Master of Technology

Unit 2/6: Computational Intelligence I

Basic Concepts of Neural Networks

Dr. Zhu Fangming
Institute of Systems Science
National University of Singapore

Objectives

- To introduce basic concepts of neural networks
- To introduce single-layer perceptron and percetron learning
- To introduce multilayer feedforward networks and the backpropogation learning rules
- To discuss some important issues in training multilayer networks

Outline

- Single-layer perceptrons
- Multilayer feedforward networks
- Backpropagation learning
- Important issues in training

Single-layer Perceptron

- The activation function employed is a hard-limiting function
  - » $I = \Sigma_i x_i w_i$
  - » hard-limiting function

$$f(I) = \begin{cases} 0 & \text{if } I < \theta \text{ (threshold)} \\ 1 & \text{otherwise} \end{cases}$$

- *Perceptron learning algorithm as a formula*

$\Delta W_i = \alpha\,(T - O)\,X_i = \alpha \times \text{error} \times \text{input-}i$

$W_i\,(t+1) = W_i\,(t) + \Delta W_i$

where $W_i\,(t)$ — the weight at time t (or t-th iteration)

ΔW_i — the change made to weight W_i

X_i — the i-th input

α — learning step $(0 < \alpha < 1)$

T — target output

O — actual output

NN Learning: Single-layer Perceptron ...

- ***Perceptron learning rule***
  - » **If the output is ONE and should be ONE or if the output is ZERO and should be ZERO (*no error*), do nothing (no change to weights).**
  - » **If the output is ZERO (inactive) and should be ONE (active), increase the weight values on all active input links.**
  - » **If the output is ONE (active) and should be ZERO (inactive), decrease the weight values on all active input links.**

Example: Learning in Single Perceptron

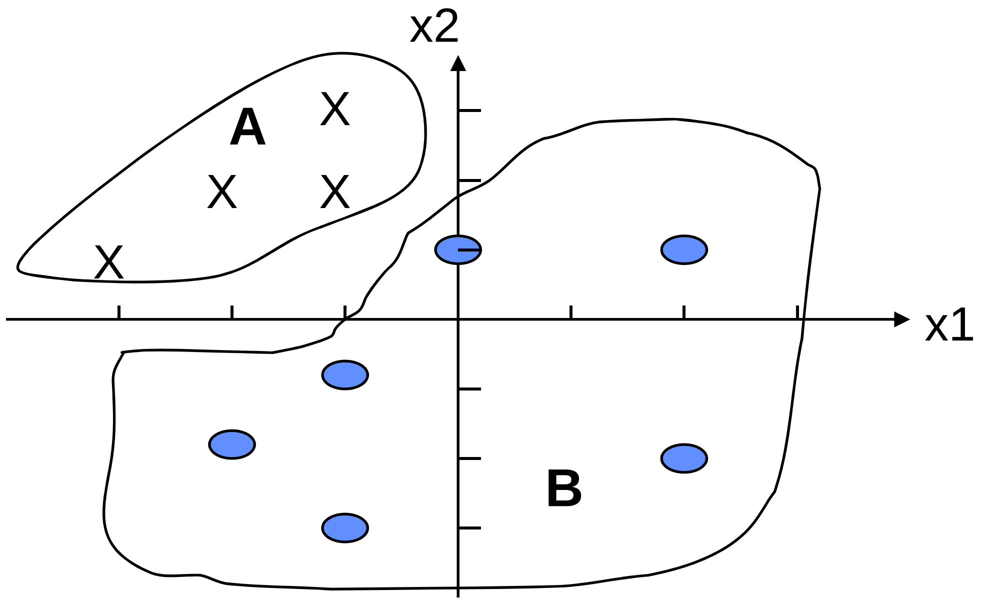
- **An example of pattern classification on x1-x2 space**

  Class A with four patterns given

  (-3, 1), (-2, 2), (-1, 2), (-1, 3)

  Class B with six patterns given

  (-1, -1), (-2, -2), (-1, -3) (2, -2), (2, 1), (0, 1)



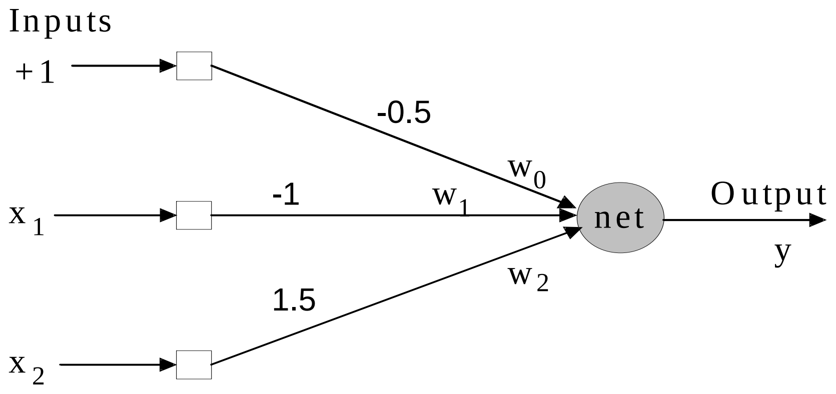
Example: Learning in Single Perceptron ...

- Single perceptron
  - » $I = \Sigma_i x_i w_i$
  - » hard-limiting function

$$y = f(I) = \begin{cases} 0 & \text{if } I \leq 0 \\ 1 & \text{otherwise} \end{cases}$$

  - » y = 1 indicate class A
  - » y = 0 indicate class B
  - » let initial weights be

    w0 = -0.5, w1 = -1, w2 = 1.5   learning step $\alpha = 0.5$
  - » using the 10 given patterns for training (assuming the sequence as appearance)



Example: Learning in Single Perceptron ...

