CHAPTER 6

Artificial neural networks:

Supervised learning

- Introduction, or how the brain works
- The neuron as a simple computing element
- The perceptron
- Multilayer neural networks
- Accelerated learning in multilayer neural networks
- The Hopfield network
- Bidirectional associative memories (BAM)
- Summary

© Negnevitsky, Pearson Education, 2002

Introduction, or how the brain works

Machine learning involves adaptive mechanisms that enable computers to learn from experience, learn by example and learn by analogy. Learning capabilities can improve the performance of an intelligent system over time. The most popular approaches to machine learning are **artificial neural networks** and **genetic algorithms**. This lecture is dedicated to neural networks.

© Negnevitsky, Pearson Education, 2002

- A **neural network** can be defined as a model of reasoning based on the human brain. The brain consists of a densely interconnected set of nerve cells, or basic information-processing units, called **neurons**.
- The human brain incorporates nearly 10 billion neurons and 60 trillion connections, *synapses*, between them. By using multiple neurons simultaneously, the brain can perform its functions much faster than the fastest computers in existence today.

© Negnevitsky, Pearson Education, 2002

- Each neuron has a very simple structure, but an army of such elements constitutes a tremendous processing power.
- A neuron consists of a cell body, **soma**, a number of fibers called **dendrites**, and a single long fiber called the **axon**.

© Negnevitsky, Pearson Education, 2002

Biological neural network

© Negnevitsky, Pearson Education, 2002

Neural networks to the rescue

- Neural networks are configured for a specific application, such as pattern recognition or data classification, through a learning process
- In a biological system, learning involves adjustments to the synaptic connections between neurons

➔ same for artificial neural networks (ANNs)

© Negnevitsky, Pearson Education, 2002

Where can neural network systems help

- when we can't formulate an algorithmic solution.
- when we ***can*** get lots of examples of the behavior we require.

  'learning from experience'
- when we need to pick out the structure from existing data.

© Negnevitsky, Pearson Education, 2002

- Our brain can be considered as a highly complex, non-linear and parallel information-processing system.
- Information is stored and processed in a neural network simultaneously throughout the whole network, rather than at specific locations. In other words, in neural networks, both data and its processing are **global** rather than local.
- Learning is a fundamental and essential characteristic of biological neural networks. The ease with which they can learn led to attempts to emulate a biological neural network in a computer.

© Negnevitsky, Pearson Education, 2002

- An artificial neural network consists of a number of very simple processors, also called **neurons**, which are analogous to the biological neurons in the brain.
- The neurons are connected by weighted links passing signals from one neuron to another.
- The output signal is transmitted through the neuron's outgoing connection. The outgoing connection splits into a number of branches that transmit the same signal. The outgoing branches terminate at the incoming connections of other neurons in the network.

© Negnevitsky, Pearson Education, 2002

Architecture of a typical artificial neural network

© Negnevitsky, Pearson Education, 2002

Analogy between biological and artificial neural networks

Biological Neural Network	***Artificial Neural Network***
Soma	Neuron
Dendrite	Input
Axon	Output
Synapse	Weight

© Negnevitsky, Pearson Education, 2002

The neuron as a simple computing element

Diagram of a neuron

© Negnevitsky, Pearson Education, 2002

- The neuron computes the weighted sum of the input signals and compares the result with a **threshold value**, θ. If the net input is less than the threshold, the neuron output is –1. But if the net input is greater than or equal to the threshold, the neuron becomes activated and its output attains a value +1.
- The neuron uses the following transfer or **activation function**:

$$X = \sum_{i=1}^{n} x_i w_i \qquad Y = \begin{cases} +1, & \text{if } X \geq \theta \\ -1, & \text{if } X < \theta \end{cases}$$

- This type of activation function is called a **sign function**.

© Negnevitsky, Pearson Education, 2002

Activation functions of a neuron

Step function	*Sign function*	*Sigmoid function*	*Linear function*
$Y^{step} = \begin{cases} 1, \text{ if } X \geq 0 \\ 0, \text{ if } X < 0 \end{cases}$	$Y^{sign} = \begin{cases} +1, \text{if } X \geq 0 \\ -1, \text{if } X < 0 \end{cases}$	$Y^{sigmoid} = \dfrac{1}{1+e^{-X}}$	$Y^{linear} = X$

© Negnevitsky, Pearson Education, 2002

Can a single neuron learn a task?

