Training Techniques and Tips (for Backpropagation, and in some cases, more generally)

Two References:

Neural Networks Tricks of the Trade, Orr and Muller, eds., Springer LNCS 1524

Backpropagator's Review
http://www.dontveter.com/bpr/bpr.html

Not a Panacea

- Backpropagation seems like a wonderful idea, with a well-founded theory.
- However, it may need significant help to solve certain problems.
- Usually you cannot simply throw raw data at it and expect it to give good results.

Scrub the Data Set

- If the network is to learn a function, make sure that the samples are functional, i.e. that they don't specify conflicting outputs for the same input value.

- For example, if clinical outcomes are the output, it is possible that two patients with the same symptoms have different outputs; presenting these to the network will mean that it will never fully converge.

Normalize the Inputs

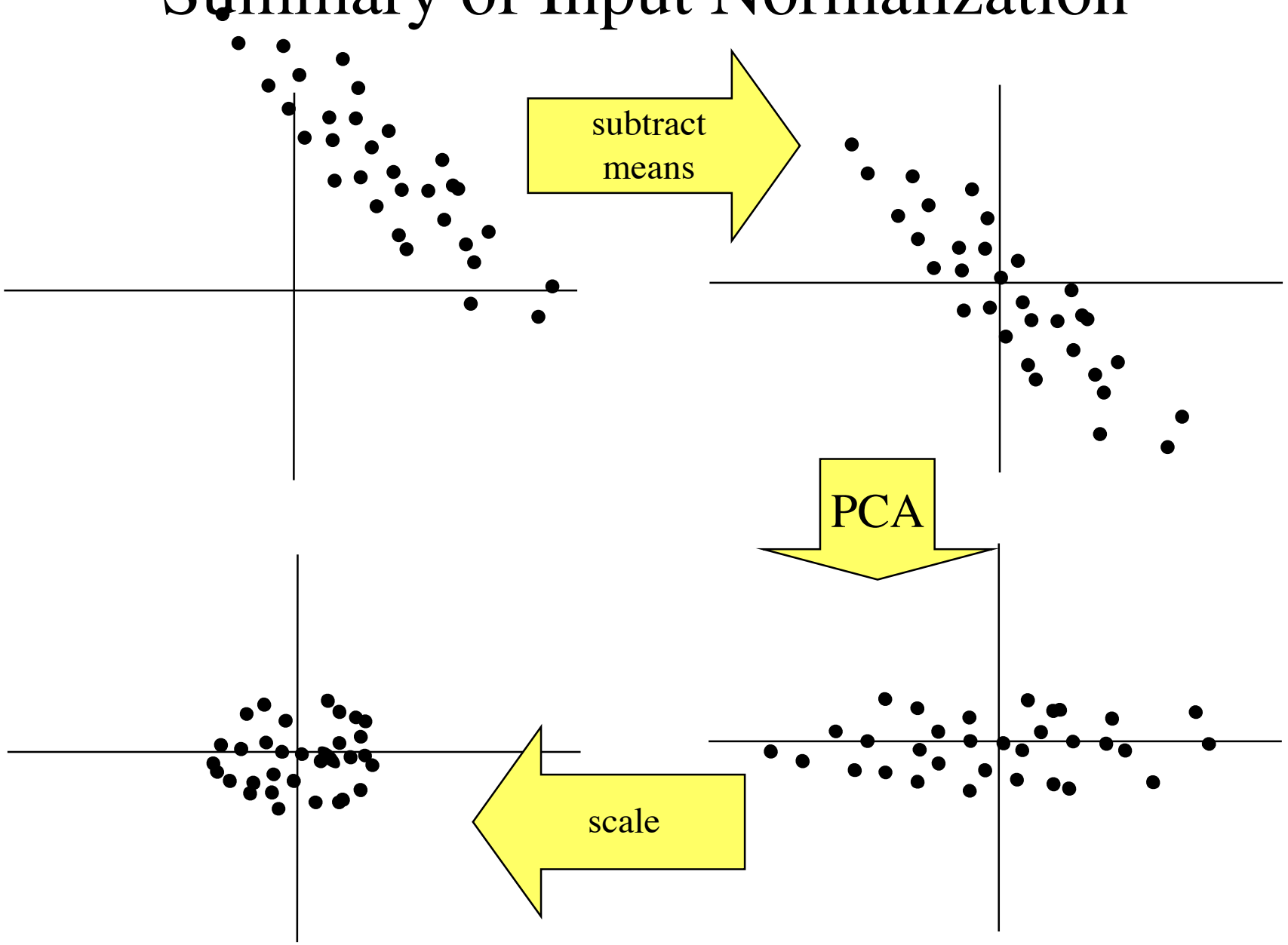
- Better if **mean** of a particular variable is near 0.
  - Then weight changes are less likely to be synchronized, since some will be positive, others negative.
  - Therefore, **subtract the actual mean** from the variable before training.
- Better if the variables are **scaled** to have similar auto-covariances, defined as

  (sum-of-squares of variable)/(number of samples)
  - Then the **weights will learn at similar rates**.
  - Exception: When some variables are known in advance to be of less significance.

