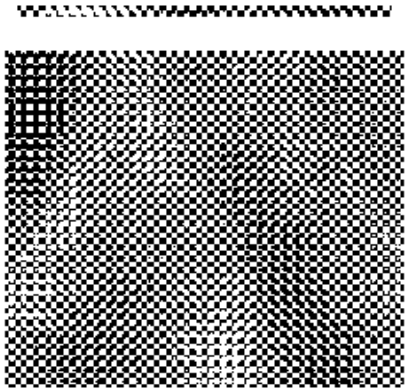
Engine management

- The behaviour of a car engine is influenced by a large number of parameters
  - temperature at various points
  - fuel/air mixture
  - lubricant viscosity.
- Major companies have used neural networks to dynamically tune an engine depending on current settings.

ALVINN

Drives 70 mph on a public highway





30 outputs for steering

4 hidden units

30x32 pixels as inputs

30x32 weights into one out of four hidden unit

Driving – ALVINN

Pomerleau (1989) constructed a neural network controller ALVINN for driving a car on a winding road. The inputs were a 30×32 pixel image from a video camera, and an 8×32 image from a range finder. These were fed into a hidden layer of 29 units, and from there to a line of 45 output units corresponding to direction to drive.

The network was originally trained using back-propagation on 1200 simulated road images. After about 40 epochs the network could drive at about 5mph – the speed being limited by the speed of the computer that the neural network was running on.

In a later study the network learnt by watching how a human steered, and by using additional views of what the road would look like at positions slightly off course. After about three minutes of training, ALVINN was able to take over and continue to drive. ALVINN has successfully driven at speeds up to 70mph and for distances of over 90 miles on a public highway north of Pittsburgh. (Apparently, actually being inside the vehicle during the test drive was a big incentive for the researchers to develop a good neural network!)

Signature recognition

- Each person's signature is different.
- There are structural similarities which are difficult to quantify.
- One company has manufactured a machine which recognizes signatures to within a high level of accuracy.
  - Considers speed in addition to gross shape.
  - Makes forgery even more difficult.

Sonar target recognition

- Distinguish mines from rocks on sea-bed
- The neural network is provided with a large number of parameters which are extracted from the sonar signal.
- The training set consists of sets of signals from rocks and mines.

Stock market prediction

- “Technical trading” refers to trading based solely on known statistical parameters; e.g. previous price
- Neural networks have been used to attempt to predict changes in prices.
- Difficult to assess success since companies using these techniques are reluctant to disclose information.

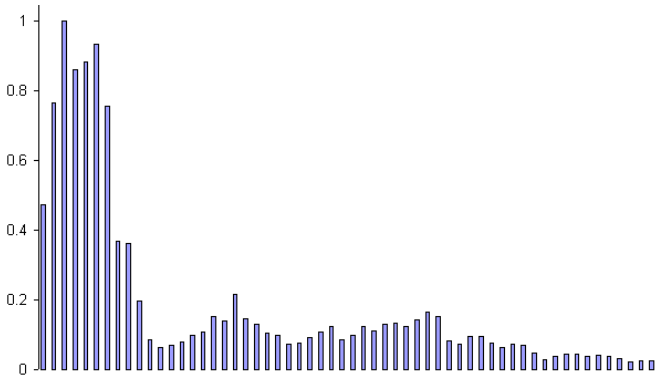
Mortgage assessment

- Assess risk of lending to an individual.
- Difficult to decide on marginal cases.
- Neural networks have been trained to make decisions, based upon the opinions of expert underwriters.
- Neural network produced a 12% reduction in delinquencies compared with human experts.

