CISC/CMPE452/COGS 400
Backpropagation II

Ch. 3 Text book

Farhana Zulkernine

NN Application Design

- How can we design networks for particular applications?
- Designing NNs is basically an engineering task.
- As we discussed before, for example, there is no formula to determine the optimal number of hidden units in a BPN for a given task.

NN Application Design (cont…)

- We need to address the following issues for a successful application design:
  1. Appropriate data representation.
  2. Size of training set.
  3. Number of nodes at the different layers.
  4. a) Initial weight values and b) update frequency.
  5. Learning rate.
  6. Optimization by using Momentum.
  7. Training the network and evaluating its performance.
     a) Batch and per-pattern training.
     b) Generalization and performance evaluation.

1. Data Representation

- Typically conversion to binary data value is required when we have categorical input data such as color.
- Analog data can also be converted to 1 and 0s based on some selected threshold θ (if input > θ set it to 1 otherwise set it to 0).
- **Data Preprocessing:** Data transformation techniques can also include value range adjustment, feature selection for better classification of test data, removal of outliers, handle missing data etc.

Encoded Output

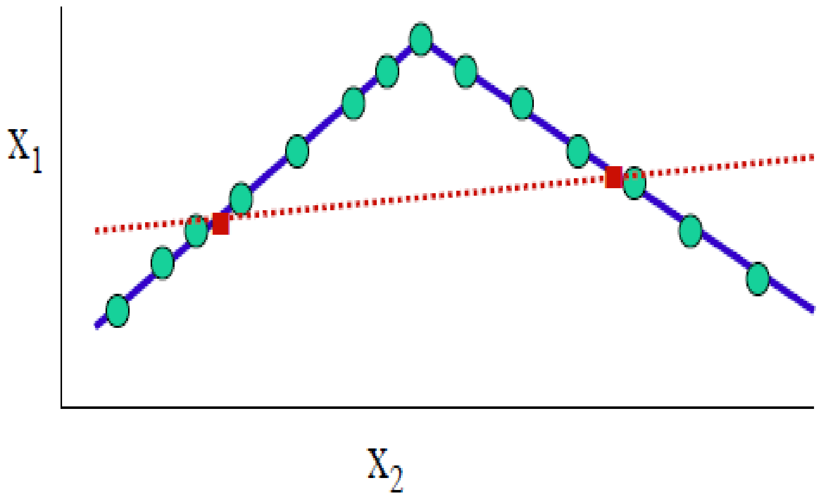
- Can we use n processors for 2^n classes? For example, 8 classes can be expressed as 000,001,…,111 using 3 processor outputs.
  - The problem is that the same weights on the output node would be required to "learn" all the classes causing conflicts.
  - Causes **Crosstalk**.

Crosstalk

- Different training samples or output nodes may require conflicting changes for the same weight, making it difficult to learn the most appropriate value for a weight.
- Cause too much **'crosstalk'** as different trials will try to set the same weight in different ways.
- Such networks usually **need more hidden neurons and longer training**, and **their ability to generalize is weaker** than for the one-neuron-per-feature-value networks.

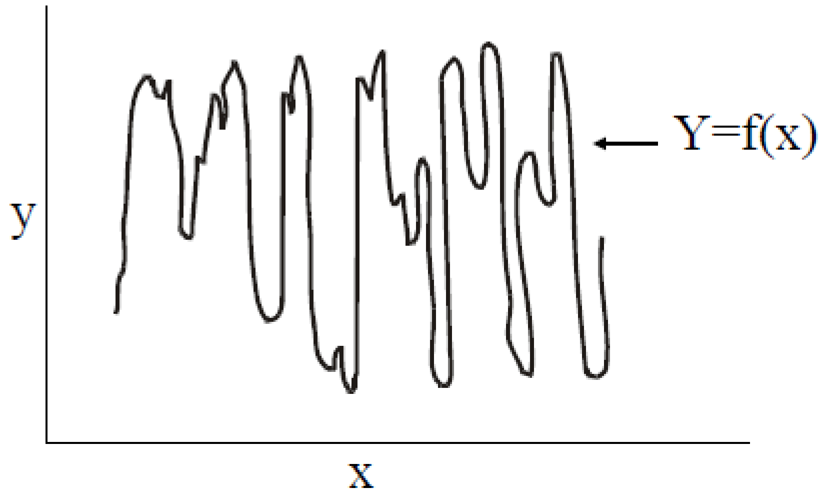
2. Size of Training Set

- The nature of the problem determines the number of training samples that are required.
- A backpropagation network will likely do very well with the ● training trials.