Decorrelate the Inputs

- Better if no two input variables are correlated.
- Correlated inputs analogous to having linearly dependent variables in a linear system.
- A technique called PCA (Principal Components Analysis), aka Karhunen-Loeve Expansion, can be used to remove linear correlations.
- We will look at PCA in more detail later; PCA itself can be done by a PCA neural network.

Summary of Input Normalization



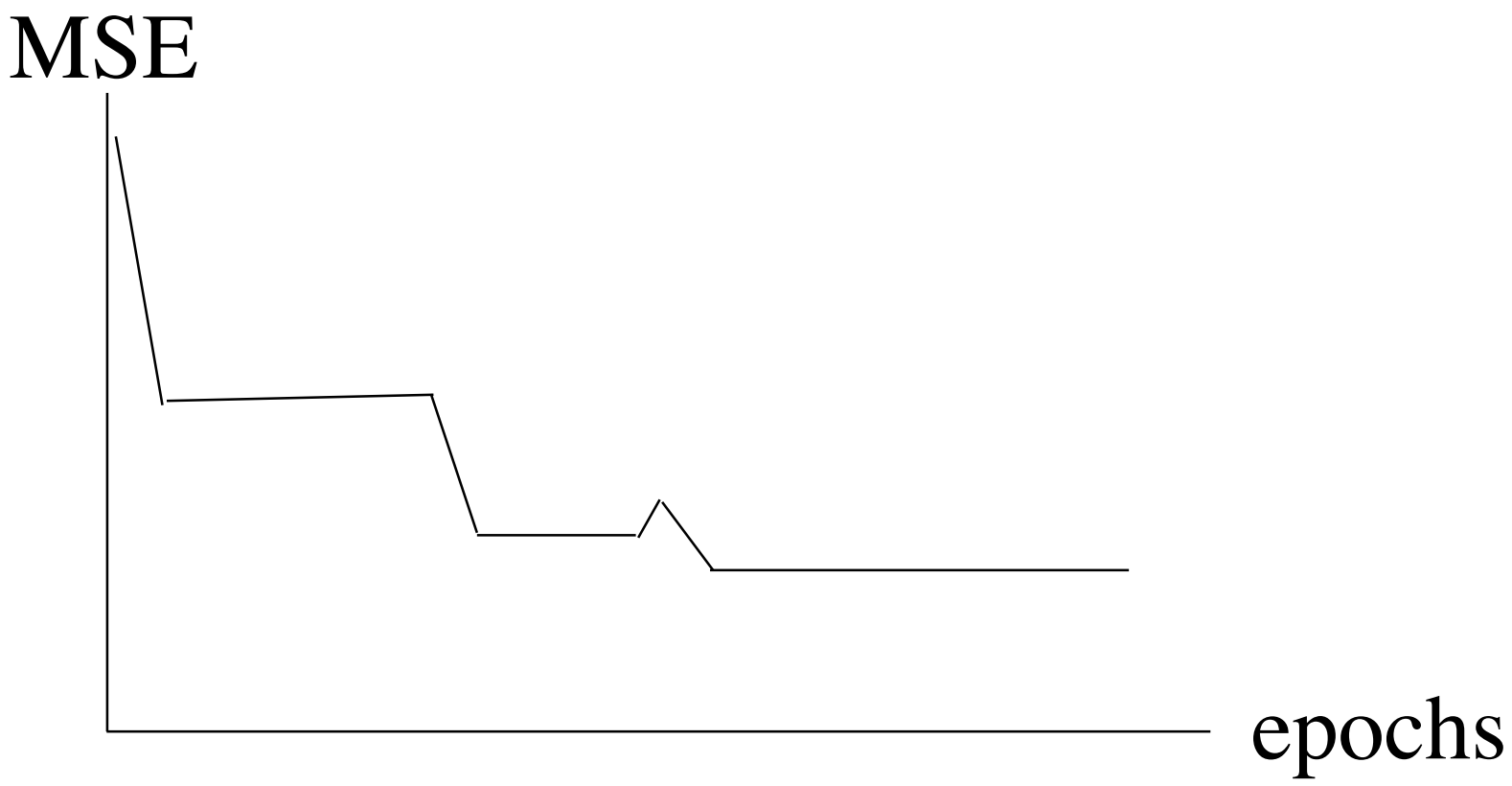
Training Time

- It can be shown (Blum and Rivest, 1992) that training a network using backpropagation is NP hard (which essentially means that it is likely to require exponential time as a function of problem size **in the worst case**). Jirí Síma (2002) showed this to be true even for 1-neuron.

- **But this is not *every* case**. It does not deter practical applications from using it.