- for class A: No error for all four patterns
  (-3, 1), (-2, 2), (-1, 2), (-1, 3)
- for class B: No error for five patterns
  (-1,-1), (-2,-2), (-1,-3), (2,-2), (2,1)
- there is an error for the pattern (0, 1) of class B
- learning

$$\Delta W_0 = \alpha\,(T - O)\,X_0 = 0.5 \times (0 - 1) \times 1 = -0.5$$

$$\Delta W_1 = \alpha\,(T - O)\,X_1 = 0.5 \times (0 - 1) \times 0 = 0$$

$$\Delta W_2 = \alpha\,(T - O)\,X_2 = 0.5 \times (0 - 1) \times 1 = -0.5$$

- modified weights

$$W'_0 = -0.5 + (-0.5) = -1$$

$$W'_1 = -1$$

$$W'_2 = 1.5 + (-0.5) = 1$$

- with the new set of weights, all ten patterns are classified correctly (verify by yourself)

ISS National University of Singapore

Single Perceptron and Linear Separability — An Example in 2-dimensional Space —

- Equation for straight line

$$x_2 = a \times x_1 + b$$

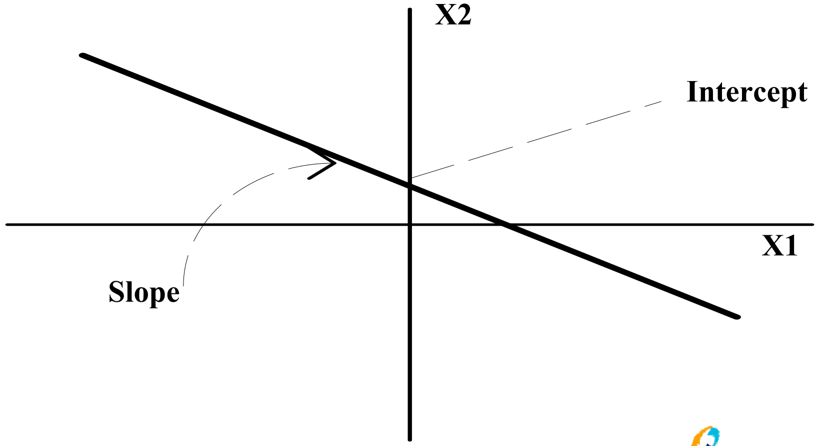
- Weighted sum for single perceptron

$$w_0 + w_1 \times x_1 + w_2 \times x_2 = 0$$

$$x_2 = -(w_1/w_2)\, x_1 - (w_0/w_2)$$

$-(w_0/w_2)$ — intercept

$-(w_1/w_2)$ — slope



Single Perceptron and Linear Separability...

For previous example

- **N_0, the original neuron**

  w0 = -0.5, w1 = -1, w2 = 1.5 slope = 2/3, intercept = 1/3

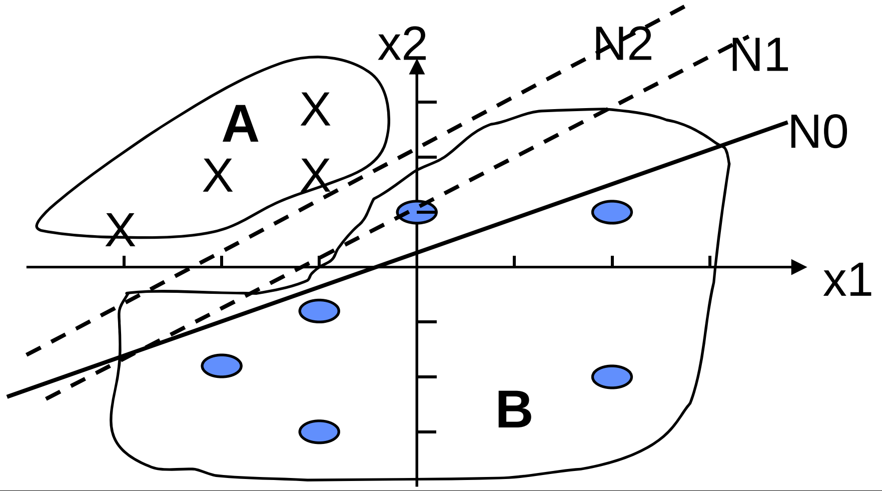
- **N_1, the neuron after learning**

  w0 = -1, w1 = -1, w2 = 1 slope = 1, intercept = 1

  » **another set of weights represent the same line:**

  w0 = -2, w1 = -2, w2 = 2

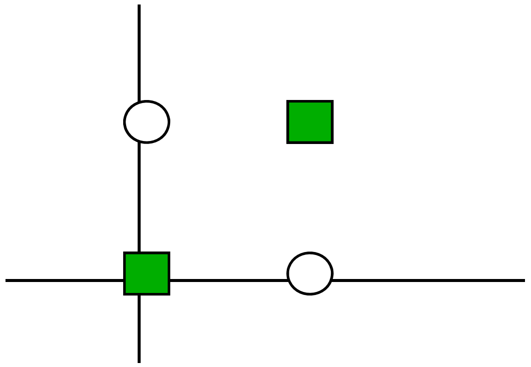
- **N_2, a neuron can also solve the same problem (for your exercise)**



Single Perceptron and Linear Separability ...

XOR Problem

- four patterns
  - » (0, 0) -> 0 (false)
  - » (0, 1) -> 1 (true)
  - » (1, 0) -> 1 (true)
  - » (1, 1) -> 0 (false)
- initial weights
  - » w0 = 1, w1 = 1, w2 = 1
  - » learning step $\alpha = 0.5$



- **Can we train a perceptron to solve this problem? Why?**

Single Perceptron and Linear Separability ...

- In the activation function

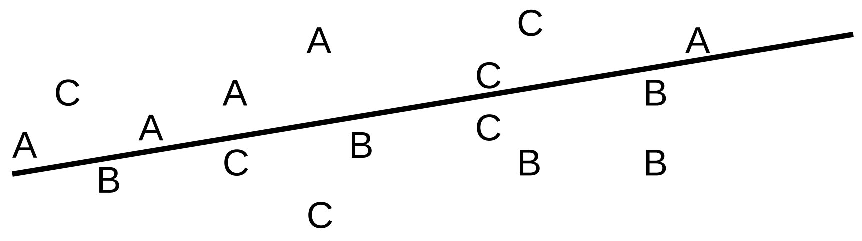
$$\sum_{i=1}^{n} W_i X_i = \theta$$

forms a hyperplane in the n-dimensional space, dividing the space into two halves. Using θ as a threshold to produce output value (1 or 0) means classifying the instances to two classes. When n = 2, the hyperplane becomes a line.

