Backpropagation

Wen Yu

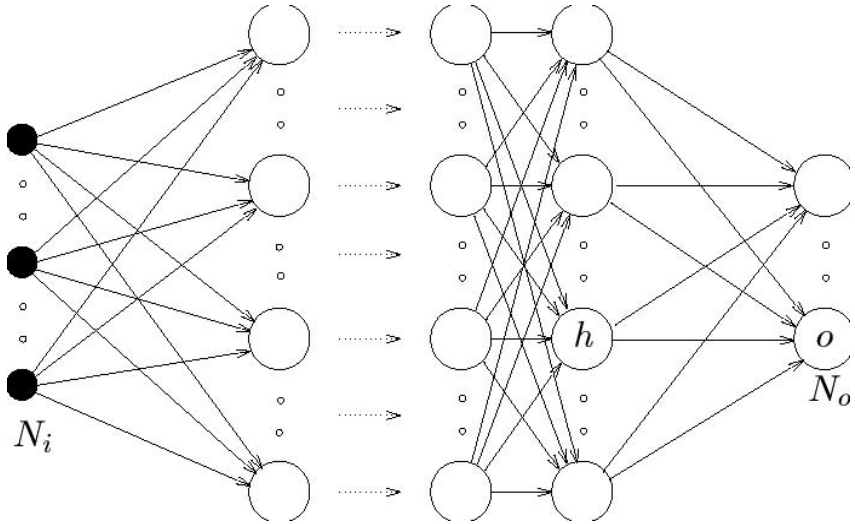
https://www.ctrl.cinvestav.mx/~yuw/
yuw@ctrl.cinvestav.mx

Backpropagation

- Delta rule: Update of weights in output layer, it is not applicable to hidden layer because we don't know the desired values for hidden nodes
- Solution: Propagating errors from output nodes to hidden nodes

BACKPROPAGATION (BP) learning
Proposed first by Werbos (1974), current formulation by Rumelhart, Hinton, and Williams (1986)
Error backpropagation can be continued downward if the net has more than one hidden layer

MultiLayer Perceptron (MLP)



Gradient descent (steepest descent)

Delta rule

$$w(k+1) = w(k) - \eta e(k) u(k)$$

where

$$e(k) = y(k) - d(k)$$

where $y(k)$ is neuron output, $d(k)$ is the desired output, $u(k)$ is the input to the weight $w(k)$.

Stochastic gradient descent (SGD): the instantaneous sum of the squared output errors

$$J(k) = \frac{1}{2} \sum_{k=1}^{l} e^2(k)$$

Because $\frac{\partial}{w(k)} J(k)$ only includes $e(k)$,

$$w(k+1) = w(k) - \eta e(k) u(k)$$

In each iteration k, the gradient is only evaluated at a single $e_o(k)$. This is the key difference between stochastic gradient descent and batched gradient descent.

Gradient descent

Gradient descent is a method for unconstrained mathematical optimization. It is a first-order iterative algorithm.
If $f(x)$ is differentiable in a neighborhood of a point, then $f(x)$ decreases fastest if it goes from a in the direction of the negative gradient of $f(x)$ at a

$$a(k+1) = a(k) - \eta \bigtriangledown f$$

until the local minimum

$$\bigtriangledown f = 0$$

Chain rule

For single layer NN

$$y(k) = w(k)\, u(k)$$

both $w(k)$ and $u(k)$ are vector, $d(k)$ is the desired output,

$$e(k) = y(k) - d(k)$$

We want to move $w(k)$ such taht $J(k)$ decreases fastest,

$$J(k) = \frac{1}{2} \sum_{k=1}^{n} e^2(k)$$

Chain rule

Gradient descent method is

$$w\left(k+1\right)=w\left(k\right)-\eta\bigtriangledown J$$

where

$$\begin{aligned}\bigtriangledown J &= \frac{\partial J}{\partial w(k)}=\frac{\partial J}{\partial e(k)}\frac{\partial e(k)}{\partial w(k)}\\ &= e\left(k\right)\frac{\partial[y(k)-d(k)]}{\partial w(k)}=e\left(k\right)\frac{\partial[w(k)u(k)-d(k)]}{\partial w(k)}\\ &= e\left(k\right)u\left(k\right)\end{aligned}$$

