Using Neural Networks for Classification on Real Data

Kevin Sim

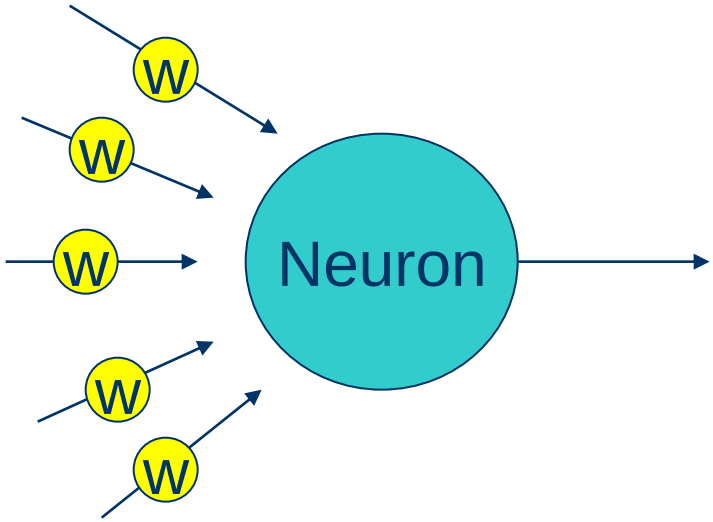
Slides courtesy of Prof. Emma Hart

Overview

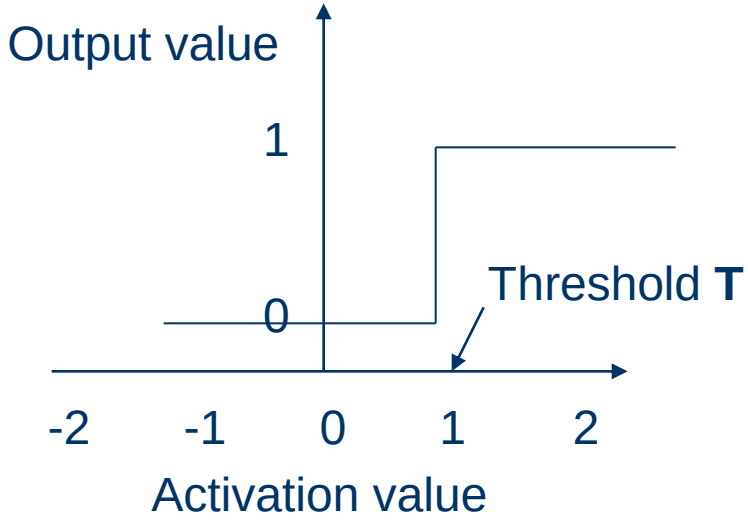
- Recap
- What kind of problems can we use NNs for
- Dealing with data
- Designing the neural network
- Training the network
- Backpropagation
- Testing the network

Recap Simple Perceptrons

if $\sum w_i x_i \geq T$ **output 1, otherwise 0**

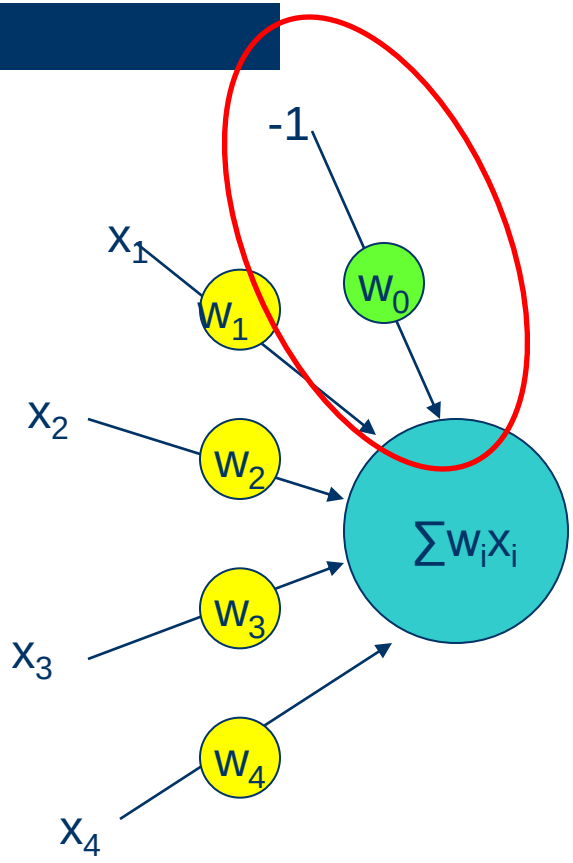


Step Function:



Recap Simple Perceptrons

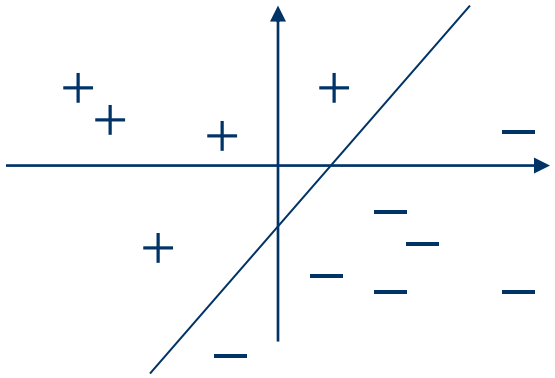
- We can ignore the threshold by adding an extra input to those required by the application
- The extra input is called the **bias**
- *It always has input **value -1**
- It has a **weight w_0** which needs to be calculated

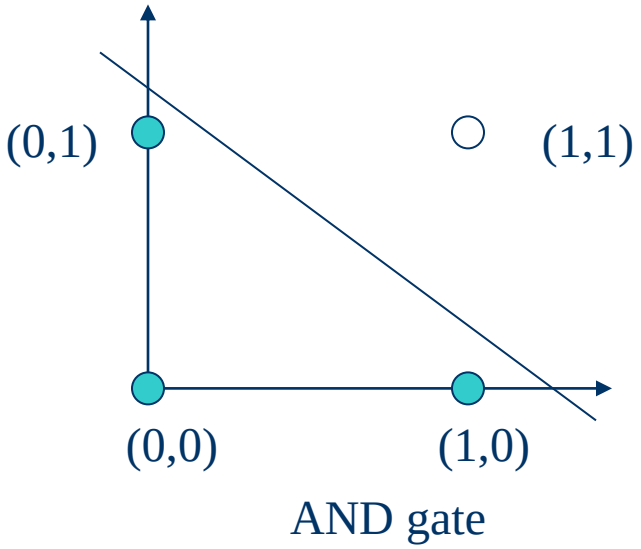


that in practice, most software assumes the input is actually +1 and the weight is –w (which achieves the same effect)

Recap Simple Perceptrons

- From last week: Simple Perceptrons are only useful if data is **linearly separable**