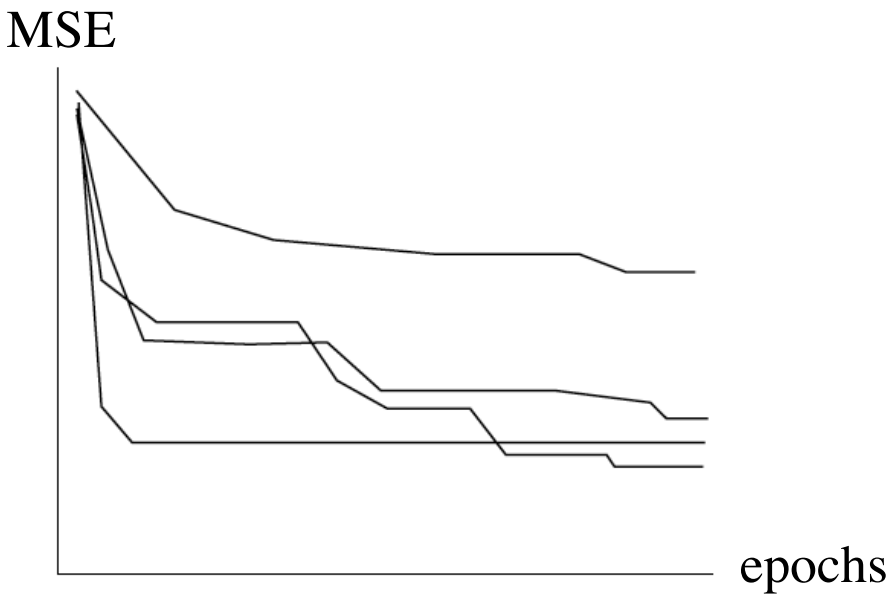
- But training time is a significant consideration for large problems.

Typical BP Training Profile

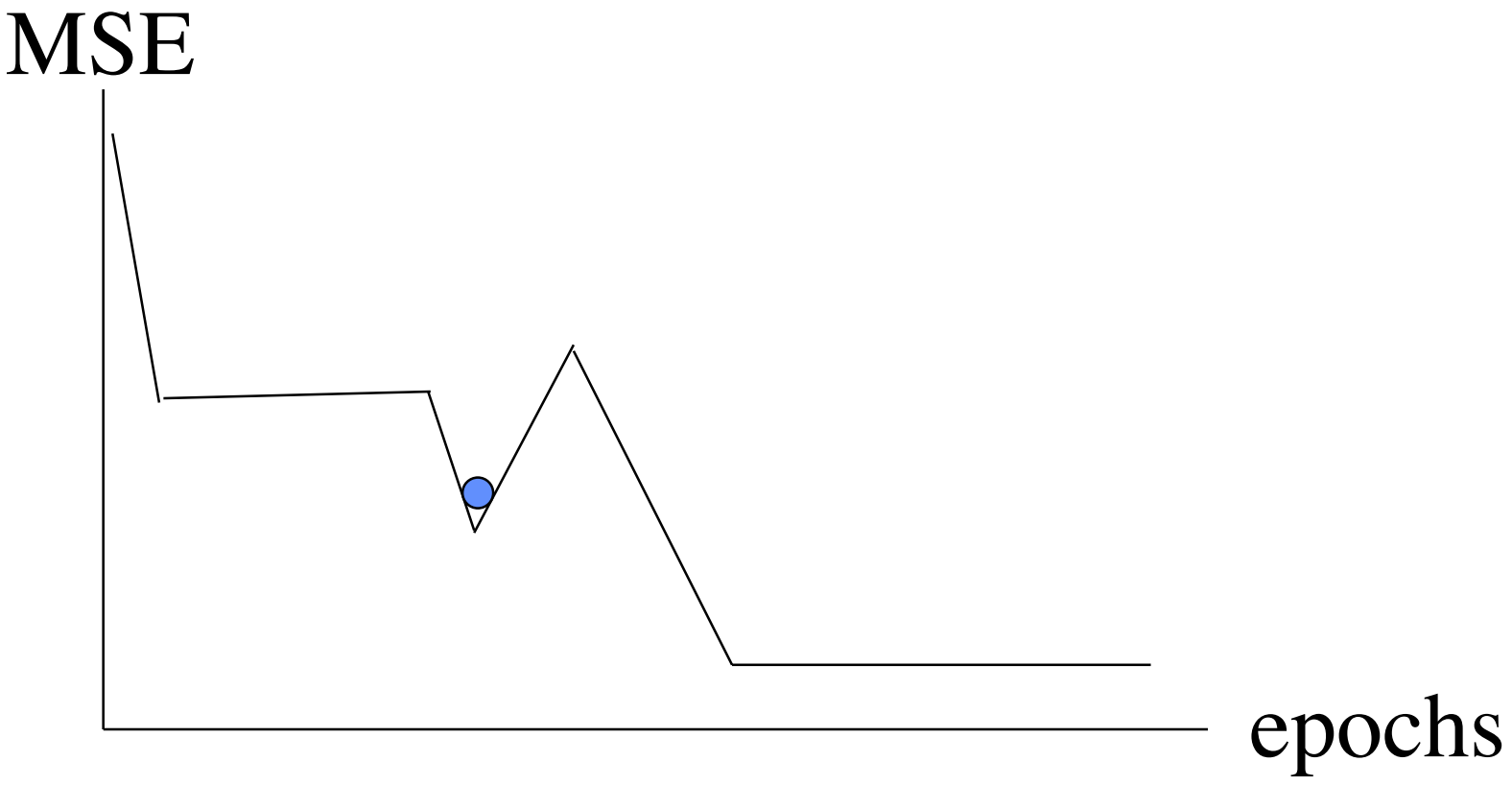


Weight Initialization Will Matter

- Weights should be randomly initialized
- Do not use the same weights or all 0's, as all neurons would then train to be alike.
- You will get different ultimate behaviors based on the chosen starting weights.
- It is worthwhile training multiple times, then choosing the best.



