UFME7K-15-M Intelligent and Adaptive Systems

Multi-Layer Perceptrons. Approximation, generalisation and validation

Charlie Sullivan

14 February 2018

Intelligent Adaptive Systems Lecture 4

Dr J. Charlie Sullivan

charlie.sullivan@brl.ac.uk

Generalisation as Learning

- **Generalisation** is one of the most important aspects of learning

Generalisation as Learning

- **Generalisation** is one of the most important aspects of learning
- It is an example of **inductive inference**

Generalisation as Learning

- **Generalisation** is one of the most important aspects of learning
- It is an example of **inductive inference**
- It is what MLP networks are good at

Generalisation as Learning

- **Generalisation** is one of the most important aspects of learning
- It is an example of **inductive inference**
- It is what MLP networks are good at
- We are focussing on function approximation tasks

Generalisation as Learning

- **Generalisation** is one of the most important aspects of learning
- It is an example of **inductive inference**
- It is what MLP networks are good at
- We are focussing on function approximation tasks
    - regression

Generalisation as Learning

- **Generalisation** is one of the most important aspects of learning
- It is an example of **inductive inference**
- It is what MLP networks are good at
- We are focussing on function approximation tasks
  - regression
  - fitting curves surfaces etc

Generalisation as Learning

- **Generalisation** is one of the most important aspects of learning
- It is an example of **inductive inference**
- It is what MLP networks are good at
- We are focussing on function approximation tasks
  - regression
  - fitting curves surfaces etc
- The same principles apply also to classification tasks (pattern recognition)

Generalisation as Learning

- **Generalisation** is one of the most important aspects of learning
- It is an example of **inductive inference**
- It is what MLP networks are good at
- We are focussing on function approximation tasks
  - regression
  - fitting curves surfaces etc
- The same principles apply also to classification tasks (pattern recognition)
- Note that we are only considering what are now considered to be fairly small problems and other approaches (e.g. Deep Learning) are more appropriate in problems with much larger datasets

MLPs for Function Approximation

- MLP Neural networks are very powerful function approximators:

MLPs for Function Approximation

- MLP Neural networks are very powerful function approximators:
  - Any boolean function can be realised by a MLP with one hidden layer.

MLPs for Function Approximation

- MLP Neural networks are very powerful function approximators:
  - Any boolean function can be realised by a MLP with one hidden layer.
  - Any bounded continuous function can be approximated with arbitrary precision by a MLP with one hidden layer. (Cybenko,1989)

MLPs for Function Approximation

- MLP Neural networks are very powerful function approximators:
    - Any boolean function can be realised by a MLP with one hidden layer.
    - Any bounded continuous function can be approximated with arbitrary precision by a MLP with one hidden layer. (Cybenko,1989)
- Therefore, they incorporate the danger of overfitting.

How can we design a MLP network?

- the size of the network matters:

How can we design a MLP network?

- the size of the network matters:
    - the larger the number of hidden neurons, the more powerful the representation capacity

How can we design a MLP network?

- the size of the network matters:
  - the larger the number of hidden neurons, the more powerful the representation capacity
  - Tradeoff: being able to reasonably learn (avoid underfitting), but not just memorise training data (overfitting)

How can we design a MLP network?

- the size of the network matters:
  - the larger the number of hidden neurons, the more powerful the representation capacity
  - Tradeoff: being able to reasonably learn (avoid underfitting), but not just memorise training data (overfitting)
  - Unfortunately, there is no analytical way to determine the right network size from the training data (Bishop, 1995)

How can we design a MLP network?

- the size of the network matters:
  - the larger the number of hidden neurons, the more powerful the representation capacity
  - Tradeoff: being able to reasonably learn (avoid underfitting), but not just memorise training data (overfitting)
  - Unfortunately, there is no analytical way to determine the right network size from the training data (Bishop, 1995)
  - We will try to find some **heuristics** to help design a good network

What are the symptoms of overfitting?

- overfitting:

What are the symptoms of overfitting?

- overfitting:
    - validation error increases while training error decreases

What are the symptoms of overfitting?

- overfitting:
  - validation error increases while training error decreases
- successful learning:

What are the symptoms of overfitting?

- overfitting:
  - validation error increases while training error decreases
- successful learning:
  - validation error and training error monotonically decrease

What are the symptoms of overfitting?