AND gate

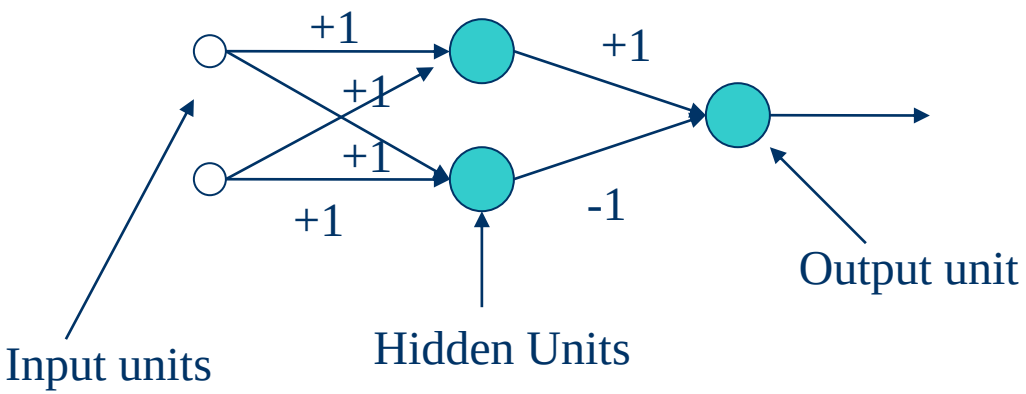
Multi Layer Perceptrons

Most problems involve data that isn't linearly separable

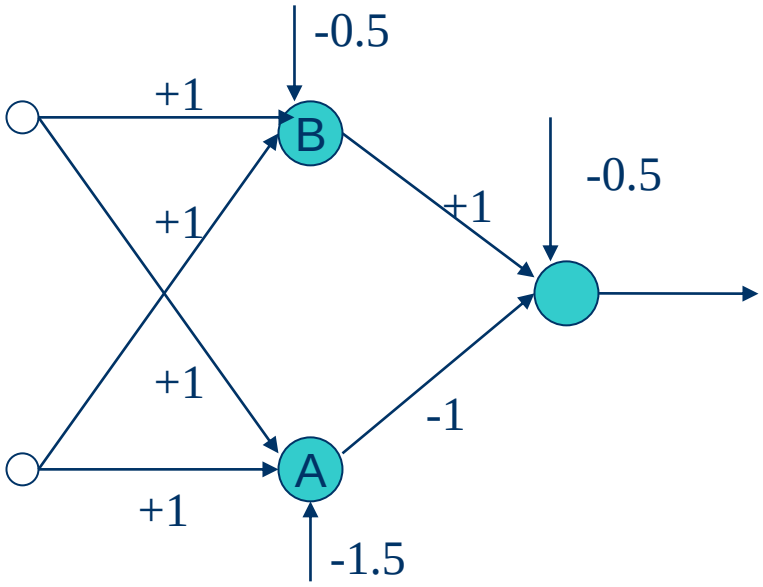
- Single Perceptrons can be combined into multi-layer networks
- In theory – any function can be approximated in this way

Multilayer perceptrons

- A network that solves XOR
  - *3 neurons are used*
  - *2 in a 'hidden layer' and 1 for the output*



Learning XOR

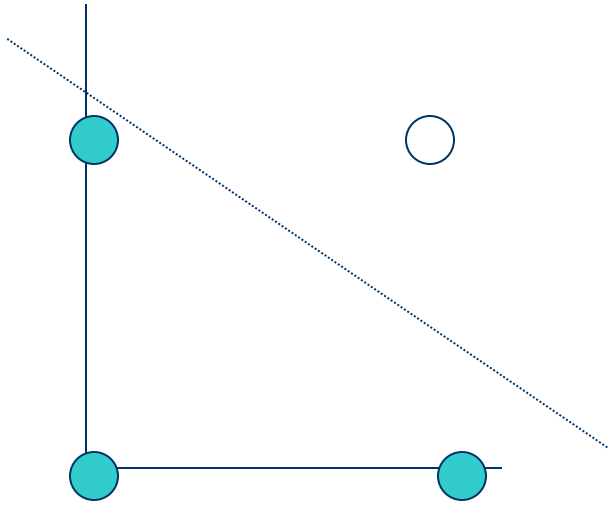


input		Hidden Unit A	Hidden Unit B	output
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Output of hidden units becomes **input** to output unit

Role of the Hidden Units

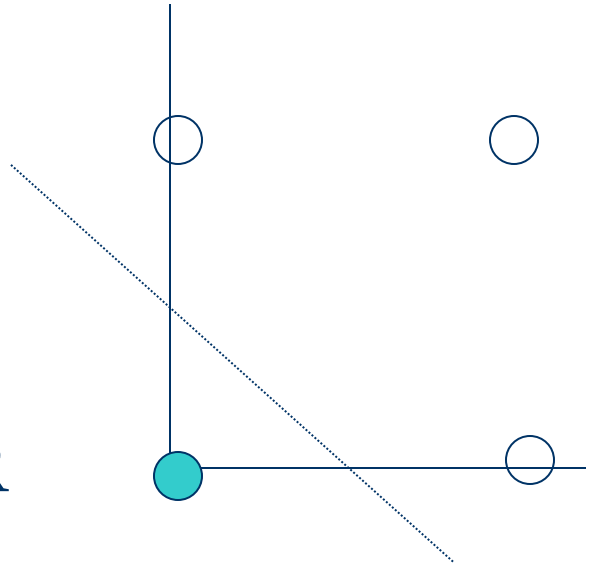
input		Hidden Unit A	Hidden Unit B	output
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0



Hidden Unit A is learning AND

Role of the Hidden Units

input		Hidden Unit A	Hidden Unit B	output
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0



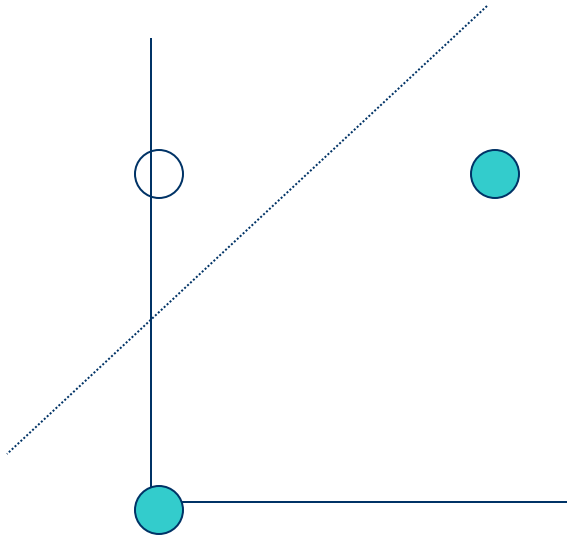
Hidden Unit B is learning OR

Effect of Hidden Units on Output

input		Hidden Unit 1	Hidden Unit 2	output
1	1	1	1	0
1	0	0	1	1
0	1	0	1	1
0	0	0	0	0

Output unit's *input* is the *output* of hidden units

Note that now there are only 3 possible input pairs



Training multilayer Networks

- The simple perceptron learning rule can't be applied to a multi-layer network
  - As there is a knock-on effect with the output of one node being the input of the next
- For any given input pattern, we can only easily calculate the error at the final output node(s)
  - we don't know what the outputs for the hidden nodes should be
- There are 2 ways to find the required weights:
  - **Backpropagation** (for supervised learning, this week)
  - Evolutionary Algorithm (unsupervised learning, next week)

Classification with MLPs

- **What kind of problems can we use NNs for**
- Dealing with data
- Designing the neural network
- Training the network (for classification)
- Backpropagation
- Testing the network

What kind of problems are NNs good at ?

- Instances represented by many input-output pairs
- Target function can be discrete-valued, real-valued, or a vector of real/discrete attributes
- Training examples can contain errors
- Long training times area acceptable
- Fast evaluation of the learned target functions is required
- The ability of humans to understand the learned target function is not important

And when they are not very good ....

- If you need to understand the reasons behind output:
  - E.g medical applications – predicting a heart attack
- No relationship between input and output variables
  - E.g day of the week -> weather forecast
- No relationship between the past and the future
  - E.g predicting the lottery numbers for next week
- Training data is very biased:
  - Only have examples from one particular class (e.g trying to predict class of degree but only have examples of students who obtained 2(1) and 3rd.)

