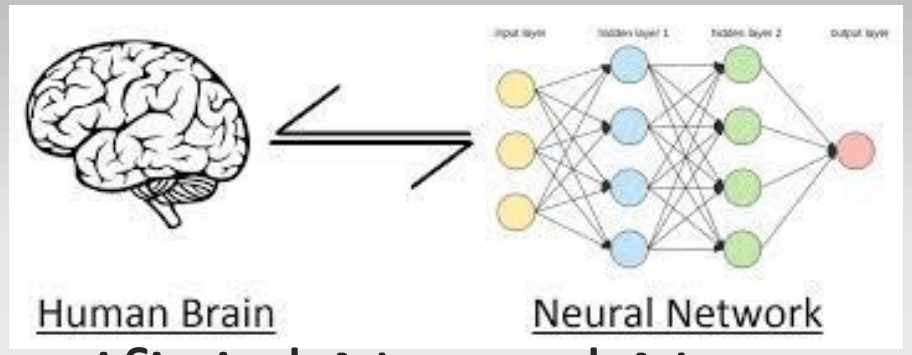
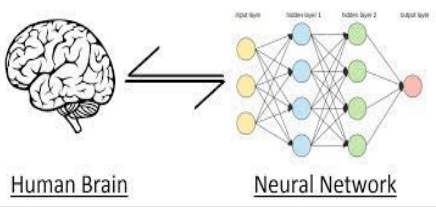


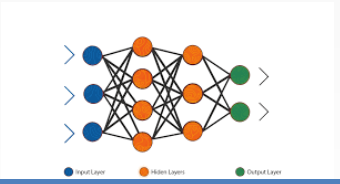
Artificial Neural Network and Deep Learning
Lecture 4

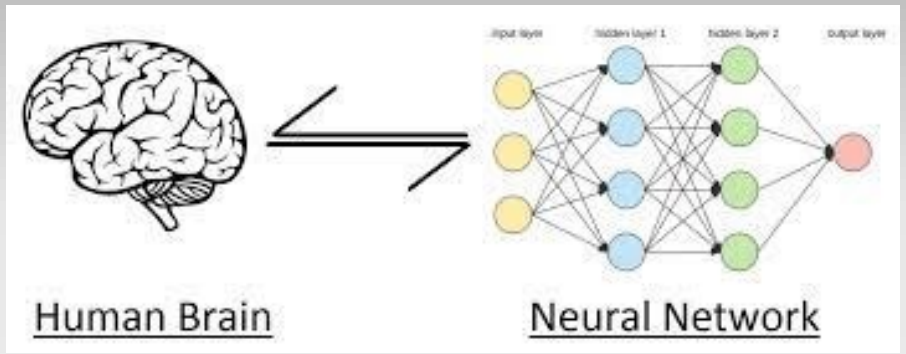
Chapter 04: Multilayer Perceptron (MLP) Network



Agenda

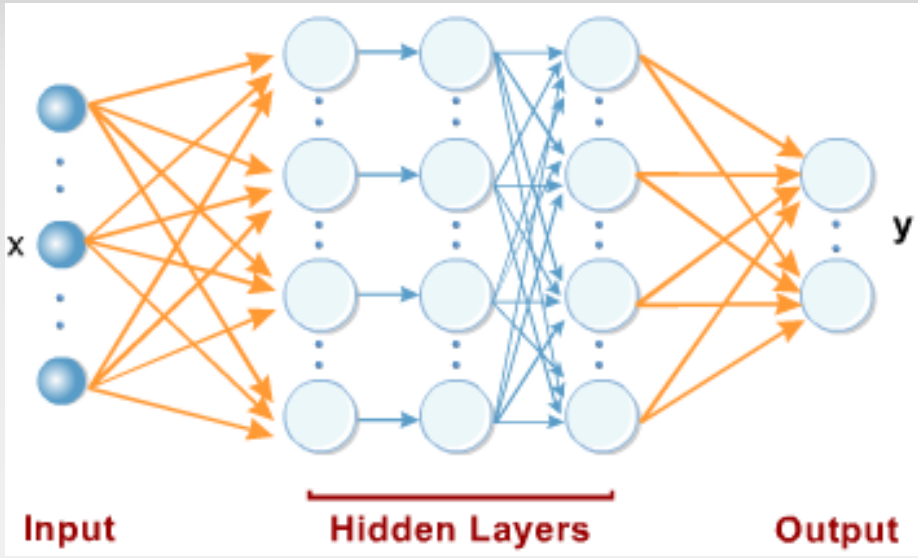
- Multilayer Perceptrons networks (Recap)
- Differentiable of Activation functions
- XOR Problem
- Backpropagation Improvement
- Local Minima problem
- Overtraining problem
- Learning with adaptive learning rate
- Preprocessing





Multilayer Perceptrons networks (Recap)

Architecture of Multilayer Perceptrons networks



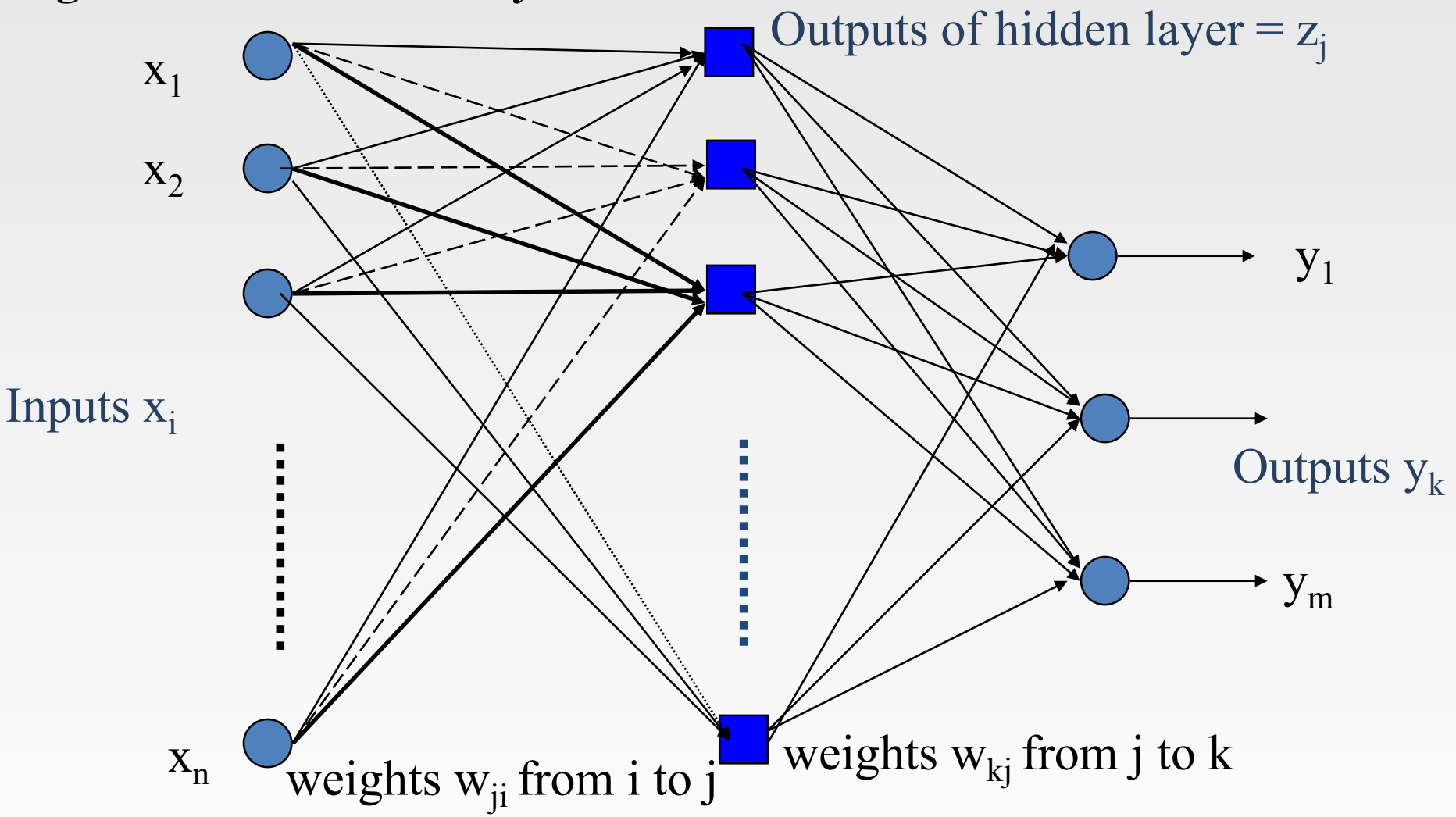
• It is a feed-forward (full connection between nodes in adjacent layers, no connection within a layer)

•The multilayer perceptron consists of three or more layers (an *input* and an output layer with one or more *hidden layers*) of nonlinearly activating nodes that is *differentiable*.

•The most commonly used for non-linear activation function are **sigmoid functions.**

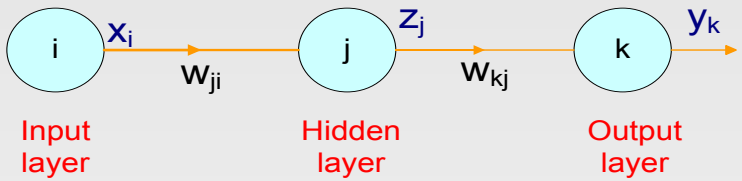
Two-layer networks

We will concentrate on two layer, but could easily generalize to more layers



Back-propagation algorithm, Summary

1. Initialize the weights to small random values.
2. **repeat**
   - **for each training example <(x_1,…x_n),d> Do**



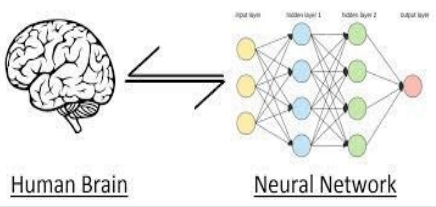
     i. Input the instance (x_1,…,x_n) to the network and compute the outputs from each layer in the network : (**forward step**)
        - For hidden layer: $z_j=f(net_j)$, since $net_j= \Sigma w_{ji}x_i$
        - For output layer: $y_k=f(net_k)$, since $net_k=\Sigma w_{kj}z_j$

     ii. Compute the errors signal δ_k in the output layer and propagate them to the hidden layer (**backward step**):
        - For each output unit k: $\delta_k=(d_k-y_k)\, f'(net_k)$,
        - For each hidden unit j: $\delta_j= f'(net_j)\Sigma_k \delta_k w_{kj}$

     iii. Update the weights in both layers according to: (**forward step**)

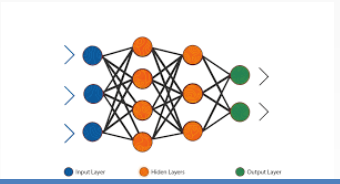
$$\Delta w_{ji} = \eta\, \delta_j\, x_i \;,\; \Delta w_{kj} = \eta\, \delta_k\, z_j$$

   - **end for loop**
3. **until overall error E_{av} becomes acceptably low**



Agenda

- Multilayer Perceptrons networks (Recap)
- Differentiable of Activation functions
- XOR Problem
- Backpropagation Improvement
- Local Minima problem
- Overtraining problem
- Learning with adaptive learning rate
- Preprocessing



Differentiable activation functions

- Since the BP algorithm requires computation of the **gradient of the error function at each iteration step**, we must guarantee the continuity and differentiability of the error function. Obviously we have to use a kind of activation function other than the step function and the linear function used in perceptrons.
- But linear function is continuity and differentiability: A multi-layer network with linear activation functions would be no more powerful than a single-layer network.
- What makes a multilayer perceptron different is that each neuron uses a *nonlinear* activation function which was developed to model the frequency of action potentials, or firing, of biological neurons in the brain. This function is modeled in several ways, but must always be normalizable and differentiable.

The Activation Function

Differentiability is the only *requirement* that an activation function has to satisfy in the BP Algorithm.

- This is required to compute the δ for each neuron.