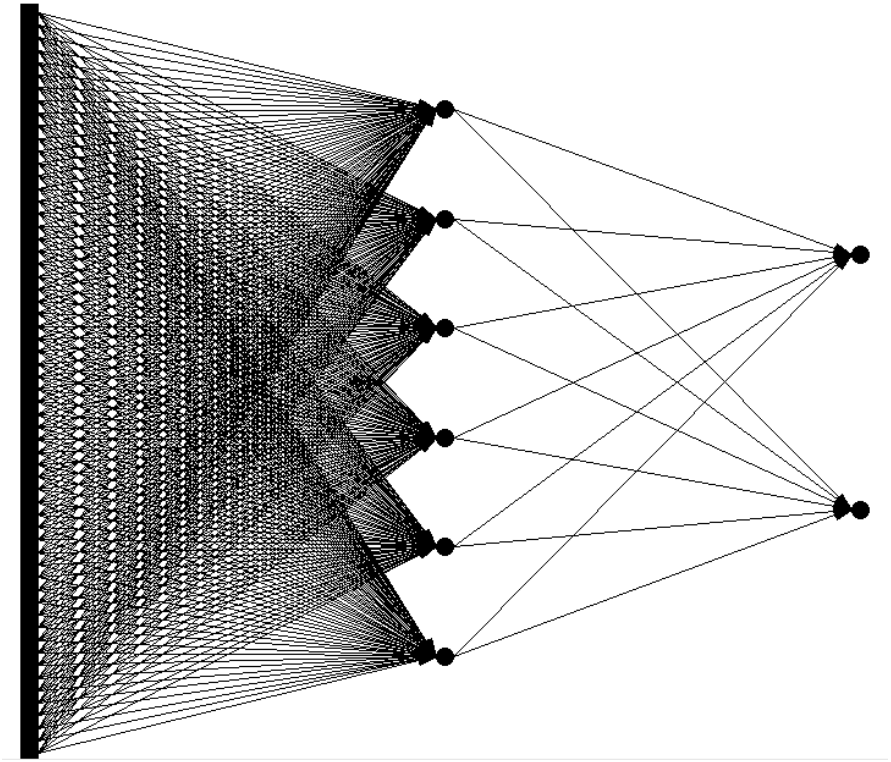
Example: Voice Recognition

- Task: Learn to discriminate between two different voices saying “Hello”

- Data
  - Sources
    - Steve Simpson
    - David Raubenheimer
  - Format
    - Frequency distribution (60 bins)
    - Analogy: cochlea

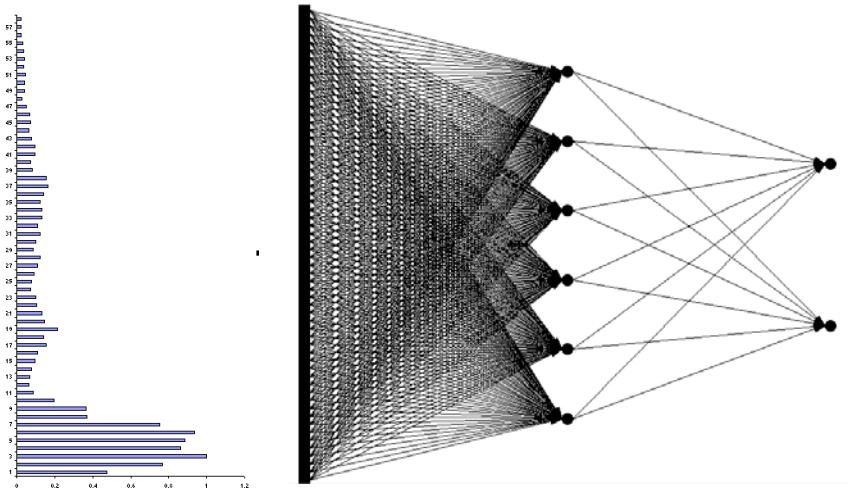


- Network architecture
  - Feed forward network
    - 60 input (one for each frequency bin)
    - 6 hidden
    - 2 output (0-1 for “Steve”, 1-0 for “David”)