So

$$w\left(k+1\right)=w\left(k\right)-\eta u\left(k\right)e\left(k\right)$$

Chain rule

Single layer NN

$$y(k) = \phi[w(k)\, u(k)]$$

$$\begin{aligned}
\nabla J &= \frac{\partial J}{\partial w(k)} = \frac{\partial J}{\partial e(k)} \frac{\partial e(k)}{\partial w(k)} \\
&= e(k) \frac{\partial [y(k) - d(k)]}{\partial w(k)} = e(k) \frac{\partial [\phi[w(k)u(k)] - d(k)]}{\partial w(k)} \\
&= e(k) \frac{\partial [\phi[w(k)u(k)] - d(k)]}{\partial [w(k)u(k) - d(k)]} \frac{\partial [w(k)u(k) - d(k)]}{\partial w(k)} \\
&= \phi' u(k)\, e(k)
\end{aligned}$$

So

$$w(k+1) = w(k) - \eta \phi' u(k)\, e(k)$$

Chain rule

Two-layer NN

$$y(k) = V(k)\,\phi\left[W(k)\,u(k)\right]$$

$$J(k) = \frac{1}{2}\sum_{k=1}^{n} e_o^2(k), \qquad e_o(k) = y(k) - d(k)$$

here

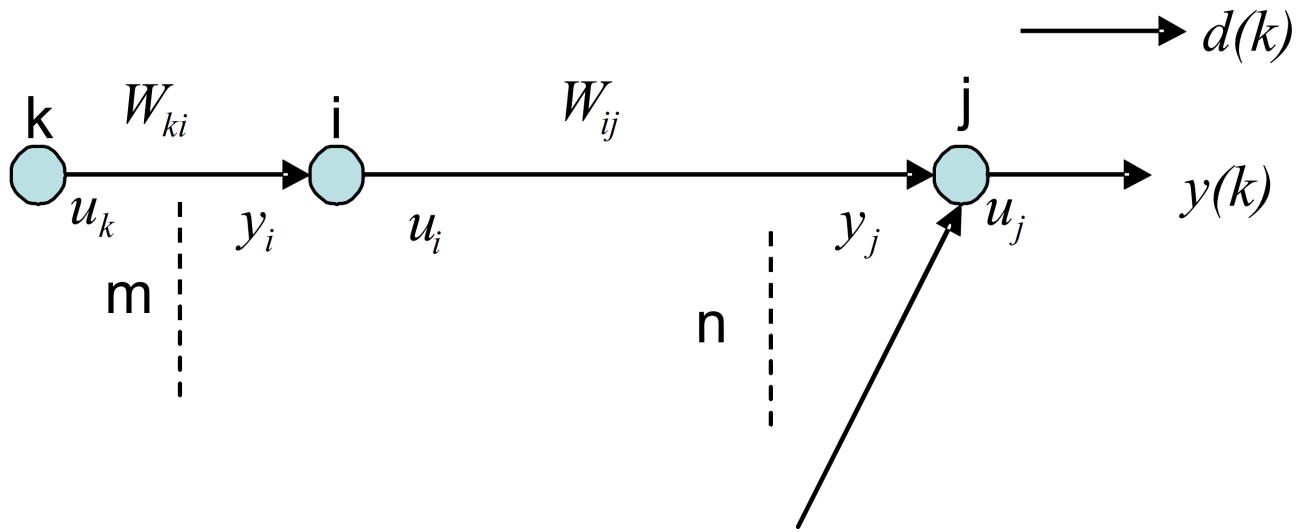
$$\begin{aligned}\nabla J &= \frac{\partial J}{\partial v_i(k)} = \frac{\partial J}{\partial e_o(k)}\frac{\partial e_o(k)}{\partial v_i(k)} \\ &= e(k)\frac{\partial[y(k)-d(k)]}{\partial v_i(k)} = e(k)\frac{\partial[V(k)\phi[\cdot]-d(k)]}{\partial v_i(k)} \\ &= e(k)\,\phi[\cdot]\end{aligned}$$

Gradient descent method

$$v_i(k+1) = v_i(k) - \eta\nabla J = v_i(k) - \eta e(k)\,\phi[\cdot]$$

Chain rule

$$y(k) = W_1(k)\,\phi_1\left[W_2(k)\,\phi_2\left[W_2(k)\,\phi_3 \cdots u(k)\right]\right]$$