Dealing with data

- What kind of problems can we use NNs for
- **Dealing with data**
- Designing the neural network
- Training the network (for classification)
- Backpropagation
- Testing the network

Dealing with data

- Factors to consider
  - Available data
    - categorical, numerical, discrete, ordered
  - input encoding
    - Transforming the data
  - output encoding

Dealing with data

- **Real world data is likely to be messy**
- Neural networks need numerical inputs
- Neural networks work best with inputs in range 0-1

- Might have missing values for some attributes
- Might have irrelevant values
- Need to clean a data-set before trying to use with an NN

Dealing with data

- Real world data is likely to be messy
- **Neural networks need numerical inputs**
- Neural networks work best with inputs in range 0-1

- Real data has attributes of mixed type:
- Continuous (1.34, 4.78 etc.)
- Discrete (1,2,3)
- Categorical (red, blue, yellow)

Dealing with data

- Real world data is likely to be messy
- Neural networks need numerical inputs
- **Neural networks work best with inputs in range 0-1**

- Real data has varied ranges:

  0 to 10,000

  -100 to 100

Example

- Let's use an example to illustrate how we can deal with data:
- An estate agent want to use a neural network to predict house prices
- Input = collected data
- Output = price of house

Attribute	Type	Values
Location Rating	discrete	1-10
House Condition	discrete	1-10
Bedrooms	discrete	1-8
Bathrooms	discrete	1-6
Living Area	continuous	$50m^2$-$250m^2$
Land Area	continuous	$100m^2$-$1000m^2$
Garage	categorical	yes/no
Heating System	categorical	Wood, oil, gas, electric
PRICE	continuous	52,500-225,000

Dealing with data: Continuous

- Continuous data that varies between two preset values (minimum and maximum) can easily be *normalised* to gives values between 0 and 1
  - (WEKA does this for you automatically)

$$\text{New value} = \frac{\text{actual value - minimum value}}{\text{maximum value - minimum value}}$$

An Example

- Area of houses in the training set varies between 59 and 231 square metres
- Set minimum to 50 and maximum to 250
  - Leave some room at extremes for values that might not appear in your data
- An area of say 121m^2 is then mapped to

$$\text{New value of } 121 = \frac{121-50}{250-50} = 0.355$$

How do we convert the output to a price ?

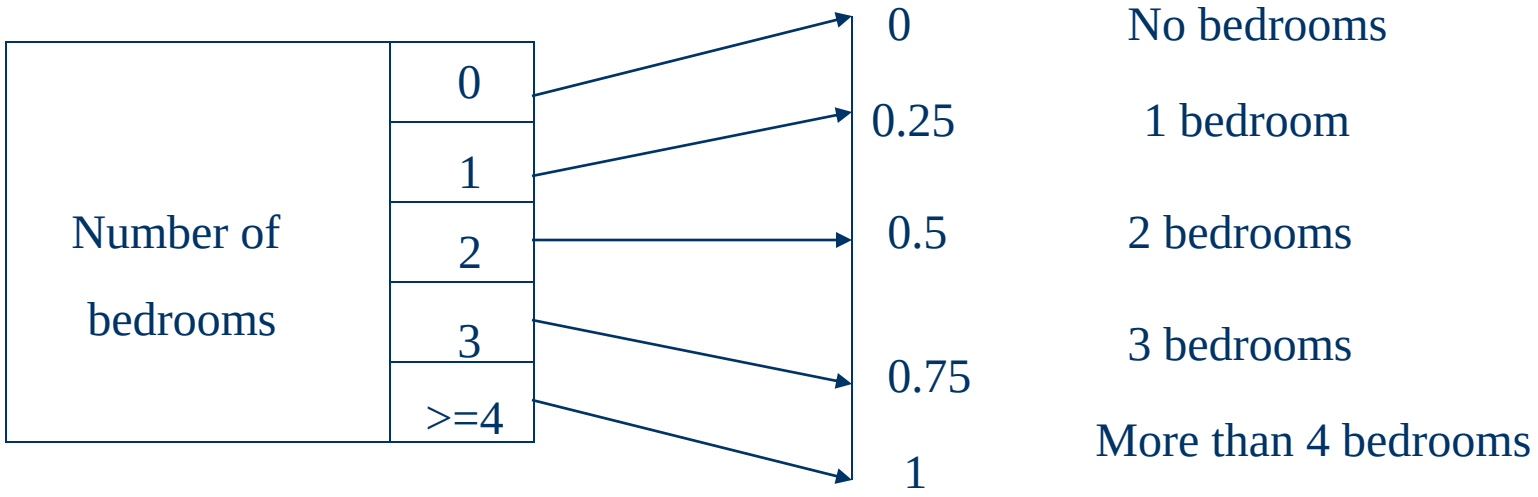
- We can do the same for the output:
- Prices vary between £52500 and £225,000
- Set up output so that £50000 maps to 0, and £25000 0 maps to 1

$$\text{output} = \frac{120{,}920 - 50{,}000}{250{,}000 - 50{,}000} = 0.3546$$

Note that the NN is unlikely to be good on new data that is outside of the boundaries you set so choose them carefully

Discrete Data

- Discrete data such as number of bedrooms, number of bathrooms, also has minimum and maximum values
- Convert to numerical values on a scale:



Categorical Data

- For data with a fixed number of categories *n*, represent as n attributes where one attribute is 1 and the rest are 0
- This is called a 1-of-N encoding (it uses N inputs of which 1 is set to 1)

Wood	1	0	0	0
Oil	0	1	0	0
Gas	0	0	1	0
Electric	0	0	0	1

Categorical Data

- Don’t convert to a scale!
  - Wood 0.2
  - Oil 0.4
  - Gas 0.6
  - Electric 0.8
- Why ?
  - It introduce as false ordering into the data
  - In categorical data, there is no ordering

Categorical Data

- Also avoid converting to a ‘binary’ representation
- E.g the 4 categorical values could be represented by 2 neurons:
  - Wood 00
  - Oil 01
  - Gas 10
  - Electric 11
- Again this introduce a false notion of order and relationship

Dealing with Data

- Using these rules, transform each record in your data set to a suitable format

location	condition	bedrooms	Bathrooms	Living area	Land area	Garage	Heating	PRICE
7	8	3	1	121	682	yes	oil	£120,920

location	condition	bedrooms	Bathrooms	Living area	Land area	Garage		Heating				PRICE
0.7	0.8	0.75	0.5	0.355	0.468	0	1	0	1	0	0	0.3546

Dealt with data!

- Now we have transformed each record in the dataset, we are ready to **design the neural network**
- **Note:**
  - WEKA will default to normalise all numeric data (in the GUI or the libraries)
  - It will automatically use the 1-of-N representation for nominal data
  - If for some reason you want to do it before using a classifier, you can do it in Excel

DESIGNING THE NEURAL NETWORK

Neural Network Architecture

- A neural network design has three steps:
  1. Select number of input neurons
  2. Select number of output neurons
  3. Select number of hidden layers and number of neurons in each hidden layer

Steps 1 and 2

- The first two steps are easy:
- The number of input and output neurons are determined by our transformed data:
  - Input neurons = 12
  - Output neurons = 1

location	condition	Bedrooms	Bathrooms	Living area	Land area	Garage		Heating				PRICE
0.7	0.8	0.75	0.5	0.355	0.468	0	1	0	1	0	0	0.3546

input

output

Hidden Layers

- How many hidden layers ?
  - **Too many makes computation too difficult**
  - **Too few cannot detect complex patterns**
- Weka uses a heuristic to get a start point:
  - 1 hidden layer with (inputs+outputs)/2 neurons
- Start with a small number and increase as necessary:
  - E.g start with 1 hidden layer
  - Choose small number of neurons
- This is not an exact science…
  - There are no rules – other than trial and error