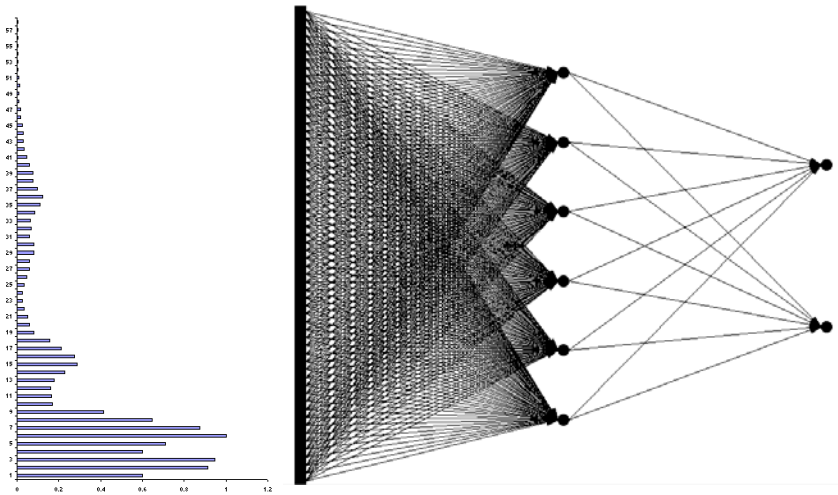


- Presenting the data

Steve

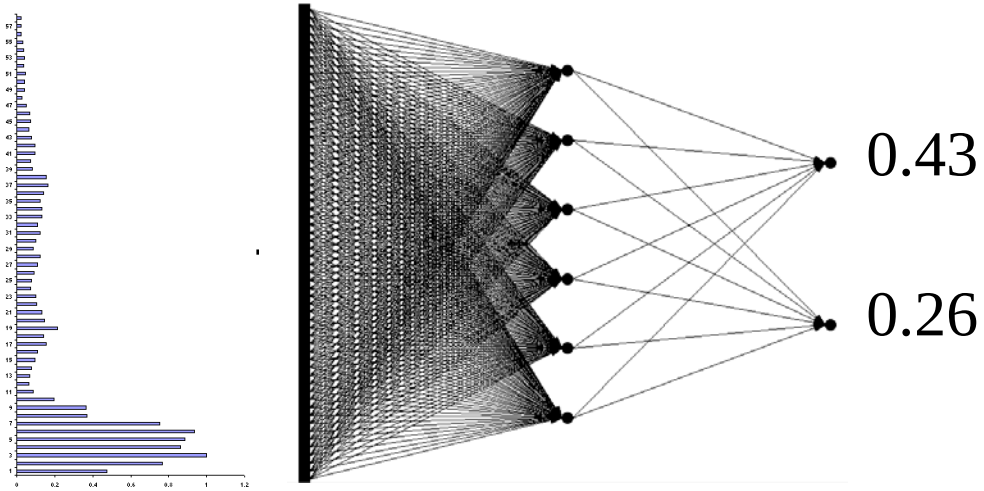


David

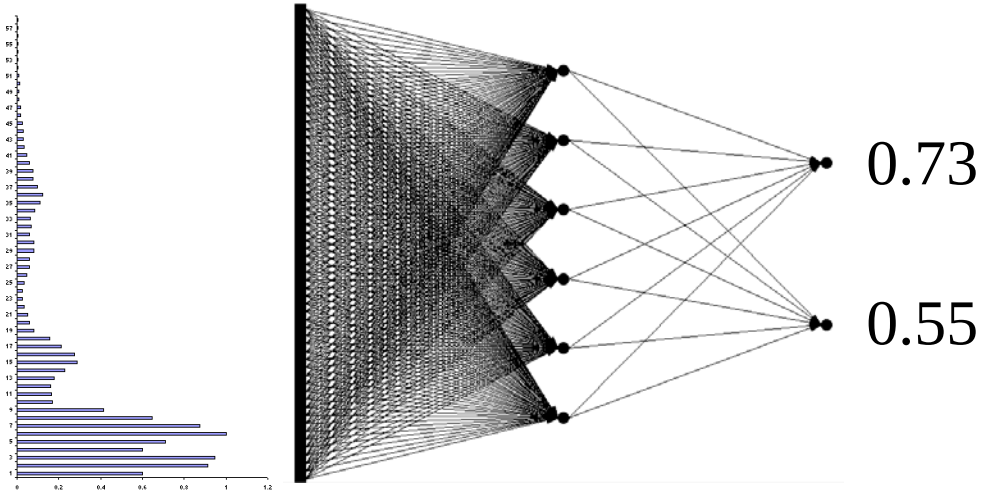


- Presenting the data (untrained network)