Gradient descent

$$w_{ij}(k+1) = w_{ij}(k) - \eta \bigtriangledown J = w_{ij}(k) - \eta \frac{\partial J(k)}{\partial w_{ij}(k)}$$

where $J(k) = \frac{1}{2}\sum_{k=1}^{n} e_o^2(k)$, $e_o(k) = y(k) - d(k)$

Chain rule

Chain rule

$$J(k)=\frac{1}{2}\sum_{k=1}^{n}e_o^2(k),\qquad e_o(k)=y(k)-d(k)$$

$$\frac{\partial J(k)}{\partial w_{ij}(k)}=\frac{\partial J(k)}{\partial e_o(k)}\frac{\partial e_o(k)}{\partial u_j(k)}\frac{\partial u_j(k)}{\partial y_j(k)}\frac{\partial y_i(k)}{\partial w_{ij}(k)}$$

$$\begin{aligned}\frac{\partial J(k)}{\partial e_o(k)}&=e_o(k),\qquad \frac{\partial e_o(k)}{\partial u_j(k)}=1\\ \frac{\partial u_j(k)}{\partial y_j(k)}&=\frac{\partial \phi[y_j(k)]}{\partial y_j(k)}=\phi_j'\\ \frac{\partial y_j(k)}{\partial w_{ij}(k)}&=\frac{\partial [u_i(k)w_{ij}(k)]}{\partial w_{ij}(k)}=u_i(k)\end{aligned}$$

So

$$\bigtriangledown J=\frac{\partial J(k)}{\partial w_{ij}(k)}=e_o(k)\phi' u_i(k)$$

then

$$w_{ij}(k+1)=w_{ij}(k)-\eta\bigtriangledown J=w_{ij}(k)-\eta e_o(k)\phi_j' u_i(k)$$

Chain rule

Chain rule

$$\frac{\partial J(k)}{\partial w_{ki}(k)} = \frac{\partial J(k)}{\partial e_o(k)} \frac{\partial e_o(k)}{\partial u_j(k)} \frac{\partial u_j(k)}{\partial y_j(k)} \frac{\partial y_j(k)}{\partial u_i(k)} \frac{\partial u_i(k)}{\partial y_i(k)} \frac{\partial y_i(k)}{\partial w_{ki}(k)}$$

$$\begin{aligned}
\frac{\partial J(k)}{\partial e_o(k)} &= e_o(k), \qquad \frac{\partial e_o(k)}{\partial u_j(k)} = 1 \\
\frac{\partial u_j(k)}{\partial y_j(k)} &= \phi_j' \\
\frac{\partial y_j(k)}{\partial u_i(k)} &= \frac{\partial [u_i(k) w_{ij}(k)]}{\partial u_i(k)} = w_{ij} \\
\frac{\partial u_i(k)}{\partial y_i(k)} &= \phi_i' \\
\frac{\partial y_i(k)}{\partial w_{ki}(k)} &= \frac{\partial [u_i(k) w_{ki}(k)]}{\partial w_{ki}(k)} = u_k(k)
\end{aligned}$$

So

$$w_{ki}(k+1) = w_{ki}(k) - \eta \frac{\partial J(k)}{\partial w_{ki}(k)} = w_{ki}(k) - \eta e_o(k) \phi_j' w_{ij} \phi_i' u_k(k)$$