- *Linear separability*
  » A data set is called "linearly separable", when a linear hyper plane exists to place the instances of one class on one side and those of the other class on the other side.

- A single perceptron can only solve a classification problem when it is linearly separable.

- Many classification problems are not linearly separable.
  » Sets A and B are linearly separable
  » Sets A and C, sets B and C are not linearly separable



Perceptron Learning Convergence Theorem

If

- » **given a set of input vectors, each of them with a desired output, and**
- » **each training case is presented to the network with positive probability**

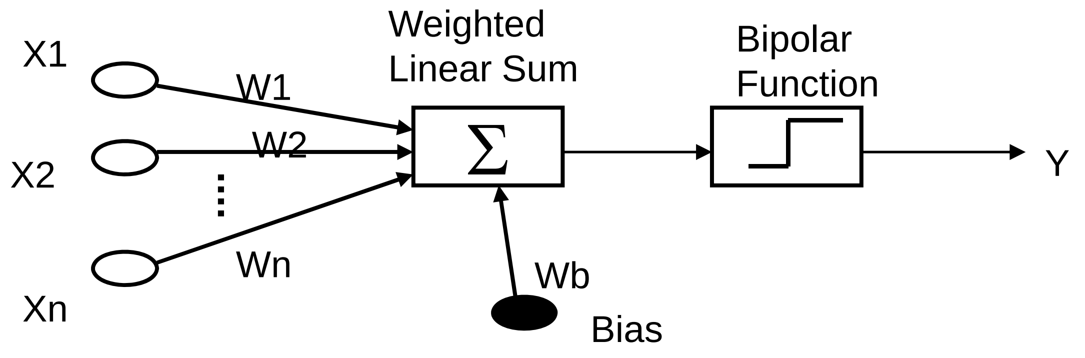
Then

- » **there is a procedure guaranteed to find a set of weights that give correct outputs if-and-only-if a set of weights exist for the task**

A single perceptron will converge only when the problem is linearly separable

ADALINE & Its Learning Rule

- ADALINE (ADAptive LINear Element) (Widrow, 1959)
  - » a single unit similar to the perceptron
  - » has a single output which receives multiple inputs, takes the weighted linear sum of the inputs and passes it to a bipolar function (which produces either +1 or -1, depending on the polarity of the sum)
- MADALINE (Multiple ADALINE)
  - » a network of ADALINEs
- Learning rule of ADALINE
  - » Widrow-Hoff rule (LMS, least mean square)



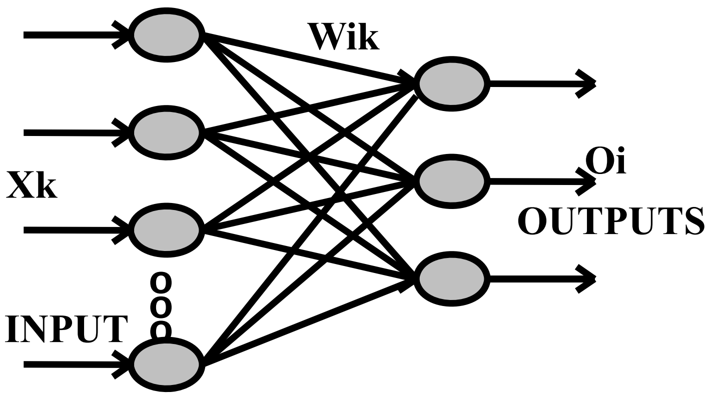
Widrow-Hoff Delta Rule

- Linear Outputs: $O_i^p = \sum_k W_{ik} X_k^p$
  ith output for pattern p
- Target Output: T_i^p
- Error Measure:

$$E(W) = \tfrac{1}{2}\sum_{ip}\left(T_i^p - O_i^p\right)^2 = \tfrac{1}{2}\sum_{ip}\left(T_i^p - \sum_k w_{ik} x_k^p\right)^2$$

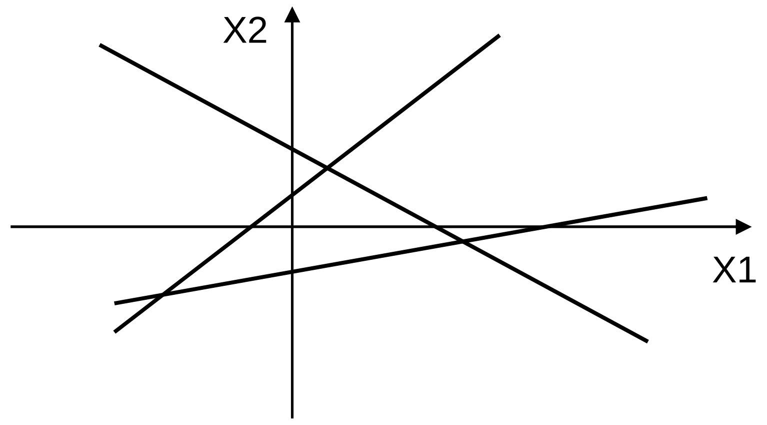
- Learning algorithm:
  - » A form of gradient descent learning: change weight W_{ik} proportional to the negative derivative of error. η is learning rate.