- But with only 🟥 training trials, it would likely conclude a function such as the straight dotted line.

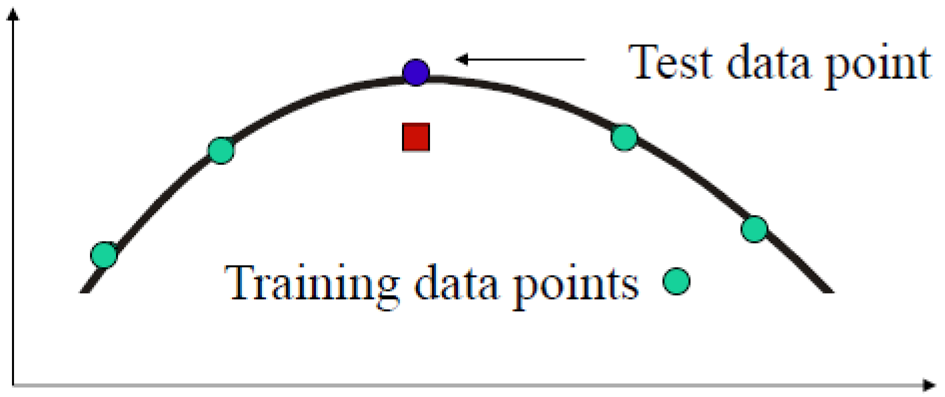
Size of TS (cont…)



- Obviously, the more abrupt the changes in f(x), the more points on x that must be used as training trial.
- Backpropagation, and most neural networks, only work well when similar inputs result in similar output.

Size of TS (cont…)

- The backpropagation network can be thought of as interpolating from the training samples that it has seen.
- Suppose the underlying function being modelled is:



- If the network is trained using ● points, and then it responds as 🟥 to the test data point that should be ●
- Is the network correct, or in error?

Size of Training Dataset

1. Statistical analyses suggest at least five to ten times as many training samples as the number of weights to be trained.
2. Baum and Haussler (1989) suggest as a necessary condition.

$$\text{no. of patterns} > \frac{\text{no. of weights}}{(1 - \text{expected accuracy on test set})}$$

E.g., if a network contains 27 weights and the desired test set accuracy is 95%, then the size of the training set should be at least $27/0.05 = 540$

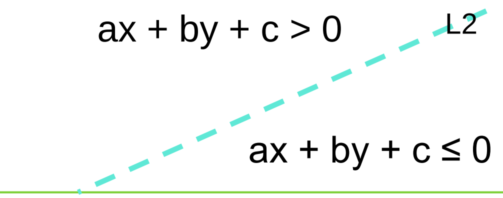
3. Number of Nodes: Input & Output

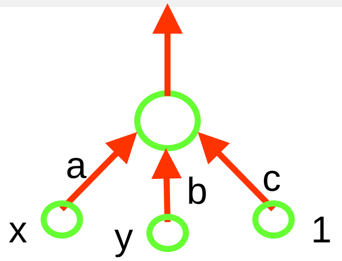
- Compute how many bits (1 and 0) would be needed to represent all input data → # of input nodes.
- Compute how many bits (1 and 0) would be needed to represent all possible output classes → # of output nodes.

Number of Hidden Nodes

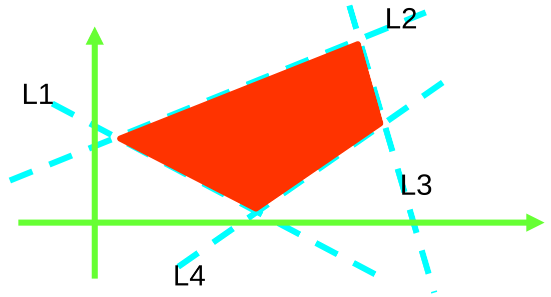
- For a 1-hidden layer network, we can estimate the number of necessary hyperplanes to solve a classification problem in d-dimensional input space, thus determining the number of necessary nodes in the hidden layer. (the cube problem)
- Certain clustering procedures may be applied as a pre-processing step to determine the above.

