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

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

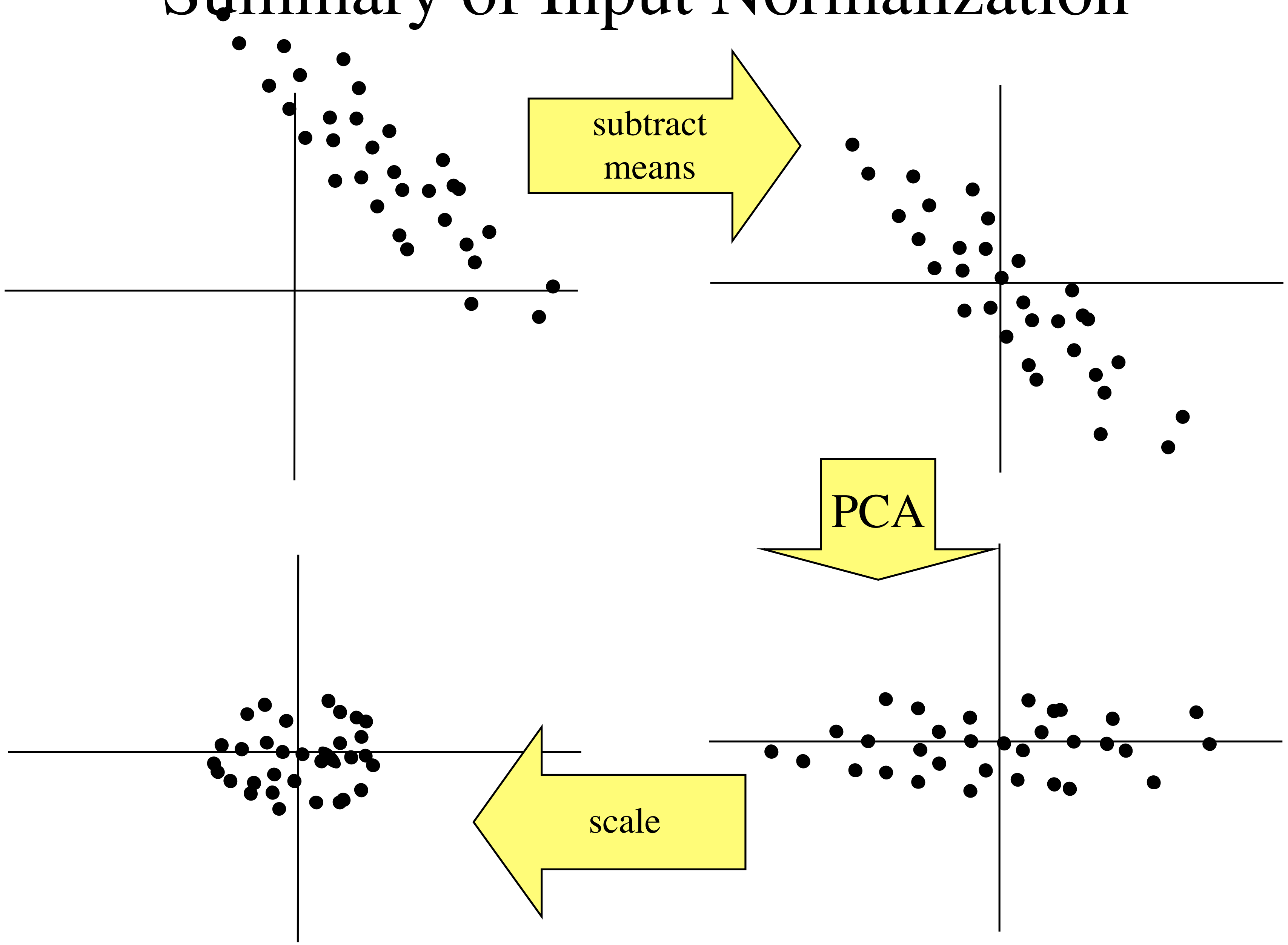
  (sum-of-squares of variable)/(number of samples)

  - 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 later; PCA itself can be done by a PCA neural network.