- overfitting:
  - validation error increases while training error decreases
- successful learning:
  - validation error and training error monotonically decrease
- underfitting:

What are the symptoms of overfitting?

- overfitting:
  - validation error increases while training error decreases
- successful learning:
  - validation error and training error monotonically decrease
- underfitting:
  - validation and training error remain large

Performance plot showing overfitting

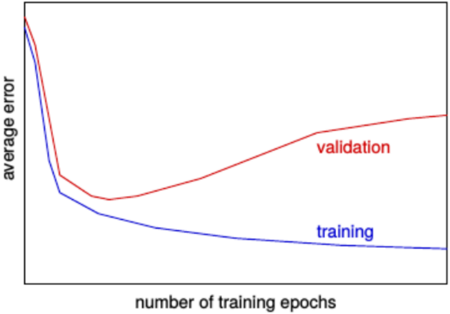


Figure 1: Overfitting

Performance plot showing underfitting

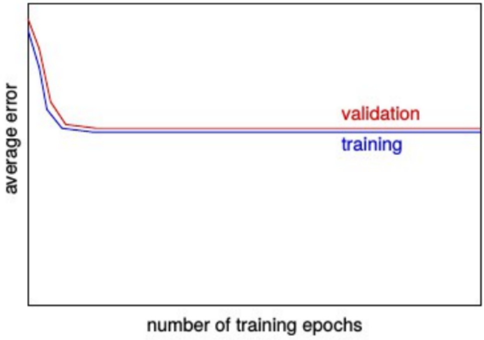


Figure 2: Underfitting

Which is a better fit to the data?

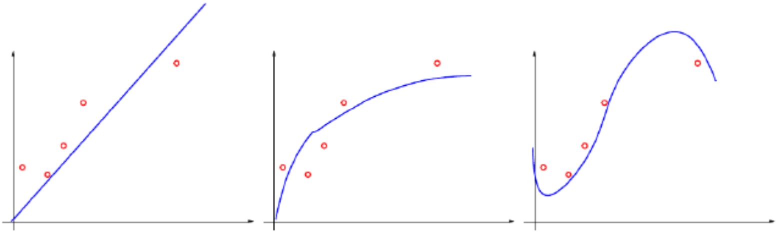
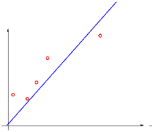
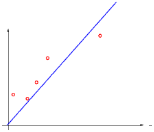
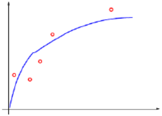


Figure 3: Some alternative curves to fit the data

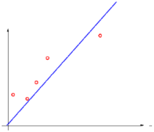
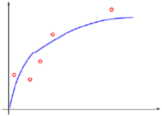
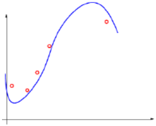
Which is a better fit to the data?

- Underfit 

Which is a better fit to the data?

- Underfit 
- OK 

Which is a better fit to the data?

- Underfit 
- OK 
- Over-fit 

Training and Validation

- learning algorithms try to find a hypothesis that fits the training data in the best way

Training and Validation

- learning algorithms try to find a hypothesis that fits the training data in the best way
- we would like to find a hypothesis that works well for all data sets that can be derived from the true function

Training and Validation

- learning algorithms try to find a hypothesis that fits the training data in the best way
- we would like to find a hypothesis that works well for all data sets that can be derived from the true function
- assume that the input patterns are taken from some probability distribution P

Training and Validation

- learning algorithms try to find a hypothesis that fits the training data in the best way
- we would like to find a hypothesis that works well for all data sets that can be derived from the true function
- assume that the input patterns are taken from some probability distribution P
- the best hypothesis $f(\mathbf{x})$ should minimise the expected error (generalisation error)

$$E[1/2(f(\mathbf{x}) - f^*(\mathbf{x}))^2]$$

Training and Validation

- learning algorithms try to find a hypothesis that fits the training data in the best way
- we would like to find a hypothesis that works well for all data sets that can be derived from the true function
- assume that the input patterns are taken from some probability distribution P
- the best hypothesis $f(\mathbf{x})$ should minimise the expected error (generalisation error)

$$E[1/2(f(\mathbf{x}) - f^*(\mathbf{x}))^2]$$

- here E means the expectation value over all possible x

Training and Validation

- learning algorithms try to find a hypothesis that fits the training data in the best way
- we would like to find a hypothesis that works well for all data sets that can be derived from the true function
- assume that the input patterns are taken from some probability distribution P
- the best hypothesis $f(\mathbf{x})$ should minimise the expected error (generalisation error)

$$E[1/2(f(\mathbf{x}) - f^*(\mathbf{x}))^2]$$

- here E means the expectation value over all possible x
- Note we cannot calculate the expected error since we don't know P or $f^*(\mathbf{x})$