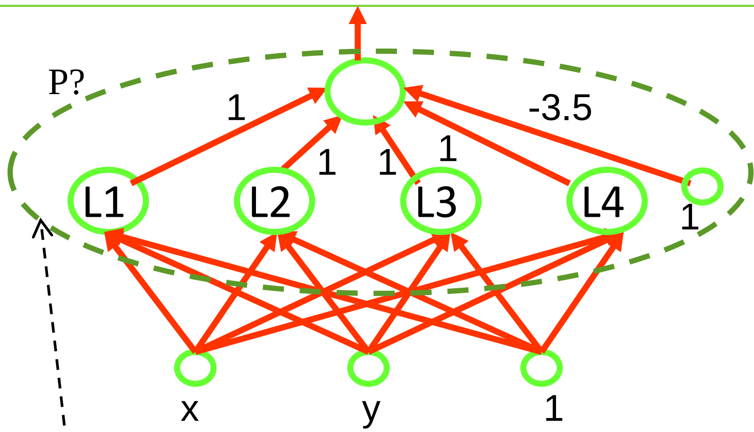
Hidden Nodes (cont…)



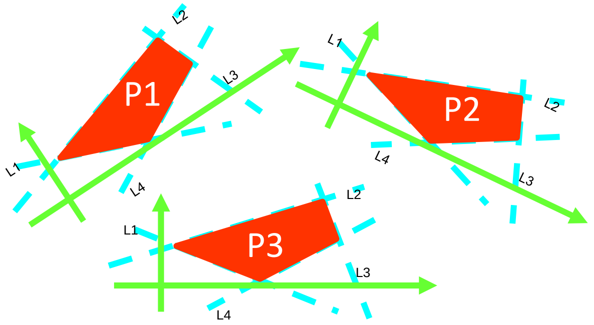


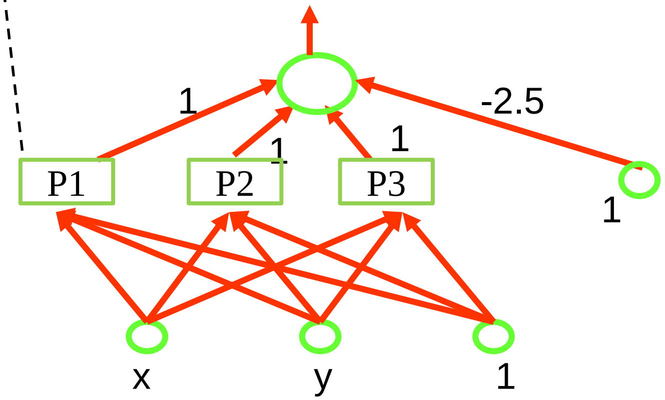
Network with a single node (step function)





One hidden layer network that realizes the convex region. Each hidden node realizes one of the lines bounding the region