Summary of Input Normalization



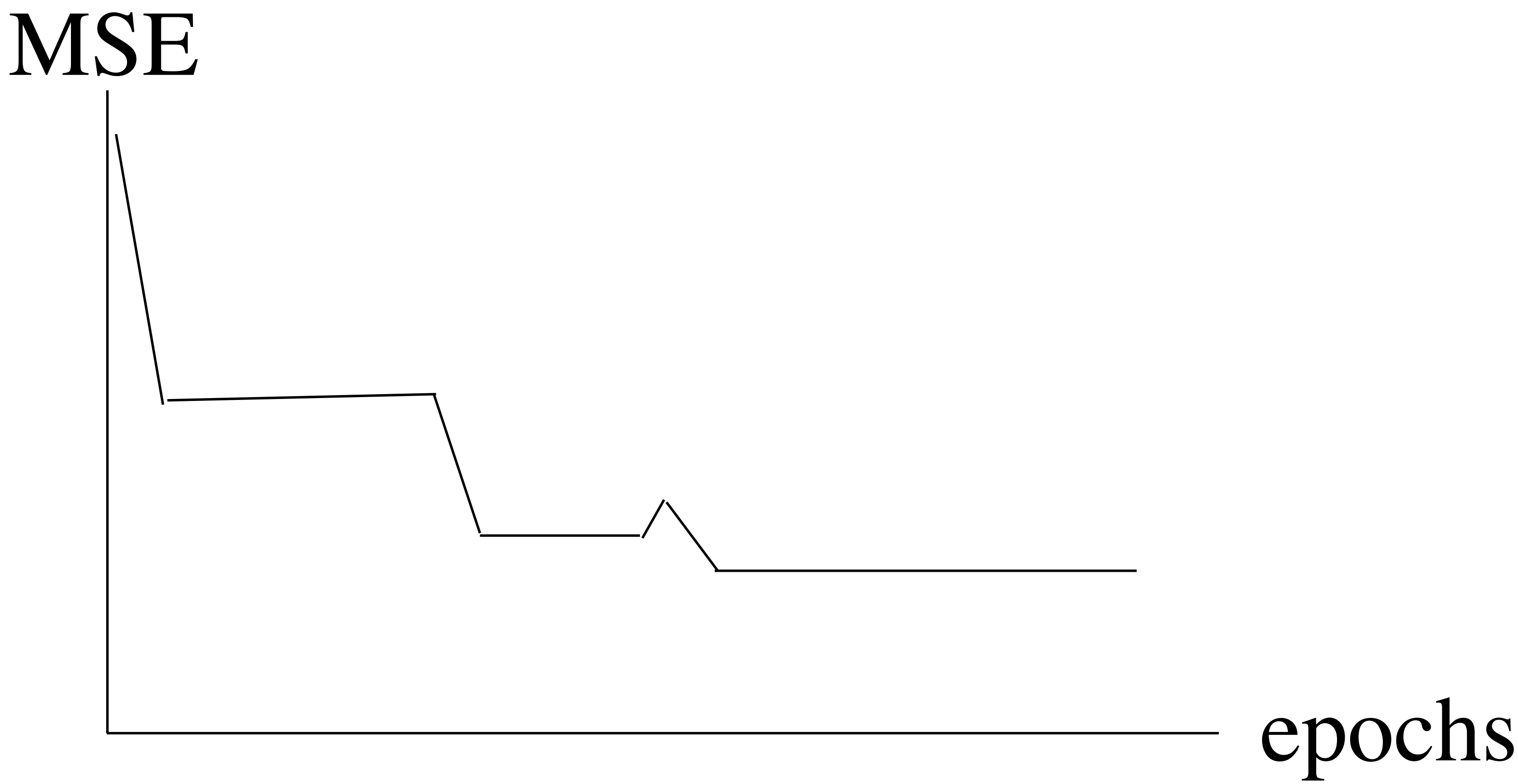
Training Time

- 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). Jií íma (2002) showed this to be true even in the 1-neuron case.