Sigmoidal functions are commonly used, since they satisfy such a condition:

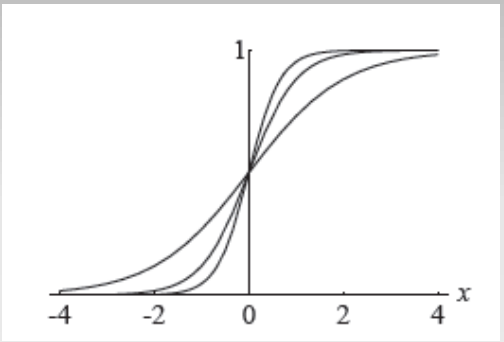
- Logistic Function

$$\varphi(v) = \frac{1}{1+\exp(-av)}, \qquad a > 0 \quad \Rightarrow \quad \varphi'(v) = \frac{a\exp(-av)}{1+\exp(-av)} = a\varphi(v)[1-\varphi(v)]$$

- Hyperbolic Tangent Function

$$\varphi(v) = a\tanh(bv), \qquad a, b > 0 \quad \Rightarrow \quad \varphi'(v) = \frac{b}{a}[a-\varphi(v)][a+\varphi(v)]$$

1- Logistic **Sigmoid** functions



- Its general form is defined by:

$$y_j = f(net_j(t)) = \frac{1}{1+\exp(-a.net_j(t))} \qquad a>0 \quad and \quad -\infty < net_j(t) < \infty$$

- Differentiating it

$$f'(net_j(t)) = \frac{a\exp(-a.net_j(t))}{\left[1+\exp(-a.net_j(t))\right]^2} = a\frac{1}{\left[1+\exp(-a.net_j(t))\right]}\frac{\exp(-a.net_j(t))}{\left[1+\exp(-a.net_j(t))\right]}$$

$$= a\left[\frac{1}{1+\exp(-a.net_j(t))}\right]\left[\frac{1+\exp(-a.net_j(t))-1}{1+\exp(-a.net_j(t))}\right]$$

$$= a y_j(t)\left[1-y_j(t)\right]$$

- Local gradient: - For output node

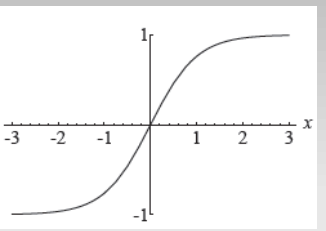
$$\delta_k = (d_k - y_k) * f'(net_k) = a*(d_k - y_k)*y_k*(1-y_k)$$

- For hidden node:

$$\delta_j = f'(v_j)\sum \delta_k w_{kj}$$

$$= a * z_j * (1-z_j) * \sum \delta_k w_{kj}$$

2- Hyperbolic tangent function



- Another form of sigmoid nonlinearity is:

$$y_j = f(net_j(t)) = a\tanh(b\,net_j(t)) \quad a>0 \quad and \quad b>0$$

- Differentiating it

$$f'(net_j(t)) = ab * \sec h^2(b.net_j(t)) = ab(1-\tanh^2(bnet_j(t)))$$

$$= \frac{b}{a}\left[a - y_j(t)\right]\left[a + y_j(t)\right]$$

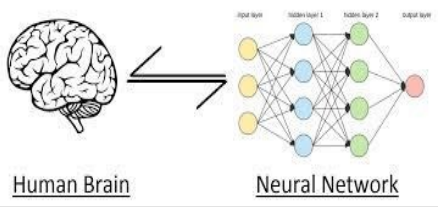
- Local gradient: - For output node:

$$\delta_k = (d_k - y_k) * f'(v_k) = \frac{b}{a} * (d_k - y_k) * \left[a - y_k(t)\right]\left[a + y_k(t)\right]$$

- For hidden node:

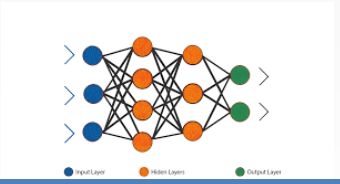
$$\delta_j = \phi'(net_j)\sum \delta_k w_{kj}$$

$$= \frac{b}{a} * \left[a - z_j(t)\right]\left[a + z_j(t)\right] * \sum \delta_k w_{kj}$$



Agenda

- Multilayer Perceptrons networks (Recap)
- Differentiable of Activation functions
- XOR Problem
- Backpropagation Improvement
- Local Minima problem
- Overtraining problem
- Learning with adaptive learning rate
- Preprocessing



XOR Problem

- The XOR function **could not be solved by a single layer perceptron network**.
  In single-layer perceptron, **there are no hidden neurons**. Consequently, it cannot classify input patterns that are not linearly separable.
- The XOR function consists of two inputs X and Y and one output F, we will consider four different patterns to solve the problem as illustrate in the following table:

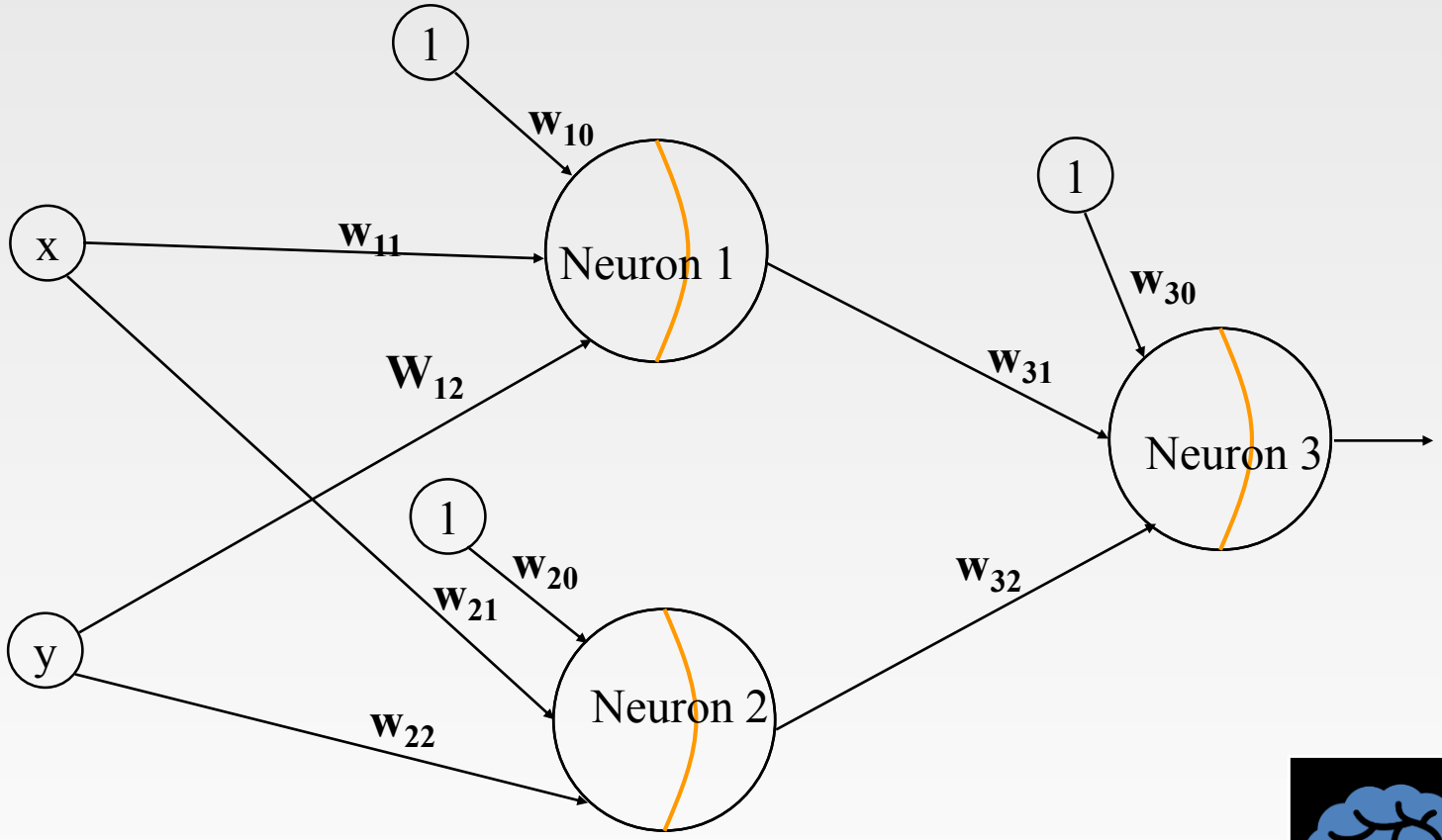
X	Y	F
0	0	0
0	1	1
1	0	1
1	1	0

XOR Architecture

We may solve the XOR problem by using a single hidden layer with two neurons as in the following figure:

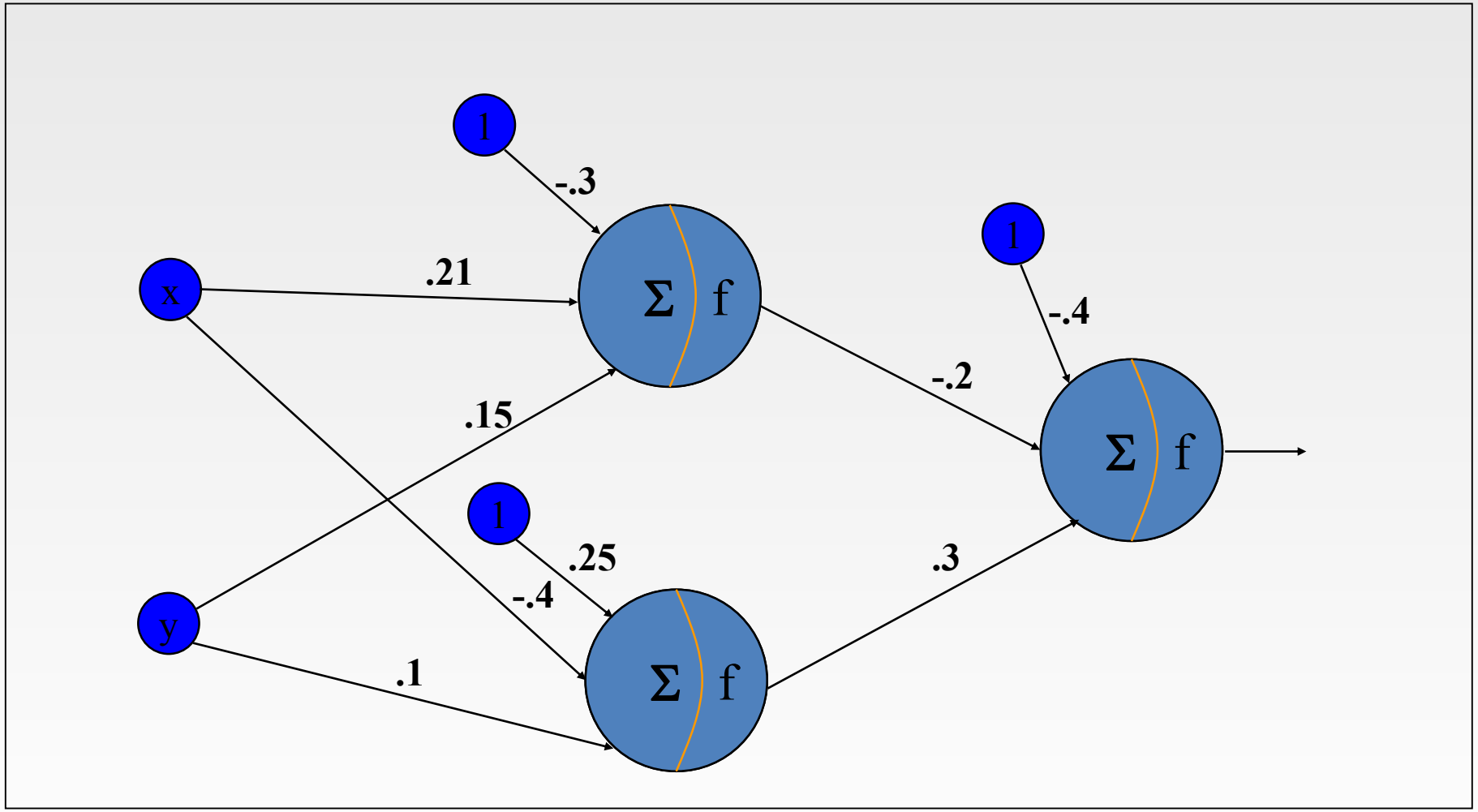
- Each computational neuron has a bias.

- Each computational neuron uses a logistic sigmoid function for its activation function.



Step 1: Initial Weights

Randomly assign small weight values.