The Dreaded Local Minimum



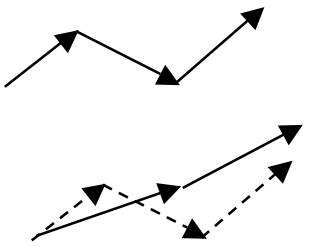
- The more weights, the more likely this is to happen.
- Could "jiggle" (add noise to) the weights
- Could add noise to the input (creating more samples with similar expectations)
- Could try multiple runs, as on previous slide.

Using Momentum

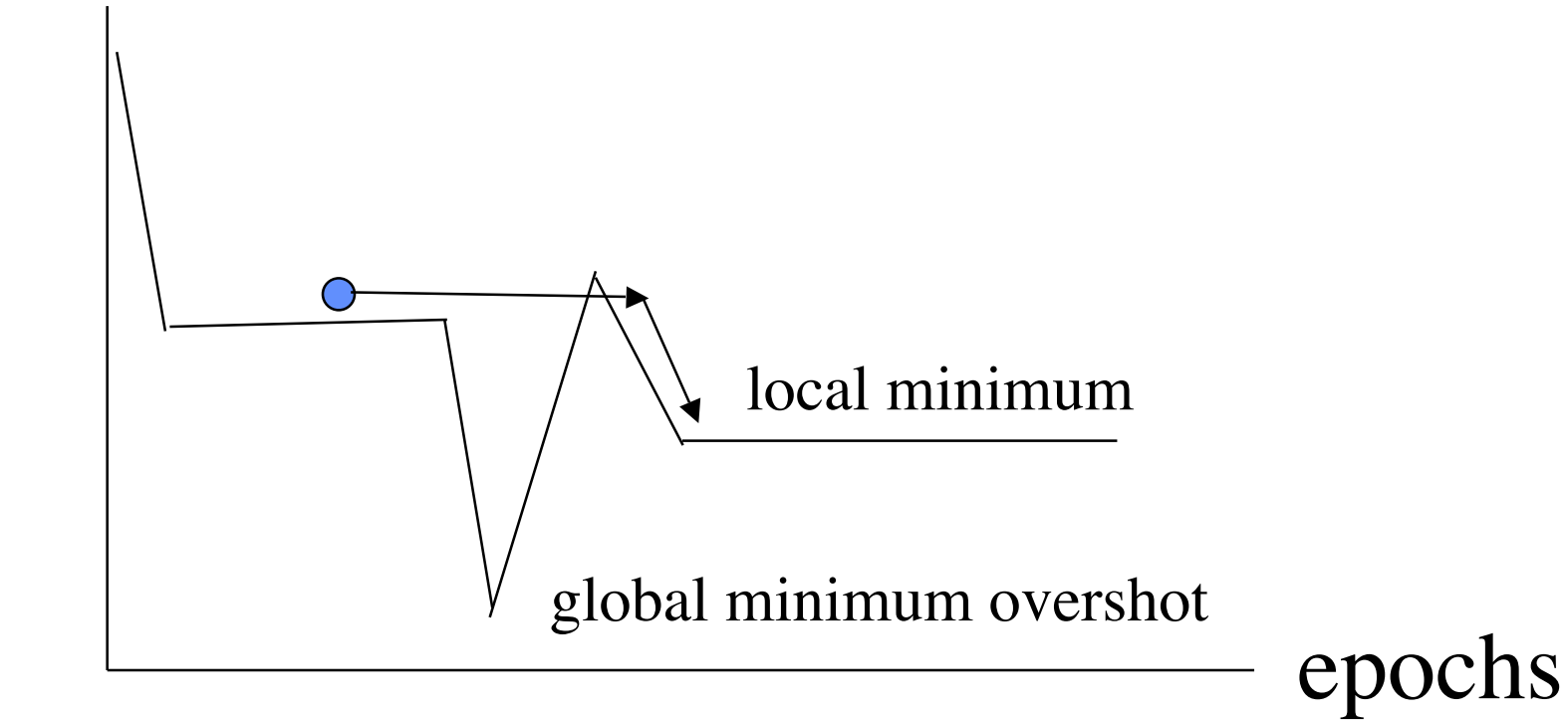
- A momentum term is one that adds some fraction μ of the previous weights to the new weights.
- More specifically,

$$\Delta_{\text{actual}} W = (1 - \mu)\, \Delta_{\text{gradient-descent}} W + \mu W$$

- The purpose of momentum is to
  - accelerate convergence
  - smooth convergence
- Addition is vectors addition, so:
  - without momentum
  - with momenutm



Too Much Momentum can be Bad



It can also cause oscillation.

Momentum should start low and gradually increase (Hinton).