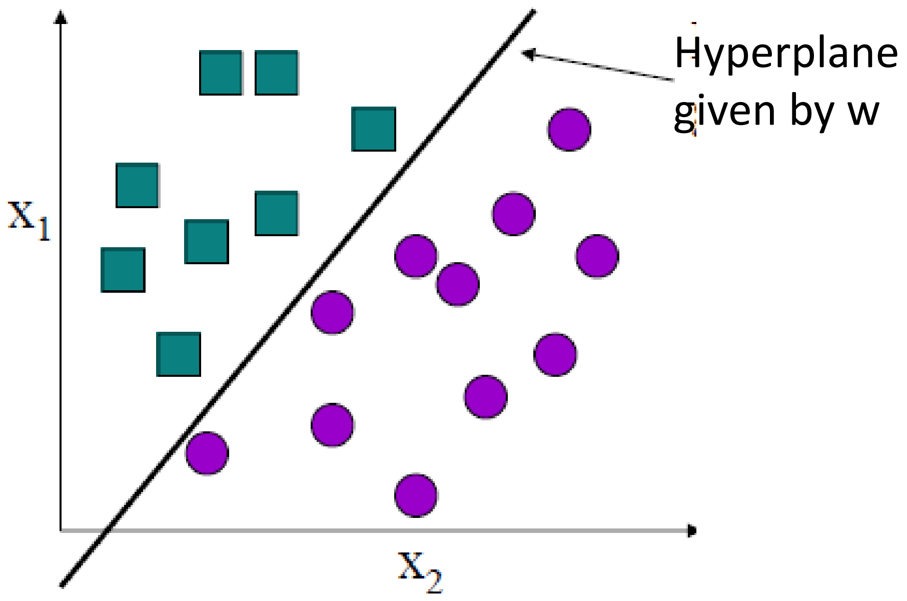


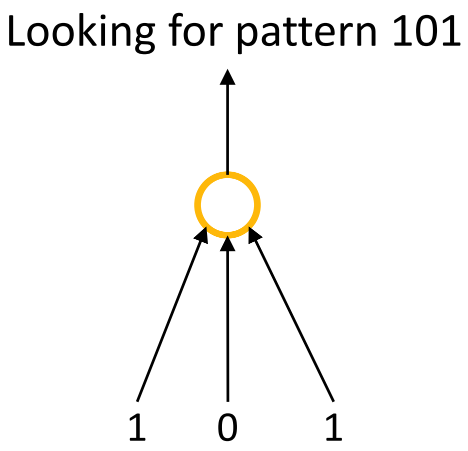


Two hidden layer network that realizes the union of the convex region. Each box represents one hidden layer network realizing one convex region.

Number of Hidden Nodes for Feature Detection Problems

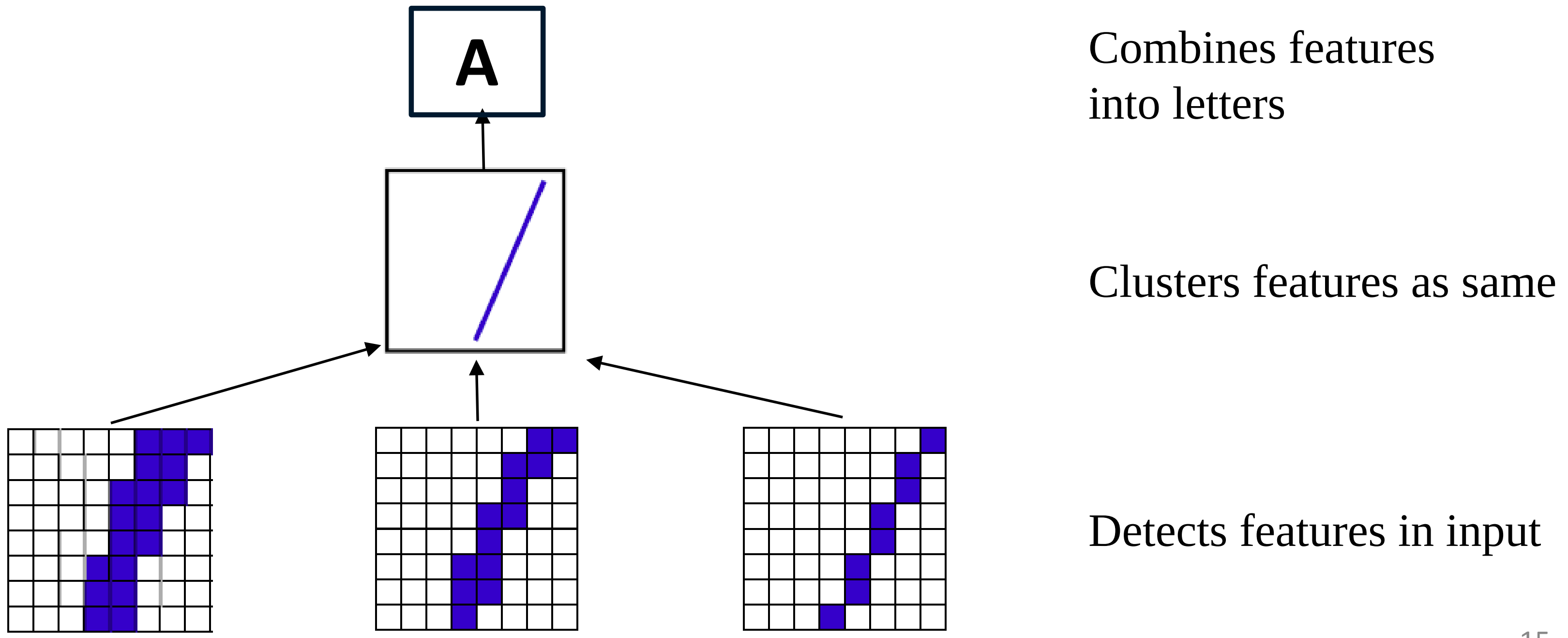
- For any processing node, we can think that it separates its input with a hyperplane. We can also think of it as a feature detector.