Ockhams Razor

- 14th century philosopher William of Ockham stated:
  - The simplest hypothesis is preferred
- Lots of hidden layers and neurons maps to a very complex function
  - Need to take care that we are not learning the data exactly
  - Rather, we are learning patterns from the data so the neural network can generalise

TRAINING THE NETWORK

Training

- You have some data and an architecture…
- You now need to **train** the network
  - Training means teaching the network to find patterns in a data-set
  - Once it's learnt, the network should produce correct classifications on **unseen** data

Dealing with data first

- First, split the data in two separate sets:
  - A training set and a test set
  - Usually in the proportion ~70:30

Used to train the network to recognise patterns

Used to test the trained network to see if it has generalised

- Why ?
  - We can only test if the network has generalised properly by giving it unseen data

Dealing with data first

- Be careful to ensure that the training and test sets both contain examples of each of the classes you are trying to predict
  - The network can't learn without examples
- Typically, place every 5th record in data set into training or select randomly
- *E.g in house data, training set should contain examples of houses in all price ranges (low, medium, high)*
- *In a credit-application example, training set needs to contain both yes and no examples*

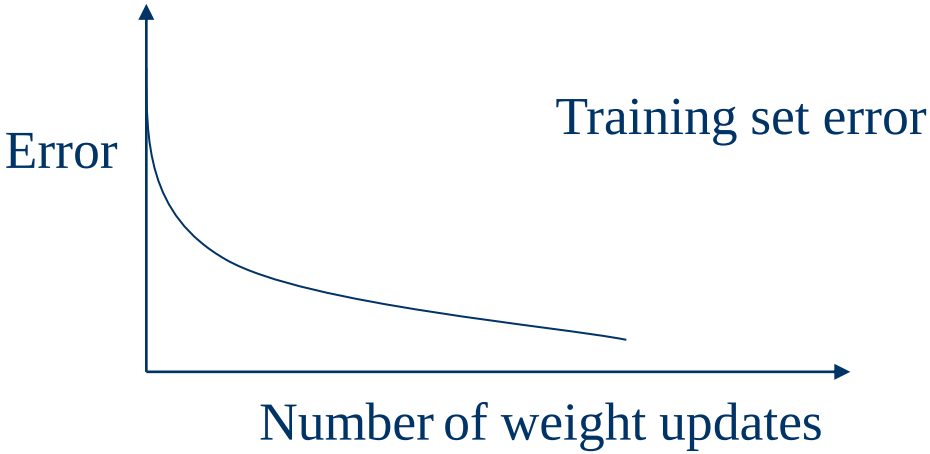
Training the Network

1. Take the prepared training set of examples:
2. For each record in the training set:
   a) Pass the input data through the network
   b) Measure the (squared output) error at the output by comparing the actual result to the desired result
   c) Apply a learning algorithm to change weights
   (usually backpropagation..)
3. Repeat set (2) until error converges

(Remains stationary or reaches a value close to zero)

Training Procedures

- Usually stop training when the network converges, but…

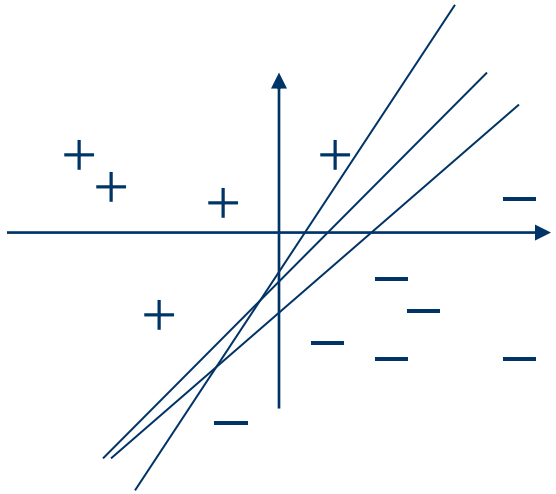


-it is important to beware of **overfitting**

What is overfitting ?

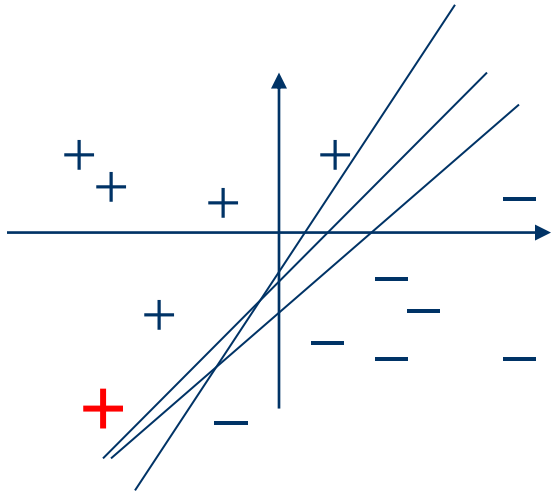
- It is possible to ‘overtrain’ a network
  - E.g teach it to the extent it simply memorises the patterns in the test set rather than generalises from them
  - Occurs it you train too long or have too many hidden nodes
- In this case, you will get **excellent** performance on the **training** set (very small error) but **very weak** performance when you try the **test set**
- For example:
  - Consider a network you are training to recognise letters in a printed document
  - An overfitted network trained on Helvetica font might not recognise the same letter in the Times Roman font

What is overfitting ?



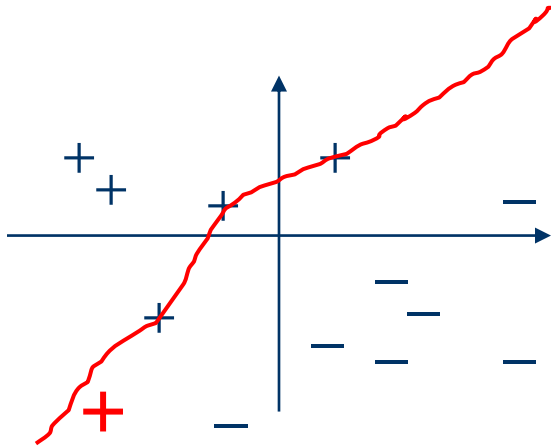
- We want to learn patterns in the data
- So that we can generalise to unseen instances

What is overfitting ?



- We want to learn patterns in the data
- So that we can generalise to unseen instances

What is overfitting ?



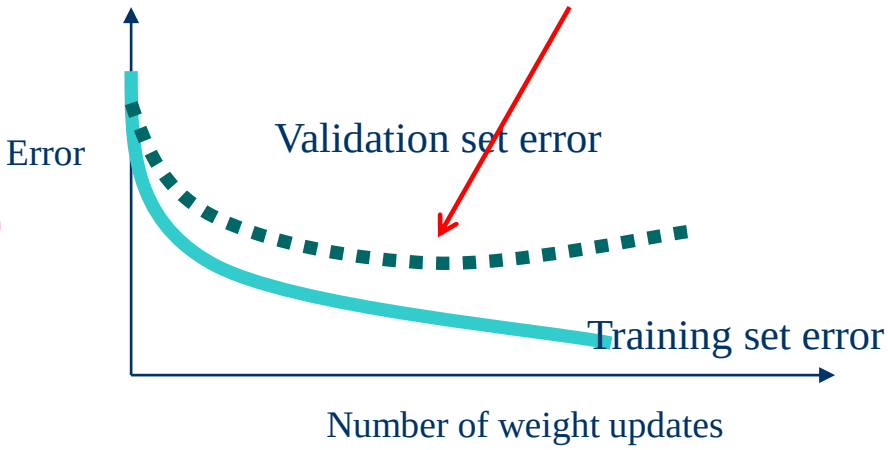
- We don’t want to model the data exactly
- Our network will perform badly when presented with unseen instances

Preventing overfitting

Divide the training data into a *training* set and *validation* set

- Update weights based on error in **training set**
- **Monitor** how well the network performs on a separate **validation set**
  - **But don't adjust any weights based on this performance**
- Choose the set of weights which give the lowest error on the validation set

At this point here, validation error starts to increase, indicating the network is starting to memorise training data



Training the Network

- What if it doesn't converge ?
- Change the architecture
  - increase number of neurons in the hidden layer (start with 2, increase to 5,10, 20….)
- Change the learning parameters….