Steve

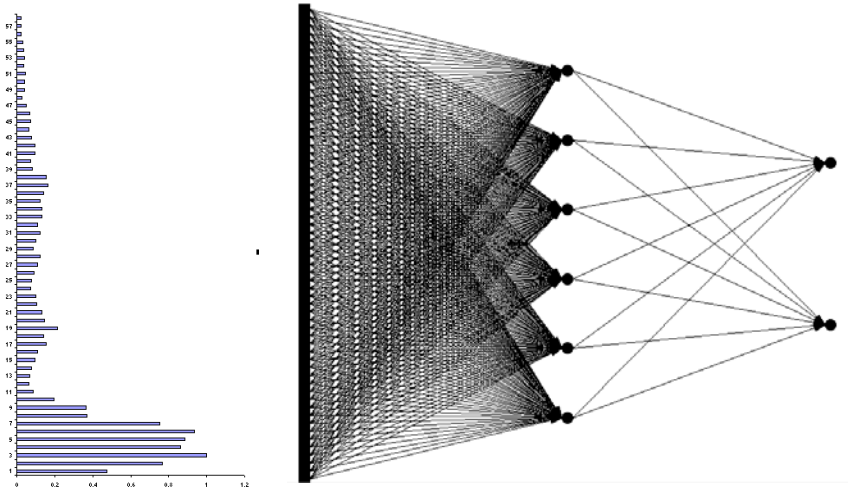


David



- Calculate error

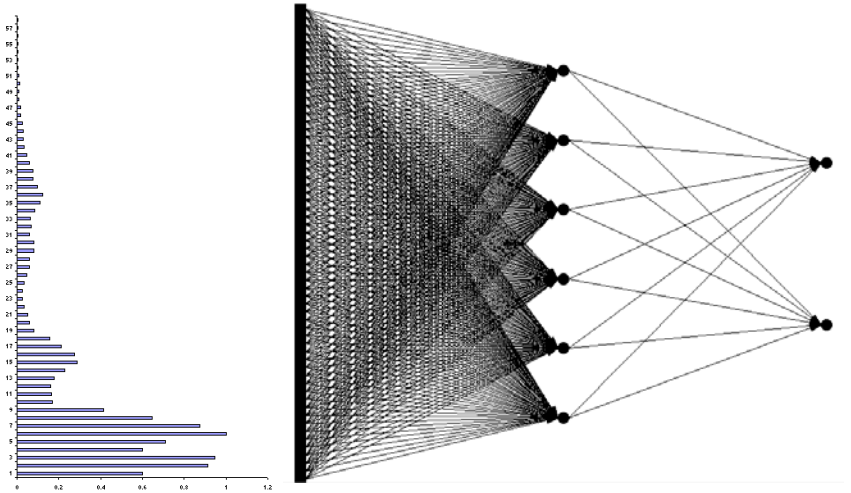
Steve



$0.43 - 0 = 0.43$

$0.26 - 1 = 0.74$

David

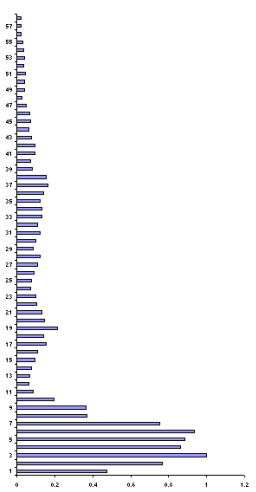


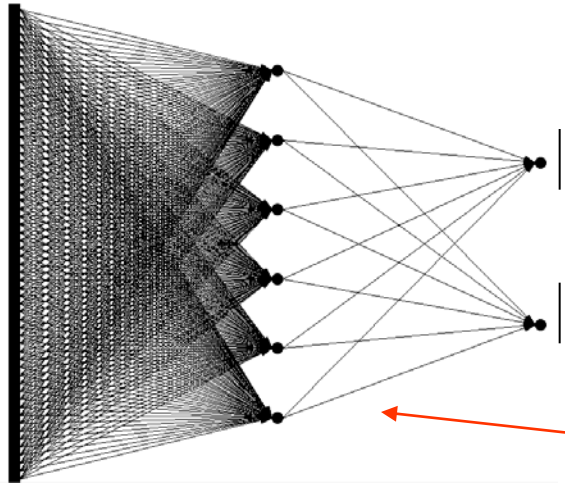
$0.73 - 1 = 0.27$

$0.55 - 0 = 0.55$