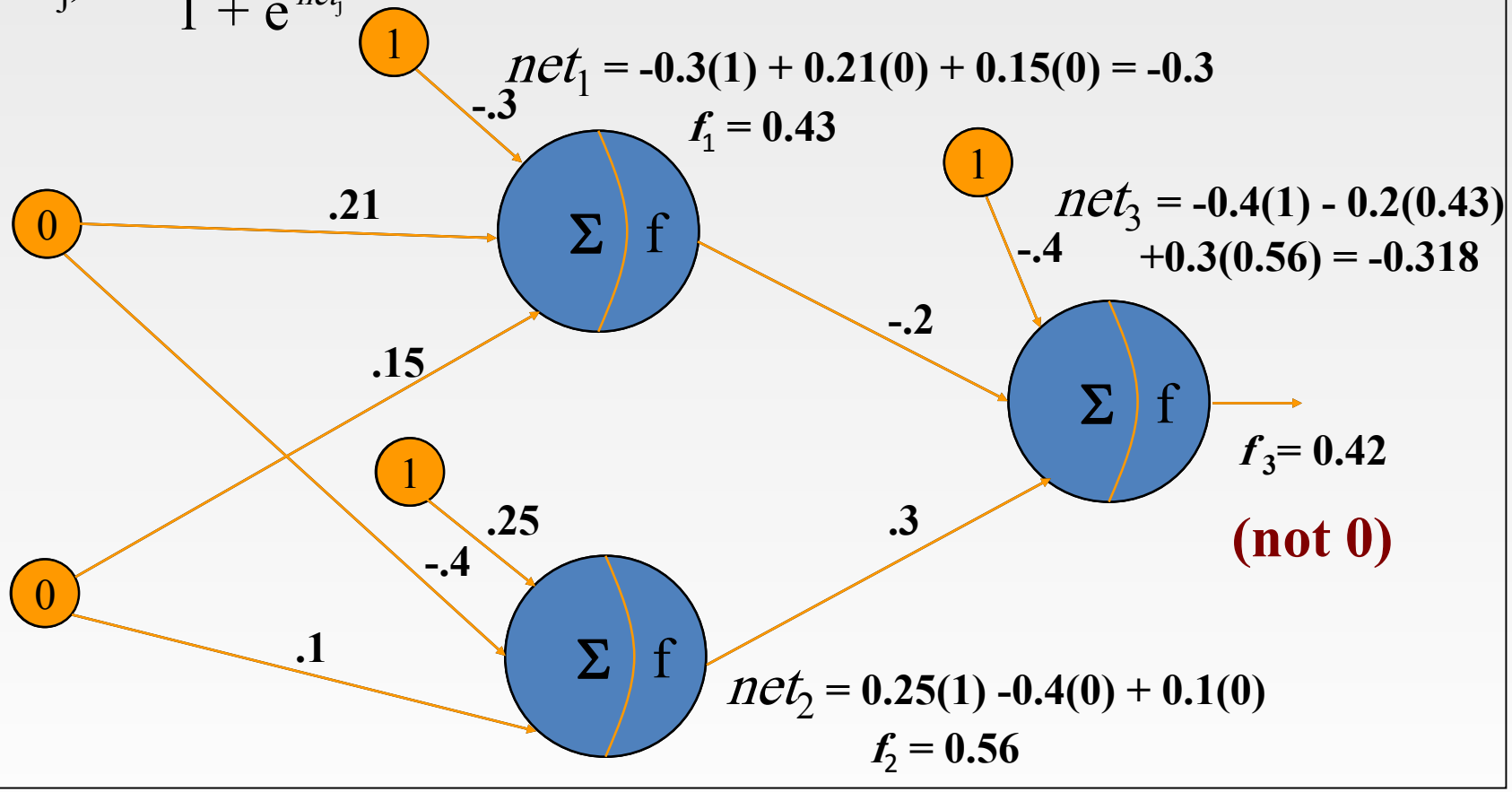


Step 2: Feedforward – 1st Pass

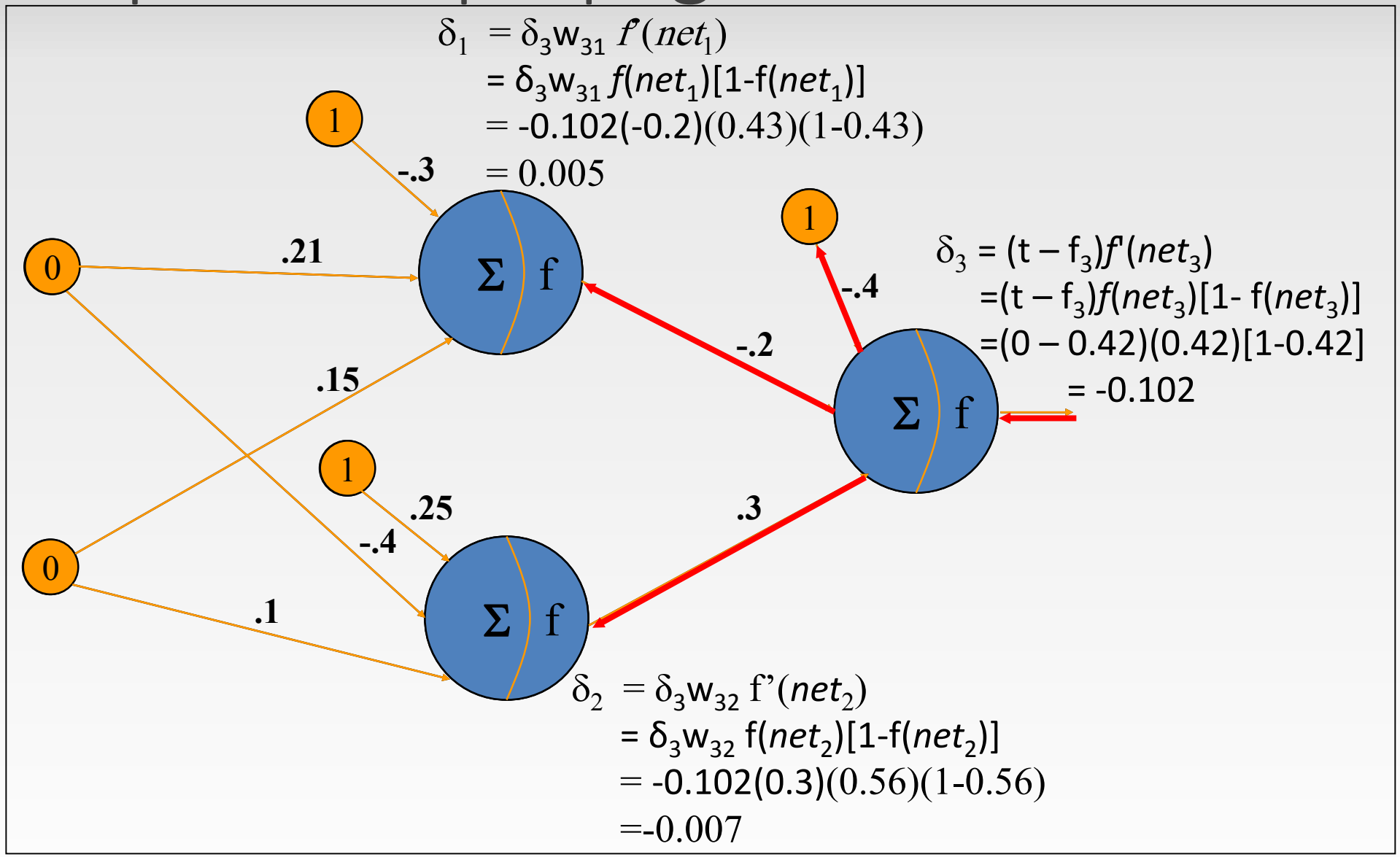
- present the first label input training pattern (0 0 0) to the network

- Activation function f:

$$f(net_j) = \frac{1}{1 + e^{-net_j}}$$



Step 3: Backpropagate – 2nd Pass

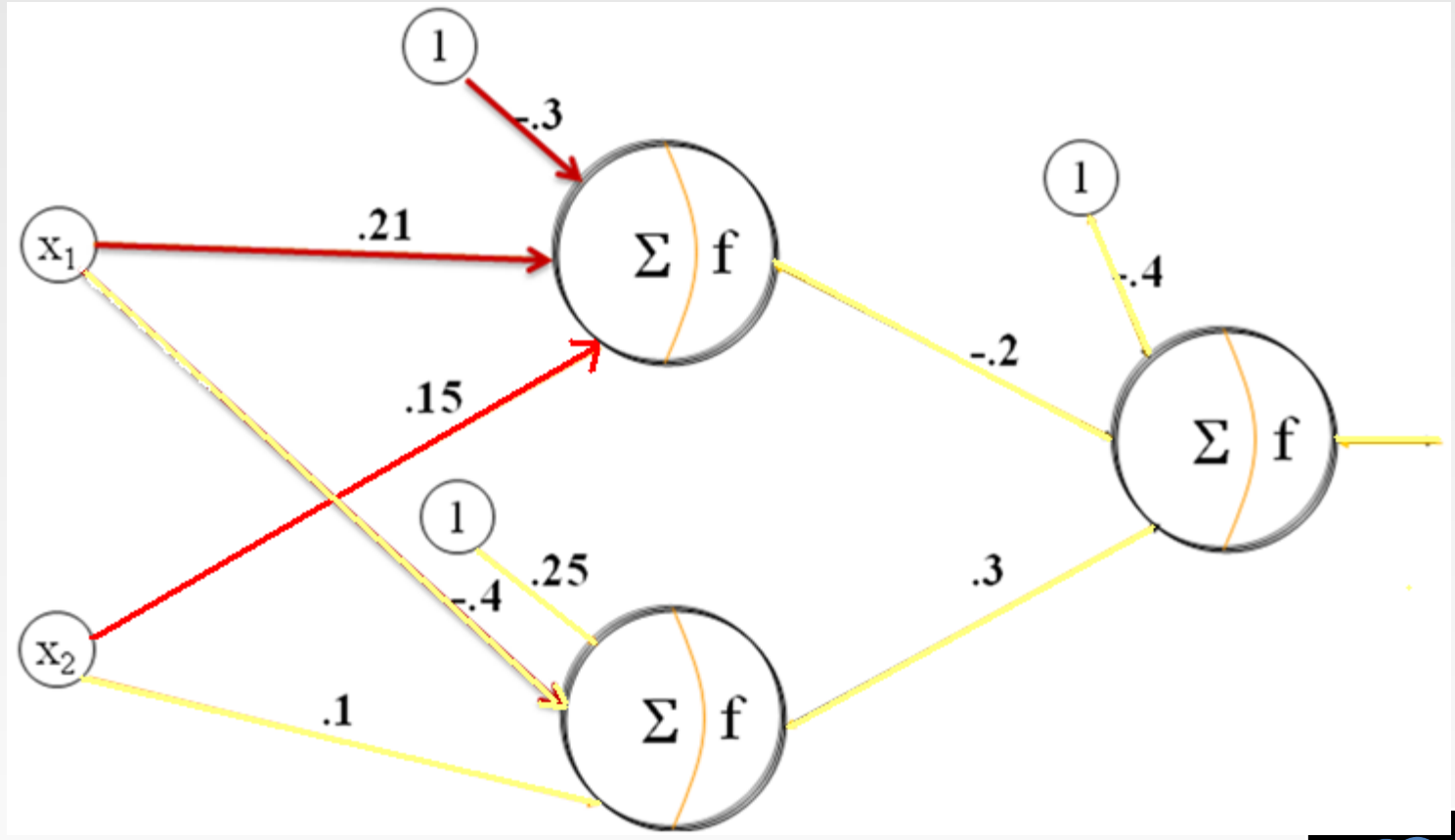


Step 4- Update the Weights

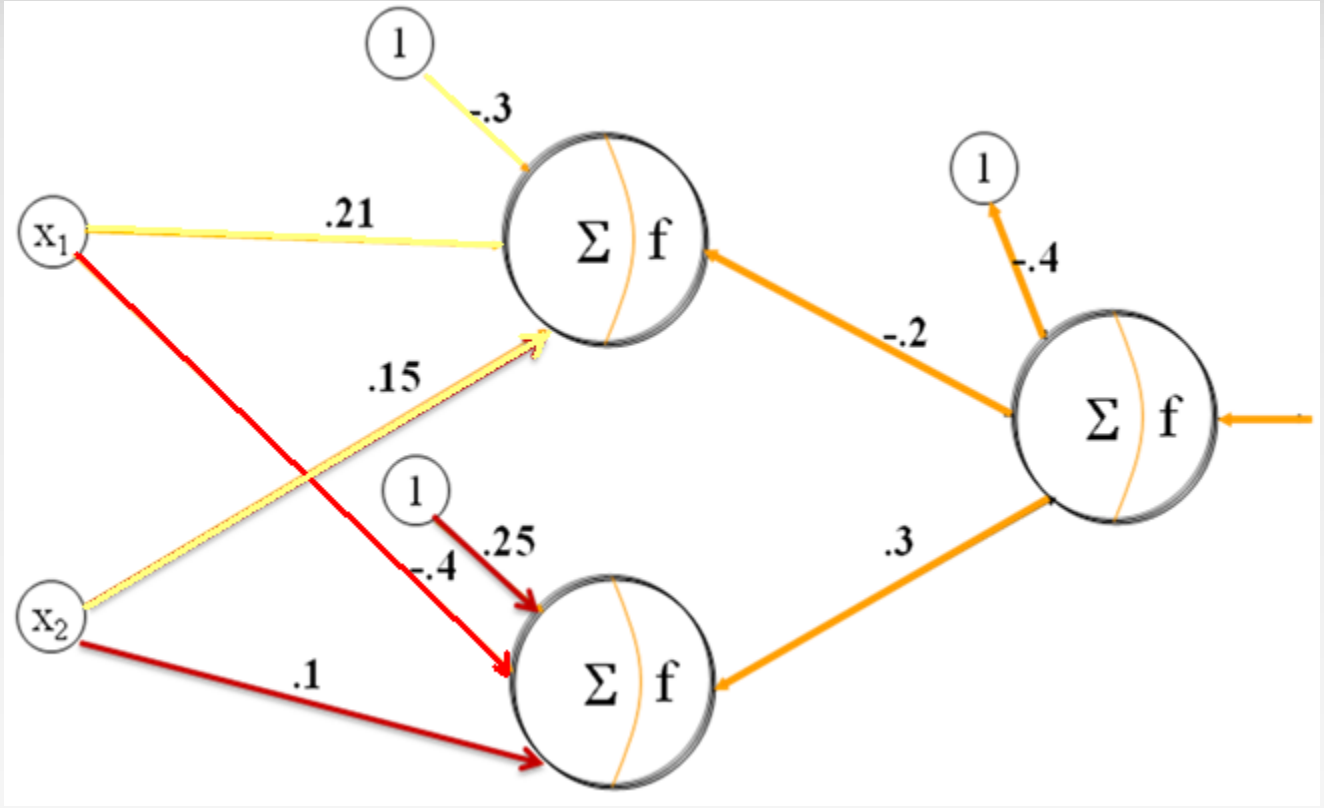
$$w'_{1(x_0)} = w_{1(x_0)} + \eta * \delta_1 * x_0$$

$$w'_{1(x_1)} = w_{1(x_1)} + \eta * \delta_1 * x_1$$

$$w'_{1(x_2)} = w_{1(x_2)} + \eta * \delta_1 * x_2$$



Step 4- Update the Weights, cont.