- In 1958, **Frank Rosenblatt** introduced a training algorithm that provided the first procedure for training a simple ANN: a **perceptron**.
- The perceptron is the simplest form of a neural network. It consists of a single neuron with *adjustable* synaptic weights and a *hard limiter*.

© Negnevitsky, Pearson Education, 2002

Single-layer two-input perceptron

© Negnevitsky, Pearson Education, 2002

The Perceptron

- The operation of Rosenblatt's perceptron is based on the **McCulloch and Pitts neuron model**. The model consists of a linear combiner followed by a hard limiter.
- The weighted sum of the inputs is applied to the hard limiter, which produces an output equal to +1 if its input is positive and –1 if it is negative.

© Negnevitsky, Pearson Education, 2002

How does the perceptron learn its classification tasks?

This is done by making small adjustments in the weights to reduce the difference between the actual and desired outputs of the perceptron. The initial weights are randomly assigned, usually in the range [–0.5, 0.5], and then updated to obtain the output consistent with the training examples.

© Negnevitsky, Pearson Education, 2002

- If at iteration p, the actual output is $Y(p)$ and the desired output is $Y_d(p)$, then the error is given by:

$$e(p) = Y_d(p) - Y(p)$$

where $p = 1, 2, 3, \ldots$

Iteration p here refers to the pth training example presented to the perceptron.

- If the error, $e(p)$, is positive, we need to increase perceptron output $Y(p)$, but if it is negative, we need to decrease $Y(p)$.

© Negnevitsky, Pearson Education, 2002

The perceptron learning rule

$$w_i(p+1) = w_i(p) + \alpha \cdot x_i(p) \cdot e(p)$$

where $p = 1, 2, 3, \ldots$

α is the **learning rate**, a positive constant less than unity.

The perceptron learning rule was first proposed by **Rosenblatt** in 1960. Using this rule we can derive the perceptron training algorithm for classification tasks.

© Negnevitsky, Pearson Education, 2002

Perceptron's training algorithm

Step 1: Initialisation

Set initial weights $w_1, w_2, \ldots, w_n$ and threshold θ to random numbers in the range [–0.5, 0.5].

If the error, $e(p)$, is positive, we need to increase perceptron output $Y(p)$, but if it is negative, we need to decrease $Y(p)$.

© Negnevitsky, Pearson Education, 2002

Perceptron's training algorithm (continued)

Step 2: Activation

Activate the perceptron by applying inputs $x_1(p)$, $x_2(p), \ldots, x_n(p)$ and desired output $Y_d(p)$.

Calculate the actual output at iteration $p = 1$

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]$$

where n is the number of the perceptron inputs, and *step* is a step activation function.

© Negnevitsky, Pearson Education, 2002

Perceptron's training algorithm (continued)

Step 3: Weight training

Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

where $\Delta w_i(p)$ is the weight correction at iteration p.

The weight correction is computed by the **delta rule**:

$$\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p)$$

Step 4: Iteration

Increase iteration p by one, go back to *Step 2* and repeat the process until convergence.

© Negnevitsky, Pearson Education, 2002

Example of perceptron learning: the logical operation *AND*

Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	–0.1	0	0	0.3	–0.1
	0	1	0	0.3	–0.1	0	0	0.3	–0.1
	1	0	0	0.3	–0.1	1	–1	0.2	–0.1
	1	1	1	0.2	–0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	–1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	–1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	–1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

Threshold: $\theta = 0.2$; learning rate: $\alpha = 0.1$

© Negnevitsky, Pearson Education, 2002

Multilayer neural networks

- A multilayer perceptron is a feedforward neural network with one or more hidden layers.
- The network consists of an **input layer** of source neurons, at least one middle or **hidden layer** of computational neurons, and an **output layer** of computational neurons.
- The input signals are propagated in a forward direction on a layer-by-layer basis.

© Negnevitsky, Pearson Education, 2002

Multilayer perceptron with two hidden layers

© Negnevitsky, Pearson Education, 2002

What does the middle layer hide?