One Example Study

Advances in Engineering Software
Volume 30, Issue 4, April 1999, Pages 291-302

doi:10.1016/S0965-9978(98)00071-4 | How to Cite or Link Using DOI

Analysis of learning rate and momentum term in backpropagation neural network algorithm trained to predict pavement performance

Nii O. Attoh-Okine

Department of Civil and Environmental Engineering, Florida International University, Miami, FL 33199, USA

Learning rate η	Momentum term α	Remarks
0.005	0.9	Gross overlearning from the beginning
0.001	0.5	Relatively few cycles needed for learni
0.001	0.9	Relatively few cycles needed for learni
0.2	0.4	About 1000 cycles needed for learning
0.3	0.5	About 1000 cycles needed for learning
0.5	0.5	About 800 cycles needed for learning

Choose samples so there is maximum information content from one sample to the next.

- Shuffle the training set so that successive samples rarely belong to the same class.
- Present input examples that produce a large error more frequently than ones that produce a small error.

Prefer tansig for inner layers

- Prefer tansig (hyperbolic tangent) rather than logsig for **inner** layers.
  - tansig output is symmetric about 0, logsig is not.
  - tansig will more likely produce outputs close to 0 for the **next stage** of the network, which centers their activation value.

Piecewise Quadratic Approx. to tanh (faster to compute)

x	f(x)
x > 1.92033	0.96016
0 < x <= 1.92033	0.96016 - 0.26037 * (x - 1.92033)^2
-1.92033 < x < 0	0.26037 * (x + 1.92033)^2 - 0.96016
x <= -1.92033	-0.96016

Derivative: $\tanh'(x) = 1-\tanh^2(x)$ can still be used.

Choice of Target Values

- Choosing target values of +1, -1 for a tansig causes the neuron to be driven toward the **saturation region**.
- To get into this region, the weights are large and may become **"stuck"** because small gradient values will not change them sufficiently.
- For inner neurons, scale the tansig to avoid this, e.g.
  - f(x) = 1.7159 tanh(2x/3), which has a maximum 2nd derivative where the function's value is +/- 1.

Weight Initialization

- Assuming that the training set has been normalized and the modified tansig is used.
- Draw the initial weights from a distribution, such as a uniform distribution, with mean 0 and standard deviation **1/sqrt(m)** where m is the **fan-in** (number of inputs to the node).
- This increases likelihood that the input to the sigmoid will have a standard deviation of 1
  (since the latter is the sqrt of the sum of the squares of the weights, for normalized input).

Learning Rates

- **Ideally**, each **weight** should have its own learning rate.
  See the Neural Networks Tricks of the Trade, Orr and Muller, eds., LNCS 1524 for how to choose learning rate based on 2nd derivatives.

- As a substitute, each neuron, or each layer could have its own learning rate.

- Learning rates should be proportional to the sqrt of the number of inputs to the neuron.

- Weights in **earlier layers should be larger** than those in later layers, since the earlier layers tend to have a smaller 2nd derivative contribution to the MSE.