$$w'_{2(x_0)} = w_{2(x_0)} + \eta * \delta_2 * x_0$$

$$w'_{2(x_1)} = w_{2(x_1)} + \eta * \delta_2 * x_1$$

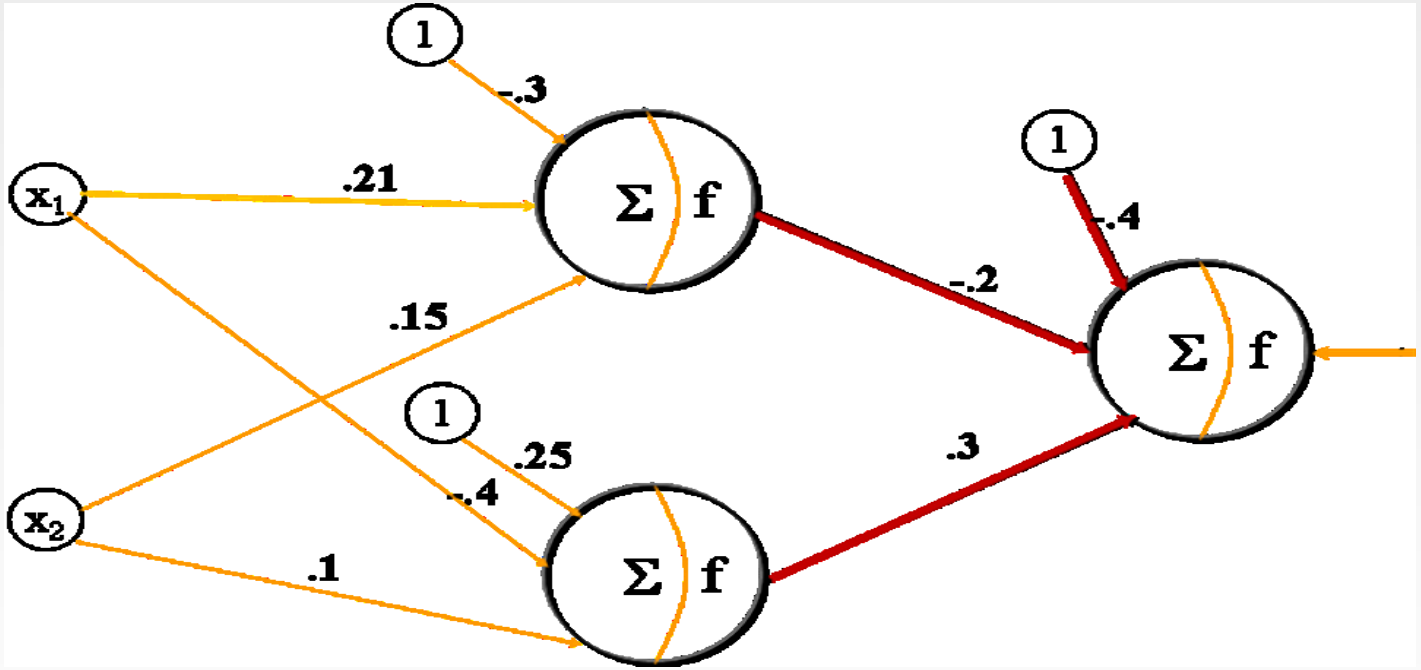
$$w'_{2(x_2)} = w_{2(x_2)} + \eta * \delta_2 * x_2$$

Step 4 - Update the Weights, cont.

$$w'_{30} = w_{30} + \eta * \delta_3 * f_0 \quad , \quad f_0 = bise = 1$$

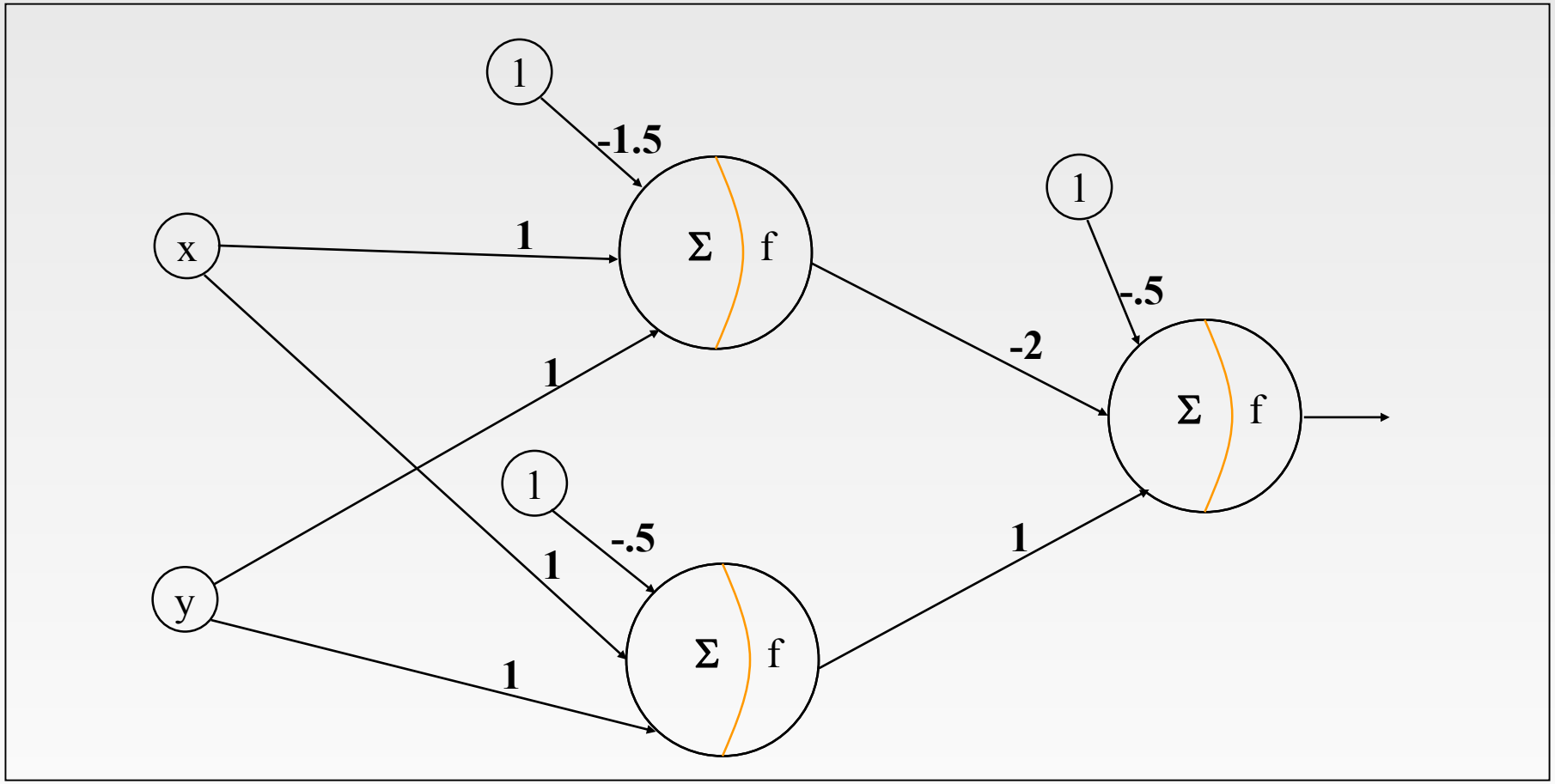
$$w'_{31} = w_{31} + \eta * \delta_3 * f_1$$

$$w'_{32} = w_{32} + \eta * \delta_3 * f_2$$

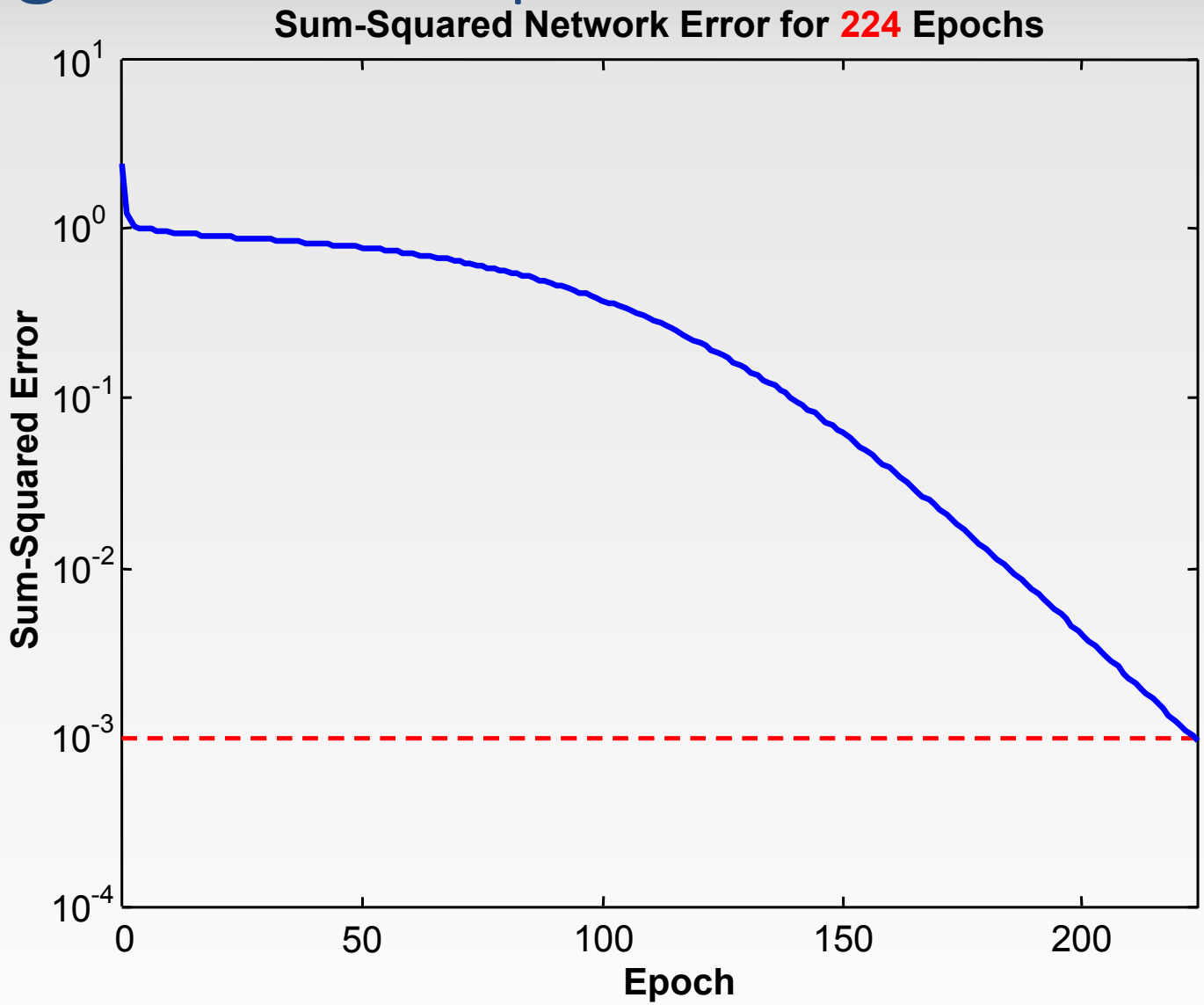


Final Result

The training process is repeated until the sum of squared errors is less than 0.001.