- A hidden layer "hides" its desired output. Neurons in the hidden layer cannot be observed through the input/output behaviour of the network. There is no obvious way to know what the desired output of the hidden layer should be.
- Commercial ANNs incorporate three and sometimes four layers, including one or two hidden layers. Each layer can contain from 10 to 1000 neurons. Experimental neural networks may have five or even six layers, including three or four hidden layers, and utilise millions of neurons.

© Negnevitsky, Pearson Education, 2002

Back-propagation neural network

- Learning in a multilayer network proceeds the same way as for a perceptron.
- A training set of input patterns is presented to the network.
- The network computes its output pattern, and if there is an error – or in other words a difference between actual and desired output patterns – the weights are adjusted to reduce this error.

© Negnevitsky, Pearson Education, 2002

- In a back-propagation neural network, the learning algorithm has two phases.
- First, a training input pattern is presented to the network input layer. The network propagates the input pattern from layer to layer until the output pattern is generated by the output layer.
- If this pattern is different from the desired output, an error is calculated and then propagated backwards through the network from the output layer to the input layer. The weights are modified as the error is propagated.

© Negnevitsky, Pearson Education, 2002

Three-layer back-propagation neural network

© Negnevitsky, Pearson Education, 2002

The back-propagation training algorithm

Step 1: Initialisation

Set all the weights and threshold levels of the network to random numbers uniformly distributed inside a small range:

$$\left(-\frac{2.4}{F_i},\ +\frac{2.4}{F_i}\right)$$

where F_i is the total number of inputs of neuron i in the network. The weight initialisation is done on a neuron-by-neuron basis.

© Negnevitsky, Pearson Education, 2002

Step 2: Activation

Activate the back-propagation neural network by applying inputs $x_1(p), x_2(p), \ldots, x_n(p)$ and desired outputs $y_{d,1}(p), y_{d,2}(p), \ldots, y_{d,n}(p)$.

(a) Calculate the actual outputs of the neurons in the hidden layer:

$$y_j(p) = sigmoid\left[\sum_{i=1}^{n} x_i(p) \cdot w_{ij}(p) - \theta_j\right]$$

where n is the number of inputs of neuron j in the hidden layer, and *sigmoid* is the *sigmoid* activation function.

© Negnevitsky, Pearson Education, 2002

Step 2: Activation (continued)

(b) Calculate the actual outputs of the neurons in the output layer:

$$y_k(p) = sigmoid\left[\sum_{j=1}^{m} x_{jk}(p)\cdot w_{jk}(p) - \theta_k\right]$$

where m is the number of inputs of neuron k in the output layer.

© Negnevitsky, Pearson Education, 2002

Step 3: Weight training

Update the weights in the back-propagation network propagating backward the errors associated with output neurons.

(*a*) Calculate the error gradient for the neurons in the output layer:

$$\delta_k(p) = y_k(p) \cdot [1 - y_k(p)] \cdot e_k(p)$$

where $$e_k(p) = y_{d,k}(p) - y_k(p)$$

Calculate the weight corrections:

$$\Delta w_{jk}(p) = \alpha \cdot y_j(p) \cdot \delta_k(p)$$

Update the weights at the output neurons:

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$$

© Negnevitsky, Pearson Education, 2002

Step 3: Weight training (continued)

(b) Calculate the error gradient for the neurons in the hidden layer:

$$\delta_j(p) = y_j(p) \cdot [1 - y_j(p)] \cdot \sum_{k=1}^{l} \delta_k(p)\, w_{jk}(p)$$

Calculate the weight corrections:

$$\Delta w_{ij}(p) = \alpha \cdot x_i(p) \cdot \delta_j(p)$$

Update the weights at the hidden neurons:

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

© Negnevitsky, Pearson Education, 2002

Step 4: Iteration

Increase iteration *p* by one, go back to *Step 2* and repeat the process until the selected error criterion is satisfied.

As an example, we may consider the three-layer back-propagation network. Suppose that the network is required to perform logical operation *Exclusive-OR*. Recall that a single-layer perceptron could not do this operation. Now we will apply the three-layer net.

© Negnevitsky, Pearson Education, 2002

Three-layer network for solving the Exclusive-OR operation

© Negnevitsky, Pearson Education, 2002

- The effect of the threshold applied to a neuron in the hidden or output layer is represented by its weight, θ, connected to a fixed input equal to –1.
- The initial weights and threshold levels are set randomly as follows:
  $w_{13} = 0.5$, $w_{14} = 0.9$, $w_{23} = 0.4$, $w_{24} = 1.0$, $w_{35} = -1.2$, $w_{45} = 1.1$, $\theta_3 = 0.8$, $\theta_4 = -0.1$ and $\theta_5 = 0.3$.