- This alone does not deter practical applications from using it.

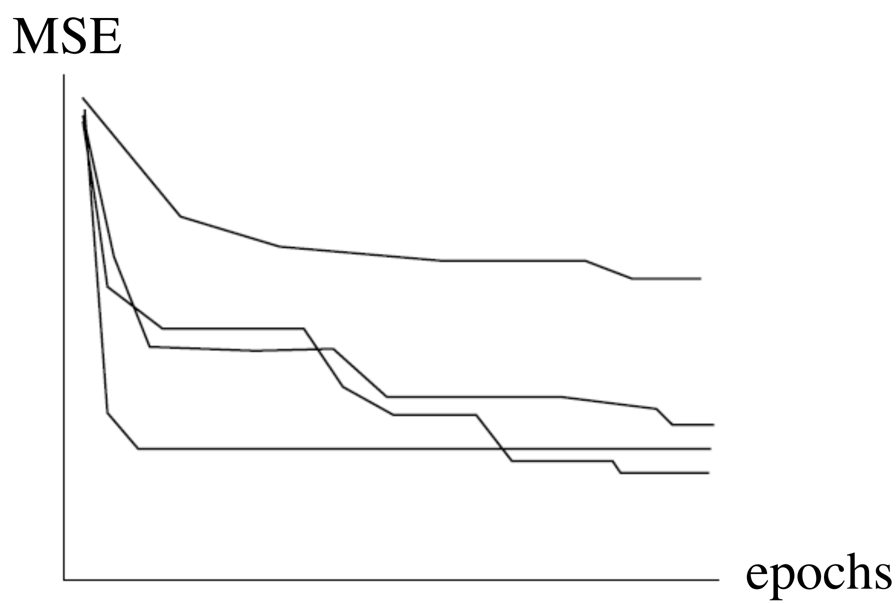
- But training time is a significant consideration for large problems.