Learning curve for operation *Exclusive-OR*

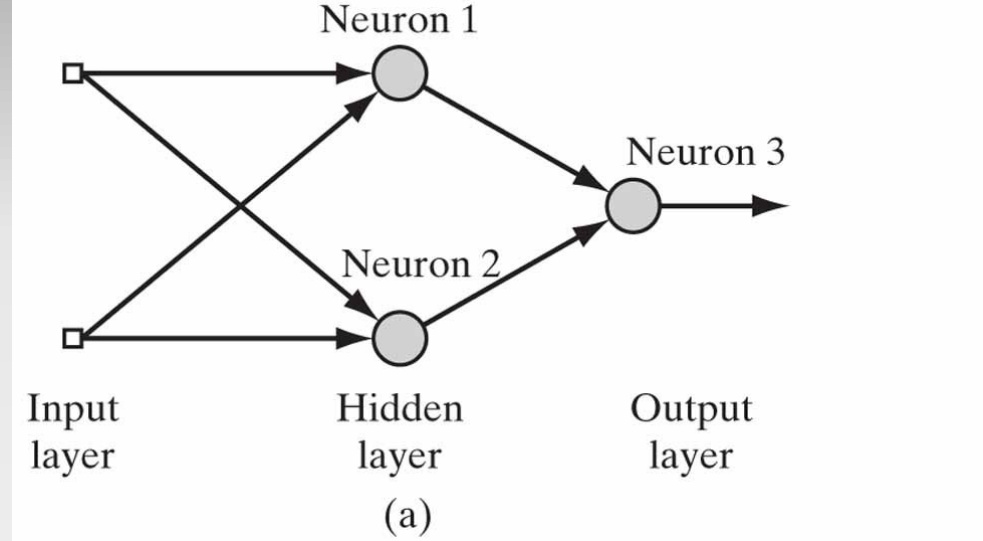


Decision boundaries

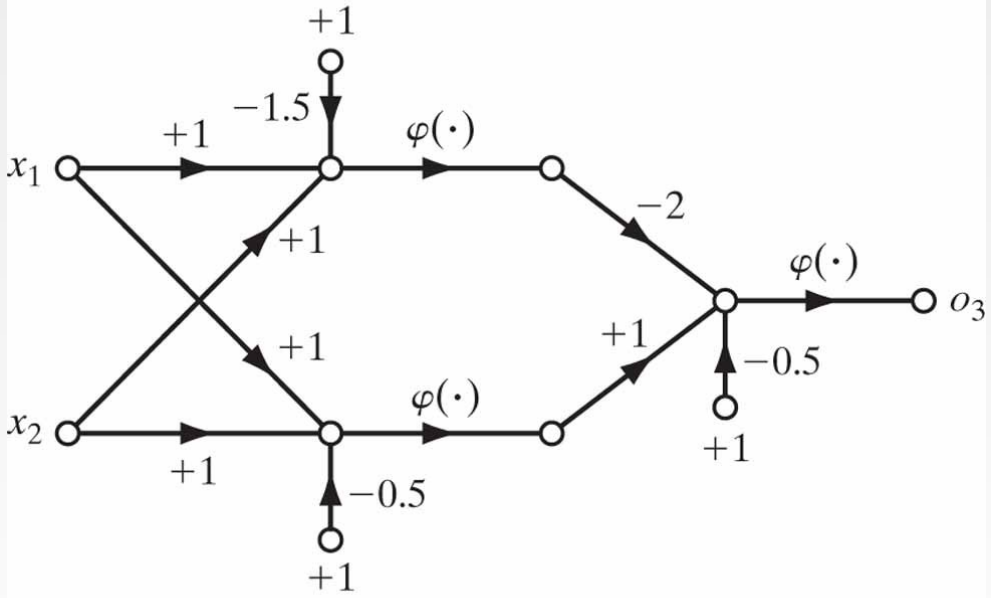
$$w_{11} = w_{12} = +1$$
$$b_1 = -\frac{3}{2}$$

$$w_{21} = w_{22} = +1$$
$$b_2 = -\frac{1}{2}$$

$$w_{31} = -2, \qquad w_{32} = +1$$
$$b_3 = -\frac{1}{2}$$



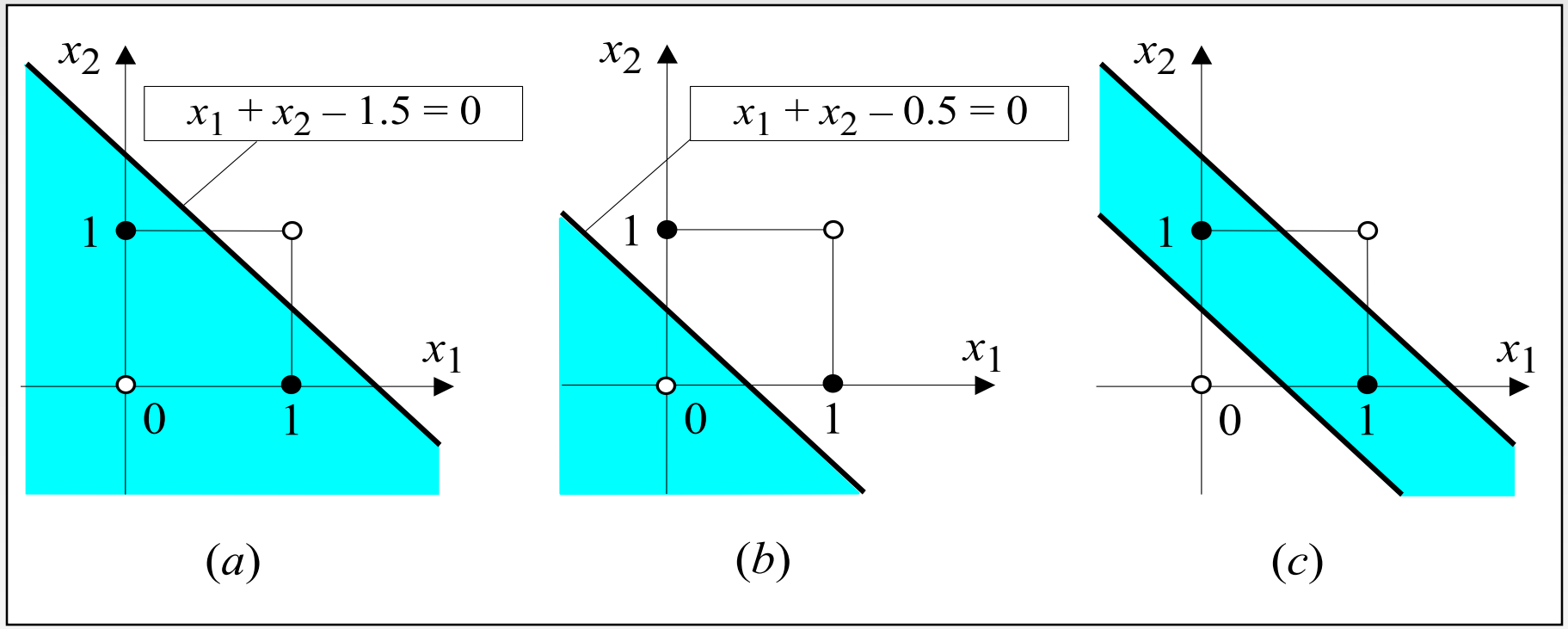
(a)



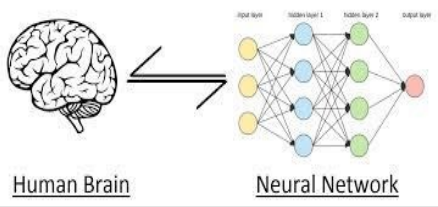
(b)

(a) Architectural graph of network for solving the XOR problem. (b) Signal-flow graph of the network.

Decision boundaries, cont.

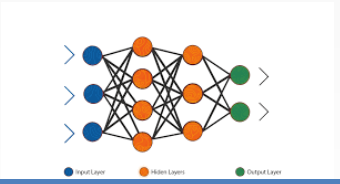


(*a*) Decision boundary constructed by hidden neuron 1;
(*b*) Decision boundary constructed by hidden neuron 2;
(*c*) Decision boundaries constructed by the complete network.

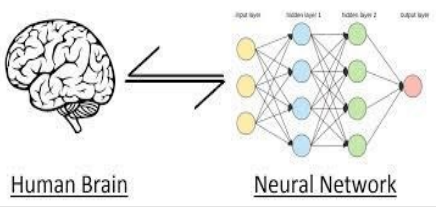


Agenda

- Multilayer Perceptrons networks (Recap)
- Differentiable of Activation functions
- XOR Problem
- Backpropagation Improvement
- Local Minima problem
- Overtraining problem
- Learning with adaptive learning rate
- Preprocessing



Advantages and limitations of the Backpropagation Using Gradient Descent

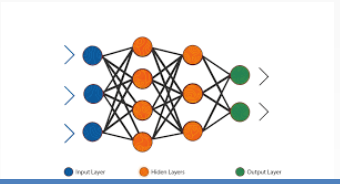


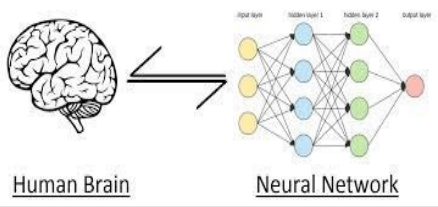
Advantages

- Relatively simple implementation
- Standard method after 80s and generally works well to solve linear & non linear problem

Limitations

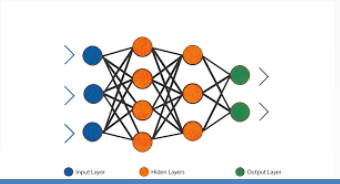
- The convergence obtained from backpropagation learning is very slow.
- If train too long, you run the risk of overfitting!!!!
- The convergence in backpropagation learning is not guaranteed.
- Can get stuck in local minima resulting in sub-optimal solutions

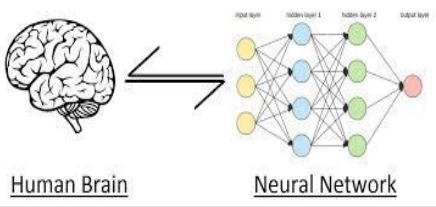




Backpropagation Improvement History

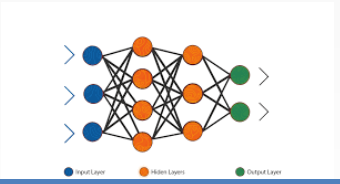
- Momentum (Rumelhart et al, 1986)
- Bounded Weights (Stinchcombe and White, 1990)
- Adaptive Learning Rates (Smith, 1993; LeCun et al, 1993)
- Normalizing Input Values (LeCun et al, 1993)
- Penalty Terms (eg. Saito & Nakano, 2000)

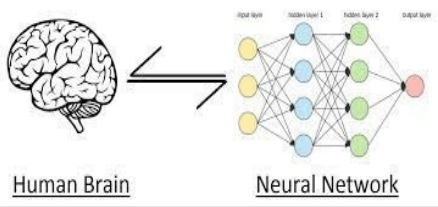




Agenda

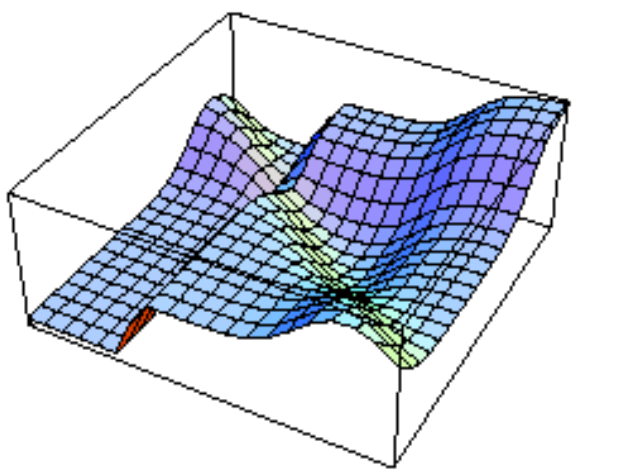
- Multilayer Perceptrons networks (Recap)
- Differentiable of Activation functions
- XOR Problem
- Backpropagation Improvement
- Local Minima problem
- Overtraining problem
- Learning with adaptive learning rate
- Preprocessing



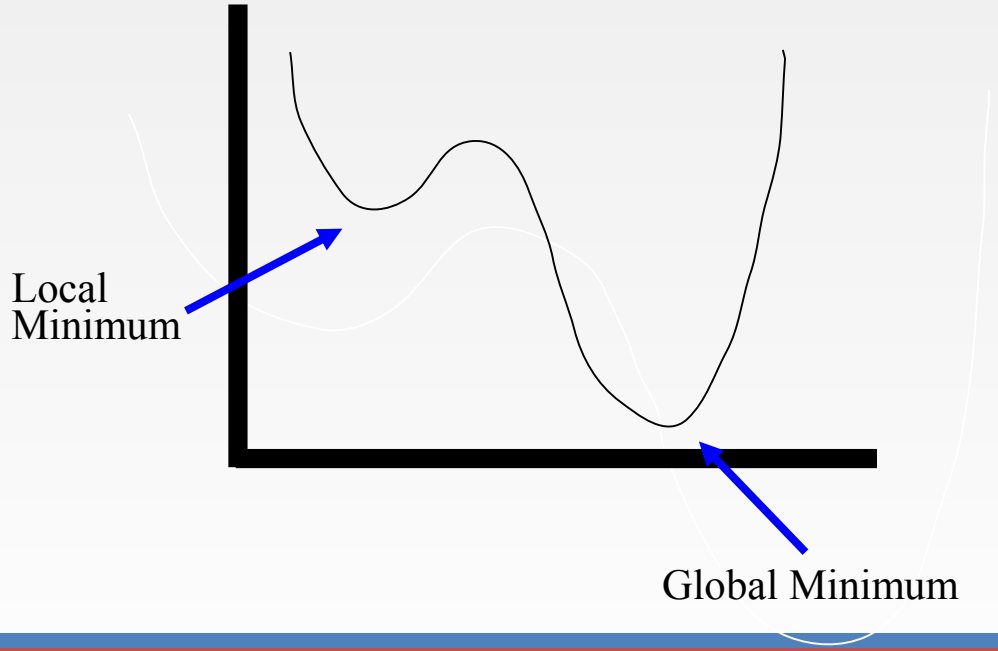


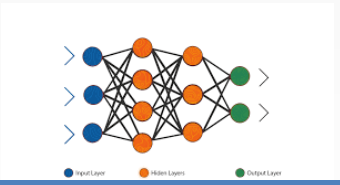
Local Minima

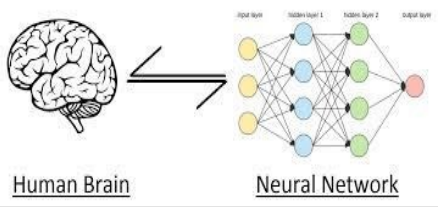
- It is possible that the backpropagation network may converge into a *local minima* instead of the desired global minimum
- The following figure shows an example of a local minimum with a higher error level than in other regions. The gradient descent algorithm will not converge to the global minum.



