UFME7K-15-M Intelligent and Adaptive Systems

Introduction to Artificial Neural Networks

Charlie Sullivan

7 February 2018

Intelligent Adaptive Systems Lecture 3

Dr J. Charlie Sullivan

charlie.sullivan@brl.ac.uk

What is an ANN?

- A parallel computing device inspired by natural networks of neurons (brains)

What is an ANN?

- A parallel computing device inspired by natural networks of neurons (brains)
- Consist of many interconnected processing units (nodes or neurons)

What is an ANN?

- A parallel computing device inspired by natural networks of neurons (brains)
- Consist of many interconnected processing units (nodes or neurons)
- Each processor is aware only of signals it receives from its input connections (analogous to synapses in biological neurons)

What is an ANN?

- A parallel computing device inspired by natural networks of neurons (brains)
- Consist of many interconnected processing units (nodes or neurons)
- Each processor is aware only of signals it receives from its input connections (analogous to synapses in biological neurons)
- Each processor has very low processing "power"

What is an ANN?

- A parallel computing device inspired by natural networks of neurons (brains)
- Consist of many interconnected processing units (nodes or neurons)
- Each processor is aware only of signals it receives from its input connections (analogous to synapses in biological neurons)
- Each processor has very low processing "power"
- ANNs can learn or be trained

Some real Neurons

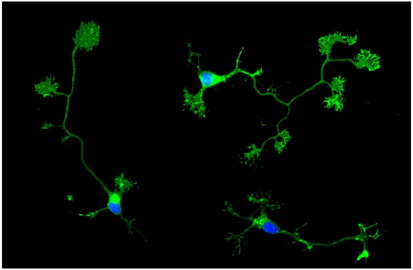


Figure 1: Hippocampal Neurons

- For some reason, no lecture on ANN is complete without a picture of some real neurons

Some real Neurons

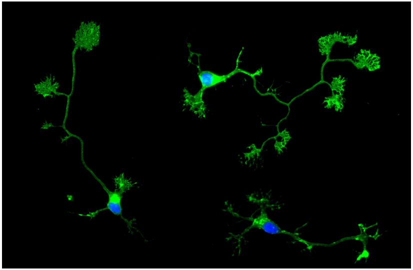


Figure 1: Hippocampal Neurons

- For some reason, no lecture on ANN is complete without a picture of some real neurons
  - I don't know why

Some real Neurons

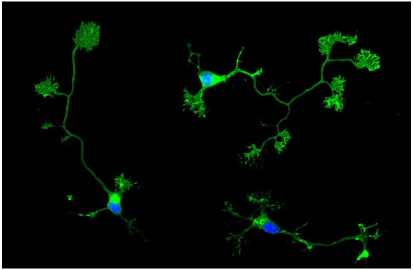


Figure 1: Hippocampal Neurons

- For some reason, no lecture on ANN is complete without a picture of some real neurons
  - I don't know why
  - Please do NOT attempt to draw real neurons on your exam scripts!

Early History

- 1943, McCulloch and Pitts showed that even simple types of neural networks could, in principle, compute any arithmetic or logical function

Early History

- 1943, McCulloch and Pitts showed that even simple types of neural networks could, in principle, compute any arithmetic or logical function
  - In order to describe how neurons in the brain might work, they modeled a simple neural network using electrical circuits.

Early History

- 1943, McCulloch and Pitts showed that even simple types of neural networks could, in principle, compute any arithmetic or logical function
  - In order to describe how neurons in the brain might work, they modeled a simple neural network using electrical circuits.
- 1949, Donald Hebb wrote The Organization of Behavior,

Early History

- 1943, McCulloch and Pitts showed that even simple types of neural networks could, in principle, compute any arithmetic or logical function
  - In order to describe how neurons in the brain might work, they modeled a simple neural network using electrical circuits.
- 1949, Donald Hebb wrote The Organization of Behavior,
  - neural pathways are strengthened each time they are used,

Early History

- 1943, McCulloch and Pitts showed that even simple types of neural networks could, in principle, compute any arithmetic or logical function
  - In order to describe how neurons in the brain might work, they modeled a simple neural network using electrical circuits.
- 1949, Donald Hebb wrote The Organization of Behavior,
  - neural pathways are strengthened each time they are used,
  - a concept fundamentally essential to the ways in which humans learn.