Approximating the error

- if we are given a set of examples $(\mathbf{x}^{(1)}, t^{(1)}), \ldots, (\mathbf{x}^{(n)}, t^{(n)})$ with $\mathbf{x}^{(i)} \sim P$ and $t^{(i)} = f^*(\mathbf{x}^{(i)})$,

Approximating the error

- if we are given a set of examples $(\mathbf{x}^{(1)}, t^{(1)}), \ldots, (\mathbf{x}^{(n)}, t^{(n)})$ with $\mathbf{x}^{(i)} \sim P$ and $t^{(i)} = f^*(\mathbf{x}^{(i)})$,
- we can approximate the expected error by the mean training error:

$$\frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{2}(f(\mathbf{x})^{(i)} - t^{(i)})^2\right)$$

Approximating the error

- if we are given a set of examples $(\mathbf{x}^{(1)}, t^{(1)}), \ldots, (\mathbf{x}^{(n)}, t^{(n)})$ with $\mathbf{x}^{(i)} \sim P$ and $t^{(i)} = f^*(\mathbf{x}^{(i)})$,
- we can approximate the expected error by the mean training error:

$$\frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{2}(f(\mathbf{x})^{(i)} - t^{(i)})^2\right)$$

- if the approximation is good: a hypothesis learned on the training set will also perform well on other data sets

Approximating the error

- if we are given a set of examples $(\mathbf{x}^{(1)}, t^{(1)}), \ldots, (\mathbf{x}^{(n)}, t^{(n)})$ with $\mathbf{x}^{(i)} \sim P$ and $t^{(i)} = f^*(\mathbf{x}^{(i)})$,
- we can approximate the expected error by the mean training error:

$$\frac{1}{n}\sum_{i=1}^{n}\left(\frac{1}{2}(f(\mathbf{x})^{(i)} - t^{(i)})^2\right)$$

- if the approximation is good: a hypothesis learned on the training set will also perform well on other data sets
- if the approximation is bad: a hypothesis learned on the training set will perform poorly on other data sets

Validation

- Validation is the process of checking the performance of a learned function on independent validation/test data

Validation

- Validation is the process of checking the performance of a learned function on independent validation/test data
- simple approach for validation:

Validation

- Validation is the process of checking the performance of a learned function on independent validation/test data
- simple approach for validation:
  - before training, split the available data set into two disjoint subsets:

Validation

- Validation is the process of checking the performance of a learned function on independent validation/test data
- simple approach for validation:
  - before training, split the available data set into two disjoint subsets:
  - training set and validation set

Validation

- Validation is the process of checking the performance of a learned function on independent validation/test data
- simple approach for validation:
  - before training, split the available data set into two disjoint subsets:
  - training set and validation set
  - apply training only on the training set

Validation

- Validation is the process of checking the performance of a learned function on independent validation/test data
- simple approach for validation:
  - before training, split the available data set into two disjoint subsets:
  - training set and validation set
  - apply training only on the training set
  - test the learned functions on the validation set

Validation

- Validation is the process of checking the performance of a learned function on independent validation/test data
- simple approach for validation:
  - before training, split the available data set into two disjoint subsets:
  - training set and validation set
  - apply training only on the training set
  - test the learned functions on the validation set
- disadvantage: only a subset of available data is used for training, only a subset of available data is used for testing

Validation

- Validation is the process of checking the performance of a learned function on independent validation/test data
- simple approach for validation:
  - before training, split the available data set into two disjoint subsets:
  - training set and validation set
  - apply training only on the training set
  - test the learned functions on the validation set
- disadvantage: only a subset of available data is used for training, only a subset of available data is used for testing
- Other approaches may be better :

Validation

- Validation is the process of checking the performance of a learned function on independent validation/test data
- simple approach for validation:
  - before training, split the available data set into two disjoint subsets:
  - training set and validation set
  - apply training only on the training set
  - test the learned functions on the validation set
- disadvantage: only a subset of available data is used for training, only a subset of available data is used for testing
- Other approaches may be better :
  - Cross-validation, e.g. stratified k-fold XVAL