A local minimum of the error function

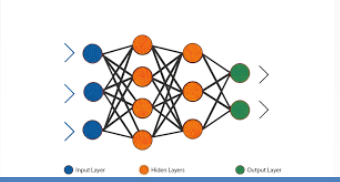


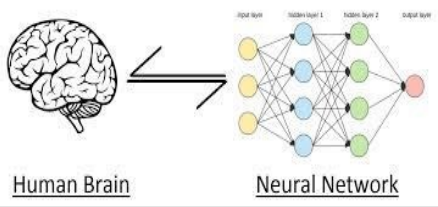




A strategies for dealing with local minima

1. Try nets with different number of hidden layers and hidden nodes (they may lead to different error surfaces, some might be better than others)
2. Try different initial weights (different starting points on the surface)
   - Repeat the training process several times with new random weights, and pick the network that gives the best generalization performance.
3. Train and test a committee or ensemble of networks, then use the average of the network outputs.
4. Another answer is to utilize *momentum (will discuss in next section)*, which gradually increases the weight adjustment.



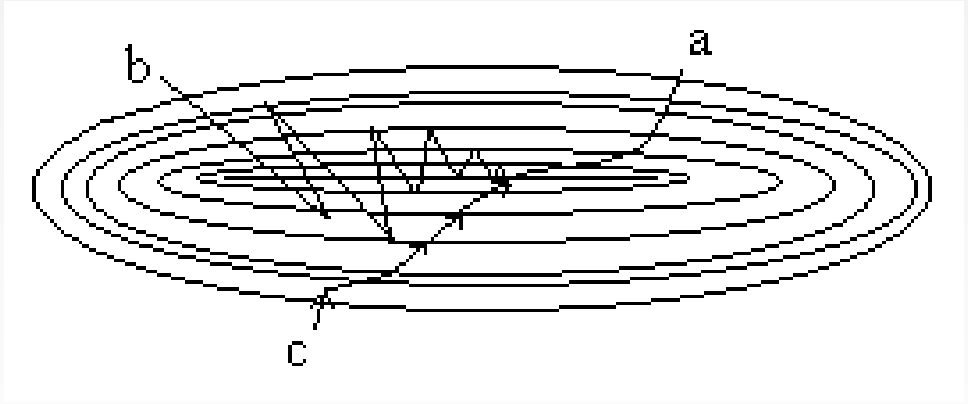


Momentum term

- The update weight rule,

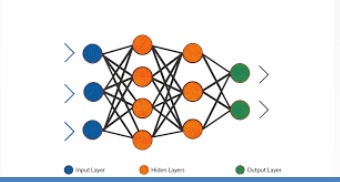
$$w_{ij}(t+1) = w_{ij}(t) - \eta \frac{\delta E}{\delta w_{ij}}$$

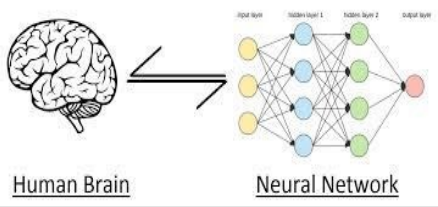
- make back-propagation very slow, especially when η is very small.
- If the learning-rate η is too large, the gradient descent will oscillate widely.
- In the following figure:



a) For small learning rate;

b) for large learning rate: note the oscillations.



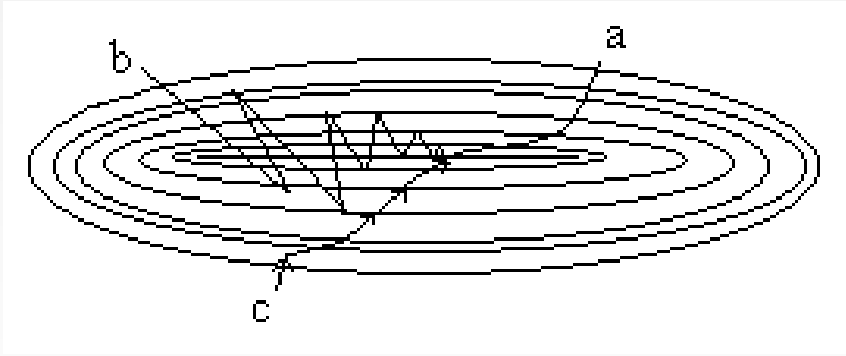


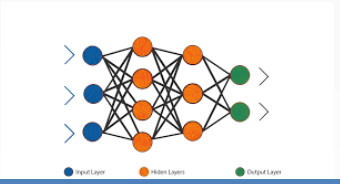
Momentum term (Cont.)

One way to speedup the learning process and to avoid oscillation (*avoiding instability*) at large η, is to make the change in weight dependent of the past weight change by adding a momentum term.

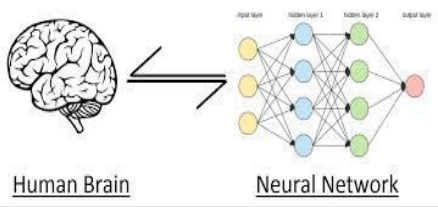
$$\Delta w_{ij}(t) = -\eta \frac{\delta E}{\delta w_{ij}} + \alpha\left(w_{ij}(t) - w_{ij}(t-1)\right)$$

Where α is a positive number called the momentum term. This equation called *generalized delta rule*.





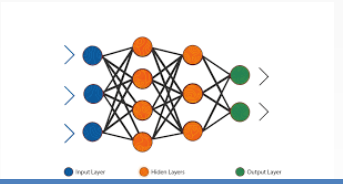
c) with large learning rate and momentum term added



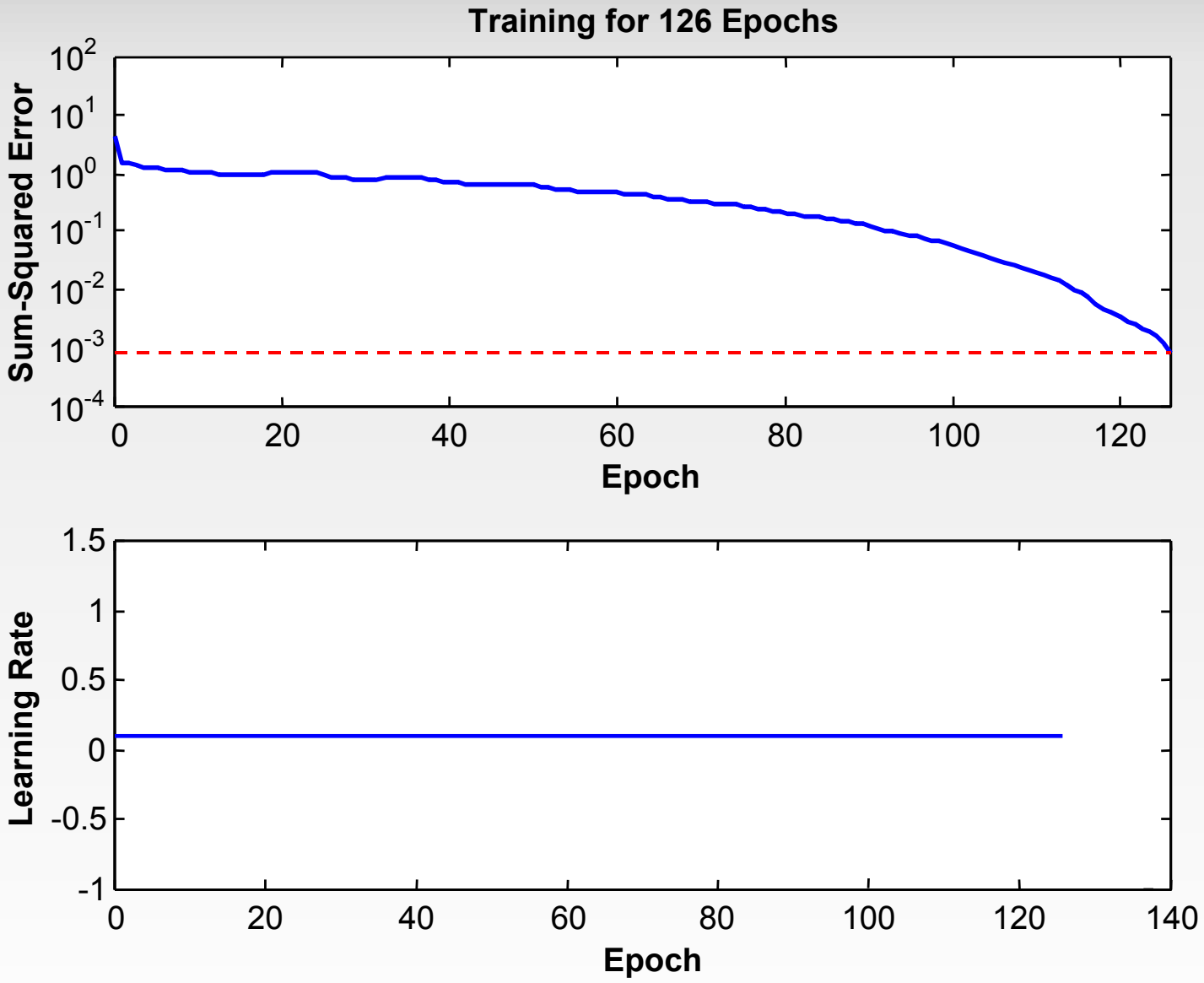
Momentum term, cont.

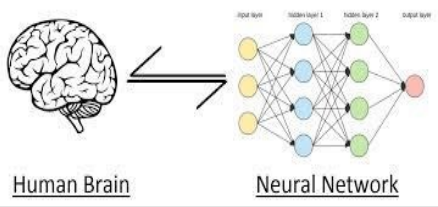
Adding the momentum to the weight and bias has the advantages:

- Speed up the convergence.
- Provides some immunity to local minima.
- Avoid sudden change of directions of weight update (smoothing the learning process)



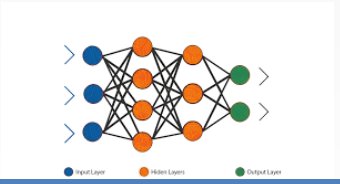
Learning with momentum for XOR

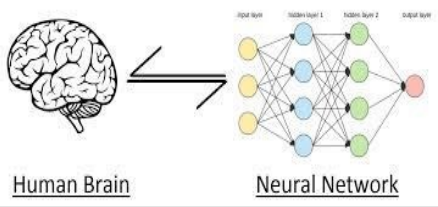




Agenda

- Multilayer Perceptrons networks (Recap)
- Differentiable of Activation functions
- XOR Problem
- Backpropagation Improvement
- Local Minima problem
- Overtraining problem
- Learning with adaptive learning rate
- Preprocessing

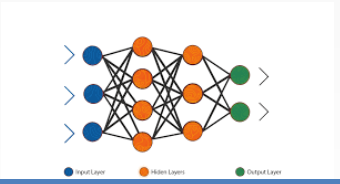


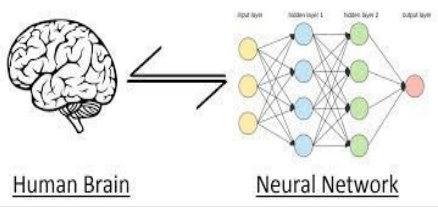


When to Stop Learning

Learn until error on the training set is below some threshold

- Bad idea! Can result in overfitting
  - If you match the training examples too well, your performance on the real problems may suffer



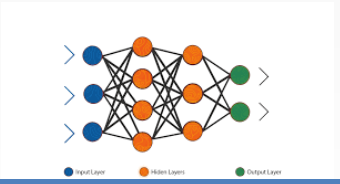


Introduction

The hope is that the network becomes well trained so that it learns enough about the past to generalize to the future.

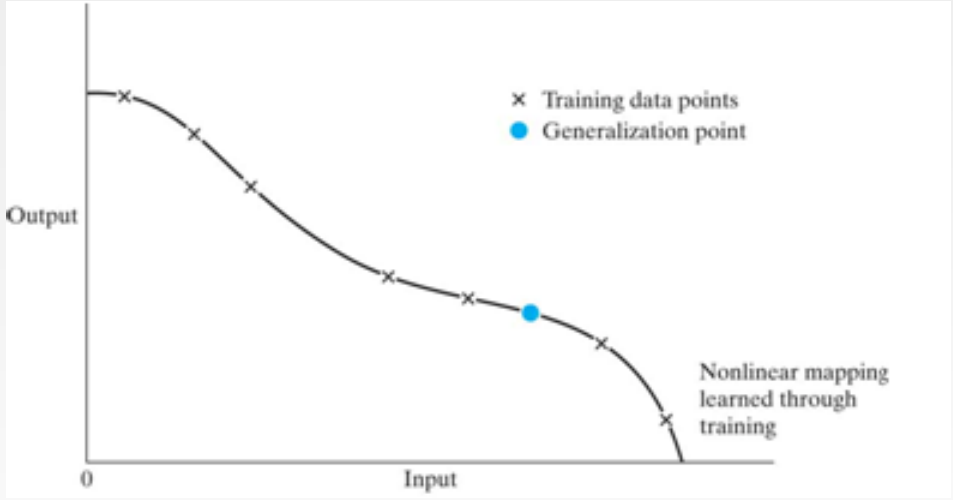
A network is said to *generalize well* when the network *input-output* mapping is *correct* (or nearly so) for the *test* data.