Early History

- 1943, McCulloch and Pitts showed that even simple types of neural networks could, in principle, compute any arithmetic or logical function
  - In order to describe how neurons in the brain might work, they modeled a simple neural network using electrical circuits.
- 1949, Donald Hebb wrote The Organization of Behavior,
  - neural pathways are strengthened each time they are used,
  - a concept fundamentally essential to the ways in which humans learn.
  - If two nerves fire at the same time the connection between them is enhanced.

Early History

- 1943, McCulloch and Pitts showed that even simple types of neural networks could, in principle, compute any arithmetic or logical function
  - In order to describe how neurons in the brain might work, they modeled a simple neural network using electrical circuits.
- 1949, Donald Hebb wrote The Organization of Behavior,
  - neural pathways are strengthened each time they are used,
  - a concept fundamentally essential to the ways in which humans learn.
  - If two nerves fire at the same time the connection between them is enhanced.
- As computers became more advanced in the 1950's, it was finally possible to simulate a hypothetical neural network.

Early History

- 1943, McCulloch and Pitts showed that even simple types of neural networks could, in principle, compute any arithmetic or logical function
  - In order to describe how neurons in the brain might work, they modeled a simple neural network using electrical circuits.
- 1949, Donald Hebb wrote The Organization of Behavior,
  - neural pathways are strengthened each time they are used,
  - a concept fundamentally essential to the ways in which humans learn.
  - If two nerves fire at the same time the connection between them is enhanced.
- As computers became more advanced in the 1950's, it was finally possible to simulate a hypothetical neural network.
  - The first step towards this was made by IBM research laboratories.

Adaptive Linear Element Networks

- 1959, Widrow and Hoff of Stanford developed two models:-

Adaptive Linear Element Networks

- 1959, Widrow and Hoff of Stanford developed two models:-
    - ADALINE was developed to recognize binary patterns

Adaptive Linear Element Networks

- 1959, Widrow and Hoff of Stanford developed two models:-
  - ADALINE was developed to recognize binary patterns
    - if it was reading streaming bits from a phone line, it could predict the next bit

Adaptive Linear Element Networks

- 1959, Widrow and Hoff of Stanford developed two models:-
  - ADALINE was developed to recognize binary patterns
    - if it was reading streaming bits from a phone line, it could predict the next bit
  - MADALINE (Multiple ADAptive LINear Elements) was the first neural network applied to a real world problem,

Adaptive Linear Element Networks

- 1959, Widrow and Hoff of Stanford developed two models:-
  - ADALINE was developed to recognize binary patterns
    - if it was reading streaming bits from a phone line, it could predict the next bit
  - MADALINE (Multiple ADAptive LINear Elements) was the first neural network applied to a real world problem,
    - an adaptive filter that eliminates echoes on phone lines.

Adaptive Linear Element Networks

- 1959, Widrow and Hoff of Stanford developed two models:-
  - ADALINE was developed to recognize binary patterns
    - if it was reading streaming bits from a phone line, it could predict the next bit
  - MADALINE (Multiple ADAptive LINear Elements) was the first neural network applied to a real world problem,
    - an adaptive filter that eliminates echoes on phone lines.
    - while the system is as ancient as air traffic control systems, it is still in commercial use.

The Perceptron

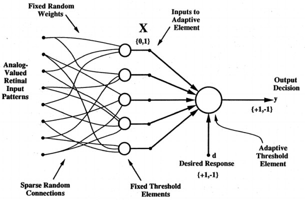


Figure 2: Perceptron

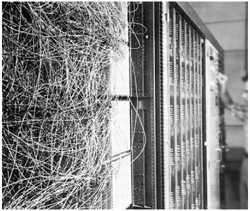
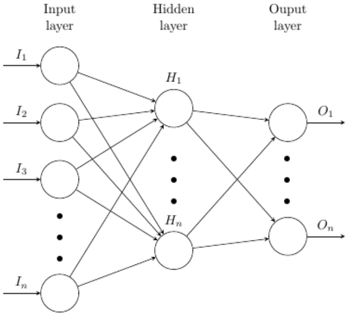


Figure 3: Early prototype

- Rosenblatt's Perceptron (1962)

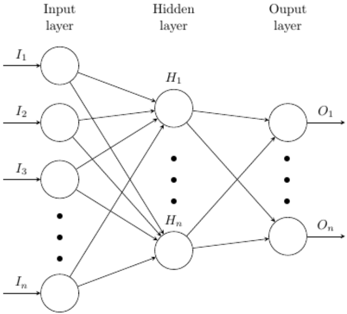
Feedforward Networks (MLP)

- we will concentrate on applications using **multilayer feedforward networks**



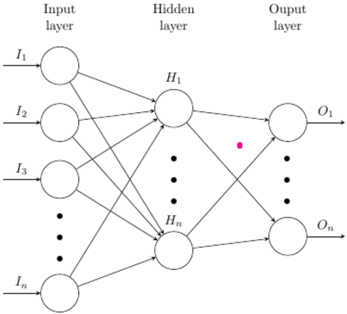
Feedforward Networks (MLP)

- we will concentrate on applications using **multilayer feedforward networks**
- often referred to as Multi-layer Perceptrons (MLP).