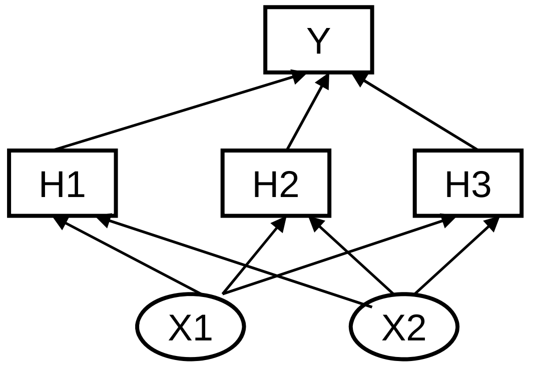
$$\Delta w_{ik} = -\eta \frac{\partial E}{\partial w_{ik}} = \eta(T_i^p - O_i^p) X_k^p = \eta \delta_i^p X_k^p$$



Multilayer Perceptron

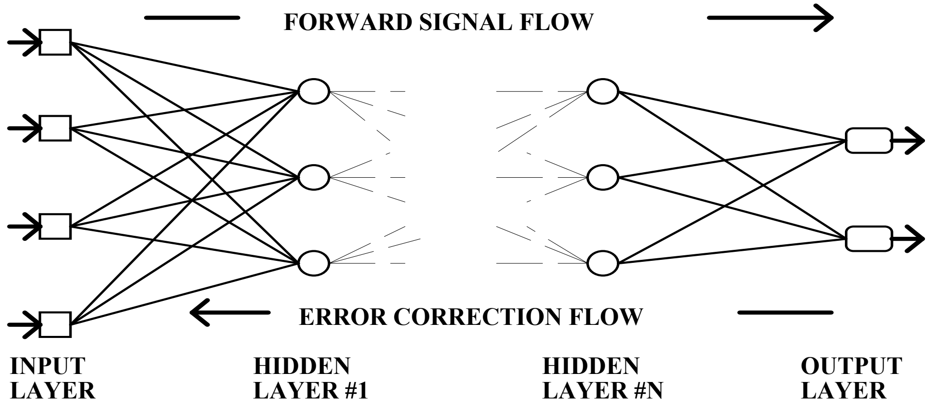
- **Multilayer perceptron is a feedforward neural network with at least one hidden layer**
- **It can form more complex decision regions to solve nonlinear classification problem**
- **Each node in the first layer (above the input layer) can create a hyperplane. Each node in the second layer can combine hyperplanes to create convex decision regions. Each node in the third layer can combine convex regions to form concave regions.**
- **The delta rule does not apply to training a multilayer network since the error of a hidden unit is not known.**
- ***The backpropagation learning method can overcome this difficulty***





Multilayer Feedforward Nets and Backpropagation Learning

- **Fully connected units**
- **Feedforward signals only**
- **One input-layer, one or more hidden-layers and one output layer**
- **Nonlinear differentiable activation functions**
- **Weights in hidden layers are adjusted to reduce aggregate errors in the output layer**
- **A two-stage process:**
  - » **Propagate signals forward and then errors backward**



Steps of Backpropagation Algorithm

1. **Initialize the weights to small random numbers**

2. **Randomly select a training pattern pair (x^p, t^p) and present the input pattern x^p to the network. Compute the corresponding network output pattern z^p**

3. **Compute the error E^p for pattern (x^p, t^p)**

4. **Backpropagate the errors according to the BP weight adjustment formulas (given later)**

5. **Test the mean square error (MSE) over *P* training patterns:**

$$E = \frac{1}{P}\sum_{p=1}^{P} E^p$$

   If the MSE is below the required threshold, stop. Otherwise, repeat steps 2-5.

6. **Test for generalization performance if appropriate**

BP Errors and Weight Updates

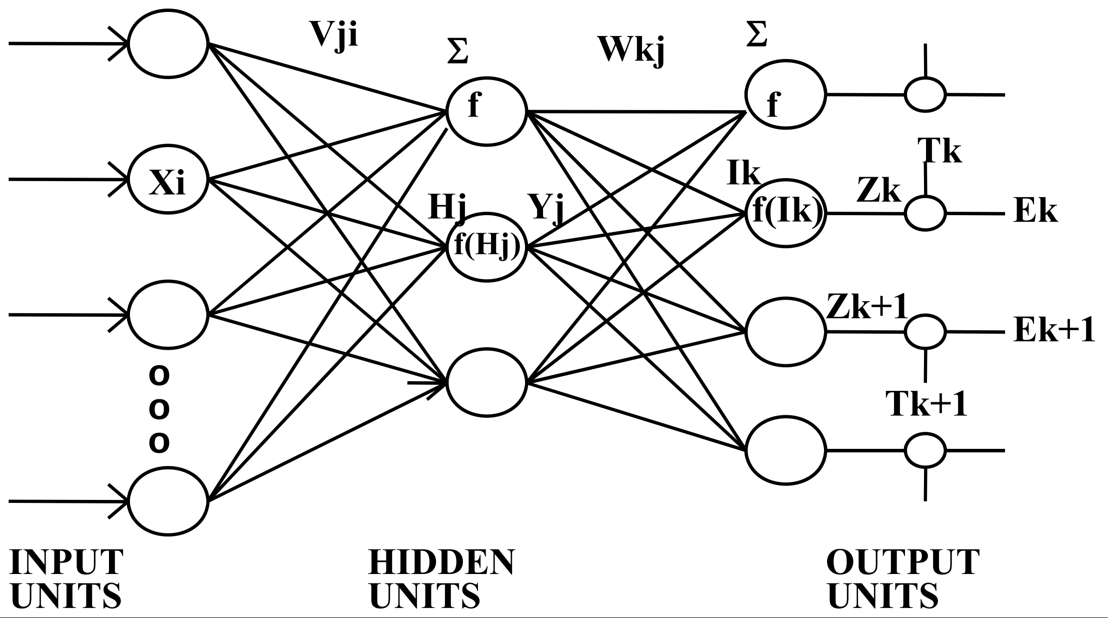
- Total error over all training patterns p and m output units:

$$E_{tot} = \sum_{p=1}^{p} E^p$$ Total error, all training patterns

$$E^p = \frac{1}{2}\sum_{k=1}^{m}(t_k - z_k)^2$$ Error for pattern p

$$z_k = f(I_k) = f(\textstyle\sum_j w_{kj} y_j) = f(\sum_j w_{kj} f(H_j)) =$$
$$= f(\textstyle\sum_j w_{kj} f(\sum_i v_{ji} x_i))$$

z_k — computed output t_k — target output



BP Errors and Weight Updates (cont.)