If the network becomes very good at classifying the training set, but poor at classifying the test set, that it is known as *overtraining problem* and solve by a standard tool in statistics, known as *cross validation*.

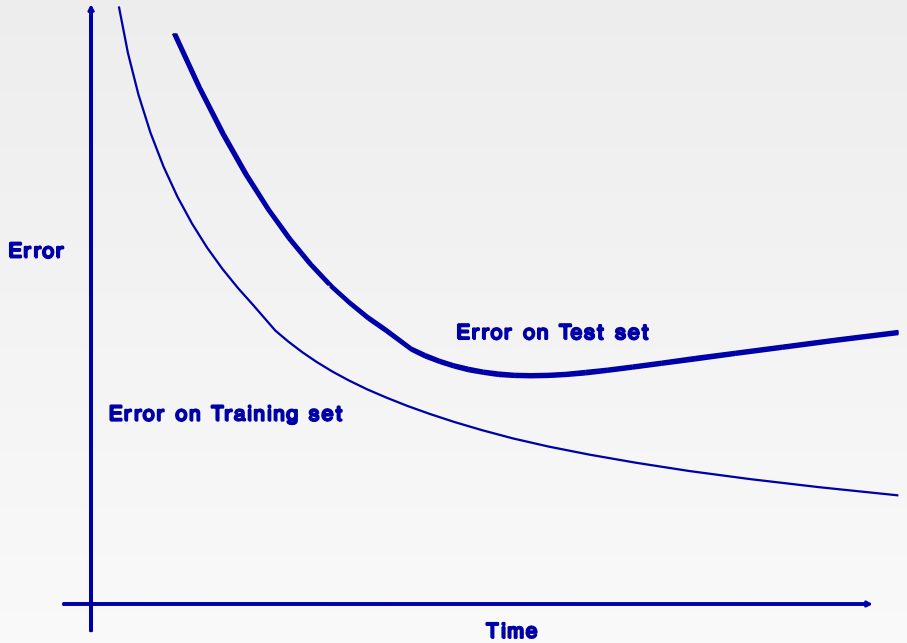


Overtraining (Overfitting)

- Over-fitting/over-training problem: It is possible to train a network too much, where the network becomes very good at classifying the training set, but poor at classifying the test set (inputs not in the training set) that it has not encountered before, i.e. it is not generalising well.



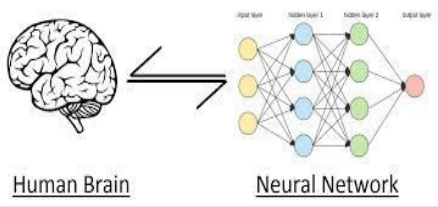
good generalization



Overfitted nonlinear mapping with poor generalization

Early stopping

- The only way to avoid the overfitting is to periodically present the test set to the network, and record the error while storing the weights at the same time, in this way an optimum set of weights giving the minimum error on the test set can be found. This technique is called early stopping.
- Early stopping technique can be implement using the statistical cross validation tool.
- Cross validation is the more powerful of the stopping criteria methods since it stops the training at the point of best generalization (i.e. the performance in the test set) is obtained.



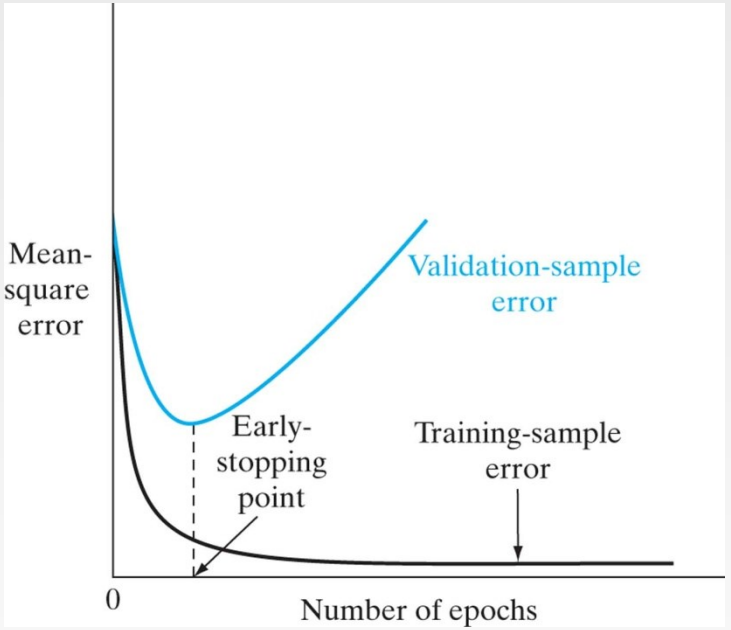
Implement Cross Validation

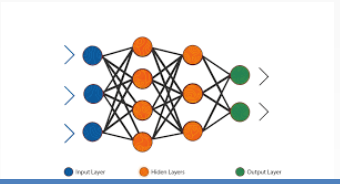
The available data set is randomly partitioned into a training set and a test set.

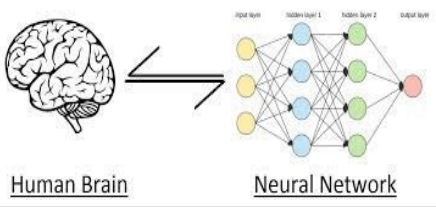
The training set is future partitioned into two disjoint subsets:

- A training subset
- A validation subset, (10 to 20 percent of the training set).

The validation subset use to see how the trained network is doing (e.g. every 100 training epochs, test the net with a validation set). When the performance starts to degrade in the validation set, training should be stopped.





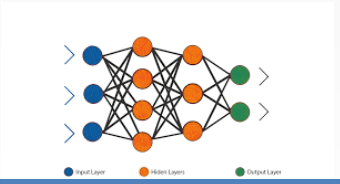


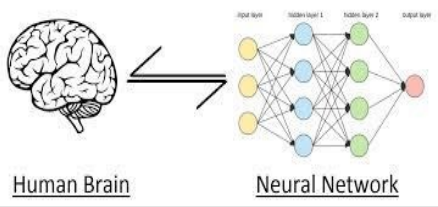
Stopping Criteria of Training

To start back-propagation, we need to load an initial value for each weight (normally a small random value), and proceed until some stopping criterion is met.

The three most commons Stopping Criteria are:

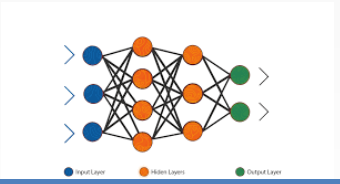
- to fix the number of iterations,
- to threshold the output mean square error,
- or to use cross validation.



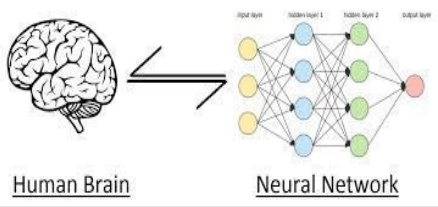


Agenda

- Multilayer Perceptrons networks (Recap)
- Differentiable of Activation functions
- XOR Problem
- Backpropagation Improvement
- Local Minima problem
- Overtraining problem
- Learning with adaptive learning rate
- Preprocessing

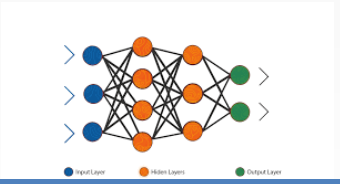


Learning with Adaptive control of the Learning Rate

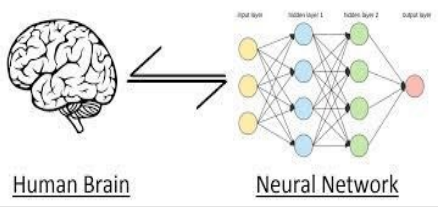


Different methods to adaptive control of the Learning Rate for back-propagation algorithm:

1. All neurons should learn at the same learning rate.
2. The last layer usually has larger local gradients than the layer at the front end of the network.
   - Hence, the learning rate parameter η should be assigned a smaller value in the last layers than in the front layers of the MLP.
3. Assigns each weight a learning rate parameter according to:
   - Neurons with many inputs should have a smaller learning rate parameter than neurons with few inputs.
4. LeCun 1993, it is suggested that for a given neuron, the learning rate parameter should be inversely propositional to the square root of the synaptic connections made to that neuron.



Learning with Adaptive control of the Learning Rate, cont.

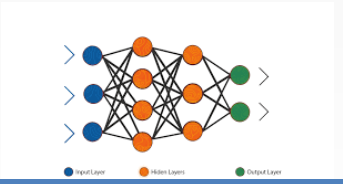


5- It assigns each weight a learning rate

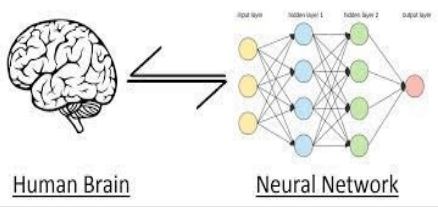
That learning rate is determined by the sign of the gradient of the error function from the last iteration

- If the signs are equal it is more likely to be a shallow slope so the learning rate is increased
- The signs are more likely to differ on a steep slope so the learning rate is decreased

This will speed up the advancement when on gradual slopes



Learning with Adaptive control of the Learning Rate, cont.



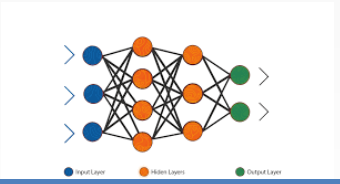
6- The learning rate play a more important role in on-line learning than in batch learning.

The on-line learning algorithm needs to be equipped with a built-in mechanism for the adaptive control of the learning rate.