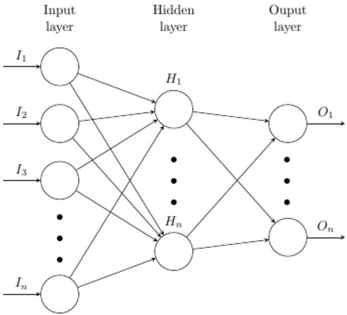
Feedforward Networks (MLP)

- we will concentrate on applications using **multilayer feedforward networks**
- often referred to as Multi-layer Perceptrons (MLP).
- the most widely used type of Neural Network



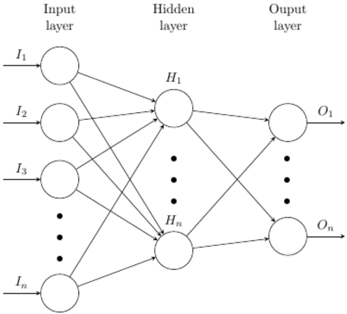
Feedforward Networks (MLP)

- we will concentrate on applications using **multilayer feedforward networks**
- often referred to as Multi-layer Perceptrons (MLP).
- the most widely used type of Neural Network
- comprise a layer of input units,



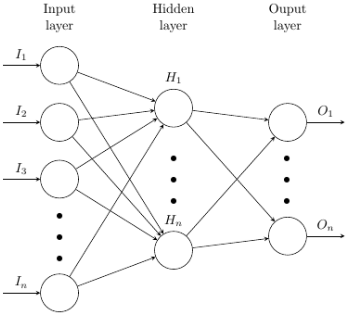
Feedforward Networks (MLP)

- we will concentrate on applications using **multilayer feedforward networks**
- often referred to as Multi-layer Perceptrons (MLP).
- the most widely used type of Neural Network
- comprise a layer of input units,
- a layer of output units



Feedforward Networks (MLP)

- we will concentrate on applications using **multilayer feedforward networks**
- often referred to as Multi-layer Perceptrons (MLP).
- the most widely used type of Neural Network
- comprise a layer of input units,
- a layer of output units
- and one or more **hidden** layers which connect them together.



What types of problem can be solved using Neural Nets?

- Neural nets can be applied in a number of different types of problem.

What types of problem can be solved using Neural Nets?

- Neural nets can be applied in a number of different types of problem.
- MATLAB provides tools to make these simpler, for example:

What types of problem can be solved using Neural Nets?

- Neural nets can be applied in a number of different types of problem.
- MATLAB provides tools to make these simpler, for example:
  - `nftool` for Curve-Fitting or Regression problems

What types of problem can be solved using Neural Nets?

- Neural nets can be applied in a number of different types of problem.
- MATLAB provides tools to make these simpler, for example:
  - `nftool` for Curve-Fitting or Regression problems
  - `nprtool` for Pattern Recognition or Classification problems

What types of problem can be solved using Neural Nets?

- Neural nets can be applied in a number of different types of problem.
- MATLAB provides tools to make these simpler, for example:
  - `nftool` for Curve-Fitting or Regression problems
  - `nprtool` for Pattern Recognition or Classification problems
  - `nctool` for Clustering problems

What types of problem can be solved using Neural Nets?

- Neural nets can be applied in a number of different types of problem.
- MATLAB provides tools to make these simpler, for example:
  - `nftool` for Curve-Fitting or Regression problems
  - `nprtool` for Pattern Recognition or Classification problems
  - `nctool` for Clustering problems
  - `ntstool` for Dynamic Time Series problems