© Negnevitsky, Pearson Education, 2002

- We consider a training set where inputs x_1 and x_2 are equal to 1 and desired output $y_{d,5}$ is 0. The actual outputs of neurons 3 and 4 in the hidden layer are calculated as

$$y_3 = sigmoid\ (x_1 w_{13} + x_2 w_{23} - \theta_3) = 1/\left[1 + e^{-(1\cdot 0.5 + 1\cdot 0.4 - 1\cdot 0.8)}\right] = 0.5250$$

$$y_4 = sigmoid\ (x_1 w_{14} + x_2 w_{24} - \theta_4) = 1/\left[1 + e^{-(1\cdot 0.9 + 1\cdot 1.0 + 1\cdot 0.1)}\right] = 0.8808$$

- Now the actual output of neuron 5 in the output layer is determined as:

$$y_5 = sigmoid\ (y_3 w_{35} + y_4 w_{45} - \theta_5) = 1/\left[1 + e^{-(-0.5250\cdot 1.2 + 0.8808\cdot 1.1 - 1\cdot 0.3)}\right] = 0.5097$$

- Thus, the following error is obtained:

$$e = y_{d,5} - y_5 = 0 - 0.5097 = -0.5097$$

© Negnevitsky, Pearson Education, 2002

- The next step is weight training. To update the weights and threshold levels in our network, we propagate the error, *e*, from the output layer backward to the input layer.
- First, we calculate the error gradient for neuron 5 in the output layer:

$$\delta_5 = y_5\,(1 - y_5)\,e = 0.5097 \cdot (1 - 0.5097) \cdot (-0.5097) = -0.1274$$

- Then we determine the weight corrections assuming that the learning rate parameter, α, is equal to 0.1:

$$\Delta w_{35} = \alpha \cdot y_3 \cdot \delta_5 = 0.1 \cdot 0.5250 \cdot (-0.1274) = -0.0067$$
$$\Delta w_{45} = \alpha \cdot y_4 \cdot \delta_5 = 0.1 \cdot 0.8808 \cdot (-0.1274) = -0.0112$$
$$\Delta \theta_5 = \alpha \cdot (-1) \cdot \delta_5 = 0.1 \cdot (-1) \cdot (-0.1274) = -0.0127$$

© Negnevitsky, Pearson Education, 2002

- Next we calculate the error gradients for neurons 3 and 4 in the hidden layer:

$$\delta_3 = y_3(1-y_3)\cdot\delta_5\cdot w_{35} = 0.5250\cdot(1-0.5250)\cdot(-0.1274)\cdot(-1.2) = 0.0381$$

$$\delta_4 = y_4(1-y_4)\cdot\delta_5\cdot w_{45} = 0.8808\cdot(1-0.8808)\cdot(-0.127\,4)\cdot 1.1 = -0.0147$$

- We then determine the weight corrections:

$$\Delta w_{13} = \alpha\cdot x_1\cdot\delta_3 = 0.1\cdot 1\cdot 0.0381 = 0.0038$$
$$\Delta w_{23} = \alpha\cdot x_2\cdot\delta_3 = 0.1\cdot 1\cdot 0.0381 = 0.0038$$
$$\Delta\theta_3 = \alpha\cdot(-1)\cdot\delta_3 = 0.1\cdot(-1)\cdot 0.0381 = -0.0038$$
$$\Delta w_{14} = \alpha\cdot x_1\cdot\delta_4 = 0.1\cdot 1\cdot(-0.0147) = -0.0015$$
$$\Delta w_{24} = \alpha\cdot x_2\cdot\delta_4 = 0.1\cdot 1\cdot(-0.0147) = -0.0015$$
$$\Delta\theta_4 = \alpha\cdot(-1)\cdot\delta_4 = 0.1\cdot(-1)\cdot(-0.0147) = 0.0015$$