Typical BP Training Profile

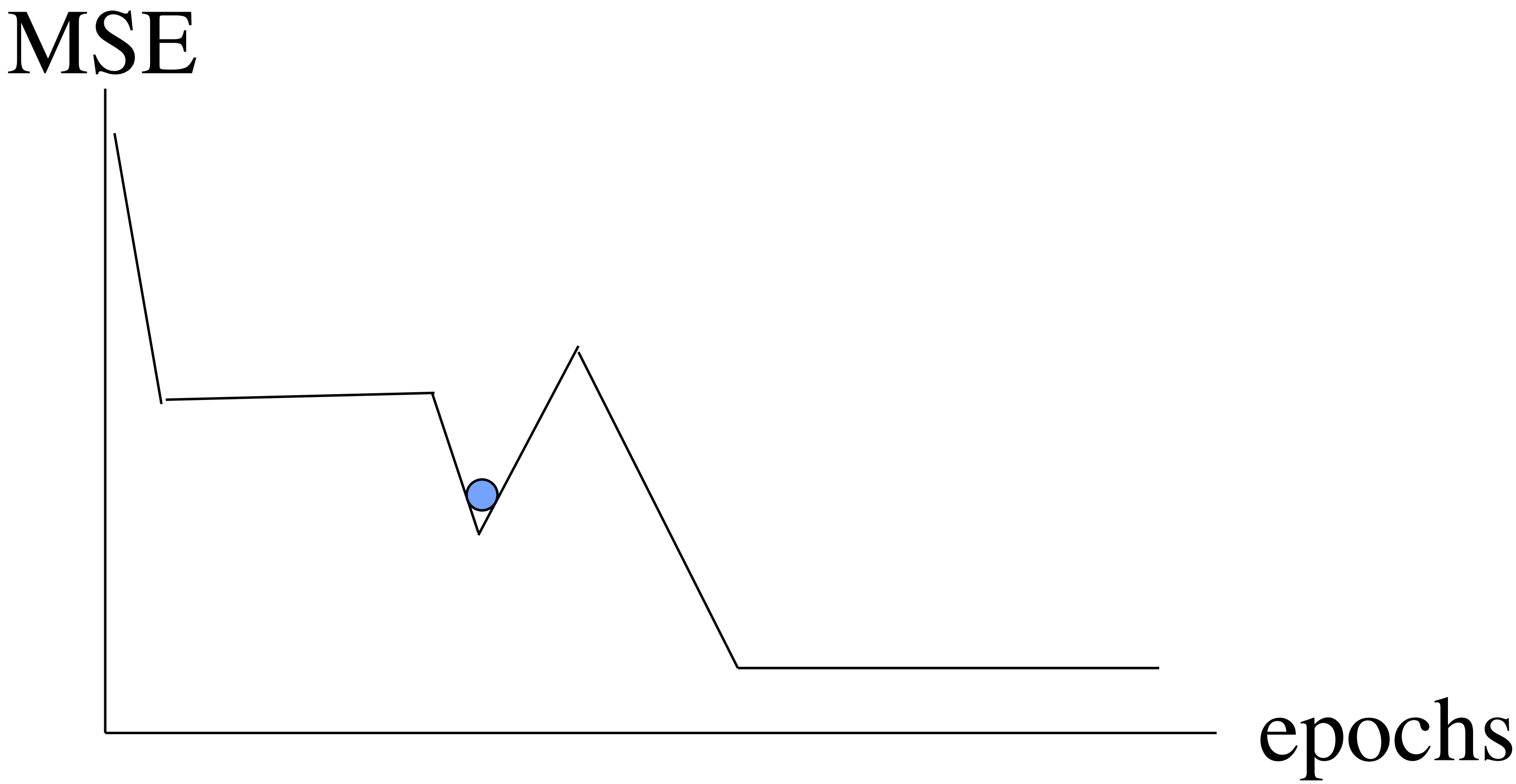


Weight Initialization Will Matter

- 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



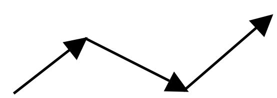
- The more weights, the more likely this is to happen.
- Could “jiggle” (add noise to) the weights
- Could add noise to the input (creating more samples with similar expectations)
- Could try multiple runs, as on previous slide.

Using Momentum

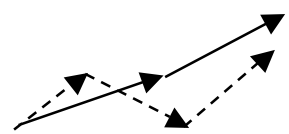
- A momentum term is one that adds some fraction μ of the previous weights to the new weights.
- More specifically,