Using the nftool

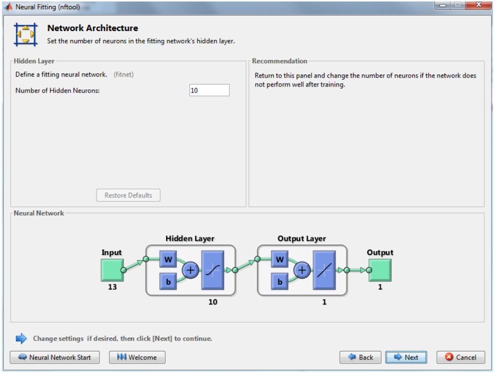


Figure 5: Network architecture

MLP Neuron

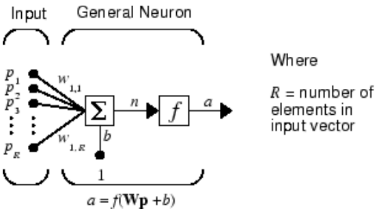


Figure 6: Single MLP Neuron

- Each input is multiplied by an appropriate weight $w_{i,j}$

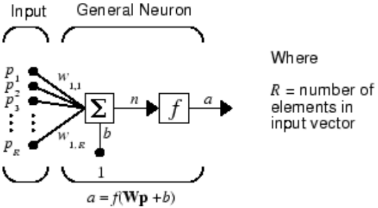


Figure 6: Single MLP Neuron

- Each input is multiplied by an appropriate weight $w_{i,j}$
- The sum of the weighted inputs and the bias forms the input to the activation (threshold) function f

Activation Functions

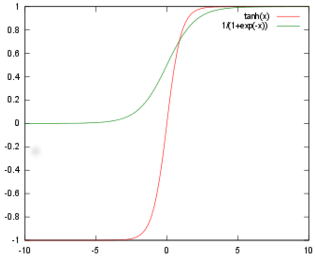


Figure 7: Typical Activation Functions

- Multilayer networks often use the logistic sigmoid transfer function $logsig(x) = 1/(1 + exp(-x))$

Activation Functions

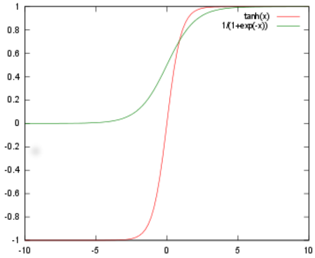


Figure 7: Typical Activation Functions

- Multilayer networks often use the logistic sigmoid transfer function $logsig(x) = 1/(1 + exp(-x))$
- generates outputs between 0 and 1 as the neuron's input goes from negative to positive infinity

Activation Functions

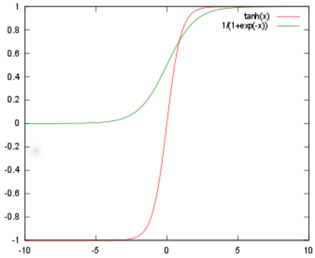


Figure 7: Typical Activation Functions

- Multilayer networks often use the logistic sigmoid transfer function $logsig(x) = 1/(1 + exp(-x))$
- generates outputs between 0 and 1 as the neuron's input goes from negative to positive infinity
- Sigmoid output neurons are often used for pattern recognition problems, while linear output neurons are used for function fitting problems

Network design for fitting

- Feedforward networks often have one or more hidden layers of sigmoid neurons followed by an output layer of linear neurons

Network design for fitting

- Feedforward networks often have one or more hidden layers of sigmoid neurons followed by an output layer of linear neurons
- Multiple layers of neurons with nonlinear transfer functions allow the network to learn nonlinear relationships between input and output vectors

Network design for fitting

- Feedforward networks often have one or more hidden layers of sigmoid neurons followed by an output layer of linear neurons
- Multiple layers of neurons with nonlinear transfer functions allow the network to learn nonlinear relationships between input and output vectors
- The linear output layer is most often used for function fitting (or nonlinear regression) problems

Network design for fitting

- Feedforward networks often have one or more hidden layers of sigmoid neurons followed by an output layer of linear neurons
- Multiple layers of neurons with nonlinear transfer functions allow the network to learn nonlinear relationships between input and output vectors
- The linear output layer is most often used for function fitting (or nonlinear regression) problems
- can be used for any kind of input to output mapping

Network design for fitting

- Feedforward networks often have one or more hidden layers of sigmoid neurons followed by an output layer of linear neurons
- Multiple layers of neurons with nonlinear transfer functions allow the network to learn nonlinear relationships between input and output vectors
- The linear output layer is most often used for function fitting (or nonlinear regression) problems
- can be used for any kind of input to output mapping
- A feedforward network with one hidden layer and enough neurons in the hidden layers, can fit any finite input-output mapping problem.