- Backprop error and adjust weights

Steve



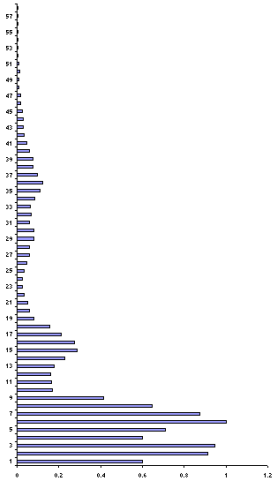


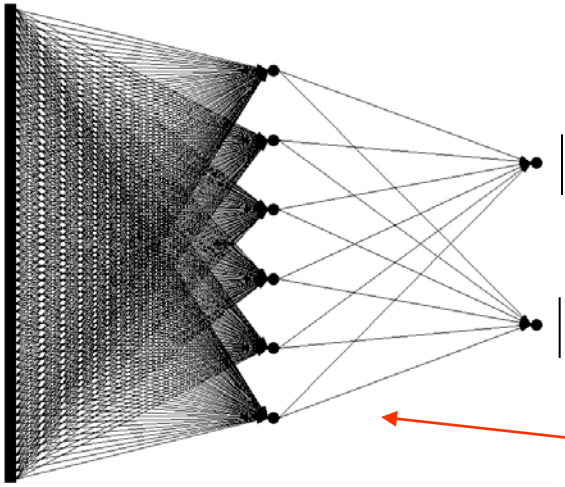
$$|0.43 - 0| = 0.43$$

$$|0.26 - 1| = 0.74$$

$$1.17$$

David



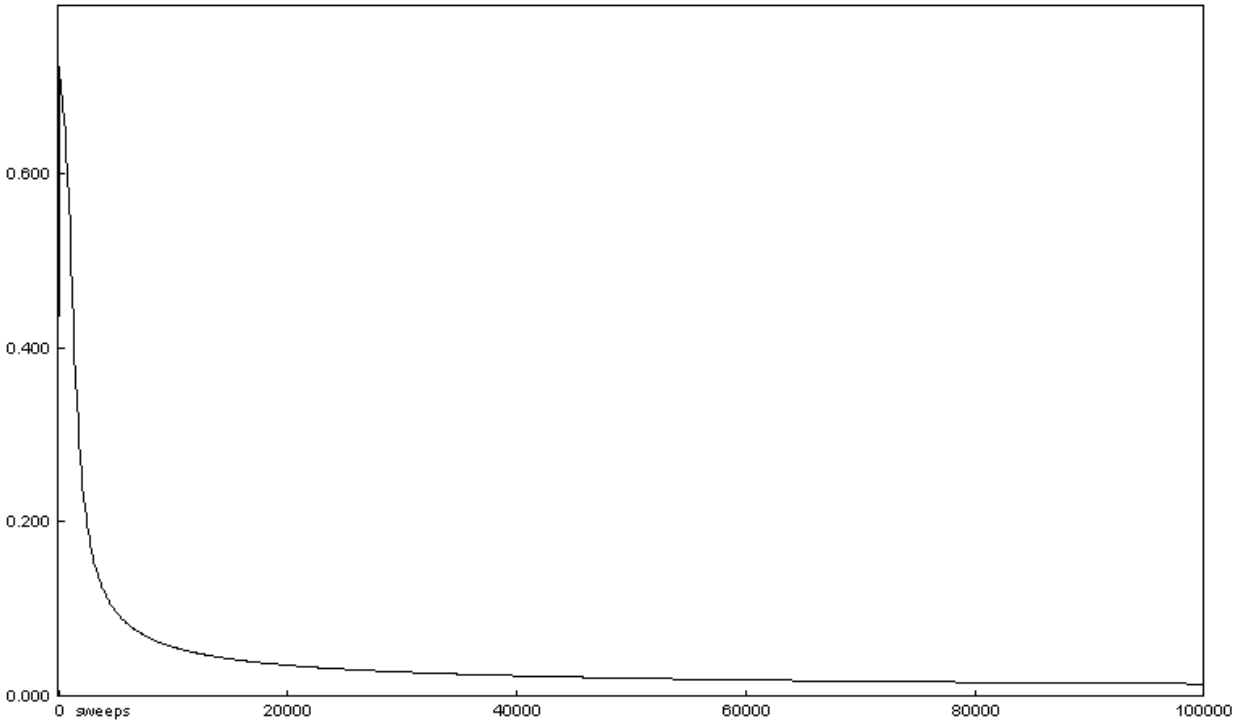


$$|0.73 - 1| = 0.27$$

$$|0.55 - 0| = 0.55$$