$$\Delta_{actual}W = (1- \mu)\, \Delta_{gd}W \; + \; \mu W$$

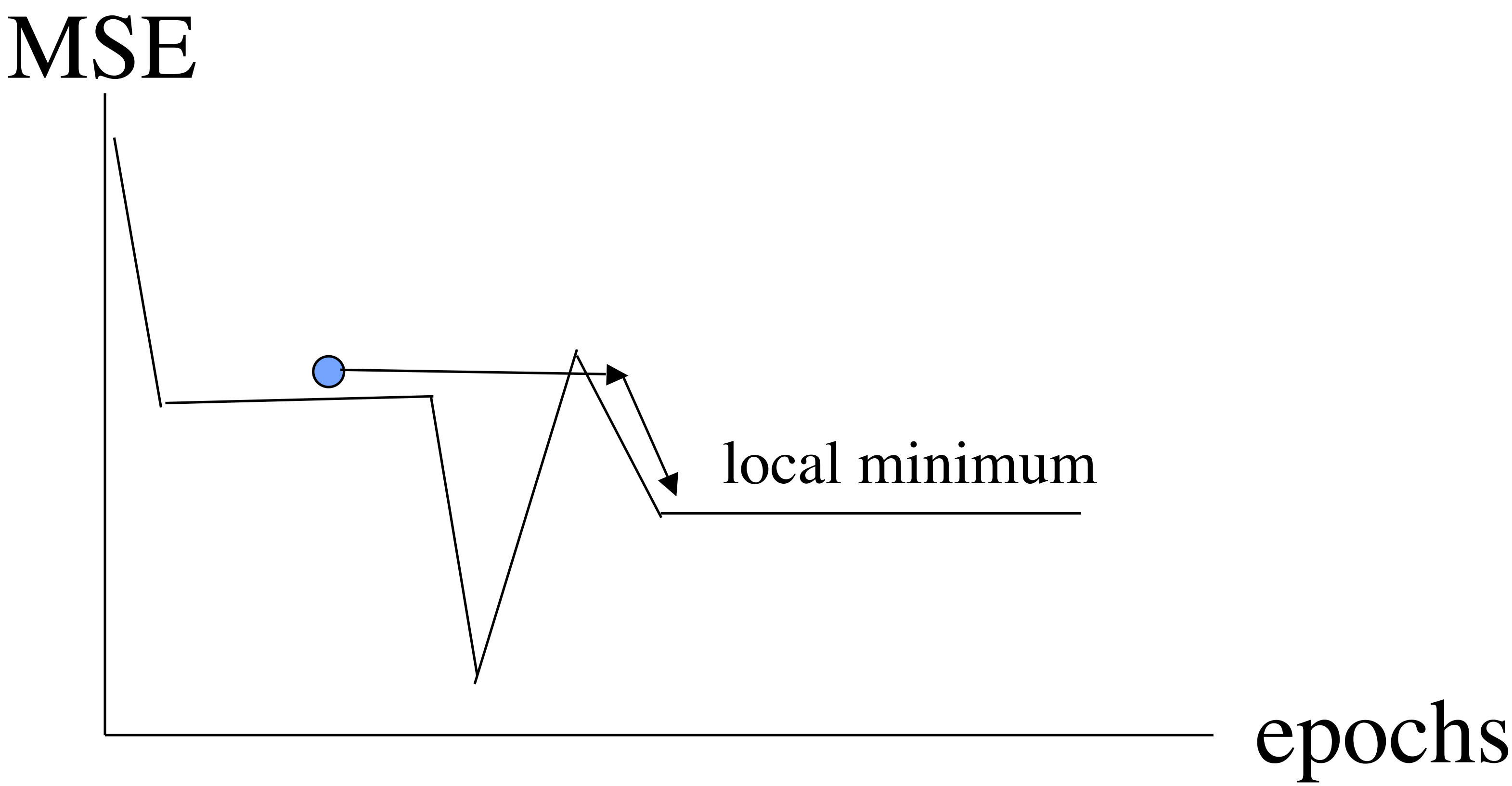
- The purpose of momentum is to
  - accelerate convergence
  - smooth convergence
- Addition is vectors addition, so:
  - without momentum



  - with momenutm



Too Much Momentum can be Bad



Choose examples with maximum information content

- 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

- 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

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.
- It may be better to **choose output targets offset** from these saturation values (e.g. .9, -.9)
- For inner neurons, scale the tansig to get a similar effect, 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 previous sigmoid 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).
- 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 of the MSE.