Second Derivatives by Layer

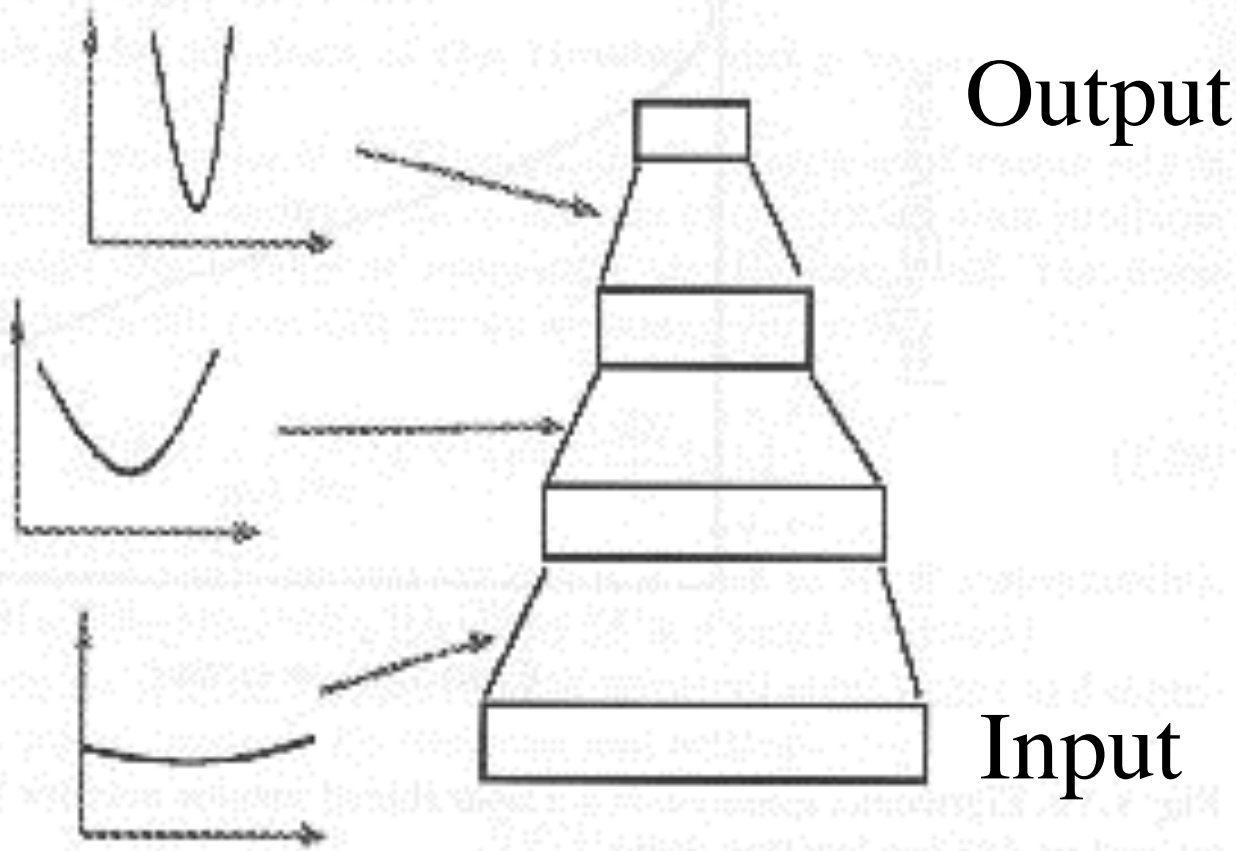


Fig. 1.21. Multilayered architecture: the second derivative is often smaller in lower layers

Validation Technique ("Cross-Validation") & Early Stopping

- Split the training set into **training** and **validation** subsets, e.g. 2:1 or 5:1 ratio.
- Train only on the training subset; **use the validation set for MSE**, every so often (e.g. every 5 epochs).
- **For early stopping:** Stop training **as soon as the validation error goes up**. Then use the weights as they were **before** the error went up.
- Rationale: Avoiding overfitting.