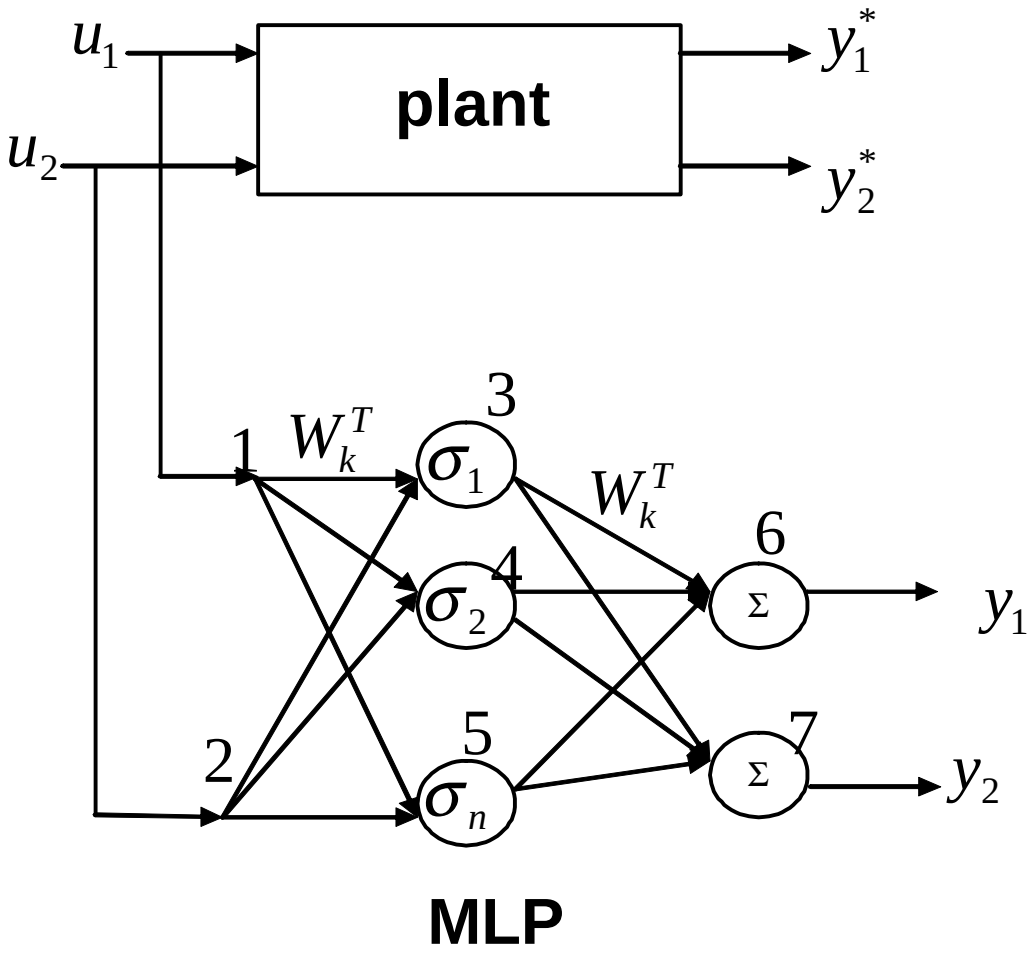
Backpropagation for MLP

$$w_{ij}(k+1) = w_{ij}(k) - \eta e_j(k) u_i(k)$$
$$w_{ki}(k+1) = w_{kj}(k) - \eta e_i(k) u_k(k)$$

where

$$e_j(k) = e_o(k) \phi'_j$$
$$e_i(k) = e_o(k) \phi'_j w_{ij} \phi'_i$$

Example


u_1
u_2
plant
y_1^*
y_2^*
1
2
3
4
5
6
7
W_k^T
W_k^T
σ_1
σ_2
σ_n
Σ
Σ
y_1
y_2

MLP

Example

Feedforward propagation

$$
\begin{aligned}
O_3 &= \sigma_3\left(w_{13}u_1 + w_{23}u_2\right)\\
O_4 &= \sigma_4\left(w_{14}u_1 + w_{24}u_2\right)\\
O_5 &= \sigma_5\left(w_{15}u_1 + w_{25}u_2\right)\\
y_1 &= O_6 = \sigma_6\left(w_{36}O_3 + w_{46}O_4 + w_{56}O_5\right)\\
y_2 &= O_7 = \sigma_7\left(w_{37}O_3 + w_{47}O_4 + w_{57}O_5\right)
\end{aligned}
$$

Example

Backpropagation

$$
\begin{aligned}
e_{o1} &= y_1 - y_1^* \\
e_{o2} &= y_2 - y_2^* \\
e_6 &= e_{o1}\dot{\sigma}_6 = e_{o1} \\
e_7 &= e_{o2}\dot{\sigma}_7 = e_{o2} \\
e_3 &= e_6\dot{\sigma}_3 w_{36} + e_7\dot{\sigma}_3 w_{37} \\
e_4 &= e_6\dot{\sigma}_4 w_{46} + e_7\dot{\sigma}_4 w_{47} \\
e_5 &= e_6\dot{\sigma}_5 w_{56} + e_7\dot{\sigma}_5 w_{57}
\end{aligned}
$$