Function approximation

- The MLP can be used as a general function approximator

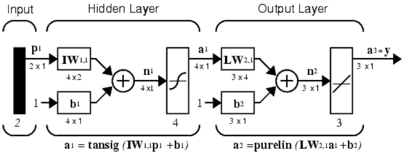


Figure 8: 3 layer MLP

- The MLP can be used as a general function approximator
    - can approximate any function with a finite number of discontinuities arbitrarily well,

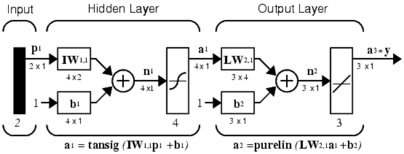


Figure 8: 3 layer MLP

- The MLP can be used as a general function approximator
    - can approximate any function with a finite number of discontinuities arbitrarily well,
    - given sufficient neurons in the hidden layer

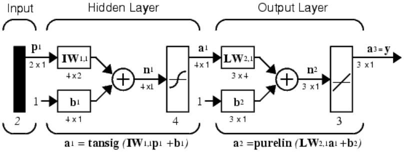


Figure 8: 3 layer MLP

Work flow

- Collect data

Work flow

- Collect data
- Create the network

Work flow

- Collect data
- Create the network
- Configure the network

Work flow

- Collect data
- Create the network
- Configure the network
- Initialise the weights and biases

Work flow

- Collect data
- Create the network
- Configure the network
- Initialise the weights and biases
- Train the network

Work flow

- Collect data
- Create the network
- Configure the network
- Initialise the weights and biases
- Train the network
- Validate the network (post-training analysis)

Work flow

- Collect data
- Create the network
- Configure the network
- Initialise the weights and biases
- Train the network
- Validate the network (post-training analysis)
- Use the network

Data subsets

- the general practice is to first divide the data into three subsets:-

Data subsets

- the general practice is to first divide the data into three subsets:-
- The first subset is the **training** set, which is used for computing the gradient and updating the network weights and biases.

Data subsets

- the general practice is to first divide the data into three subsets:-
- The first subset is the **training** set, which is used for computing the gradient and updating the network weights and biases.
- The second subset is the **validation** set.

Data subsets

- the general practice is to first divide the data into three subsets:-
- The first subset is the **training** set, which is used for computing the gradient and updating the network weights and biases.
- The second subset is the **validation** set.
  - The error on the validation set is monitored during the training process.

Data subsets

- the general practice is to first divide the data into three subsets:-
- The first subset is the **training** set, which is used for computing the gradient and updating the network weights and biases.
- The second subset is the **validation** set.
  - The error on the validation set is monitored during the training process.
  - The validation error normally decreases during the initial phase of training, as does the training set error.

Data subsets

- the general practice is to first divide the data into three subsets:-
- The first subset is the **training** set, which is used for computing the gradient and updating the network weights and biases.
- The second subset is the **validation** set.
  - The error on the validation set is monitored during the training process.
  - The validation error normally decreases during the initial phase of training, as does the training set error.
  - However, when the network begins to overfit the data, the error on the validation set typically begins to rise.

Data subsets

- the general practice is to first divide the data into three subsets:-
- The first subset is the **training** set, which is used for computing the gradient and updating the network weights and biases.
- The second subset is the **validation** set.
  - The error on the validation set is monitored during the training process.
  - The validation error normally decreases during the initial phase of training, as does the training set error.
  - However, when the network begins to overfit the data, the error on the validation set typically begins to rise.
  - The network weights and biases are saved at the minimum of the validation set error.

Data subsets

- the general practice is to first divide the data into three subsets:-
- The first subset is the **training** set, which is used for computing the gradient and updating the network weights and biases.
- The second subset is the **validation** set.
  - The error on the validation set is monitored during the training process.
  - The validation error normally decreases during the initial phase of training, as does the training set error.
  - However, when the network begins to overfit the data, the error on the validation set typically begins to rise.
  - The network weights and biases are saved at the minimum of the validation set error.
- The **test set** is not used during training, but it is used to compare different models

Data subsets

- the general practice is to first divide the data into three subsets:-
- The first subset is the **training** set, which is used for computing the gradient and updating the network weights and biases.
- The second subset is the **validation** set.
  - The error on the validation set is monitored during the training process.
  - The validation error normally decreases during the initial phase of training, as does the training set error.
  - However, when the network begins to overfit the data, the error on the validation set typically begins to rise.
  - The network weights and biases are saved at the minimum of the validation set error.
- The **test set** is not used during training, but it is used to compare different models
  - It is also useful to plot the test set error during the training process

Data subsets

- the general practice is to first divide the data into three subsets:-
- The first subset is the **training** set, which is used for computing the gradient and updating the network weights and biases.
- The second subset is the **validation** set.
  - The error on the validation set is monitored during the training process.
  - The validation error normally decreases during the initial phase of training, as does the training set error.
  - However, when the network begins to overfit the data, the error on the validation set typically begins to rise.
  - The network weights and biases are saved at the minimum of the validation set error.
- The **test set** is not used during training, but it is used to compare different models
  - It is also useful to plot the test set error during the training process
  - If the error on the test set reaches a minimum at a significantly different iteration number than the validation set error, this might indicate a poor division of the data set.