Second Derivatives by Layer

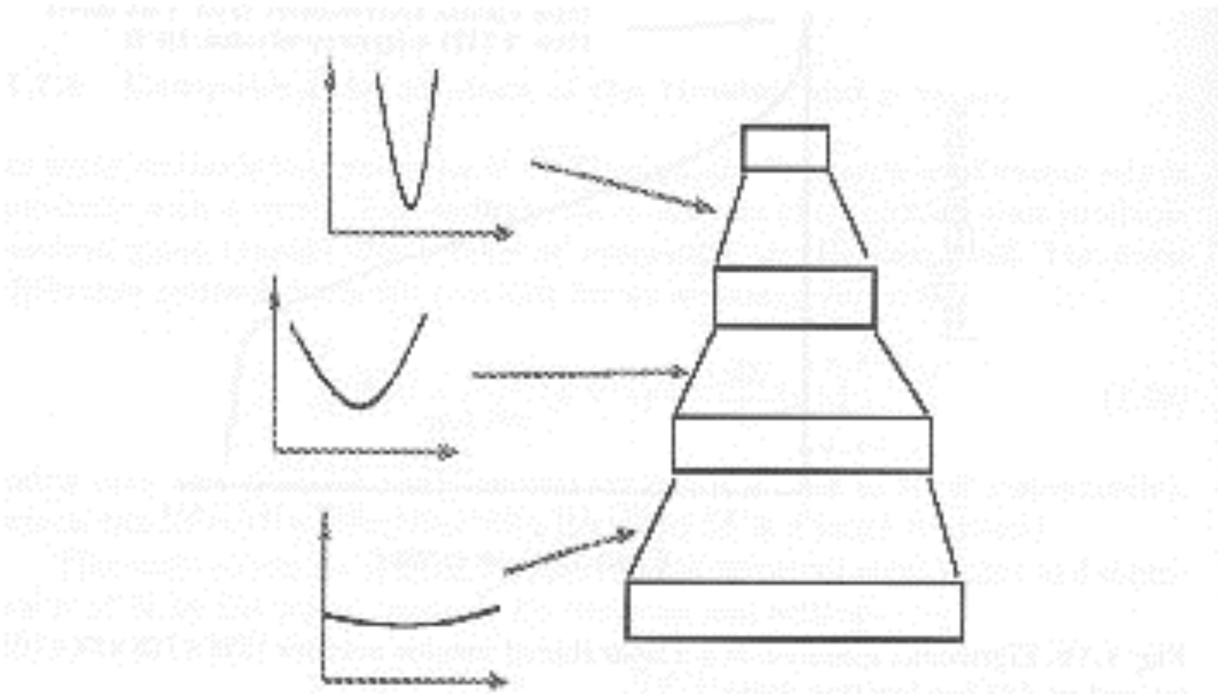


Fig. 1.21. Multilayered architecture: the second derivative is often smaller in lower layers

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.
- Rational: Even though a lower minimum might eventually be reached, the local minima tend to be fairly close in value in practice.

A Validation Error Curve

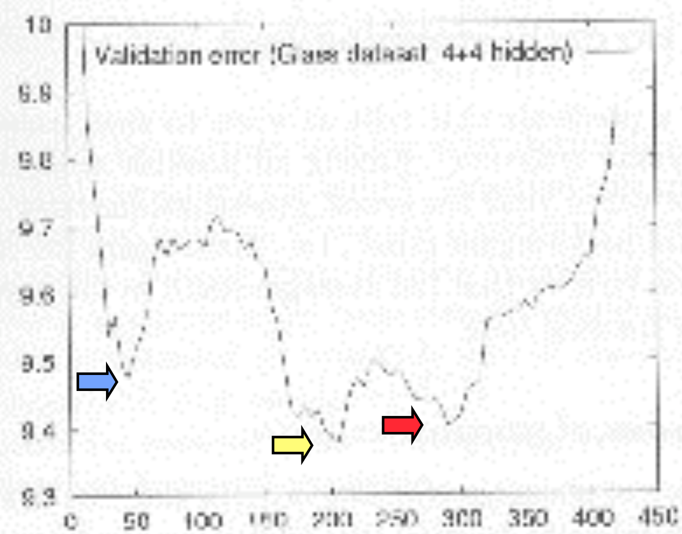


Fig. 2.2. A real validation error curve. Vertical: validation set error; horizontal: time (in training epochs).