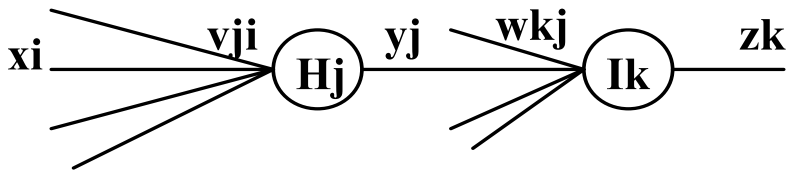
- For output units k = 1, 2, ..., m, adjust the weights to reduce the error for each pattern *p* (Gradient descent)

$$\Delta w_{kj} = -\eta \frac{\partial E^p}{\partial w_{kj}} = -\frac{\eta}{2}\sum_{k=1}^{m} \frac{\partial (t_k^p - z_k^p)^2}{\partial w_{kj}}$$

Dropping superscripts *p*

$$\frac{\partial (t_k - z_k)^2}{\partial w_{kj}} = \frac{\partial (t_k - f(I_k))^2}{\partial I_k} \frac{\partial I_k}{\partial w_{kj}} = -2(t_k - z_k) f'(I_k) y_j$$

So
$$\Delta w_{kj} = -\eta \frac{\partial E^p}{\partial w_{kj}} = -\frac{\eta}{2}\sum_{k=1}^{m} \frac{\partial (t_k^p - z_k^p)^2}{\partial w_{kj}}$$
$$= \eta (t_k - z_k) f'(I_k) y_j = \eta \delta_k y_j$$



- For hidden units, use the chain rule, get

$$\Delta v_{ji} = \eta f'(H_j) x_i \sum_k \delta_k w_{kj}$$

Summary of Backpropagation Algorithm

- **Weights initialization**

  Set all weights to random numbers following Uniform distribution in the range (-1,1)

- **Calculation of activation**

  » **the activation level of an input unit is determined by the instance presented to the network**

  » **the activation level O_j of a hidden and output unit is determined by**

  $O_j = f(\Sigma W_{ji}O_i - \theta_j)$, where W_{ji} is the weight from an input O_i, θ_j is the node threshold, *f* is the transfer function.

- **Weight Updating**

  1) **start at the output nodes and work backward to the hidden layers recursively, adjust weights by $W_{ji}^{(t+1)} = W_{ji}^{(t)} + \Delta W_{ji}$**

  where $W_{ji}^{(t)}$ is the weight from unit *i* to *j* at time t (or *t*-th iteration), ΔW_{ji} is the weight adjustment

Summary of Backpropagation Algorithm (cont.)

(Assume a sigmoid function $f(a) = 1/[1 + e^{-a}]$ is used)

- **Weight Updating (cont.)**

  2) The weight change is computed by $\Delta W_{ji} = \eta \delta_j O_i$

  where η is a trial-independent learning rate ($0 < \eta < 1$),

  δ_j is the error gradient at unit *j*

  3) The error gradient

  3a) for the output units: $\delta_j = O_j(1 - O_j)(T_j - O_j)$

  where T_j is the desired (target) output activation and O_j is the actual output activation at output unit *j*

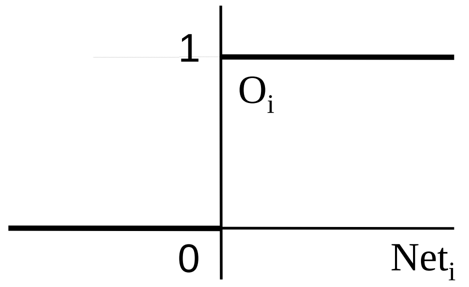
  3b) for the hidden units: $\delta_j = O_j(1 - O_j) \Sigma_k \delta_k W_{kj}$

  where δ_k is the error gradient at unit k to which a connection points from hidden unit *j*

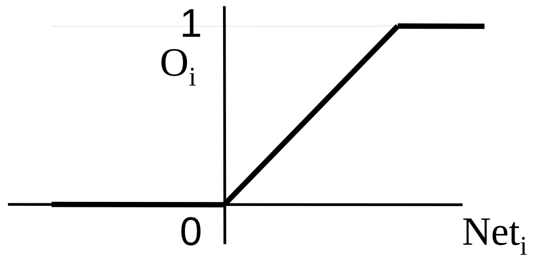
- **Repeat iterations until convergence in terms of the selected error criterion. An iteration includes presenting an instance, calculating activations, and modifying weights.**