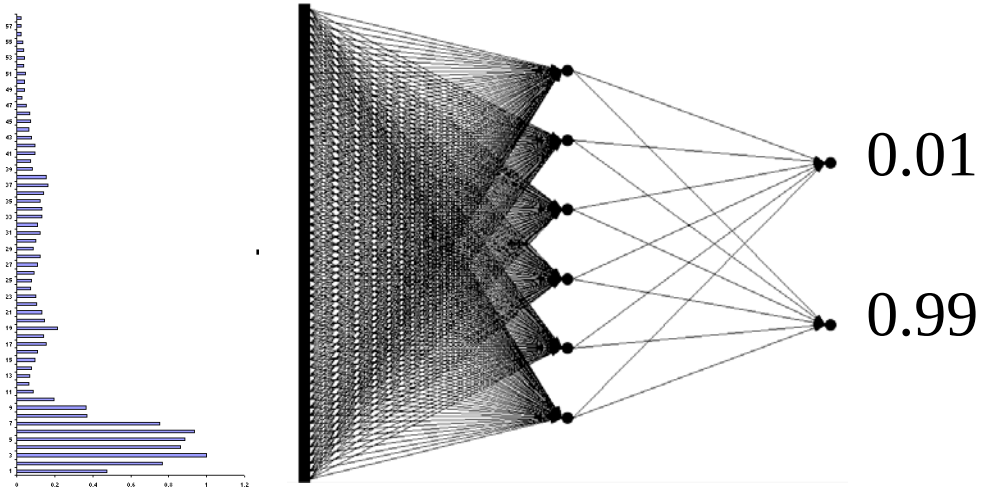
$$0.82$$

- Repeat process (sweep) for all training pairs
  - Present data
  - Calculate error
  - Backpropagate error
  - Adjust weights
- Repeat process multiple times

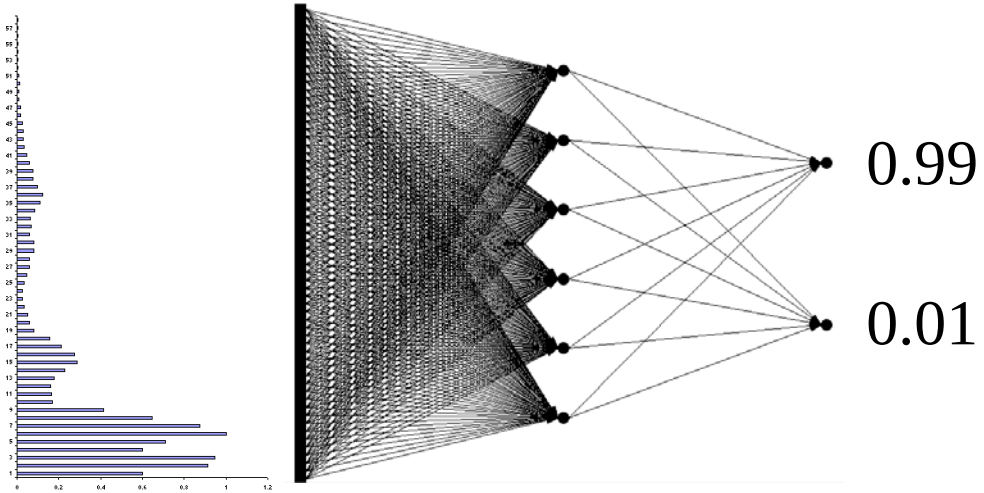


- Presenting the data (trained network)