Number of HN in Feature (cont…)

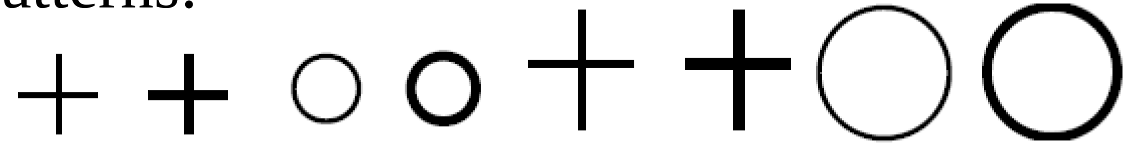
- You can sometimes anticipate the suitable number of layers by analyzing the problem, and determining the features that are likely involved.


A
Combines features into letters
Clusters features as same
Detects features in input

Number of HL (cont…)

- It is possible to estimate the number of hyperplanes needed in the hidden layer based on the numbers of features in the input data.
- Note however, that the "features" detected by your hidden nodes may not coincide with the features that define the task.
- E.g., given the features: big/small, thick/thin, circle/cross

With patterns:



- The task might be solved with 3 hidden nodes, but they will not relate to the 3 features by which the patterns were created – *hidden nodes group features and not necessary correspond to individual features*.

4. a) Initial Weight Values

- The weight values $w_{ji} \in \mathsf{R}$, and so eventually can take on any value.
- The weights must be randomly set to start, and it is sensible to start them off small e.g., (-1.0 to 1.0) or (-0.5 to 0.5).
- Recall that weight updates are smaller for large activations for sigmoid function (curve flattens as $a \rightarrow \pm\infty$).
  - Large weights → large activation.
  - This can slow down the learning process. By starting the weights with small values, we can avoid this.

Weights (cont…)

- Inputs can have different ranges of values. E.g.,

  If $x_1 = 1.0$ (range of (0.0 to 1.0)) $\approx$ max and

  $x_2 = 2.0$ (range of (0.0 to 1000.0)) ≈ 0

  x_2 is, in its own range, almost zero whereas x_1 is at its maximum.

- Despite this, when the weight updates are calculated

  $\Delta w_{ji} = c.\delta_j.x_i$ the *effect* of multiplying by x_2 is greater.

Weights (cont…)

- We could:
  - *Scale the inputs* into identical ranges → normalize the input values.
  - Run for a long time for the weights to adjust to take into account the differences in ranges.
  - Initialize the weights in a way that considers the possible input values.

4. b) Weight Update Frequency

1. Compute weight changes after each training trial, called ***per-pattern training*** mode.
2. Accumulate changes, and make them all after each epoch, called ***per-epoch*** or ***batch training*** mode.
   - An **epoch** is a cycle through all the training samples.

$$\Delta w = \sum_{p=1}^{P} \Delta w_p$$

   - Weight changes suggested by different training samples are accumulated together into a single change to occur at the end of each epoch.

Update Frequency (cont…)

- Both methods are in wide use, and each has its advantages and disadvantages. Considering per-pattern update
  - Advantages: Reducing the weight change necessary after every change, may accelerate convergence, can understand the effect of changes.
  - Disadvantages: More computation intensive, may be making changes based on one input which is not necessary or can do worse.
- Considering ***training time***, for large applications it can be large requiring several days even on the fastest processors, irrespective of whether the training method is *per-epoch* or *per-pattern*.

5. Learning Rate

- $\Delta w_{ji} = - c.\partial E/\partial w_{ji}$
- If learning rate c is too high, the changes over-react to individual errors, and negate adjustments made on the basis of previous training trials, resulting in an oscillating behaviour.
- If c is too small, the learning process can be too slow.
- Values between 0.1 and 0.9 have been used in many applications.
- Each weight in the network can be associated with its own learning rate.

Learning Rate (cont…)

There are several approaches to varying the learning rate.