© Negnevitsky, Pearson Education, 2002

- At last, we update all weights and threshold:

$$w_{13} = w_{13} + \Delta w_{13} = 0.5 + 0.0038 = 0.5038$$

$$w_{14} = w_{14} + \Delta w_{14} = 0.9 - 0.0015 = 0.8985$$

$$w_{23} = w_{23} + \Delta w_{23} = 0.4 + 0.0038 = 0.4038$$

$$w_{24} = w_{24} + \Delta w_{24} = 1.0 - 0.0015 = 0.9985$$

$$w_{35} = w_{35} + \Delta w_{35} = -1.2 - 0.0067 = -1.2067$$

$$w_{45} = w_{45} + \Delta w_{45} = 1.1 - 0.0112 = 1.0888$$

$$\theta_3 = \theta_3 + \Delta\theta_3 = 0.8 - 0.0038 = 0.7962$$

$$\theta_4 = \theta_4 + \Delta\theta_4 = -0.1 + 0.0015 = -0.0985$$

$$\theta_5 = \theta_5 + \Delta\theta_5 = 0.3 + 0.0127 = 0.3127$$

- The training process is repeated until the sum of squared errors is less than 0.001.

© Negnevitsky, Pearson Education, 2002

Learning curve for operation *Exclusive-OR*

© Negnevitsky, Pearson Education, 2002

Final results of three-layer network learning

Inputs		Desired output	Actual output	Error	Sum of squared errors
x_1	x_2	y_d	y_5	e	
1	1	0	0.0155	–0.0155	0.0010
0	1	1	0.9849	0.0151	
1	0	1	0.9849	0.0151	
0	0	0	0.0175	–0.0175	

© Negnevitsky, Pearson Education, 2002

Network represented by McCulloch-Pitts model for solving the *Exclusive-OR* operation

© Negnevitsky, Pearson Education, 2002

Accelerated learning in multilayer neural networks

- A multilayer network learns much faster when the sigmoidal activation function is represented by a **hyperbolic tangent**:

$$Y^{tanh} = \frac{2a}{1+e^{-bX}} - a$$

where a and b are constants.

Suitable values for a and b are:
$a = 1.716$ and $b = 0.667$

© Negnevitsky, Pearson Education, 2002

- We also can accelerate training by including a **momentum term** in the delta rule:

$$\Delta w_{jk}(p) = \beta \cdot \Delta w_{jk}(p-1) + \alpha \cdot y_j(p) \cdot \delta_k(p)$$

where β is a positive number ($0 \le \beta < 1$) called the **momentum constant**. Typically, the momentum constant is set to 0.95.

This equation is called the **generalised delta rule**.

© Negnevitsky, Pearson Education, 2002

Learning with momentum for operation *Exclusive-OR*

© Negnevitsky, Pearson Education, 2002

Learning with adaptive learning rate

To accelerate the convergence and yet avoid the danger of instability, we can apply two heuristics:

Heuristic 1

If the change of the sum of squared errors has the same algebraic sign for several consequent epochs, then the learning rate parameter, α, should be increased.

Heuristic 2

If the algebraic sign of the change of the sum of squared errors alternates for several consequent epochs, then the learning rate parameter, α, should be decreased.

© Negnevitsky, Pearson Education, 2002

- Adapting the learning rate requires some changes in the back-propagation algorithm.
- If the sum of squared errors at the current epoch exceeds the previous value by more than a predefined ratio (typically 1.04), the learning rate parameter is decreased (typically by multiplying by 0.7) and new weights and thresholds are calculated.
- If the error is less than the previous one, the learning rate is increased (typically by multiplying by 1.05).

© Negnevitsky, Pearson Education, 2002

Learning with adaptive learning rate

© Negnevitsky, Pearson Education, 2002

Learning with momentum and adaptive learning rate

© Negnevitsky, Pearson Education, 2002

Will a neural network work for my problem?

Neural networks represent a class of very powerful, general-purpose tools that have been successfully applied to prediction, classification and clustering problems. They are used in a variety of areas, from speech and character recognition to detecting fraudulent transactions, from medical diagnosis of heart attacks to process control and robotics, from predicting foreign exchange rates to detecting and identifying radar targets.

© Negnevitsky, Pearson Education, 2002

Case study 4: Character recognition Neural networks

Recognition of both printed and handwritten characters is a typical domain where neural networks have been successfully applied.

Optical character recognition systems were among the first commercial applications of neural networks.

© Negnevitsky, Pearson Education, 2002

We demonstrate an application of a multilayer feedforward network for printed character recognition.