Steve



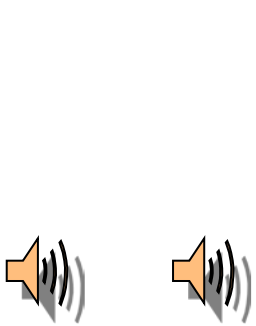
David

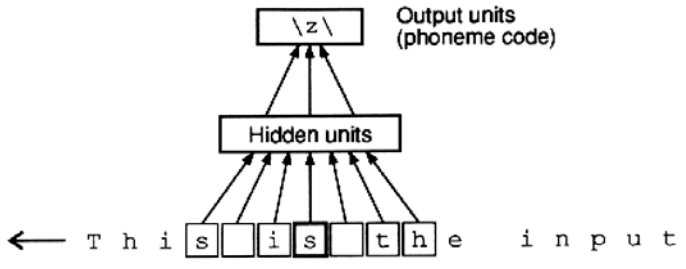


Other Applications of Feed-forward nets

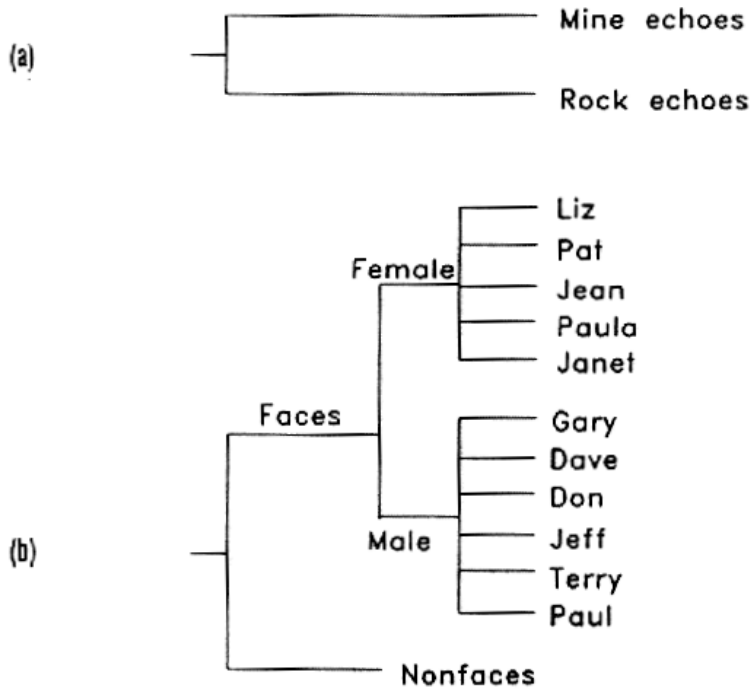
- Pattern recognition
  - Character recognition
  - Face Recognition
- Sonar mine/rock recognition (Gorman & Sejnowksi, 1988)
- Navigation of a car (Pomerleau, 1989)
- Pronunciation (NETtalk)