Validation

- Validation is the process of checking the performance of a learned function on independent validation/test data
- simple approach for validation:
  - before training, split the available data set into two disjoint subsets:
  - training set and validation set
  - apply training only on the training set
  - test the learned functions on the validation set
- disadvantage: only a subset of available data is used for training, only a subset of available data is used for testing
- Other approaches may be better :
  - Cross-validation, e.g. stratified k-fold XVAL
  - Bootstrapping

Bias and Variance

- A learning algorithm is biased for a particular input x if, when trained on each of these data sets, it is systematically incorrect when predicting the correct output for x

Bias and Variance

- A learning algorithm is biased for a particular input x if, when trained on each of these data sets, it is systematically incorrect when predicting the correct output for x
- A learning algorithm has high variance for a particular input x if it predicts different output values when trained on different training sets

Bias and Variance

- A learning algorithm is biased for a particular input x if, when trained on each of these data sets, it is systematically incorrect when predicting the correct output for x
- A learning algorithm has high variance for a particular input x if it predicts different output values when trained on different training sets
- The prediction error of a learned classifier is related to the sum of the bias and the variance of the learning algorithm.

Bias and Variance

- A learning algorithm is biased for a particular input x if, when trained on each of these data sets, it is systematically incorrect when predicting the correct output for x
- A learning algorithm has high variance for a particular input x if it predicts different output values when trained on different training sets
- The prediction error of a learned classifier is related to the sum of the bias and the variance of the learning algorithm.
- Generally, there is a tradeoff between bias and variance.

Bias and Variance

- A learning algorithm is biased for a particular input x if, when trained on each of these data sets, it is systematically incorrect when predicting the correct output for x
- A learning algorithm has high variance for a particular input x if it predicts different output values when trained on different training sets
- The prediction error of a learned classifier is related to the sum of the bias and the variance of the learning algorithm.
- Generally, there is a tradeoff between bias and variance.
  - A learning algorithm with low bias must be "flexible" so that it can fit the data well.

Bias and Variance

- A learning algorithm is biased for a particular input x if, when trained on each of these data sets, it is systematically incorrect when predicting the correct output for x
- A learning algorithm has high variance for a particular input x if it predicts different output values when trained on different training sets
- The prediction error of a learned classifier is related to the sum of the bias and the variance of the learning algorithm.
- Generally, there is a tradeoff between bias and variance.
    - A learning algorithm with low bias must be “flexible” so that it can fit the data well.
    - But if the learning algorithm is too flexible, it will fit each training data set differently, and hence have high variance.

Bias and Variance

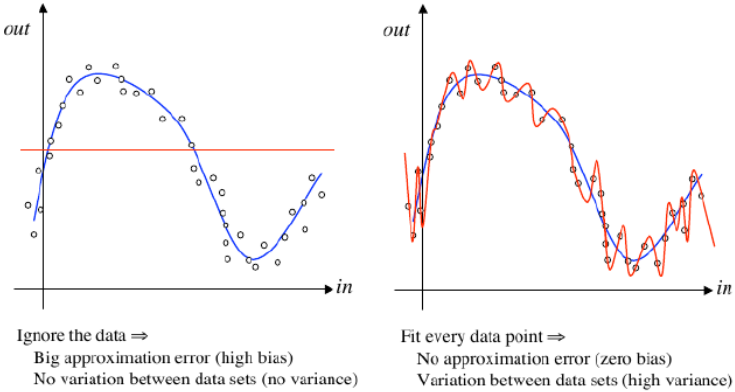


Figure 4: Bias and Variance on a Data-fitting Example

Bias/Variance Trade-off

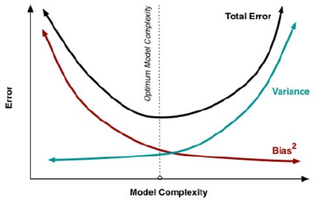


Figure 5: Bias/Variance Trade-off

- dealing with bias and variance is really about dealing with over- and under-fitting.

Bias/Variance Trade-off

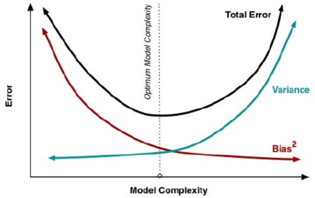


Figure 5: Bias/Variance Trade-off

- dealing with bias and variance is really about dealing with over- and under-fitting.
- Bias is reduced and variance is increased in relation to model complexity.