Learning

$$
\begin{aligned}
& w_{36}(k+1) = w_{36}(k) - \eta O_3(k) e_6(k) = w_{36}(k) - \eta O_3(k) e_{o1} \\
& \vdots \\
& w_{13}(k+1) = w_{13}(k) - \eta O_1(k) e_3(k) \\
& = w_{13}(k) - \eta u_1(k) \left[e_{o1}\dot{\sigma}_3 w_{36} + e_{o2}\dot{\sigma}_3 w_{37} \right]
\end{aligned}
$$

Backpropagation Improvement

- Momentum (Rumelhart et al, 1986)
- Adaptive Learning Rates (Smith, 1993)
- Normalizing Input Values (LeCun et al, 1998)
- Bounded Weights (Stinchcombe and White, 1990)
- Penalty Terms (eg. Saito & Nakano, 2000)
- Conjugant Gradient
- Levenberg-Marquart (damped least-squares, non-linear least squares)

BP with momentum

- the change in weight dependent of the past weight change

$$w_{ij}(k+1) = w_{ij}(k) - \eta \frac{\partial J(k)}{\partial w_{ij}(k)} + \alpha \Delta w_{ij}(k-1)$$

- Avoid sudden change of directions of weight update (smoothing the learning process)
- avoid oscillation at large learning rate, error is no longer monotonically decreasing

Problems of gradient descent

- long training process (NN)
- gradient decay (deep learning)
- local minima (NN)

Local minimum.

- local minimum. (E may not be reduced to zero) and cannot escape from the local minimum
- Not every function can be modeled by BP, depends on the shape of the error surface. Too many valleys/wells will make it easy to be trapped in local minima
- Possible solutions:
  - Try nets with different # of hidden layers and hidden nodes (they may lead to different error surfaces, some might be better than others)
  - Try different initial weights (different starting points on the surface)
  - Forced escape from local minima by random perturbation (e.g., simulated annealing)

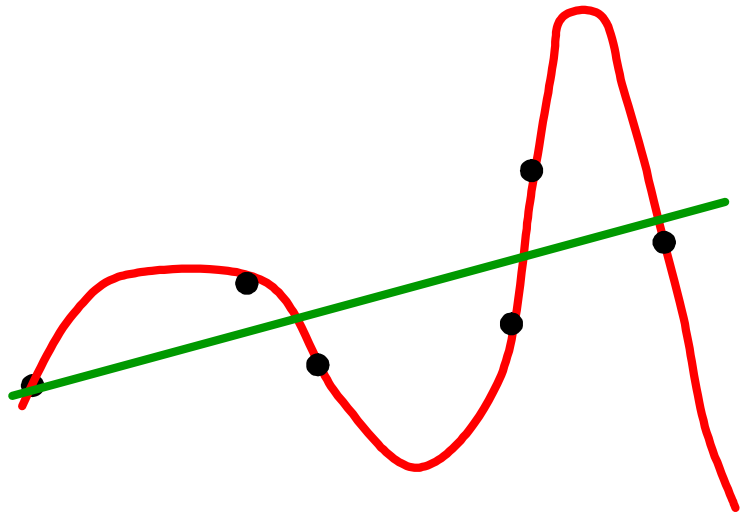
How good are MLP networks

1. The learning algorithm determines how good the error on the training set is minimized.
2. The number of learning samples determines how good the training samples represent the actual function. There is no theoretically idea number
3. Then umber of hidden units determines the 'expressive power' of the network,

Over-fitting/over-training problem

- trained net fits the training samples perfectly (E reduced to 0)
- it does not give reflect the inputs not in the training set
  - The target values may be unreliable.
  - There is sampling error. If the model is very flexible it can model the sampling error really well.

Overfitting



Which model do you believe?

- The complicated model fits the data better.
- But it is not economical

A model is convincing when it fits a lot of data surprisingly well.