Using Data Subsets in nftool

- A simple example of partitioning data

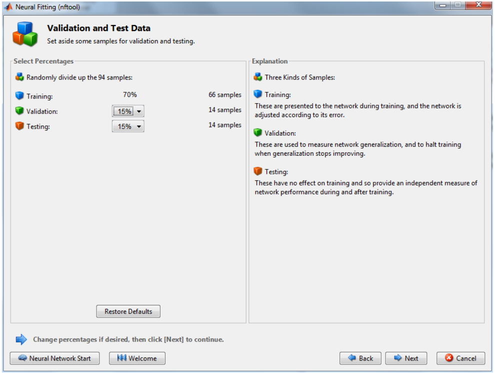


Figure 9: Simplefit Example

Training algorithms in the Matlab Toolbox

Function	Algorithm
trainlm	Levenberg-Marquardt (default)
trainbr	Bayesian Regularization
trainbfg	BFGS Quasi-Newton
trainrp	Resilient Backpropagation
trainscg	Scaled Conjugate Gradient
traincgb	Conjugate Gradient with Powell/Beale Restarts
traincgf	Fletcher-Powell Conjugate Gradient
traincgp	Polak-Ribiére Conjugate Gradient
trainoss	One Step Secant
traingdx	Variable Learning Rate Gradient Descent
traingdm	Gradient Descent with Momentum
traingd	Gradient Descent (backpropagation)

Choosing a training method

- The fastest training function is generally `trainlm`, and it is the default training function used in MATLAB's feedforwardnet

Choosing a training method

- The fastest training function is generally `trainlm`, and it is the default training function used in MATLAB's feedforwardnet
    - but note that it performs better on function fitting (nonlinear regression) problems than on pattern recognition problems.

Choosing a training method

- The fastest training function is generally `trainlm`, and it is the default training function used in MATLAB's feedforwardnet
  - but note that it performs better on function fitting (nonlinear regression) problems than on pattern recognition problems.
- The quasi-Newton method, `trainbfg`, is also quite fast

Choosing a training method

- The fastest training function is generally `trainlm`, and it is the default training function used in MATLAB's feedforwardnet
  - but note that it performs better on function fitting (nonlinear regression) problems than on pattern recognition problems.
- The quasi-Newton method, `trainbfg`, is also quite fast
- Both of these methods tend to be less efficient for large networks (with thousands of weights), since they require more memory and more computation time for these cases.

Choosing a training method

- The fastest training function is generally `trainlm`, and it is the default training function used in MATLAB's feedforwardnet
  - but note that it performs better on function fitting (nonlinear regression) problems than on pattern recognition problems.
- The quasi-Newton method, `trainbfg`, is also quite fast
- Both of these methods tend to be less efficient for large networks (with thousands of weights), since they require more memory and more computation time for these cases.
- Bayesian Regularization is also a good option in many cases

Choosing a training method

- The fastest training function is generally `trainlm`, and it is the default training function used in MATLAB's feedforwardnet
  - but note that it performs better on function fitting (nonlinear regression) problems than on pattern recognition problems.
- The quasi-Newton method, `trainbfg`, is also quite fast
- Both of these methods tend to be less efficient for large networks (with thousands of weights), since they require more memory and more computation time for these cases.
- Bayesian Regularization is also a good option in many cases
- Although widely known, the back-propagation algorithm is usually slower than the others and is not often used these days.

Choosing a training method

- The fastest training function is generally `trainlm`, and it is the default training function used in MATLAB's feedforwardnet
  - but note that it performs better on function fitting (nonlinear regression) problems than on pattern recognition problems.
- The quasi-Newton method, `trainbfg`, is also quite fast
- Both of these methods tend to be less efficient for large networks (with thousands of weights), since they require more memory and more computation time for these cases.
- Bayesian Regularization is also a good option in many cases
- Although widely known, the back-propagation algorithm is usually slower than the others and is not often used these days.
- All of these methods are gradient-based and can be prone to converging on local minima.

Backpropagation

- Small random weights are chosen to initialise the system.