Typical Nonlinear Activation Function

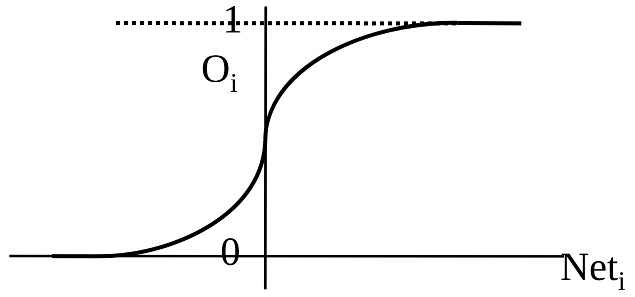
- Several typical functions



THRESHOLD FUNCTION

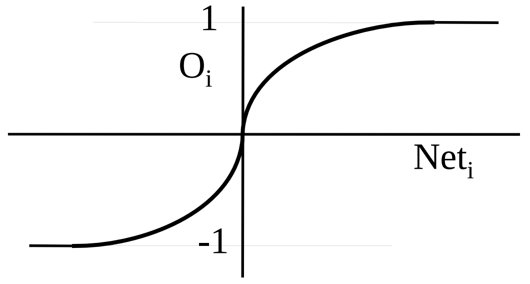


PIECEWISE LINEAR FUNCTION



SIGMOID FUNCTION

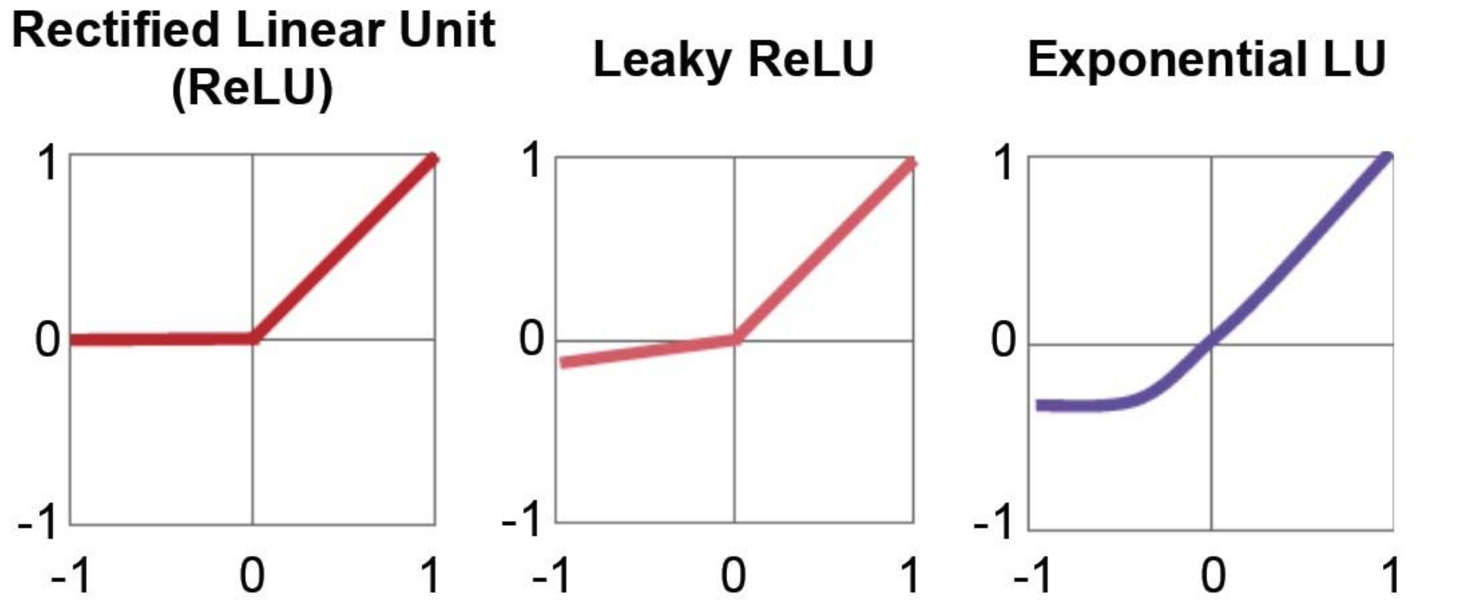
$y=1/(1+e^{-x})$



TANH FUNCTION

$y=(e^{x}-e^{-x})/(e^{x}+e^{-x})$

Typical Nonlinear Activation Function



$y=\max(0,x)$

$y=\max(\alpha x,x)$

α = small const. (e.g. 0.1)

$$y=\begin{cases} x, & x\geq 0 \\ \alpha(e^{x}-1), & x<0 \end{cases}$$

Source: Caffe Tutorial

Typical Nonlinear Activation Function

Normalizing the output vector with a sofmax function:

$$\sigma : \mathbb{R}^K \to (0,1)^K$$

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

They serve as a multinomial probability distribution.

The softmax function is often used in the final layer of a neural network-based classifier.

Source: Caffe Tutorial

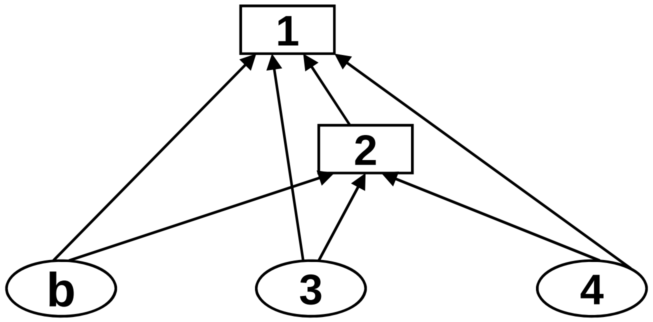
An Example of Backpropagation
— XOR problem —

- **Initial weights**

  $W_{12} = -0.02$, $W_{13} = 0.02$, $W_{14} = 0.03$, $W_{1b} = -0.01$, $W_{23} = 0.01$, $W_{24} = 0.02$, $W_{2b} = -0.01$

-

Input	Output
(1, 1)	0
(1, 0)	1
(0, 1)	1
(0, 0)	0



- **Calculation of activation** (assume b=1.0 and sigmoid activation function used)

  $O3 = O4 = 1$

  $O2 = 1/[1+e^{-(1\times W23+1\times W24+1\times W2b)}]= 1/[1+e^{-(1\times 0.01+1\times 0.02-1\times 0.01)}] = 0.505$

  $O1 = 1/[1+e^{-(0.505\times W12+1\times W13+1\times W14+1\times W1b)}]= 1/[1+e^{-(0.505\times(-0.02)+1\times 0.02+1\times 0.03-1\times 0.01)}] = 0.508$

- **Weight training (assume that the learning rate $\eta = 0.3$)**

  $\delta_1 = 0.508(1 – 0.508)(0 – 0.508) = – 0.127$ $\Delta W_{13} = 0.3\times(– 0.127)\times 1 = – 0.038$

  $\delta_2 = 0.505(1 – 0.505)[(-0.127) \times (-0.02)] = 0.0006$ $\Delta W_{23} = 0.3\times 0.0006\times 1 = 0.00018$ (omit the rest)

- **After many iterations, a set of final weights give the mean squared error of less than 0.01:**