Possible solutions:

- More and better samples, persistent exciting (PE), e.g., introducing noise into samples
- Using smaller net if possible

Using larger error bound (forced early termination)

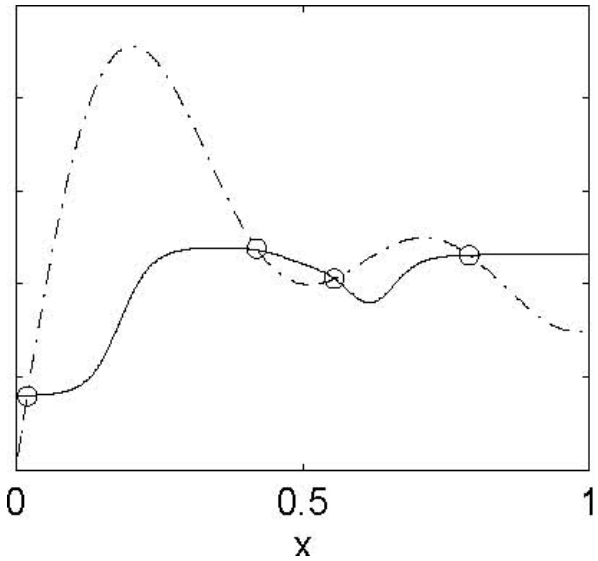
- leave some (~10%) samples as test data (not used for weight update)
- periodically check error on test data
- Learning stops when error on test data starts to increase

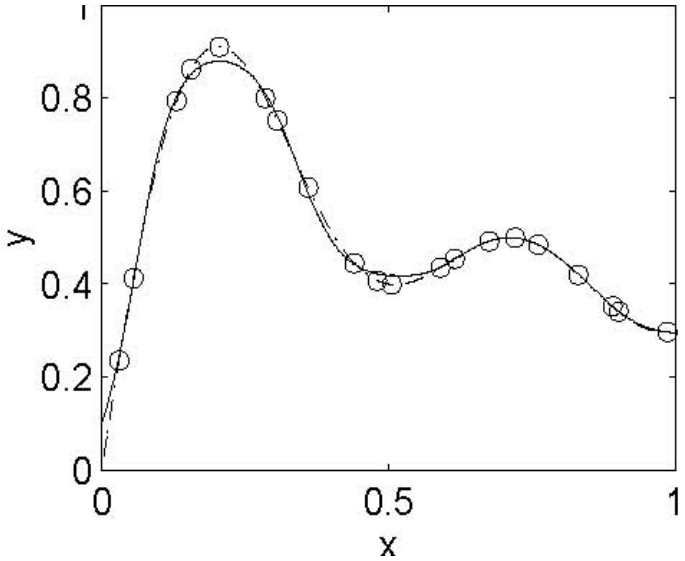
Occam's Razor

William of Ockham, a 14th-century English philosopher
Occam's Razor is the problem-solving principle that recommends searching for explanations constructed with the smallest possible set of elements.
The principle of simplicity and effectiveness:

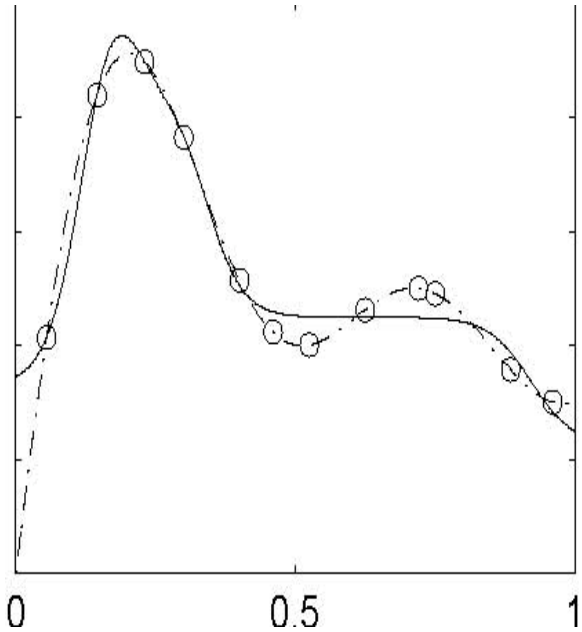
- The simplest explanation is usually the best one.
- If it is not necessary, do not increase the entity