For simplicity, we can limit our task to the recognition of digits from 0 to 9. Each digit is represented by a 5×9 bit map.

In commercial applications, where a better resolution is required, at least 16×16 bit maps are used.

© Negnevitsky, Pearson Education, 2002

Bit maps for digit recognition

© Negnevitsky, Pearson Education, 2002

How do we choose the architecture of a neural network?

- The number of neurons in the input layer is decided by the number of pixels in the bit map. The bit map in our example consists of 45 pixels, and thus we need 45 input neurons.
- The output layer has 10 neurons – one neuron for each digit to be recognised.

© Negnevitsky, Pearson Education, 2002

How do we determine an optimal number of hidden neurons?

- Complex patterns cannot be detected by a small number of hidden neurons; however too many of them can dramatically increase the computational burden.
- Another problem is **overfitting**. The greater the number of hidden neurons, the greater the ability of the network to recognise existing patterns. However, if the number of hidden neurons is too big, the network might simply memorise all training examples.

© Negnevitsky, Pearson Education, 2002

Neural network for printed digit recognition

© Negnevitsky, Pearson Education, 2002

What are the test examples for character recognition?

- A test set has to be strictly independent from the training examples.
- To test the character recognition network, we present it with examples that include “noise” – the distortion of the input patterns.
- We evaluate the performance of the printed digit recognition networks with 1000 test examples (100 for each digit to be recognised).

© Negnevitsky, Pearson Education, 2002

Learning curves of the digit recognition three-layer neural networks

© Negnevitsky, Pearson Education, 2002

Performance evaluation of the digit recognition neural networks

© Negnevitsky, Pearson Education, 2002

Can we improve the performance of the character recognition neural network?

A neural network is as good as the examples used to train it.

Therefore, we can attempt to improve digit recognition by feeding the network with "noisy" examples of digits from 0 to 9.

© Negnevitsky, Pearson Education, 2002

Performance evaluation of the digit recognition network trained with “noisy” examples

© Negnevitsky, Pearson Education, 2002

Case study 5: Prediction neural networks

As an example, we consider a problem of predicting the market value of a given house based on the knowledge of the sales prices of similar houses.

© Negnevitsky, Pearson Education, 2002

- In this problem, the inputs (the house location, living area, number of bedrooms, number of bathrooms, land size, type of heating system, etc.) are well-defined, and even standardised for sharing the housing market information between different real estate agencies.
- The output is also well-defined – we know what we are trying to predict.
- The features of recently sold houses and their sales prices are examples, which we use for training the neural network.

© Negnevitsky, Pearson Education, 2002

Network generalisation

An appropriate number of training examples can be estimated with **Widrow's rule of thumb**, which suggests that, for a good generalisation, we need to satisfy the following condition:

$$N = \frac{n_w}{e}$$

where N is the number of training examples, n_w is the number of synaptic weights in the network, and e is the network error permitted on test.

© Negnevitsky, Pearson Education, 2002

Massaging the data

Data can be divided into three main types: continuous, discrete and categorical .

- **Continuous data** vary between two pre-set values – minimum and maximum, and can be mapped, or massaged, to the range between 0 and 1 as:

$$massaged\ value = \frac{actual\ value - minimum\ value}{maximum\ value - minimum\ value}$$

© Negnevitsky, Pearson Education, 2002

- **Discrete data**, such as the number of bedrooms and the number of bathrooms, also have maximum and minimum values. For example, the number of bedrooms usually ranges from 0 to 4.

Massaging the data

© Negnevitsky, Pearson Education, 2002

- **Categorical data**, such as gender and marital status, can be massaged by using ***1 of N coding***. This method implies that each categorical value is handled as a separate input.

  For example, marital status, which can be either single, divorced, married or widowed, would be represented by four inputs. Each of these inputs can have a value of either 0 or 1. Thus, a married person would be represented by an input vector [0 0 1 0].

© Negnevitsky, Pearson Education, 2002

Feedforward neural network for real-estate appraisal

Location rating	7	0.7000
House condition	8	0.8000
Number of bedrooms	3	0.7500
Number of bathrooms	1	0.5000
Living area	121.00 m^2	0.3550
Land size	682.00 m^2	0.4682
Garage	double	1.0000
Type of heating system	wood heater	1.0000
		0.0000
		0.0000

© Negnevitsky, Pearson Education, 2002

How do we validate results?