Bias/Variance Trade-off

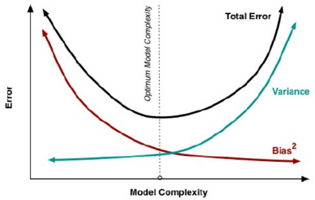


Figure 5: Bias/Variance Trade-off

- dealing with bias and variance is really about dealing with over- and under-fitting.
- Bias is reduced and variance is increased in relation to model complexity.
  - As more parameters are added to a model, the complexity of the model rises and variance becomes our primary concern while bias steadily falls

Design Questions

1. How many hidden layers?
2. How many hidden nodes per layer?
3. How many training input/target pairs needed?
4. How many random initial weight trials for each hidden node configuration?

1. How many hidden layers?

- Always begin with 1 hidden layer.

1. How many hidden layers?

- Always begin with 1 hidden layer.
  - The Multilayer Perceptron (MLP) with a single hidden layer of H, sufficiently many, sigmoidal transfer functions is a **universal approximator**

1. How many hidden layers?

- Always begin with 1 hidden layer.
  - The Multilayer Perceptron (MLP) with a single hidden layer of H, sufficiently many, sigmoidal transfer functions is a **universal approximator**
- On rare occasions it is useful to add a second hidden layer to reduce the necessary number of hidden nodes

1. How many hidden layers?

- Always begin with 1 hidden layer.
  - The Multilayer Perceptron (MLP) with a single hidden layer of H, sufficiently many, sigmoidal transfer functions is a **universal approximator**
- On rare occasions it is useful to add a second hidden layer to reduce the necessary number of hidden nodes
- $$(H_1 + H_2 < H)$$

2. How many hidden nodes per layer?

- Estimate, by trial and error, the minimum number of hidden nodes necessary for successfully approximating the underlying input-output transformation.

2. How many hidden nodes per layer?

- Estimate, by trial and error, the minimum number of hidden nodes necessary for successfully approximating the underlying input-output transformation.
- For a smooth function with N_{locmin} local minima and N_{locmax} local maxima, a reasonable lower bound is

$$H >= 2 * max(N_{locmin}, N_{locmax})$$

2. How many hidden nodes per layer?

- Estimate, by trial and error, the minimum number of hidden nodes necessary for successfully approximating the underlying input-output transformation.
- For a smooth function with N_{locmin} local minima and N_{locmax} local maxima, a reasonable lower bound is

$$H >= 2 * max(N_{locmin}, N_{locmax})$$

- can count endpoint extrema as $1/2$ a local extremum.

2. How many hidden nodes per layer?

- Estimate, by trial and error, the minimum number of hidden nodes necessary for successfully approximating the underlying input-output transformation.
- For a smooth function with N_{locmin} local minima and N_{locmax} local maxima, a reasonable lower bound is

$$H >= 2 * max(N_{locmin}, N_{locmax})$$

- can count endpoint extrema as $1/2$ a local extremum.
  - However, noise may make it difficult to identify the significant error-free extrema.

Simplefit Example

2.5 local min

2.5 local max

H >= 2.5 * 2

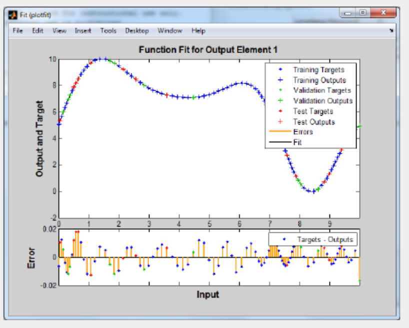


Figure 6: Simplefit function

3. How many training input/target pairs?

- The minimum number of training input/target pairs needed to adequately estimate the resulting number of weights, N_w, tends to vary linearly with H

3. How many training input/target pairs?

- The minimum number of training input/target pairs needed to adequately estimate the resulting number of weights, N_w, tends to vary linearly with H
- If the output target vectors are O-dimensional, the N_{trn} training pairs yield $N_{trneq} = N_{trn} * O$ training equations for estimating N_w unknown weights.