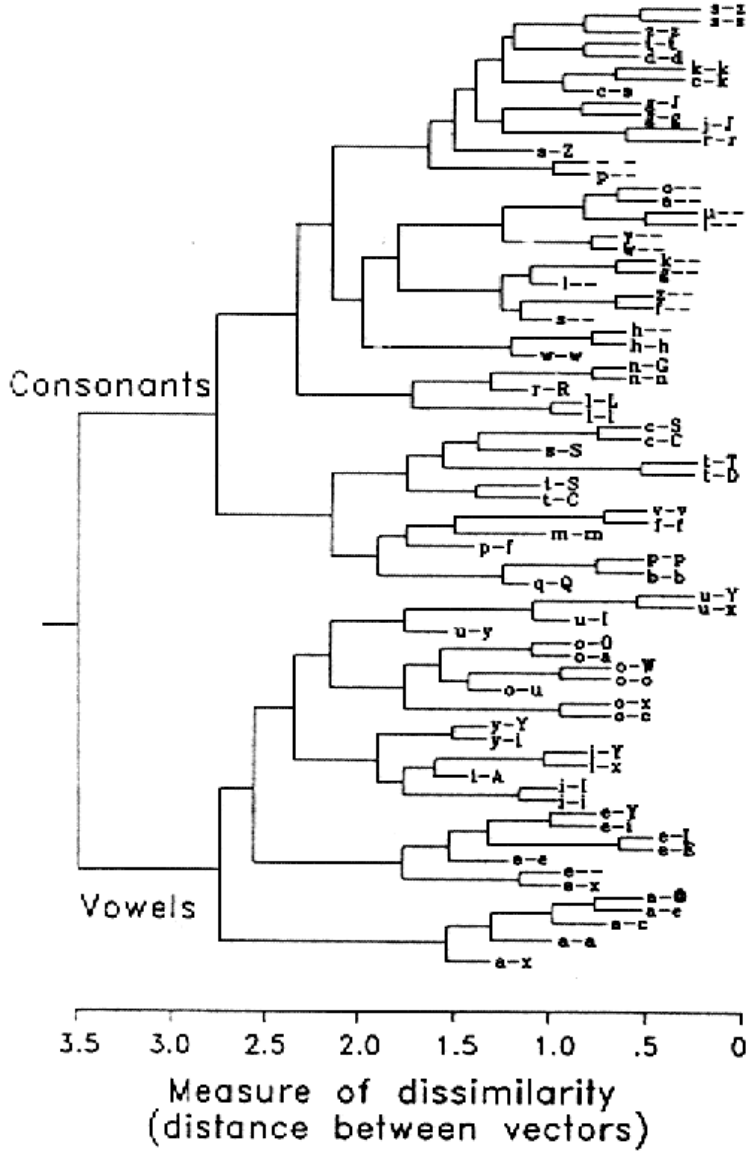
(Sejnowksi & Rosenberg, 1987)





Cluster analysis of hidden layer





FFNs as Biological Modelling Tools

- Signalling / Sexual Selection
  - Enquist & Arak (1994)
    - Preference for symmetry not selection for ‘good genes’, but instead arises through the need to recognise objects irrespective of their orientation
  - Johnstone (1994)
    - Exaggerated, symmetric ornaments facilitate mate recognition

(but see Dawkins & Guilford, 1995)

Literature & Resources

- Textbook:
  - ”Neural Networks for Pattern Recognition”, Bishop, C.M., 1996
- Software:
  - Neural Networks for Face Recognition

    http://www.cs.cmu.edu/afs/cs.cmu.edu/user/mitchell/ftp/faces.html
  - SNNS Stuttgart Neural Networks Simulator

    http://www-ra.informatik.uni-tuebingen.de/SNNS
  - Neural Networks at your fingertips

    http://www.geocities.com/CapeCanaveral/1624/

    http://www.stats.gla.ac.uk/~ernest/files/NeuralAppl.html

References / Advanced Reading List

1. Bourland, H. & Kamp, Y. (1988). Auto-association by Multilayer Perceptrons and Singular Values Decomposition. *Biological Cybernetics*, **59**, 291-294.
2. Bullinaria, J.A. (1997). Modelling Reading, Spelling and Past Tense Learning with Artificial Neural Networks. *Brain and Language*, **59**, 236-266.
3. Bullinaria, J.A. (2002). To Modularize or Not To Modularize? In *Proceedings of the 2002 U.K. Workshop on Computational Intelligence: UKCI-02*, 3-10.
4. Le Cun, Y. et al. (1989). Back-propagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, **1**, 541-551.
5. Pomerleau, D.A. (1989). ALVINN: An Autonomous Land Vehicle in a Neural Network. In D.S. Touretzky (ed.), *Advances in Neural Information Processing Systems I*, 305-313. Morgan Kaufmann.
6. Sejnowski, T.J. & Rosenberg, C.R. (1987). Parallel Networks that Learn to Pronounce English Text. *Complex Systems*, **1**, 145-168.
7. Weigend, A.S. & Gershenfeld, N.A. (1994). *Time Series Prediction: Forecasting the Future and Understanding the Past.* Addison-Wesley.

References

1. Simon Haykins – Neural Networks and Learning Machines
2. Christopher Bishop – Neural Networks in Pattern Recognition
3. Kevin Gurney – Introduction to Neural Networks
4. Hertz, Krogh, Palmer – Introduction to theory of Neural computation (Santa Fe Institute Series)
5. Handbook of Brain Sciences & Neural Networks