BACKPROPAGATION

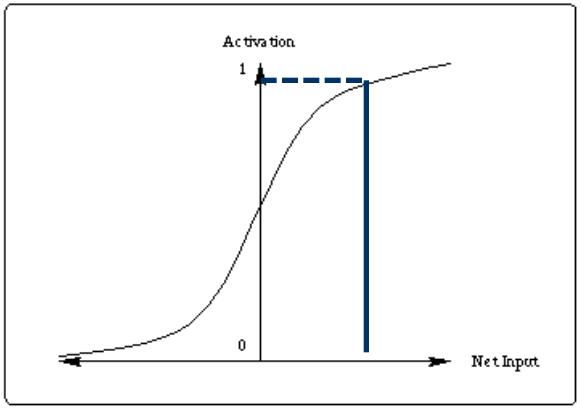
Training multilayer Networks

- The simple perceptron learning rule can't be applied to a multi-layer network
  - As there is a knock-on effect with the output of one node being the input of the next
- For any given input pattern, we can calculate the error at the final output node(s)
- Given this error, we need to apportion blame across all the weights in the network

How do the weights affect the error ?

Error = desired output – actual output

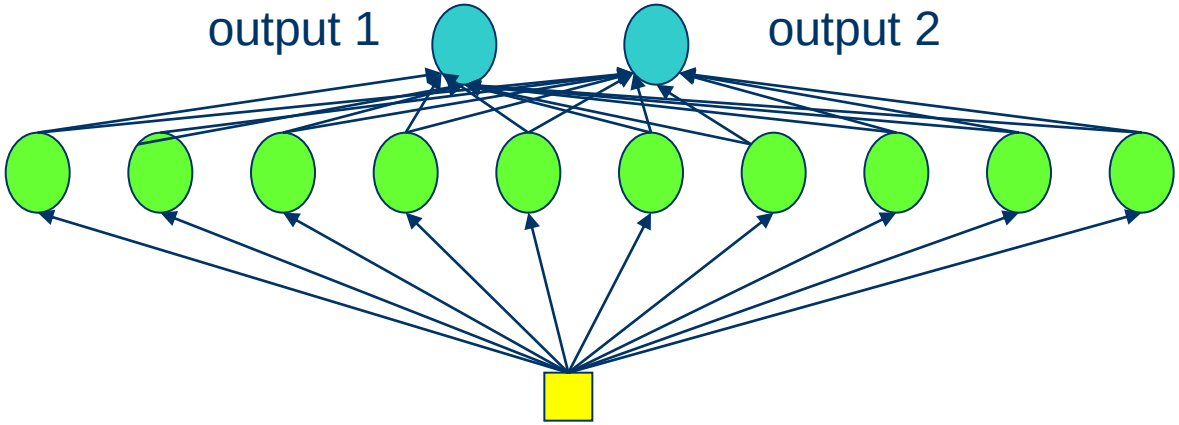
- If the output it too high:
  - Reduce the weights a bit
- If the output is too low
  - Increase the weights a bit
- Easy for output neurons
  - **We know what the desired output is!**



What about hidden neurons ?

Error = desired output – actual output

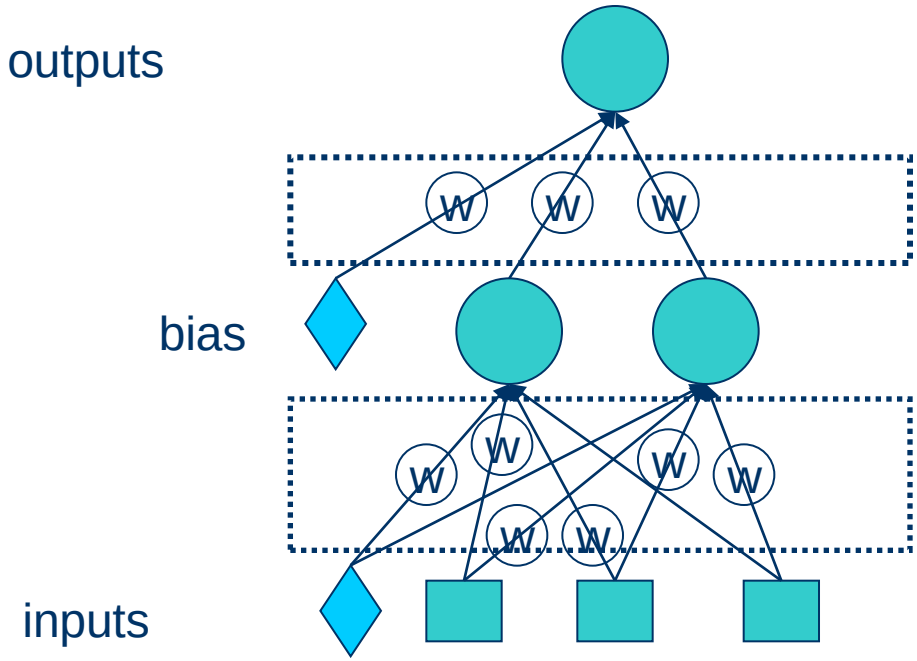
- What **should** the hidden neurons output ?
- We don't know!!



Backpropagation

- Backprop is a technique that calculates the errors and adjusts weights accordingly
- It calculates the error at the outputs
  - Adjust the weights to the output layer so that if the same input pattern is presented again, the error will be reduced
  - Repeat the process at the next layer back
  - And so on until you reach the first layer
  - The next time the same input pattern is presented, the error should be smaller
  - Repeat…….

Back propagation



Calculate error

↓

Readjust weights

↓

Calculate error

↓

Readjust weights

↓

Present input pattern(s)

Backpropagation

The Backprop Algorithm:

- For each pattern presented it:
  - Calculates error at output (easy)
  - Changes weights to output layer to reduce this error
  - Calculates error at last-but-one layer (harder)
  - Changes weights to reduce this
  - Repeat….
- The algorithm is applied each iteration until the network error is minimised

Backpropagation

- Initialize network weights (often small random values)
- **do forEach** training example
  - prediction = neural-net-output // forward pass
  - actual = known_value_from_training_record
  - compute error (prediction - actual) at the output units
  - compute d(w) for all weights from hidden layer to output layer *// backward pass*
  - compute d(w) for all weights from input layer to hidden layer *// backward pass continued*
  - update network weights
- **until** all examples classified correctly or another stopping criterion satisfied
- **return** the network

How does it actually work ?