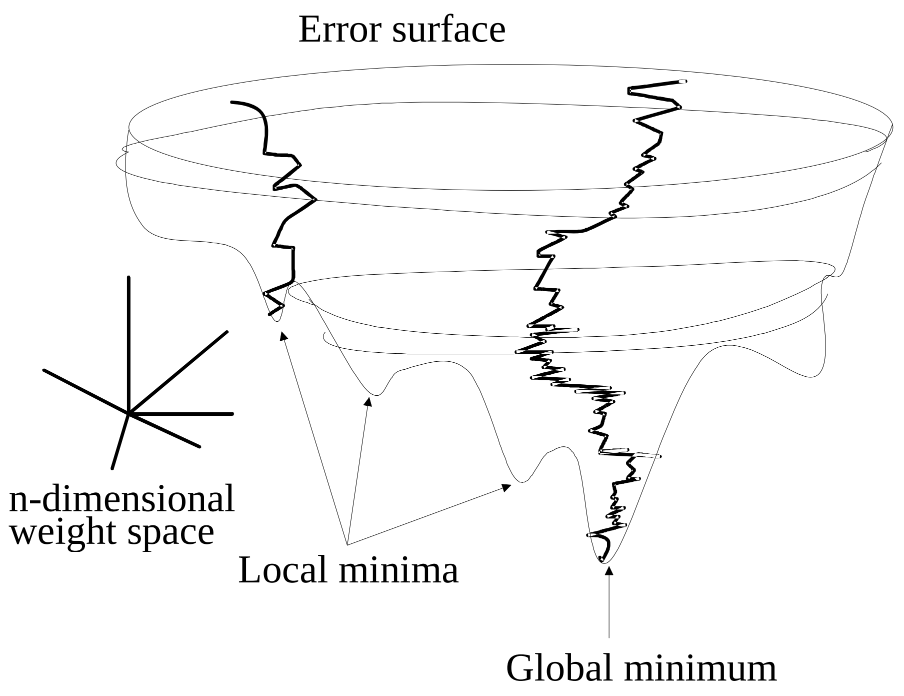
  $W_{12} = -11.30$, $W_{13} = 4.98$, $W_{14} = 4.98$, $W_{1b} = -2.16$,

  $W_{23} = 5.62$, $W_{24} = 5.62$, $W_{2b} = -8.83$

ISS National University of Singapore

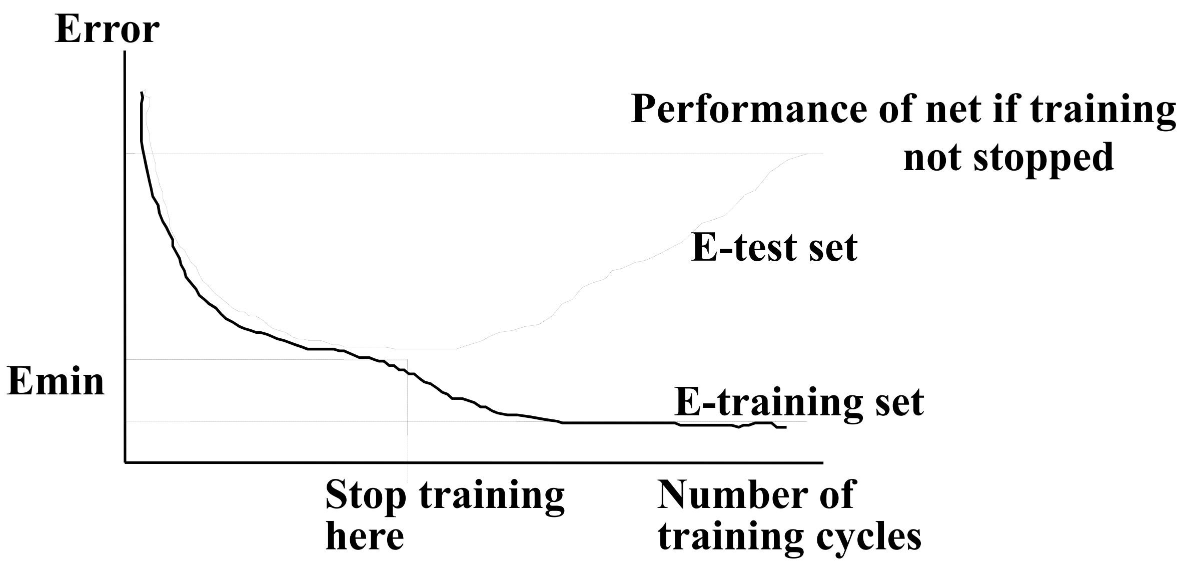
Issues of Training: Local Minimum

- **Error surface: one global minimum, but many local minima**
- **BP is never assured of finding a global minimum.**



Issues of Training: Generalization, Overtraining

- *Generalization* is the ability of a network to correctly classify a pattern it has not seen (not been trained on). NNs generalize when they recall full patterns from partial or noisy input patterns, when they recognize patterns not previously trained on or when they predict new outcomes from past behaviors.
- Networks can be *overtrained*. It means that they memorize the training set and are unable to generalize well.



Avoid Overtraining & Achieving Good Generalization

- **Overtraining can usually be avoided by:**
  - » **choosing a large training set when possible**
  - » **selecting the patterns randomly during training**
  - » **stopping the training process before excessive training occurs (monitoring the training process and stopping after the error drops to a fairly low level)**
  - » **introducing noise directly into the training patterns**
  - » **eliminate unnecessary hidden-layer nodes & weights**
  - » **Adding regularization terms such as L2 regularization**
  - » **using a combination of the above**
  - » **Using a global optimization based training algorithm, e.g. genetic algorithm (GA)**
- **Generalization is influenced by the size of training set, the architecture of the network, and the physical complexity of the problem at hand. In practice, we need the size of the training set, *N*, satisfy the condition**

$$N = O\left(\frac{W}{\varepsilon}\right)$$

where *W* is the total number of free parameters (i.e., weights and biases) in the network, ε denotes the fraction of error rate permitted on the test set, *O* denotes the order of quantity enclosed within. Eg., with an error of 10%, *N* should be about 10 times the number of free parameters in the network.

Convergence Properties

- **BP training is based on the gradient descent computations. This process iteratively searches for a set of weights W* that minimize the error function *E* over all training pattern pairs, that is**

$$E(\mathbf{W}^*) = \min_W \left\{ \sum_{p=1}^{\mathrm{P}} \mathrm{E}^p(\mathbf{W}, \mathbf{x}^p) \right\}$$

- **For such an optimization problem, a cost function is usually defined to be *minimized* with respect to a set of variables. In this case, the network variables that optimize the error function *E* over the training set are the *weight values.***

Convergence Properties…

- The chosen error measure should be differentiable and tends to zero as the collective differences between the target and computed patterns decrease over the entire training set.
- The MSE measure is one of the acceptable functions for this purpose [cross-entropy (preferred for classification), absolute error, mean error are others]. It is popular because of its statistical properties and it is better understood.
- cross-entropy cost function (error measure):