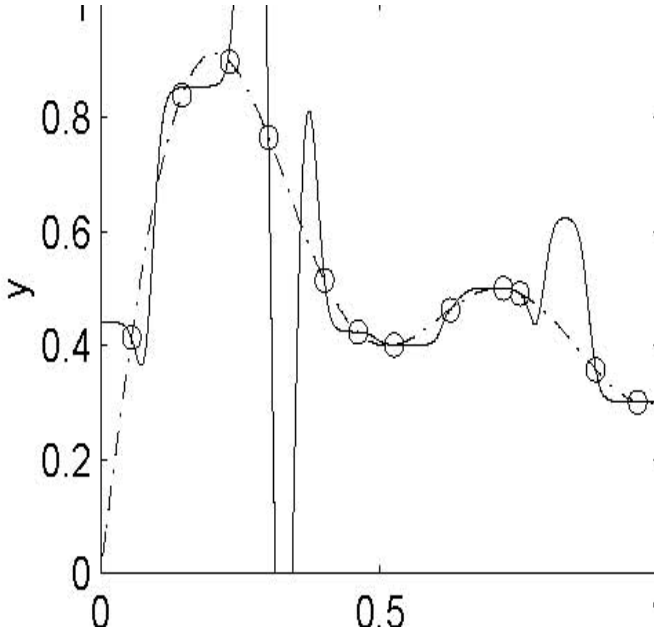
The number of learning samples



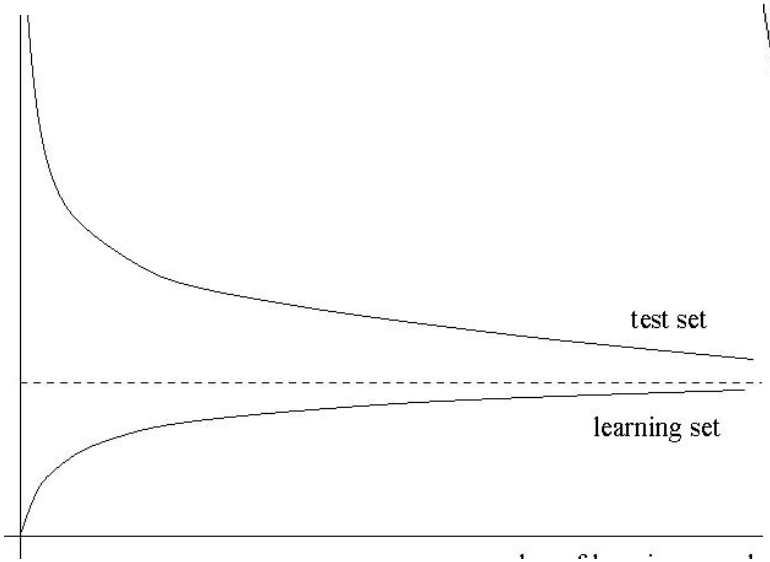


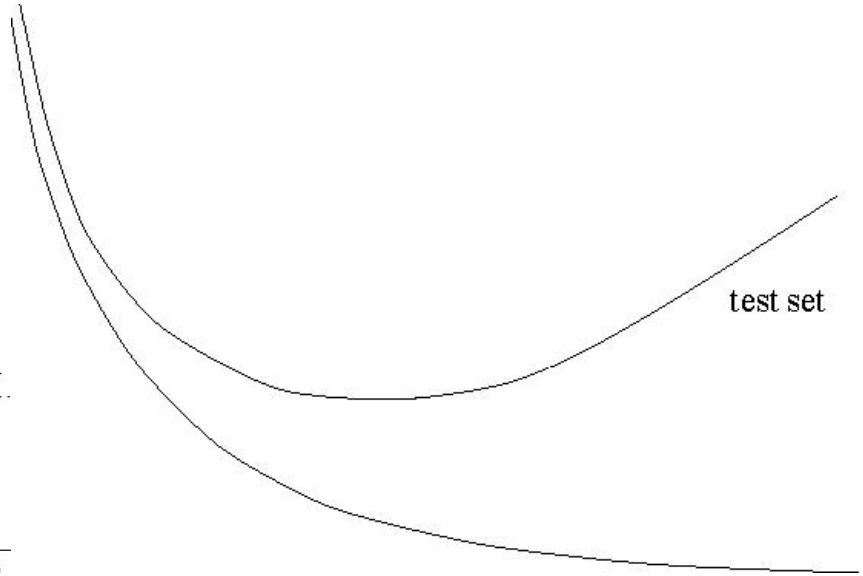
The number of hidden units: over-training





Samples and hidden units and vis error rate





Batch mode

- Weight update once per each epoch (cumulated over all P samples)
- Smoothing the training sample outliers
- Learning independent of the order of sample presentations
- Usually slower than in sequential mode