- In practice with a lot of complicated differential equations!
  - One set calculates the error at the output
  - Another calculates how to adjust the weights to the output
  - Another set calculates error in the hidden layers
  - Another set defines how to adjust weights to hidden layers

Backpropagation

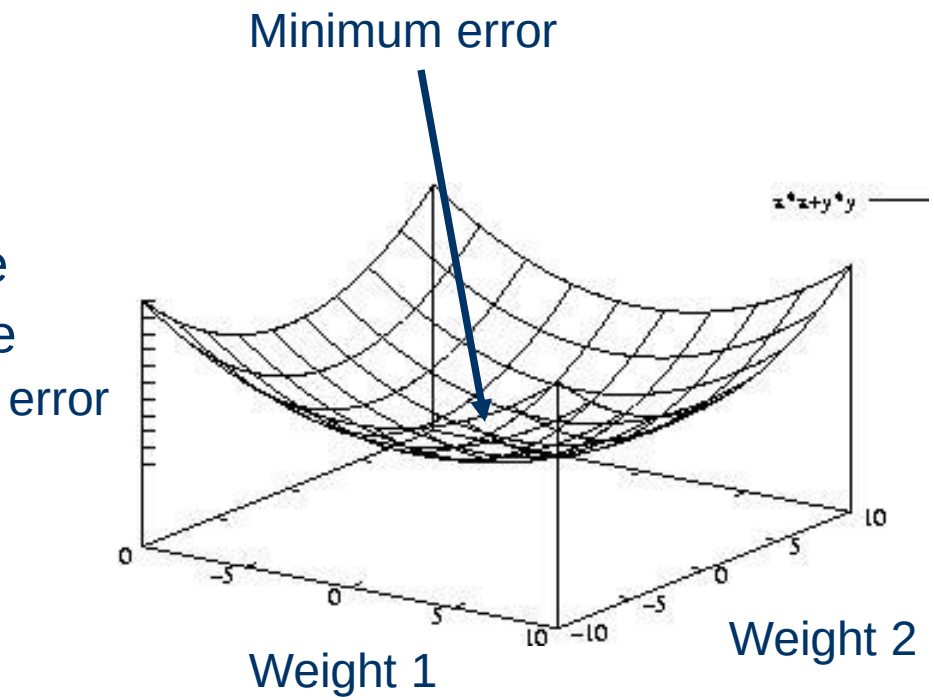
- At each iteration, the weights are adjusting according to an equation of the form:

$$w_{j,k} \leftarrow w_{j,k} + \alpha \times a_j \times \Delta_k$$

  - where α = learning rate
  - a_j = input to k_{th} neuron (output of j_{th} neuron)
  - Δ_k is the error at the j_{th} neuron
- α ***, the learning rate***, can be controlled by the user
  - Adjusting the value (between 0 and 1) controls how fast the backprop algorithm finds the correct weights

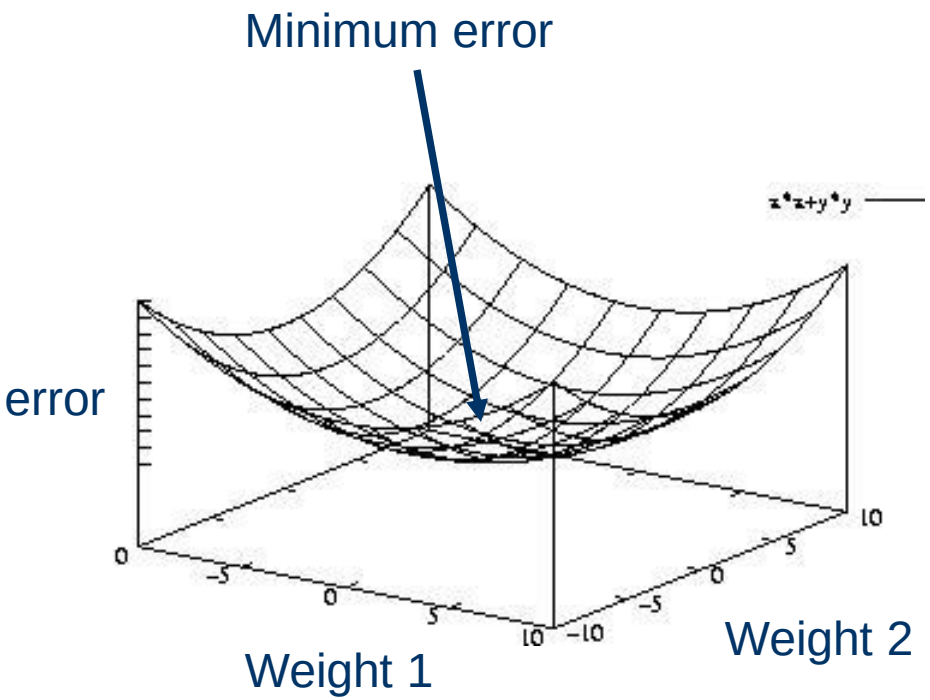
The Backpropagation technique

- Each combination of weights has an associated error
- If we could plot the high-dimensional surface of weights vs error we would have some complex surface
- Backprop is trying to find the minimum point on this surface
  - It adjust weights so that we move down the slope toward the minumum



The Backpropagation technique

- The learning rate α affects the size of the step we take down the slope
- Too small:
  - Might never reach the minum
- Too big:
  - Might overshoot

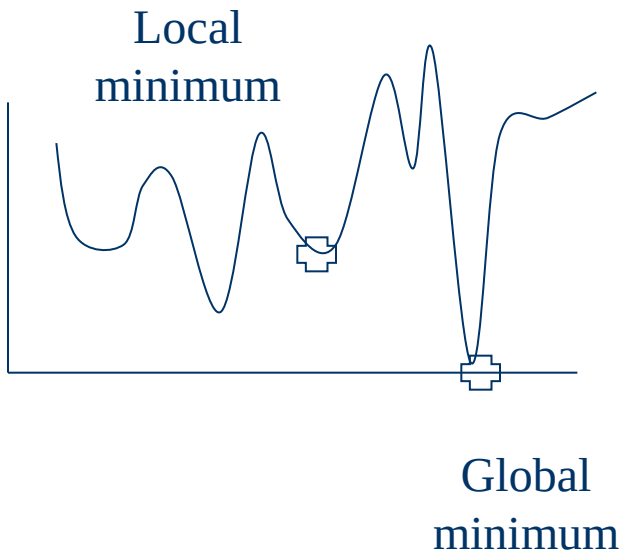


Another parameter

- Another version of backpropagation adds something called **momentum** to try and speed up learning
- Let’s look at why:

Another parameter

- The error surface might have **local optima**
- These are minima in the error surface – but are not the **global** minimum
- Momentum works by adding an extra term to the weight adjustment that encourage the algorithm to adjust the weights in the same direction as the previous step



$$w_{j,k} \leftarrow w_{j,k} + \alpha \times a_j \times \Delta_k + \lambda(\Delta w`)$$

- Typically set between 0.0 and 1.0

In practice

- All NN software/libraries has backpropagation built in
  - You don’t need to program it!
- As the software engineer, you have to supply two things:
  - The learning rate
  - The momentum term (if required)
  - Can be done in WEKA either through the GUI or in code

In practice

- It is very unlikely that you will get a good result (low error on the training set) the first time you try and train the network
- The training procedure can be time-consuming:
  - Try different architectures
  - Try different learning rates
  - Try different momentum values
  - Try representing the data differently
  - ……

TESTING

Testing

- You've prepared the data
- Trained the network
- The training error is reduced to almost zero

How well does your network work on the test data ?