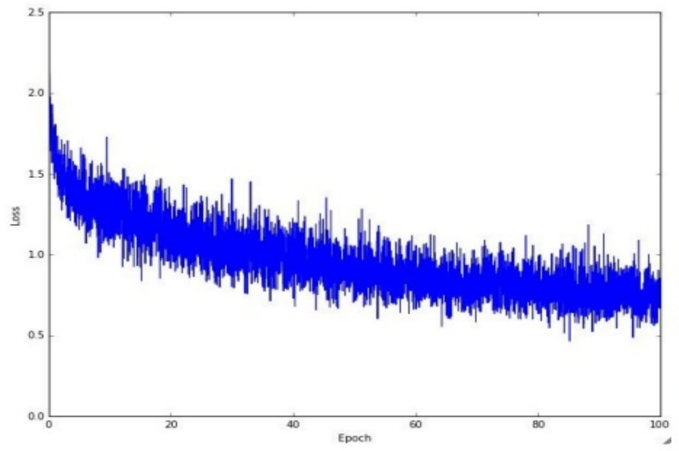
$$C = -\frac{1}{n}\sum_x \sum_j \left[y_j \ln a_j^L + (1 - y_j)\ln(1 - a_j^L) \right]$$

- The search process depends on the shape of the error surface as well as learning algorithm and the training set.

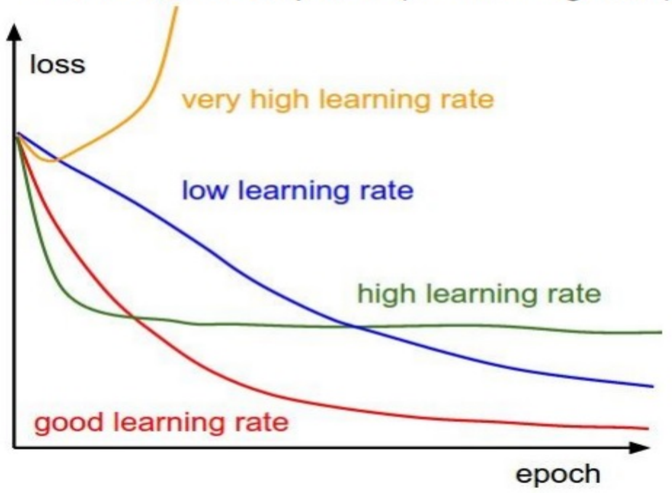
Improving the Rate of Convergence

- Weight initialization
  - » It has been shown that setting the initial weights to small (but not too small) random values, say between -1 and +1, is an effective way to avoid shallow troughs and possible entrapment at the start of the training process.
  - » Estimating initial values
    - ♦ Estimate values that are near a minimum solution rather than assigning random values
  - » LeCun suggested to draw values from a zero-mean Gaussian distribution with standard deviation 1/sqrt(N), N is the number of connections feeding into the node.
- Choosing learning rate coefficient η
  - » it determines the size of the weight adjustment made at each iteration and hence influences the rate of convergence. It should not be constant throughout the learning process for best results.

Improving the Rate of Convergence



The effects of step size (or "learning rate")



Source: Stanford Course

Learning rate can decay over time:

- ❑ step decay: e.g. decay learning rate by half every few epochs.
- ❑ exponential decay: $\eta = \eta_0 * e^{-kt}$
- ❑ 1/t decay: $\eta = \eta_0 / (1+kt)$

Improving the Rate of Convergence (cont.)

- **Adding a momentum term to the weight update rule**
  - » **It has the effect of smoothing the descent down the error curve by adding an averaging of the weight adjustments (exponential smoothing). This can prevent overshooting off the minimum.**

$$\Delta w_{ji}(t+1) = -\eta \frac{\partial E}{\partial w_{ji}(t)} + \alpha \Delta w_{ji}(t)$$

where α is the momentum coefficient ($0 < \alpha < 1$).

  - » **Separate momentum terms can be added to each set of the layer weights using different values of α for each layer to achieve even better results.**

- **The most important parameter, the number of hidden nodes, is <u>empirically chosen as a number around or close to</u>**

$$\sqrt{nin \times non}$$

where *nin* and *non* are respectively the number of input and output nodes (Timothy Masters)

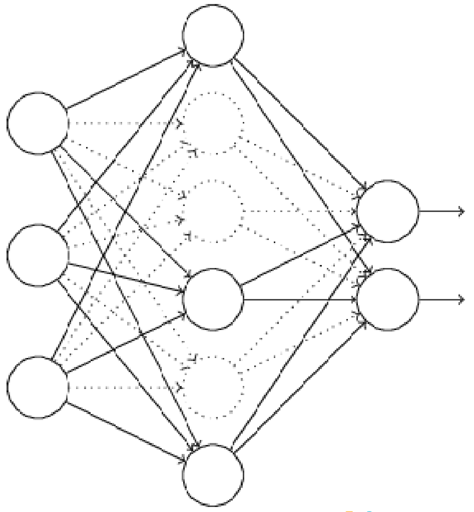
Control Overtraining/ Overfitting

- **Weight Decay (Regularization)**
  - » **To regularize the cost function so that the model parameters will not be tuned to fit the training data too well .**

L2 regularization $$C = C_0 + \frac{\lambda}{2n}\sum_w w^2$$

L1 regularization $$C = C_0 + \frac{\lambda}{n}\sum_w |w|$$

- **Dropout**
  - » **Randomly omit a certain percentage of the hidden neurons during training.**



General Design Guidelines

- Carefully select the proper MLFF network architecture for the given task (number of hidden nodes) and a good choice of activation functions. Experimentation is usually helpful. More number of hidden nodes than what is required causes "over-fitting" – making its choice a ***gamble***

- Choose a large, reliable training set and divide it into appropriate training, testing and validation sets for use in the corresponding operations

- Pre-process and analyse the training data for maximum utility and effectiveness (such as data smoothing, data normalization, etc.)

- Select good initial training parameter values (η and α) and decide on changes to be made (dynamically) during training (experiment with the use of different values). Consider using modified BP training methods (e.g. AdaGrad, RMSProp, etc.) when the network is very large or the problem complex

- Once acceptable performance has been realized by the network, optimize it by pruning nodes and weights