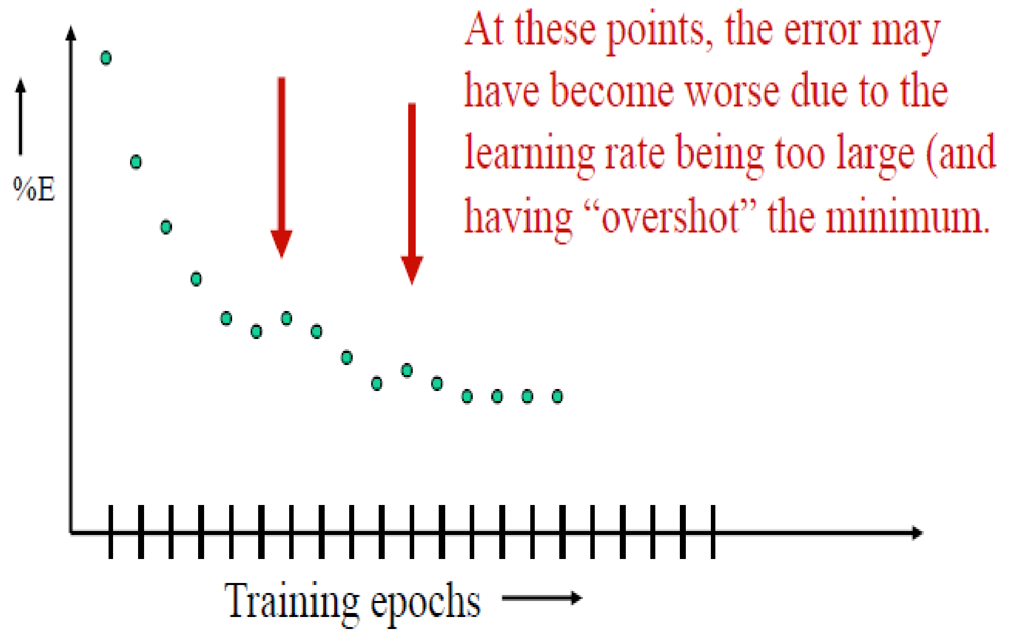
- **Steadily decrease c** : At the outset, the weights are wrong anyway - so larger changes are more acceptable. Later on, the weights are more correct, and only fine tuning is required.
- **Steadily increase c**: Start with a small c, and steadily double it until MSE worsens (more knowledge helps to learn faster)

Learning Rate (cont…)

- **Consider the error after each epoch**: Each training trial provides an error value $E = |\mathbf{d}\text{-}\mathbf{y}|^2$
  - As each epoch of trials is used, this error is accumulated. If the accumulated error is lower than the previous epoch, increase c. If it is greater, decrease c.
- **Use second derivative**: If the magnitude of the second derivative of MSE is small (2nd derivative of position w.r.t. time is acceleration), c can be high, and vice versa. However, accurate computation of the second derivative is expensive.

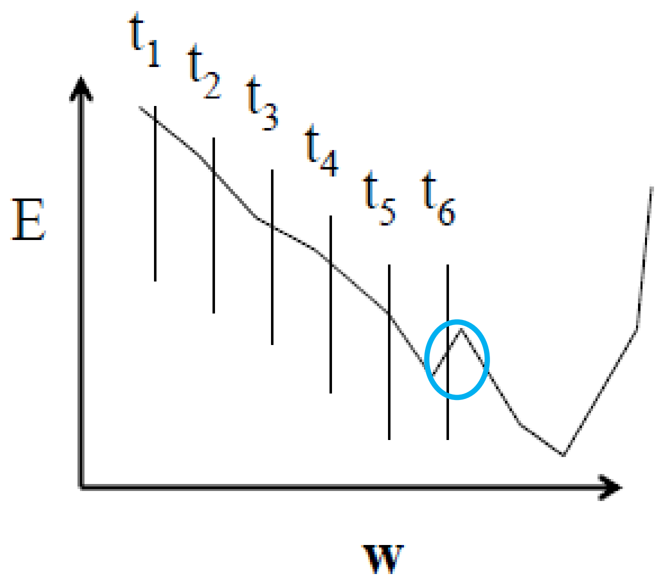
Learning Rate (cont…)

- These techniques all have two risks.
  1. While they may speed learning, they result in a greater probability of being trapped in local minima on the error surface.
  2. The expense to compute all the considerations of the error may rival the savings accomplished in learning time.