Backpropagation

- Small random weights are chosen to initialise the system.
- Learning is by successively adjusting the weights based on a set of input patterns and the corresponding set of desired output patterns (the training set).

Backpropagation

- Small random weights are chosen to initialise the system.
- Learning is by successively adjusting the weights based on a set of input patterns and the corresponding set of desired output patterns (the training set).
- During this iterative process, an input pattern is presented to the network and propagated forward to determine the resulting signal at the output units.

Backpropagation

- Small random weights are chosen to initialise the system.
- Learning is by successively adjusting the weights based on a set of input patterns and the corresponding set of desired output patterns (the training set).
- During this iterative process, an input pattern is presented to the network and propagated forward to determine the resulting signal at the output units.
- differences between the resulting output signal and the predetermined desired output signal in each output unit represent an error that is back-propagated through the network in order to adjust the weights.

Backpropagation

- Small random weights are chosen to initialise the system.
- Learning is by successively adjusting the weights based on a set of input patterns and the corresponding set of desired output patterns (the training set).
- During this iterative process, an input pattern is presented to the network and propagated forward to determine the resulting signal at the output units.
- differences between the resulting output signal and the predetermined desired output signal in each output unit represent an error that is back-propagated through the network in order to adjust the weights.
- The process continues until the the sum of root-mean-square errors from the desired output signals is less than a preset value.

Generalisation

- The aim of using a fitting network is to achieve a good fit to the whole function within the range of the data and not just at the points of the training set.

Generalisation

- The aim of using a fitting network is to achieve a good fit to the whole function within the range of the data and not just at the points of the training set.
- This is known as generalisation and it can be quantified using the test set.

Generalisation

- The aim of using a fitting network is to achieve a good fit to the whole function within the range of the data and not just at the points of the training set.
- This is known as generalisation and it can be quantified using the test set.
- The key to achieving this is in the design of the network, particularly in the number of hidden layer neurons

Generalisation

- The aim of using a fitting network is to achieve a good fit to the whole function within the range of the data and not just at the points of the training set.
- This is known as generalisation and it can be quantified using the test set.
- The key to achieving this is in the design of the network, particularly in the number of hidden layer neurons
- A good practice is to start with a small number (the default in the feedforwardnet is 10) and try increasing to see if the fit improves or gets worse.

Over-fitting

- If the number of hidden neurons is increased too much, the error at the training points can go to zero but large errors can occur between the training points

Over-fitting

- If the number of hidden neurons is increased too much, the error at the training points can go to zero but large errors can occur between the training points
- This is known as over-fitting and is one of the biggest problems in ANN design

Failure to converge

- Sometimes, gradient-based learning algorithms converge on local minima or simply diverge if the function is complicated and there are too few training points

Failure to converge

- Sometimes, gradient-based learning algorithms converge on local minima or simply diverge if the function is complicated and there are too few training points
- There are alternative training methods such as using Evolutionary Algorithms but that is beyond the scope of this introduction

Fitnet (Matlab Function)

- `fitnet` is a very easy to use method for creating feedforward neural networks (feedforwardnet) which are used to fit an input-output relationship.

```
fitnet(hiddenSizes,trainFcn)
```

Fitnet (Matlab Function)

- `fitnet` is a very easy to use method for creating feedforward neural networks (feedforwardnet) which are used to fit an input-output relationship.

```
fitnet(hiddenSizes,trainFcn)
```

- takes these arguments:

Fitnet (Matlab Function)

- `fitnet` is a very easy to use method for creating feedforward neural networks (feedforwardnet) which are used to fit an input-output relationship.

`fitnet(hiddenSizes,trainFcn)`

- takes these arguments:
  - `hiddenSizes`

Fitnet (Matlab Function)

- `fitnet` is a very easy to use method for creating feedforward neural networks (feedforwardnet) which are used to fit an input-output relationship.

`fitnet(hiddenSizes,trainFcn)`

- takes these arguments:
  - `hiddenSizes`
    - Row vector of one or more hidden layer sizes (default = 10)

Fitnet (Matlab Function)

- `fitnet` is a very easy to use method for creating feedforward neural networks (feedforwardnet) which are used to fit an input-output relationship.

`fitnet(hiddenSizes,trainFcn)`

- takes these arguments:
  - `hiddenSizes`
    - Row vector of one or more hidden layer sizes (default = 10)
  - `trainFcn`

Fitnet (Matlab Function)

- `fitnet` is a very easy to use method for creating feedforward neural networks (feedforwardnet) which are used to fit an input-output relationship.

```
fitnet(hiddenSizes,trainFcn)
```

- takes these arguments:
  - `hiddenSizes`
    - Row vector of one or more hidden layer sizes (default = 10)
  - `trainFcn`
    - Training function (default = 'trainlm')

Fitnet (Matlab Function)

- `fitnet` is a very easy to use method for creating feedforward neural networks (feedforwardnet) which are used to fit an input-output relationship.