To validate results, we use a set of examples never seen by the network.

Before training, all the available data are randomly divided into a training set and a test set.

Once the training phase is complete, the network's ability to generalise is tested against examples of the test set.

© Negnevitsky, Pearson Education, 2002

https://www.kaggle.com/adarsh140799/titanic-using-keras-and-tf

https://www.kaggle.com/alaasedeeq/predicting-the-survival-of-titanic-high-score

The Hopfield Network

- Neural networks were designed on analogy with the brain. The brain's memory, however, works by association. For example, we can recognise a familiar face even in an unfamiliar environment within 100-200 ms. We can also recall a complete sensory experience, including sounds and scenes, when we hear only a few bars of music. The brain routinely associates one thing with another.

© Negnevitsky, Pearson Education, 2002

- Multilayer neural networks trained with the back-propagation algorithm are used for pattern recognition problems. However, to emulate the human memory's associative characteristics we need a different type of network: a **recurrent neural network**.
- A recurrent neural network has feedback loops from its outputs to its inputs. The presence of such loops has a profound impact on the learning capability of the network.

© Negnevitsky, Pearson Education, 2002

- The stability of recurrent networks intrigued several researchers in the 1960s and 1970s. However, none was able to predict which network would be stable, and some researchers were pessimistic about finding a solution at all. The problem was solved only in 1982, when **John Hopfield** formulated the physical principle of storing information in a dynamically stable network.

© Negnevitsky, Pearson Education, 2002

Single-layer *n*-neuron Hopfield network

© Negnevitsky, Pearson Education, 2002

- The Hopfield network uses McCulloch and Pitts neurons with the *sign activation function* as its computing element:

$$Y^{sign} = \begin{cases} +1, & \text{if } X > 0 \\ -1, & \text{if } X < 0 \\ Y, & \text{if } X = 0 \end{cases}$$

© Negnevitsky, Pearson Education, 2002

- The current state of the Hopfield network is determined by the current outputs of all neurons, $y_1, y_2, \ldots, y_n$.

  Thus, for a single-layer n-neuron network, the state can be defined by the **state vector** as:

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

© Negnevitsky, Pearson Education, 2002

- In the Hopfield network, synaptic weights between neurons are usually represented in matrix form as follows:

$$\mathbf{W} = \sum_{m=1}^{M} \mathbf{Y}_m \mathbf{Y}_m^T - M\,\mathbf{I}$$

where M is the number of states to be memorised by the network, $\mathbf{Y}_m$ is the n-dimensional binary vector, $\mathbf{I}$ is $n \times n$ identity matrix, and superscript T denotes a matrix transposition.

© Negnevitsky, Pearson Education, 2002

- The stable state-vertex is determined by the weight matrix **W**, the current input vector **X**, and the threshold matrix θ. If the input vector is partially incorrect or incomplete, the initial state will converge into the stable state-vertex after a few iterations.
- Suppose, for instance, that our network is required to memorise two opposite states, (1, 1, 1) and (–1, –1, –1). Thus,

$$\mathbf{Y}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{Y}_2 = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \quad \text{or} \quad \mathbf{Y}_1^T = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \quad \mathbf{Y}_2^T = \begin{bmatrix} -1 & -1 & -1 \end{bmatrix}$$

where $\mathbf{Y}_1$ and $\mathbf{Y}_2$ are the three-dimensional vectors.

© Negnevitsky, Pearson Education, 2002

- The 3×3 identity matrix **I** is

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Thus, we can now determine the weight matrix as follows:

$$\mathbf{W} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & -1 & -1 \end{bmatrix} - 2 \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix}$$

- Next, the network is tested by the sequence of input vectors, $\mathbf{X}_1$ and $\mathbf{X}_2$, which are equal to the output (or target) vectors $\mathbf{Y}_1$ and $\mathbf{Y}_2$, respectively.