3. How many training input/target pairs?

- The minimum number of training input/target pairs needed to adequately estimate the resulting number of weights, N_w, tends to vary linearly with H
- If the output target vectors are O-dimensional, the N_{trn} training pairs yield $N_{trneq} = N_{trn} * O$ training equations for estimating N_w unknown weights.
- If the input vectors are I-dimensional, the number of weights for a static MLP is given by

3. How many training input/target pairs?

- The minimum number of training input/target pairs needed to adequately estimate the resulting number of weights, N_w, tends to vary linearly with H
- If the output target vectors are O-dimensional, the N_{trn} training pairs yield $N_{trneq} = N_{trn} * O$ training equations for estimating N_w unknown weights.
- If the input vectors are I-dimensional, the number of weights for a static MLP is given by
  -
$$N_w = (I+1) * H + (H+1) * O = O + (I+O+1) * H$$

4. How many random initial weight trials?

- to get an accurate estimate of error on unseen data

4. How many random initial weight trials?

- to get an accurate estimate of error on unseen data
- A conservative rule of thumb seems to be

$$N_{trials} >= 32/min(N_{trn}, N_{val}, N_{tst})$$

4. How many random initial weight trials?

- to get an accurate estimate of error on unseen data
- A conservative rule of thumb seems to be

$$N_{trials} >= 32/min(N_{trn}, N_{val}, N_{tst})$$

- But this is a statistical problem

4. How many random initial weight trials?

- to get an accurate estimate of error on unseen data
- A conservative rule of thumb seems to be

$$N_{trials} >= 32/min(N_{trn}, N_{val}, N_{tst})$$

- But this is a statistical problem
- Often a simple rule is used e.g. 10 times

Error Minimization as Gradient Descent

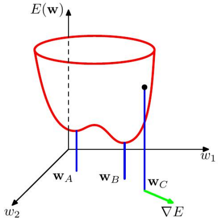


Figure 7: error surface of a network with 2 weights(Bishop, 2006)

- w_a is a local min

Error Minimization as Gradient Descent

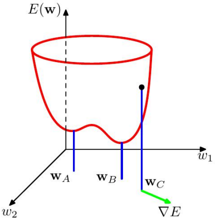


Figure 7: error surface of a network with 2 weights(Bishop, 2006)

- w_a is a local min
- w_b is the global min

Error Minimization as Gradient Descent

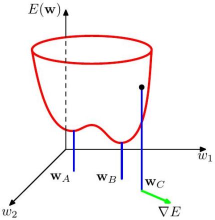


Figure 7: error surface of a network with 2 weights(Bishop, 2006)

- w_a is a local min
- w_b is the global min
- at any point w_c, the local gradient of the error surface is given by the vector ∇E

Early stopping

- stop learning when the error on the validation set has reached its minimum

Early stopping

- stop learning when the error on the validation set has reached its minimum
- needs perpetual observation of the validation error, typically combined with small initial weights

Early stopping

- stop learning when the error on the validation set has reached its minimum
- needs perpetual observation of the validation error, typically combined with small initial weights
- training is often stopped after a few iterations (10-30 iterations)

Early stopping

- stop learning when the error on the validation set has reached its minimum
- needs perpetual observation of the validation error, typically combined with small initial weights
- training is often stopped after a few iterations (10-30 iterations)
- Results are often highly variable, depending on initial random weights

Early stopping

- stop learning when the error on the validation set has reached its minimum
- needs perpetual observation of the validation error, typically combined with small initial weights
- training is often stopped after a few iterations (10-30 iterations)
- Results are often highly variable, depending on initial random weights
- Prone to getting stuck in local optima

Regularisation

- regularisation techniques:

Regularisation

- regularisation techniques:
  - approaches to improve generalisation implementing some preference rules

Regularisation

- regularisation techniques:
  - approaches to improve generalisation implementing some preference rules
- four basic principles:

Regularisation

- regularisation techniques:
  - approaches to improve generalisation implementing some preference rules
- four basic principles:
  - smaller networks should be prefered (cf. Ockham's razor)

Regularisation

- regularisation techniques:
  - approaches to improve generalisation implementing some preference rules
- four basic principles:
  - smaller networks should be prefered (cf. Ockham's razor)
  - the hypothesis set H of smaller networks is smaller than the hypothesis set of large networks

Regularisation

- regularisation techniques:
  - approaches to improve generalisation implementing some preference rules
- four basic principles:
  - smaller networks should be prefered (cf. Ockham's razor)
  - the hypothesis set H of smaller networks is smaller than the hypothesis set of large networks
  - smaller weights should be prefered