A Validation Error Curve

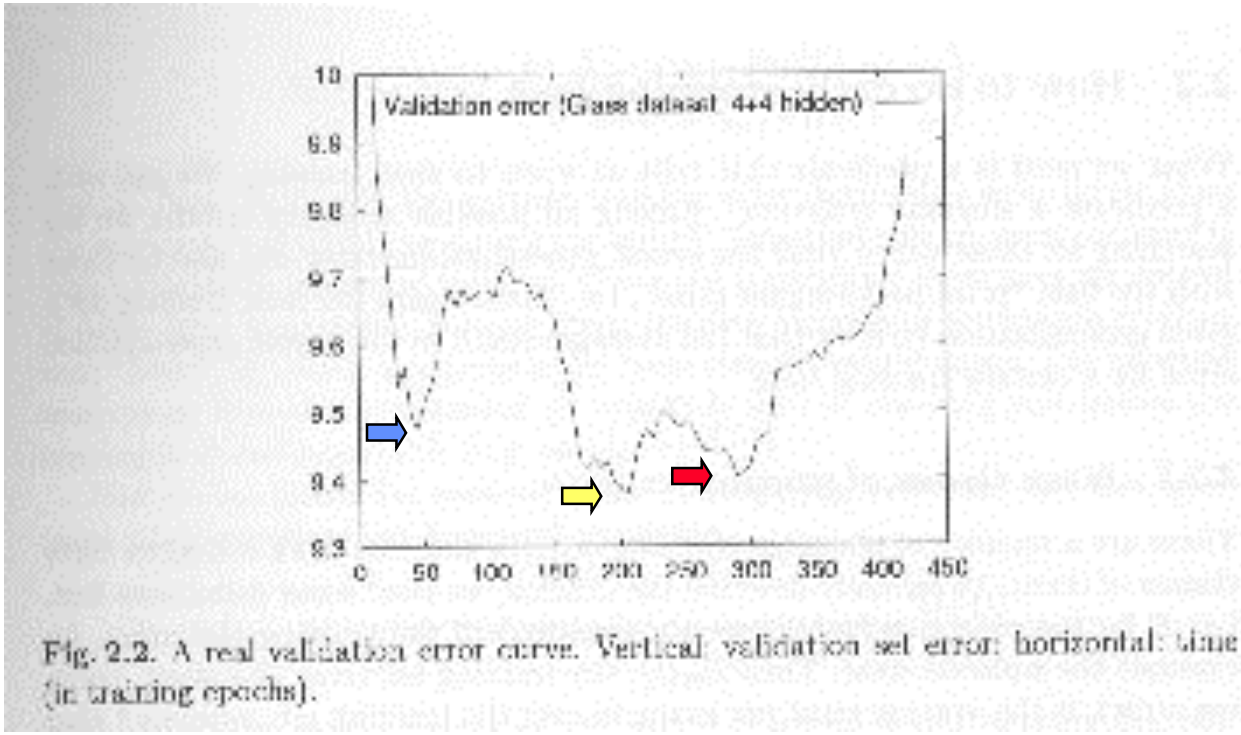
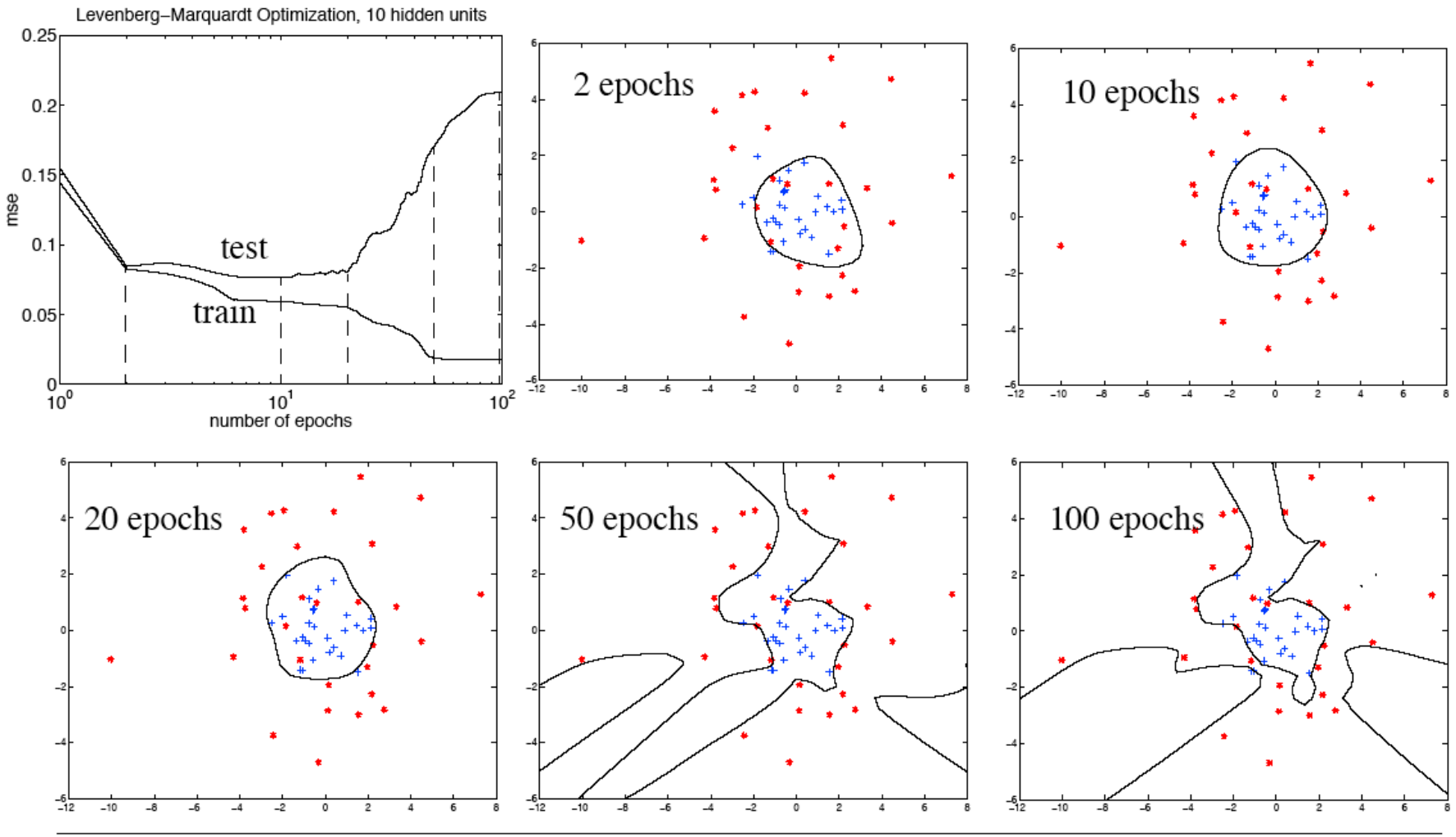


Fig. 2.2. A real validation error curve. Vertical: validation set error; horizontal: time (in training epochs).

See Neural Networks Tricks of the Trade, Orr and Muller, eds., LNCS 1524 for further refinements of the validation idea.

Over-Training Example

Overtraining Example - 10 Hidden Units



Sizing a Network

- Given a problem:
  - How many layers?
  - How many neurons per layer?

Number of Layers

- Theoretically, any function can be emulated over a given range by a network with **just two layers** total, with sufficiently many neurons in the hidden layer. [Cybenko, 1998]

- Practically, 2-3 layers suffice for large families of problems, although more may be used, especially when special feature-selection layers are used, as in the zip-code recognition network.

Neurons

- Choose number of neurons based on the assessed complexity within a layer (number of crests and valleys of a function, for example).

- Two approaches for experimental determination:
  - Start with a large number of neurons and prune.
  - Start with a small number of neurons and build up.