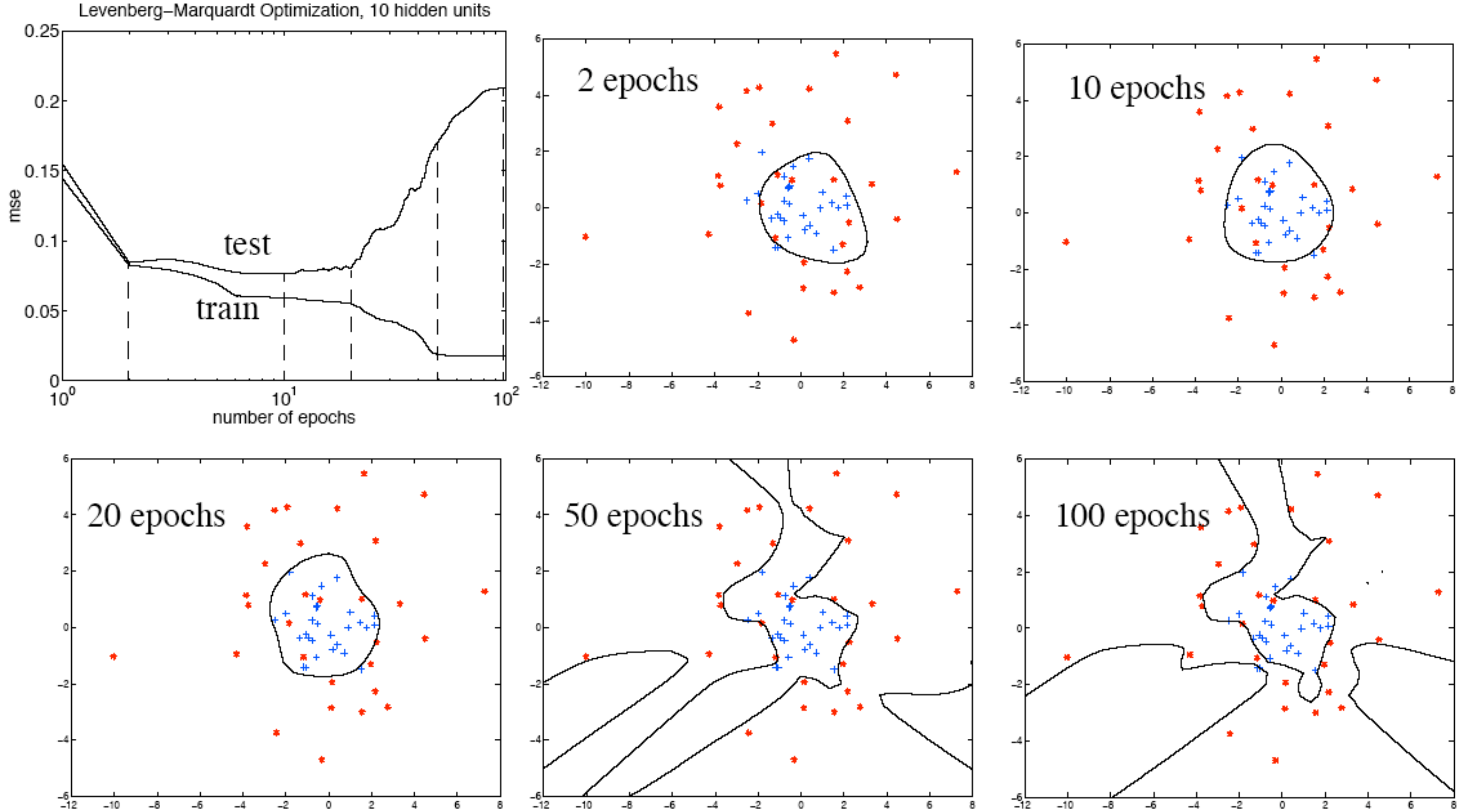
See Neural Networks Tricks of the Trade, Orr and Muller, eds., LNCS 1524 for further refinements of the validation idea.

Over-Fitting

- It is possible for a network to *over-fit* the data, meaning that it learns small variations in the data which might actually be due to noise.
- Another way of saying this is that the network does not generalize well; it is **too specialized**.
- Over-fitting can result if the network has **too many neurons** at its disposal.
- **Validation** is one technique used to help avoid over-fitting.

Over-Training Example

Overtraining Example - 10 Hidden Units



Sizing a Network

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

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.

- 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**.

Doubling Idea

- 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.

Rules of Thumb for Determining the Number of Samples Needed to Train a Given Size Network

- **Baum-Haussler rules** (1989):
- Let
  - W = the total number of weights in the network
  - a = allowed fraction of errors (e.g. .05)
  - N = the total number of neurons
- **Necessary condition for network to be trainable**:

  number of samples > W/a

  Sufficient condition for network to be trainable:

  number of samples ≥ (W/a) * log(N /a)