Regularisation

- regularisation techniques:
  - approaches to improve generalisation implementing some preference rules
- four basic principles:
  - smaller networks should be prefered (cf. Ockham's razor)
  - the hypothesis set H of smaller networks is smaller than the hypothesis set of large networks
  - smaller weights should be prefered
  - neurons with logistic activation function and small weights are almost linear, networks of linear neurons can be replaced by a single neuron

Bayesian Regularisation

- Bayesian regularization minimizes a linear combination of squared errors and weights

Bayesian Regularisation

- Bayesian regularization minimizes a linear combination of squared errors and weights
    - It also modifies the linear combination so that at the end of training the resulting network has good generalization qualities

Bayesian Regularisation

- Bayesian regularization minimizes a linear combination of squared errors and weights
  - It also modifies the linear combination so that at the end of training the resulting network has good generalization qualities
  - See MacKay (Neural Computation, Vol. 4, No. 3, 1992, pp. 415 to 447)

Bayesian Regularisation

- Bayesian regularization minimizes a linear combination of squared errors and weights
  - It also modifies the linear combination so that at the end of training the resulting network has good generalization qualities
  - See MacKay (Neural Computation, Vol. 4, No. 3, 1992, pp. 415 to 447)
  - and Foresee and Hagan (Proceedings of the International Joint Conference on Neural Networks, June, 1997) for more detailed discussions of Bayesian regularization

Bayesian Regularisation as an Optimisation Problem

- The network is trained using data set $D = \mathbf{x}^{(m)}, t^{(m)}$

Bayesian Regularisation as an Optimisation Problem

- The network is trained using data set $D = \mathbf{x}^{(m)}, t^{(m)}$
- so as to minimise the sum of the errors squared

$$E_D(\mathbf{w}) = 1/2 \sum_m \sum_i (t_i^{(m)} - y_i(\mathbf{x}^{(m)}; \mathbf{w}))^2$$

Bayesian Regularisation as an Optimisation Problem

- The network is trained using data set $D = \mathbf{x}^{(m)}, t^{(m)}$
- so as to minimise the sum of the errors squared

$$E_D(\mathbf{w}) = 1/2 \sum_m \sum_i (t_i^{(m)} - y_i(\mathbf{x}^{(m)}; \mathbf{w}))^2$$

- weight-decay Regularization adds an additional term to this objective function

$$M(w) = \beta E_D + \alpha E_W$$

Bayesian Regularisation as an Optimisation Problem

- The network is trained using data set $D = \mathbf{x}^{(m)}, t^{(m)}$
- so as to minimise the sum of the errors squared

$$E_D(\mathbf{w}) = 1/2 \sum_m \sum_i (t_i^{(m)} - y_i(\mathbf{x}^{(m)}; \mathbf{w}))^2$$

- weight-decay Regularization adds an additional term to this objective function

$$M(w) = \beta E_D + \alpha E_W$$

- where

$$E_W = 1/2 \sum_i w_i^2$$

Bayesian Regularisation algorithm (Foresee and Hagan)

0. Initialize α, β and the weights
1. Take one step of the Levenberg-Marquardt algorithm to minimize the objective function
$$F(w) = \beta E_D + \alpha E_W$$
2. Compute the effective number of parameters
$$\gamma = N - 2\alpha tr(\mathbf{H})^{-1}$$
making use of the Gauss-Newton approximation to the Hessian available in the Levenberg-Marquardt training algorithm
3. Compute new estimates for the objective function parameters
$$\alpha = \frac{\gamma}{2E_w(\mathbf{w})}$$
and
$$\beta = \frac{n - \gamma}{2E_D(\mathbf{w})}$$
4. Now iterate steps 1 through 3 until convergence.

References and Further Reading

Cybenko, G.V. (1989). Approximation by Superpositions of a Sigmoidal function, Mathematics of Control, Signals, and Systems, Vol. 2 pp. 303–314

MacKay, D (2004).; Bayesian methods for neural networks – FAQ, available on: http://www.inference.phy.cam.ac.uk/mackay/Bayes_FAQ.html

Bishop, C.M. (2006) Pattern Recognition and Machine Learning, Springer-Verlag New York, Inc., NJ, USA ©2006 ISBN:0387310738

http://crsouza.com/2009/11/neural-network-learning-by-the-levenberg-marquardt-algorithm-with-bayesian-regularization-part-1/