6. Optimization by using Momentum

- Suppose that a particular weight changes in the same direction several times in a row, then the next training trial wants to move in the opposite direction.
- The projection of the error surface for that weight might look like:



Momentum (cont…)

- BP leads the weights in a neural network to a local minimum of the MSE, possibly substantially different from the global minimum that corresponds to the best choice of weights.
- The idea of **momentum** is to give extra consideration to the general direction that has been taking recently.
- Rumelhart, Hinton, and Williams (1986) suggest

$$\Delta w_{ji}^{(t+1)} = c \,.\, \delta_j \,.\, x_i + \alpha \Delta w_{ji}^{(t)}$$

- Where α is an additional parameter. The effective weight change is an average of the change suggested by the current gradient and the weight change used in the preceding step.

Momentum (cont…)

- Problem: determine the value of α (which can change as training progresses)
  - Optimal value depends on the application and is not easy to determine a priori.
  - Can be obtained adaptively, as the learning rate. A well-chosen value can significantly reduce the number of iterations for convergence.
  - A value close to 0 implies that the past history does not have much effect on the weight change, while a value closer to 1suggests that the current error has little effect on the weight change.
- We see that momentum helps to avoid local minima, but does it reduce our ability to find or stay in global (or more significant) minima?

7. a) Batch vs Per-pattern training

- The amount of training time can be reduced by ***exploiting parallelism*** in per-epoch or batch training
  - Each of P different processors calculates weight changes for each pattern independently, followed by a phase in which all errors are summed up.
- Note also that some applications are on-line applications for which the training trials are only available one at a time anyway, and so is more conducive to per-pattern training.

b) Performance Evaluation and Generalization

- Must perform well on the training data, but rather to **perform well on new data** that the network has never processed.
- So, BPN discovers and models underlying relationship between the inputs and outputs.
- For classification problems, the fraction of misclassified samples is a useful measure:

$$E = \frac{\text{Number of misclassified samples}}{\text{Total number of samples}}$$

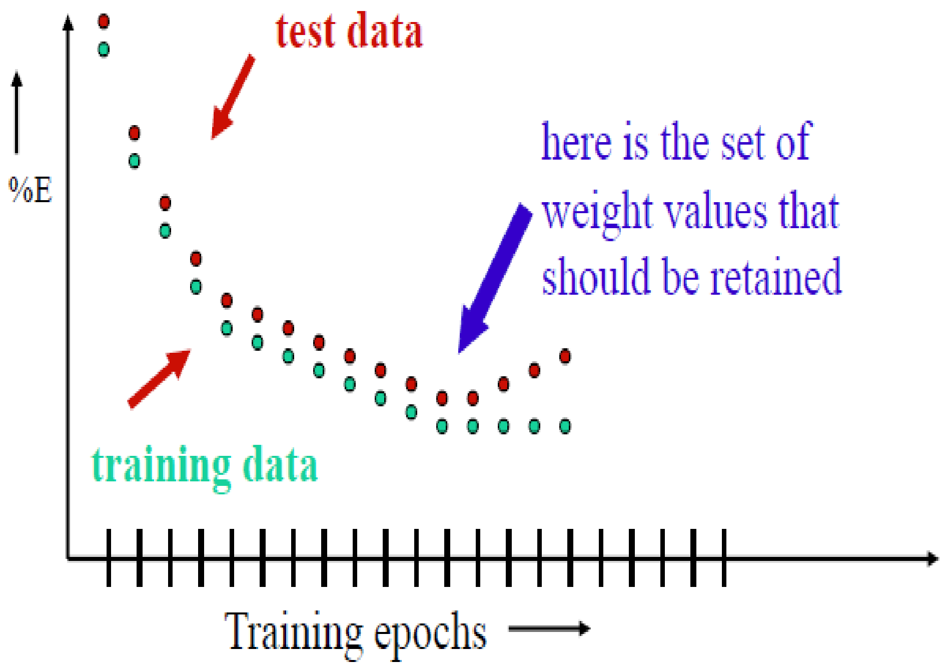
Generalization (cont…)

- We would like to know how well we could expect the network to perform on data that has not previously been encountered.
- A simple approach is to divide up the available samples into
  1. training set (80%) and
  2. test set (20%)
- Matlab uses
  1. training set (70%)
  2. validation set (15%) (new data to estimate error and retrain if needed) and
  3. test set (15%) (new data to test performance)

Generalization (cont…)

- Only training set is used to adjust the weights, and when the learning process is complete, test the network's performance with the test set.
- At the end of each training epoch evaluate performance or total error % on both the training set and the test set (if validation set is not used).
- Stop training when error increases or does not reduce any more.