`fitnet(hiddenSizes,trainFcn)`

- takes these arguments:
  - `hiddenSizes`
    - Row vector of one or more hidden layer sizes (default = 10)
  - `trainFcn`
    - Training function (default = 'trainlm')
- and returns a fitting neural network.

Example MATLAB code

- Here a fitting neural network is used to solve a simple regression problem.

```
[x,t] = simplefit_dataset;
% Create and display the network
net = fitnet(10);
% Train the network using the data in x and t
net = train(net,x,t);
view(net)
% Predict the response using the trained network
y = net(x);
% Measure the performance
perf = perform(net,y,t)
```

Plot of trained network

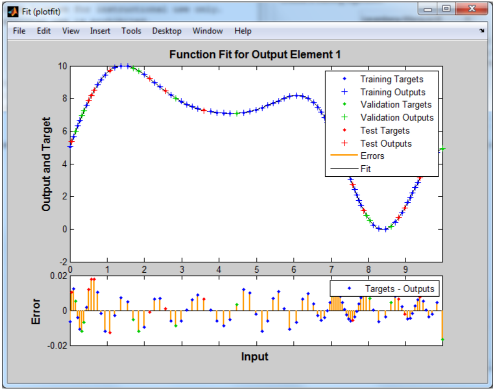


Figure 10: Plot of results of fitnet example

References and Further Reading

- This lecture is largely based on the MATLAB Neural Network Toolkit

References and Further Reading

- This lecture is largely based on the MATLAB Neural Network Toolkit
    - Familiarisation with the Users' Guide is strongly recommended: http://www.mathworks.co.uk/help/pdf_doc/nnet/nnet_ug.pdf

References and Further Reading

- This lecture is largely based on the MATLAB Neural Network Toolkit
  - Familiarisation with the Users' Guide is strongly recommended: http://www.mathworks.co.uk/help/pdf_doc/nnet/nnet_ug.pdf
- Raúl Rojas. Neural Networks. A Systematic Introduction. Springer. Berlin Heidelberg NewYork.

References and Further Reading

- This lecture is largely based on the MATLAB Neural Network Toolkit
  - Familiarisation with the Users' Guide is strongly recommended: http://www.mathworks.co.uk/help/pdf_doc/nnet/nnet_ug.pdf
- Raúl Rojas. Neural Networks. A Systematic Introduction. Springer. Berlin Heidelberg NewYork.
  - Very good but has more detail than really needed at this level

References and Further Reading

- This lecture is largely based on the MATLAB Neural Network Toolkit
  - Familiarisation with the Users' Guide is strongly recommended: http://www.mathworks.co.uk/help/pdf_doc/nnet/nnet_ug.pdf
- Raúl Rojas. Neural Networks. A Systematic Introduction. Springer. Berlin Heidelberg NewYork.
  - Very good but has more detail than really needed at this level
  - Available online at https://page.mi.fu-berlin.de/rojas/neural/neuron.pdf

References and Further Reading

- This lecture is largely based on the MATLAB Neural Network Toolkit
  - Familiarisation with the Users' Guide is strongly recommended: http://www.mathworks.co.uk/help/pdf_doc/nnet/nnet_ug.pdf
- Raúl Rojas. Neural Networks. A Systematic Introduction. Springer. Berlin Heidelberg NewYork.
  - Very good but has more detail than really needed at this level
  - Available online at https://page.mi.fu-berlin.de/rojas/neural/neuron.pdf
- Beale and Jackson, Neural Computing: An Introduction, Adam Hilger, 1991

References and Further Reading

- This lecture is largely based on the MATLAB Neural Network Toolkit
    - Familiarisation with the Users' Guide is strongly recommended: http://www.mathworks.co.uk/help/pdf_doc/nnet/nnet_ug.pdf
- Raúl Rojas. Neural Networks. A Systematic Introduction. Springer. Berlin Heidelberg NewYork.
    - Very good but has more detail than really needed at this level
    - Available online at https://page.mi.fu-berlin.de/rojas/neural/neuron.pdf
- Beale and Jackson, Neural Computing: An Introduction, Adam Hilger, 1991
    - good introductory textbook which covers all of the ANN types which we will look at