Testing Procedure

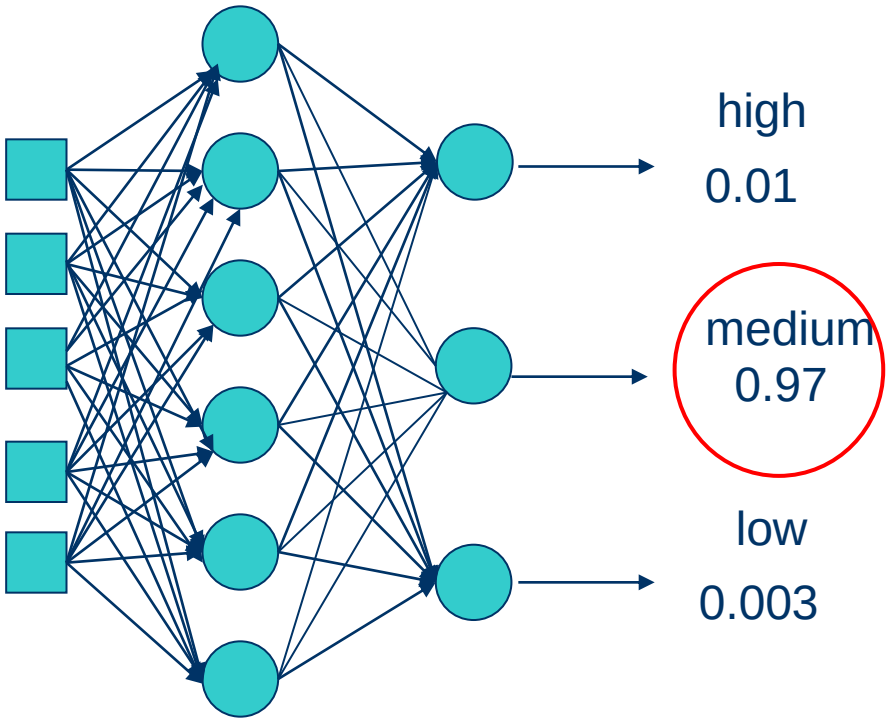
- Take your trained network
- For each record in the unseen test-set:
  - Feed the inputs into the trained network
  - Record the output for each input
  - Compare against the known desired output
- How can we quantify the performance ?

Quantifying performance:

- For **real-valued output we** can calculate the **total squared error** across the whole test set for each network we have trained
- The network with minimum error is preferred
- We might define some acceptable tolerance level depending on the application

Quantifying data:

- For some applications, we want to predict a class:
- Example: a bank rates a customer's credit risk as high medium or low depending on some collected data
- Output might be 3 neurons:
  - Neuron with highest value is the winner



Classification Rate

- In this case we can count the percentage of records for which the correct class was predicted and report that
- We can also report % correct prediction for each of the 3 classes

- Overall classification rate: **80%**

- % High 100%
- % Medium 80%
- % Low 60%

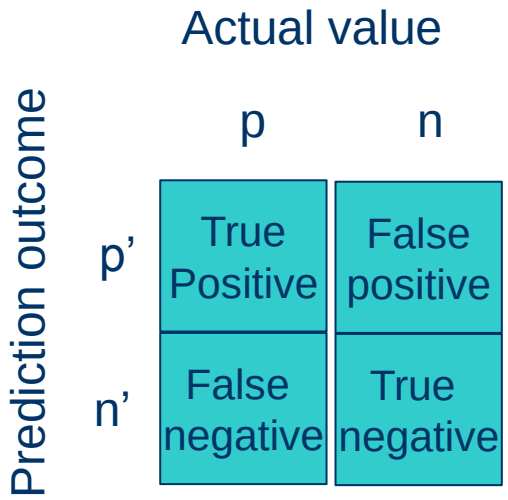
More about classification

- Consider a binary classifcation problem: e.g predicting attack/non-attack in virus detection:
  - There are P records which are attack and N records that are non-attack
- **True positive (TP)**
  - % of records classified as **attack** that were really **attack**
- **False positive (FP)**
  - % of records classified as **attack** that were **not attack**
- **True Negative (TN)**
  - % of records classifed as **non-attack** that were **non-attack**
- **False Negative (FN)**
  - % of records classified as **non-attack** that were **attack**

Illustrating Performance

- Sensitivity (sometimes called recall):
  - TP/(TP+FN)
- Accuracy
  - (TP+TN)(P+N)
- Specificity
  - TN/(FP+TN)

Confusion Matrix

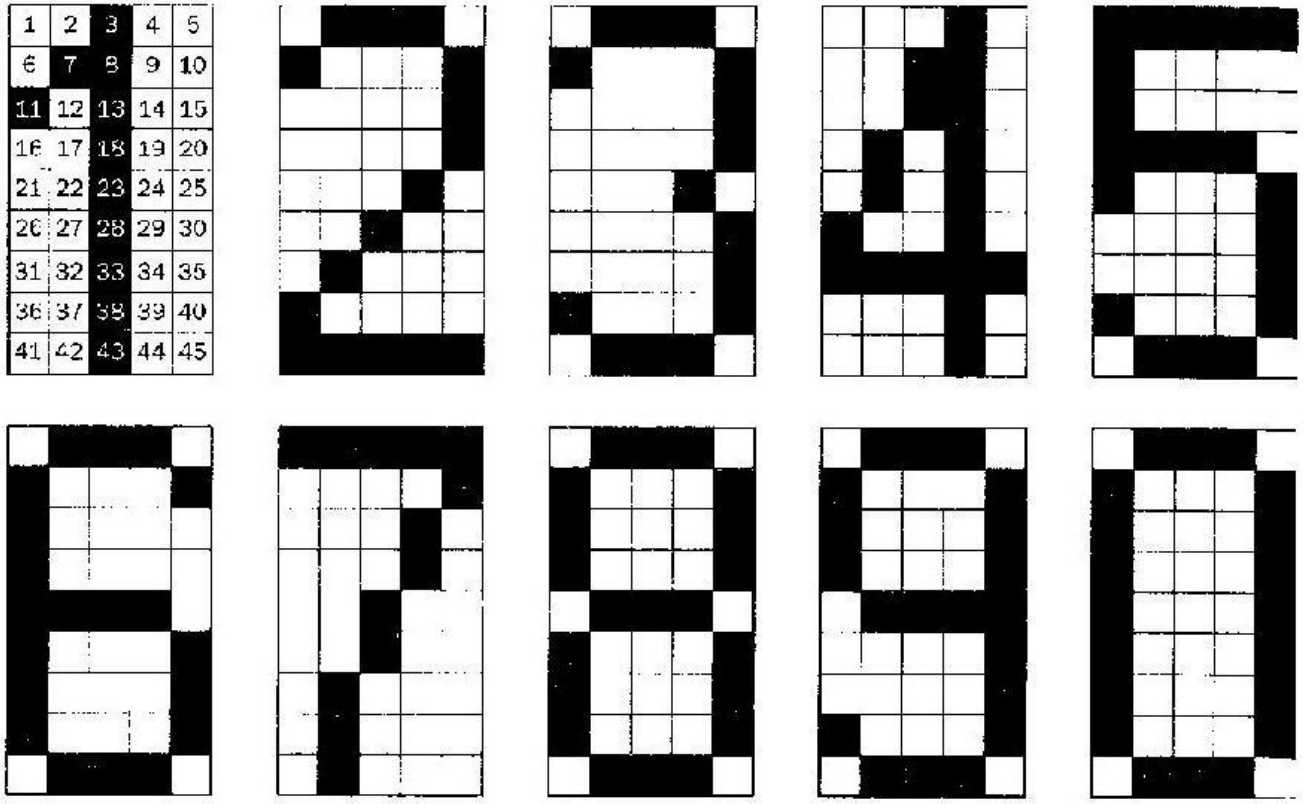


SOME EXAMPLES

Character Recognition Neural Networks

- Recognition of printed or handwritten characters is a typical NN application
- Documents can be scanned and then edited without retyping
  - Scanning divides the image into hundreds of pixel sized boxes
  - Each box is represented by a 0 or 1 (empty of full)
  - Resulting matrix of dots is called a bitmap
  - NNs learn to recognise the bit-maps as characters