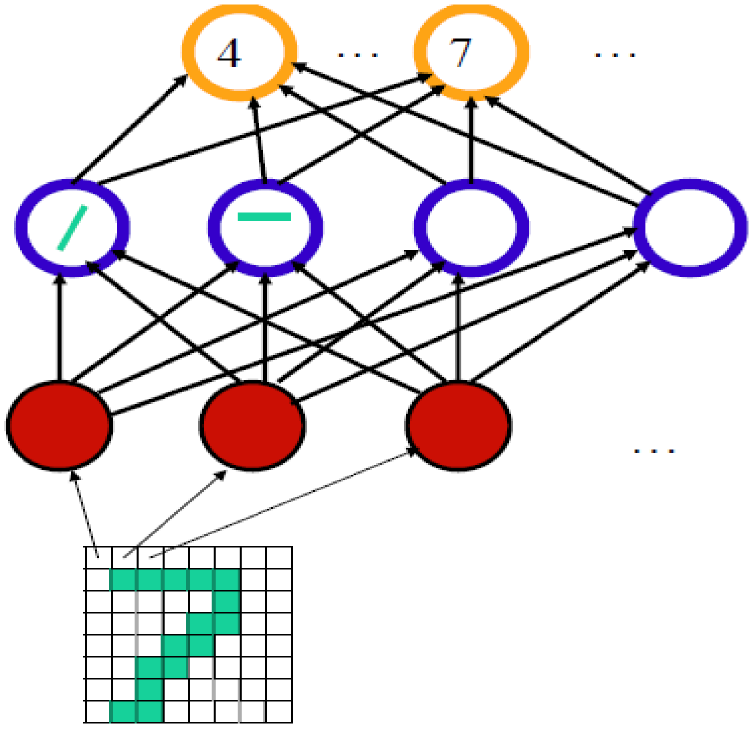
- Here is how that evaluation typically proceeds:



Solutions

- When error increases for the test data (in absence of validation set), stop training.
  - With this stopping criterion, final weights do depend on the test data (or validation data) in an indirect manner.

Example



- Consider the 1-hidden layer system that is learning the digits.
- As training proceeds, the hidden nodes may come to represent reasonable features, such as the diagonal and the horizontal line of the digit 7.

Example (cont…)

- At this point the network is doing well on the training set, and it will also do well on the test set.
- If we keep training past this point, the network learning algorithm will continue to "hunt" for ways to improve the performance of the training set.
- Suppose, by accident that all the 7's used in the training set had the property that a particular pixel was "on".

Overtraining/Overlearning

- Then the network would eventually develop a hidden node that detects this, and uses it in determining if an input is a 7.
- The result is a slight improvement on the training set, but unless all the test set 7's have the same peculiarity, the performance will get worse on the test set. This phenomenon is called **overtraining/overlearning**.
  - Learns abnormal features in the data.

Solutions (cont…)

- A network with a large number of nodes (and therefore edges) is capable of memorizing the training set but may not generalize well. For this reason, networks of smaller sizes are preferred over larger networks.
- Inject random noise into the training set in feedforward neural networks especially *when the size of the training set is small*. Each training data point (x_1,.., x_n) is modified to a point ($x_1 \pm\alpha_1$; $x_2 \pm\alpha_2$,…,$x_n \pm\alpha_n$) where each α_i is a small randomly generated displacement.

Book Examples – Ch 3

- See the examples in Ch. 3 of the book on BP networks.
- Weaning and classification of Myoelectric signals.

Example 3.7.1 Weaning - Textbook

- Weaning from mechanically assisted ventilation – discontinuing the respiratory support under certain observable conditions.
- Observe how the input data are normalized to lie between 0 and 1 (see in textbook).
- The example network uses momentum to avoid being stuck at a local minima. The learning rate used is $\eta = 0.9$ and momentum rate is $\alpha = 0.4$.

Example 3.7.2 Myoelectric signals – Textbook

- Electrical signals that correspond to muscle movements in animals - measured on the surface of the skin.
- Classification of such signals into three groups that translate directly into movements of specific parts of the body.
- Problem – Signal measurements contain significant amounts of noise, due to background electrical activity in nerves unrelated to the movement of the relevant muscles.
- Hence perfect classification is impossible.