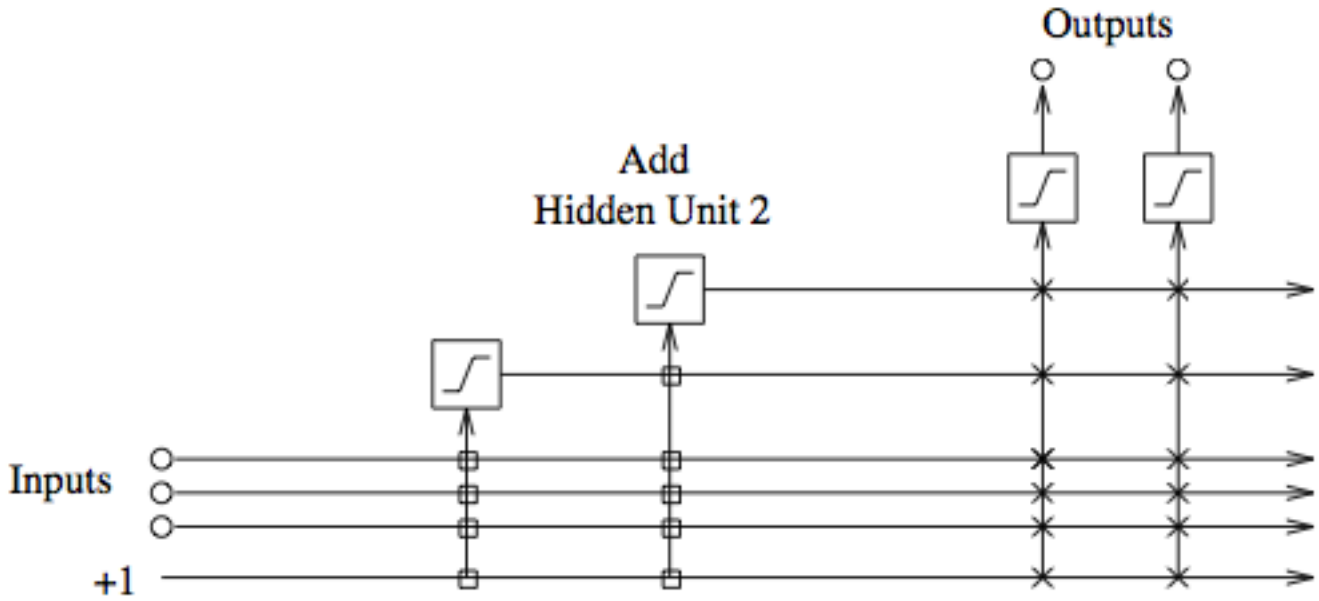
Pruning

- Negligible weights can be eliminated (forced to 0).
- If all input weights to a node are eliminated, the node can be eliminated (output forced to 0).
- If all weights to which a node feeds are negligible, the node itself can be eliminated.
- Vary weights w to see whether $\partial J/\partial w$ is significant; if not, prune the weight.
- Pruning techniques known by colorful names, such as "Optimal Brain Damage".

Building

- Cascade-Correlation Network (Fahlman) adds one neuron at a time, testing the quality of the results and stopping when they are adequate.
- Training by correlation is a technique to be explored later.
- Problem with cascade correlation is that **each added neuron is effectively a new layer**.

Cascade-Correlation Network



Doubling Heuristic

- Start with a small number of neurons in the inner layer.
- If at the conclusion of a training cycle, the MSE is inadequate, repeat with double the number of neurons.
- Or use a different factor other than 2.

Haykin's Rule of Thumb (p166)

Size of Training Set =

O(#Weights/allowable error)

i.e. $N = O(W/\varepsilon)$

where O is "big-oh" notation.

Using Baum-Haussler Rule to Determine Number of Hidden Nodes

The Baum-Haussler rule [10] can be used to determine the maximum number of hidden nodes.

Let $$N_{hmax} = \frac{N_{trains} E_{tolerance}}{N_{inputs} + N_{outputs}}$$

then $N_{hiddens} \leq N_{hmax}$ Where

N_{hmax} is the maximum number of hidden nodes,

$N_{hiddens}$ is the number of nodes in the hidden layer,

$N_{outputs}$ is the number of output nodes and

$E_{tolerance}$ is the error tolerance,

N_{trains} is calculated by:

$$N_{trains} = N_{set} * N_{inputs}$$

Here N_{set} is the number of data points and N_{inputs} is the number of input nodes.

Using the Bayesian Information Criterion (BIC)

Schwarz [11] and Rissanen [12] suggested a method called Bayesian Information Criterion (BIC) to find the optimal number of nodes in hidden layer. The criterion is as follows:

$$BIC = M\ln(MSE) + P\ln(M)$$

where M is the number of data points used in the training step. MSE is the mean squared error of the network. $MSE = \frac{SE}{M}$

P is the number of parameters (weights) used.

Combining Baum-Haussler and BIC:

Refer to 2007 paper:

Fifth International Conference on Software Engineering Research, Management and Applications

Determination the Number of Hidden Nodes of Recurrent Neural Networks for River Flow and Stock Price Forecasting

Suwarin Pattamavorakun
Science and Technology Faculty,
Rajamangala University of Technology,
Pathumthani, Thailand
suwarinp@yahoo.com

Suwat Pattamavorakun
Business Administration Faculty,
Rajamangala University of Technology,
Pathumthani, Thailand
suwatpat@yahoo.com

0-7695-2867-8/07 $25.00 © 2007 IEEE
DOI 10.1109/SERA.2007.63