© Negnevitsky, Pearson Education, 2002

- First, we activate the Hopfield network by applying the input vector **X**. Then, we calculate the actual output vector **Y**, and finally, we compare the result with the initial input vector **X**.

$$\mathbf{Y}_1 = sign\left\{\begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}\right\} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\mathbf{Y}_2 = sign\left\{\begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix}\begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}\right\} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

© Negnevitsky, Pearson Education, 2002

- The remaining six states are all unstable. However, stable states (also called **fundamental memories**) are capable of attracting states that are close to them.
- The fundamental memory (1, 1, 1) attracts unstable states (–1, 1, 1), (1, –1, 1) and (1, 1, –1). Each of these unstable states represents a single error, compared to the fundamental memory (1, 1, 1).
- The fundamental memory (–1, –1, –1) attracts unstable states (–1, –1, 1), (–1, 1, –1) and (1, –1, –1).
- Thus, the Hopfield network can act as an **error correction network**.

© Negnevitsky, Pearson Education, 2002

Storage capacity of the Hopfield network

- **Storage capacity is** or the largest number of fundamental memories that can be stored and retrieved correctly.
- The maximum number of fundamental memories M_{max} that can be stored in the n-neuron recurrent network is limited by

$$M_{max} = 0.15\,n$$

© Negnevitsky, Pearson Education, 2002

Figure 5. An example of the behavior of a Hopfield net when used as a content-addressable memory. A 120 node net was trained using the eight exemplars shown in (A). The pattern for the digit "3" was corrupted by randomly reversing each bit with a probability of .25, and then applied to the net at time zero. Outputs at time zero and after the first seven iterations are shown in (B).

© Negnevitsky, Pearson Education, 2002

Hopfield Model

- The Hopfield model is used as an auto-associative memory to store and recall a set of bitmap images.
- Associative recall of images, given incomplete or corrupted version of a stored image the network can recall the original

© Negnevitsky, Pearson Education, 2002

Bidirectional associative memory (BAM)

- The Hopfield network represents an **autoassociative** type of memory – it can retrieve a corrupted or incomplete memory but cannot associate this memory with another different memory.
- Human memory is essentially **associative**. One thing may remind us of another, and that of another, and so on. We use a chain of mental associations to recover a lost memory. If we forget where we left an umbrella, we try to recall where we last had it, what we were doing, and who we were talking to. We attempt to establish a chain of associations, and thereby to restore a lost memory.

© Negnevitsky, Pearson Education, 2002

- To associate one memory with another, we need a recurrent neural network capable of accepting an input pattern on one set of neurons and producing a related, but different, output pattern on another set of neurons.
- **Bidirectional associative memory (BAM)**, first proposed by **Bart Kosko**, is a heteroassociative network. It associates patterns from one set, set *A*, to patterns from another set, set *B*, and vice versa. Like a Hopfield network, the BAM can generalise and also produce correct outputs despite corrupted or incomplete inputs.

© Negnevitsky, Pearson Education, 2002

BAM operation

(*a*) Forward direction.

(*b*) Backward direction.

© Negnevitsky, Pearson Education, 2002

The basic idea behind the BAM is to store pattern pairs so that when *n*-dimensional vector **X** from set *A* is presented as input, the BAM recalls *m*-dimensional vector **Y** from set *B*, but when **Y** is presented as input, the BAM recalls **X**.

© Negnevitsky, Pearson Education, 2002

- To develop the BAM, we need to create a correlation matrix for each pattern pair we want to store. The correlation matrix is the matrix product of the input vector **X**, and the transpose of the output vector $\mathbf{Y}^T$. The BAM weight matrix is the sum of all correlation matrices, that is,

$$\mathbf{W} = \sum_{m=1}^{M} \mathbf{X}_m \ \mathbf{Y}_m^T$$

where M is the number of pattern pairs to be stored in the BAM.

© Negnevitsky, Pearson Education, 2002

Stability and storage capacity of the BAM

- The BAM is **unconditionally stable**. This means that any set of associations can be learned without risk of instability.
- The maximum number of associations to be stored in the BAM should not exceed the number of neurons in the smaller layer.
- The more serious problem with the BAM is **incorrect convergence**. The BAM may not always produce the closest association. In fact, a stable association may be only slightly related to the initial input vector.

© Negnevitsky, Pearson Education, 2002

Associative Network Types

1. Auto-associative: X = Y

*Recognize noisy versions of a pattern

2. Hetero-associative Bidirectional: X <> Y

BAM = Bidirectional Associative Memory

*Iterative correction of input and output

© Negnevitsky, Pearson Education, 2002