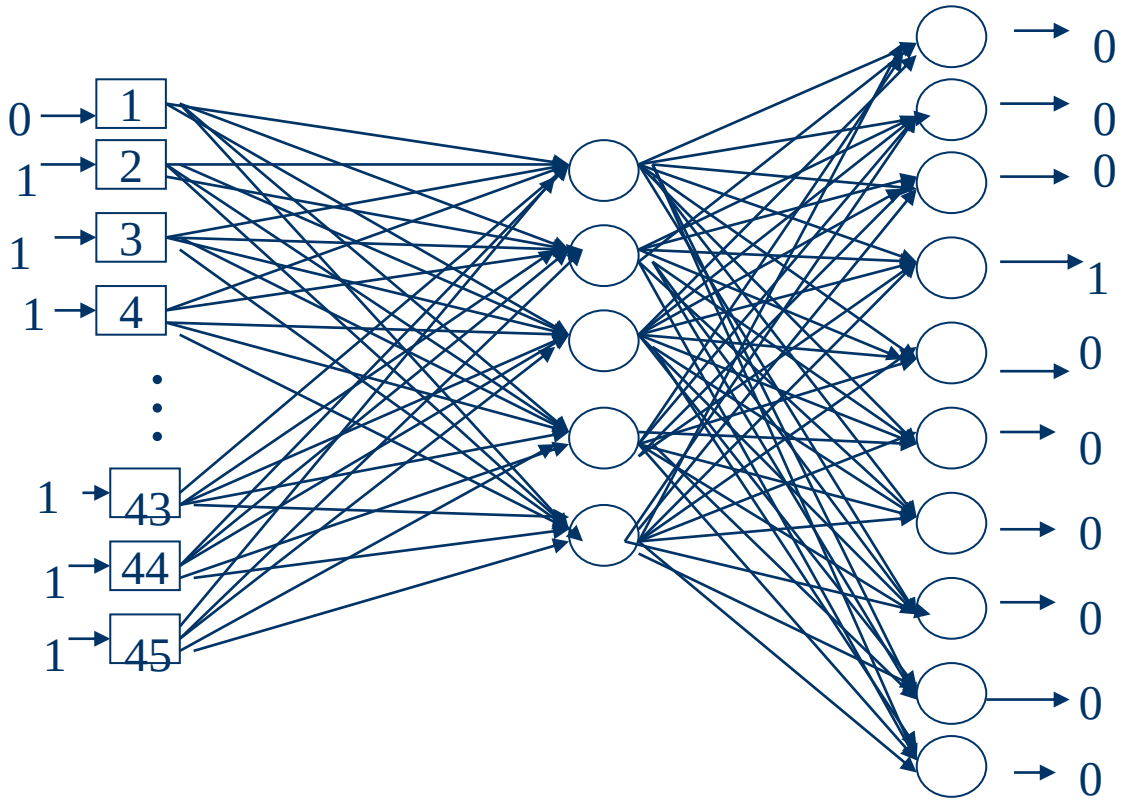
Character Bit Map Input Data



Character Recognition

- Task is to recognise digits 0-9
- Each digit represented by a 5x9 bitmap
  - 1 neuron for pixel = 45 input neurons, with value 0 or 1
- (In commercial applications, use higher resolution, e.g 16x16)
- Network must output the correct digit, given some (possibly noisy) input
  - 10 output neurons, one has value 1, the others 0

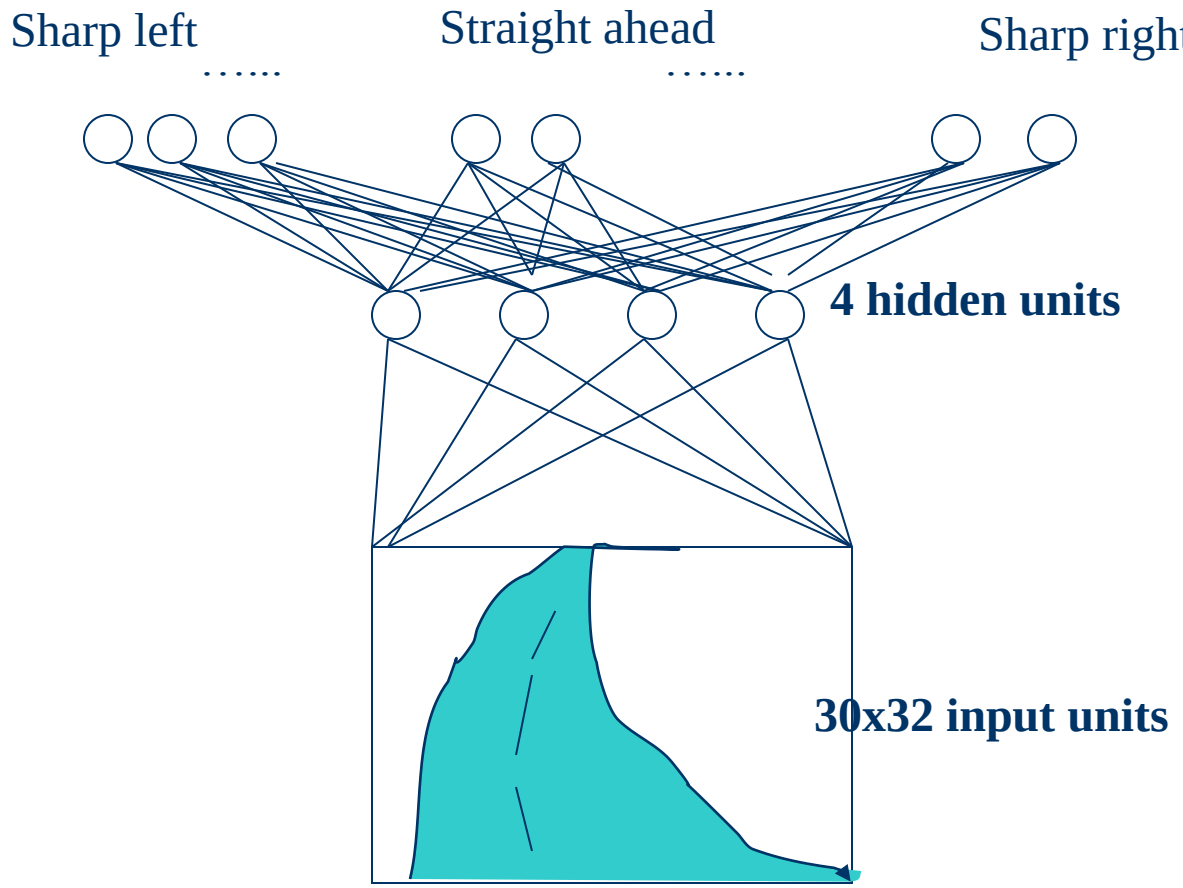
Neural Network for Digit Recognition



ALVINN - another real example

- NN to steer an autonomous vehicle driving at normal speeds on public roads
- Input is 30x32 grid of pixel intensities obtained from a camera (real values)
- Output is direction to steer - there are 30 outputs corresponding to particular directions, winner is output with highest value
- ALVINN has been used to drive at speeds of 70 mph for 90 miles (in left hand lane!) on public highways with other vehicles present

ALVINN


Sharp left
Straight ahead
Sharp right
4 hidden units
30x32 input units

Summary

In this lecture you have learned

- What kind of problems can we use NNs for
- Dealing with data
- Designing the neural network
- Training the network
- Backpropagation
- Testing the network