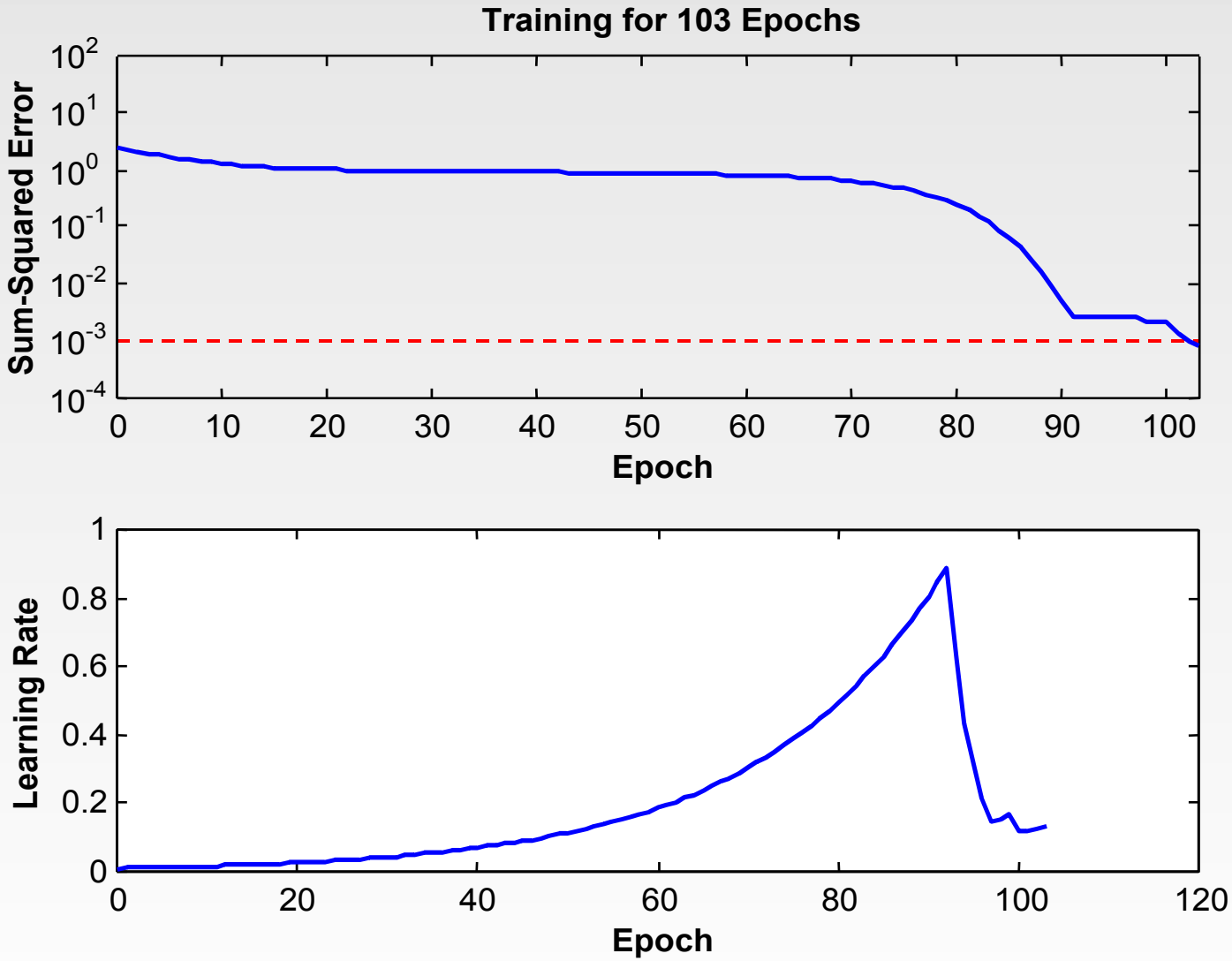
Darken and Mody 1991, let

$$\eta(t) = \frac{\tau}{t + \tau}\eta_0$$

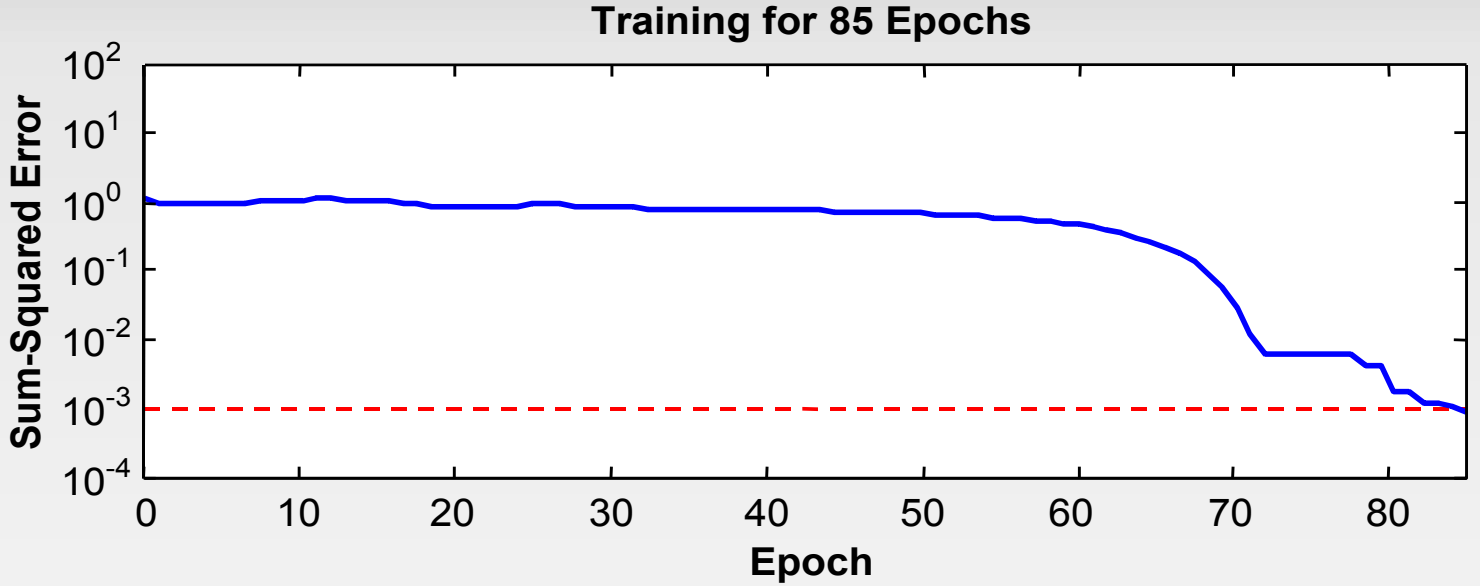
Account for dependence of the learning rate on time t, where η and τ are positive tuning parameters refer to the iteration # and total number of iterations.

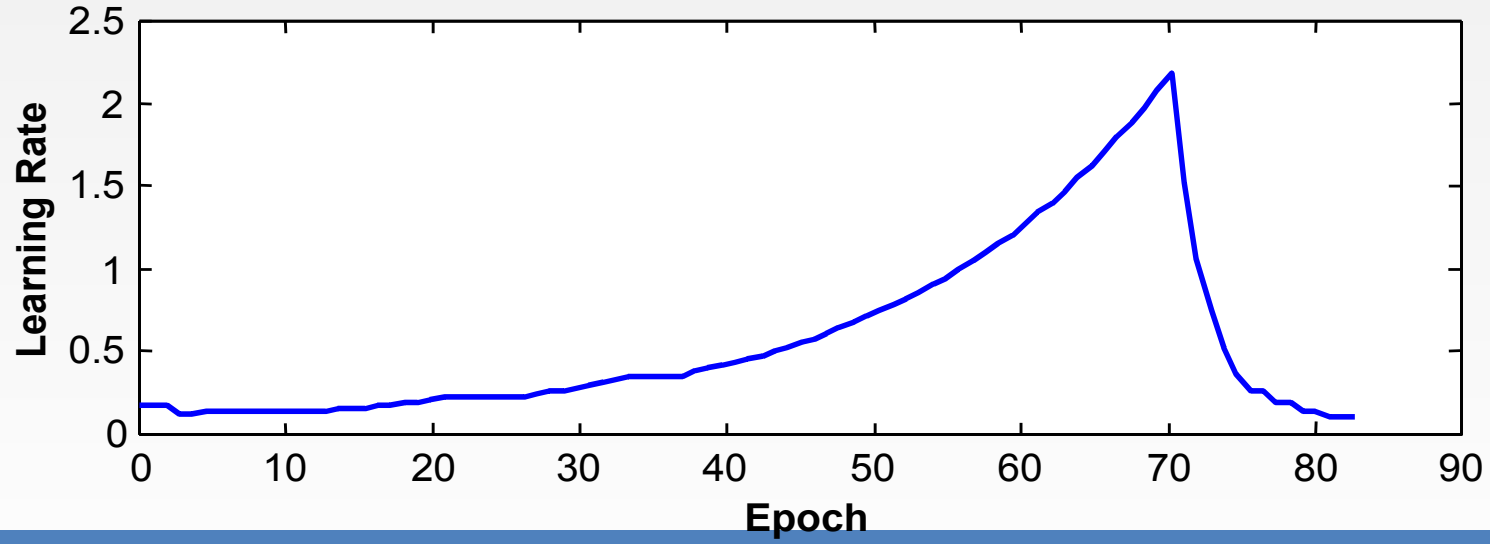


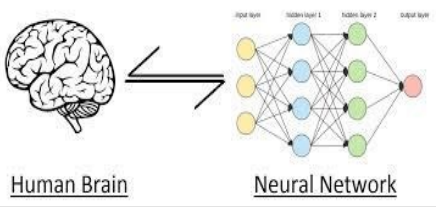
Learning with Adaptive control of the Learning Rate, cont.



Learning with Adaptive control of the Learning Rate and using momentum term, cont.

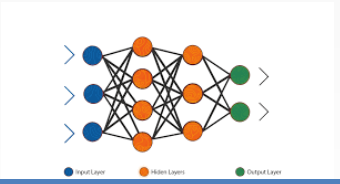


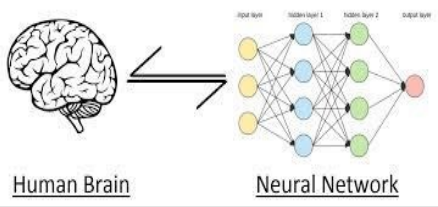




Agenda

- Multilayer Perceptrons networks (Recap)
- Differentiable of Activation functions
- XOR Problem
- Backpropagation Improvement
- Local Minima problem
- Overtraining problem
- Learning with adaptive learning rate
- Preprocessing





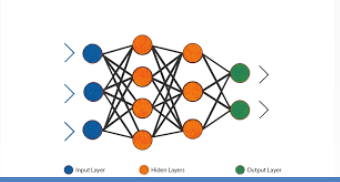
Why Preprocessing ?

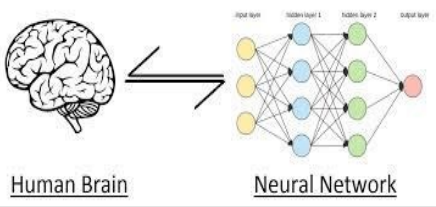
1- The Curse of Dimensionality

- The quantity of training data grows exponentially with the dimension of the input space
- In practice, we only have limited quantity of input data
  - Increasing the dimensionality of the problem leads to give a poor representation of the mapping

2- Inputs of the neural net are often of different types with different orders of magnitude (E.g. Pressure, Temperature, etc.)

- It is necessary to normalize the data so that they have the same impact on the model. And this makes the cost function faster to optimize.





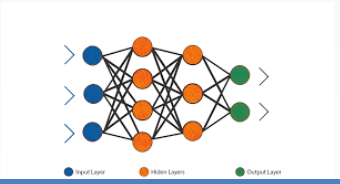
Preprocessing methods

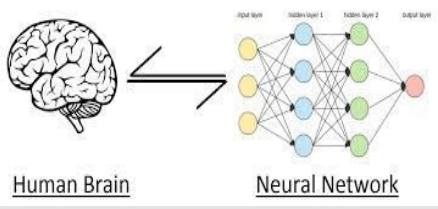
To solve Curse of Dimensionality problem, need to Component reduction

- Build new input variables in order to reduce their number
- No Lost of information about their distribution
- Reduction methods (PCA, CCA, etc.)

Normalization

- Translate input values so that they can be exploitable by the neural network

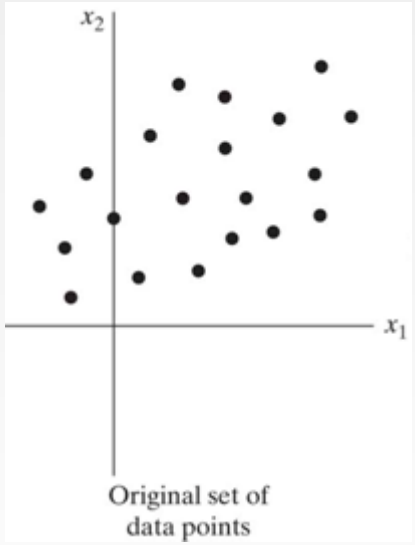


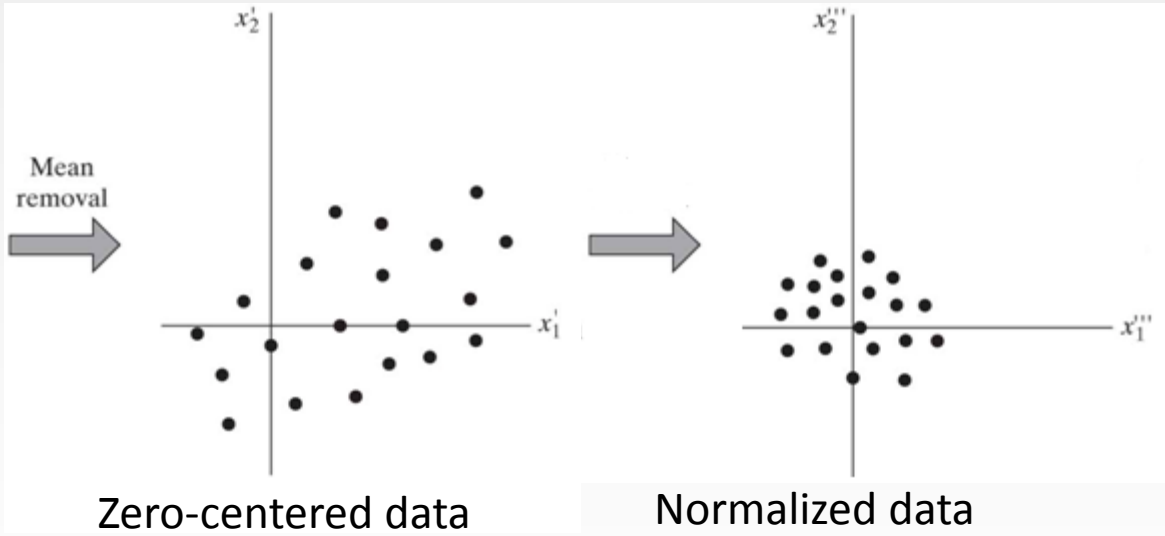


Preprocess the data by: Normalizing the input training data

Each input value should be preprocessed so that its men value, averaged over the entire training sample, is close to zero (LeCun et al, 1993).

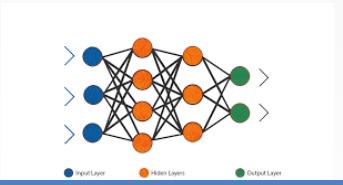
Training data should be scaled in the region (-1, +1) or (0, +1) to ensure initial operation in the high sensitivity region of the activation functions.

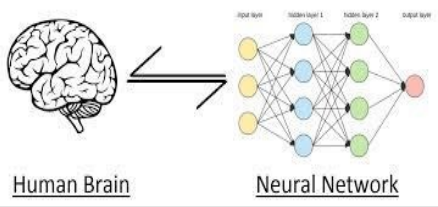




Zero-centered data

Normalized data



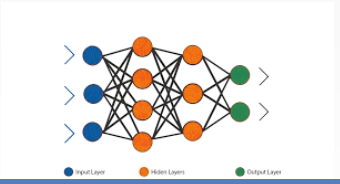


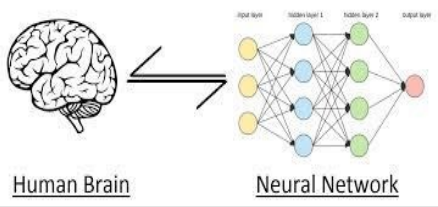
Normalize Training samples, cont.

Real input data values are standardized (scaled), normalized so that they all have ranges from 0 – 1.

$$\text{newValue} = \frac{\text{originalValue} - \text{minimumValue}}{\text{maximumValue} - \text{minimumValue}}$$

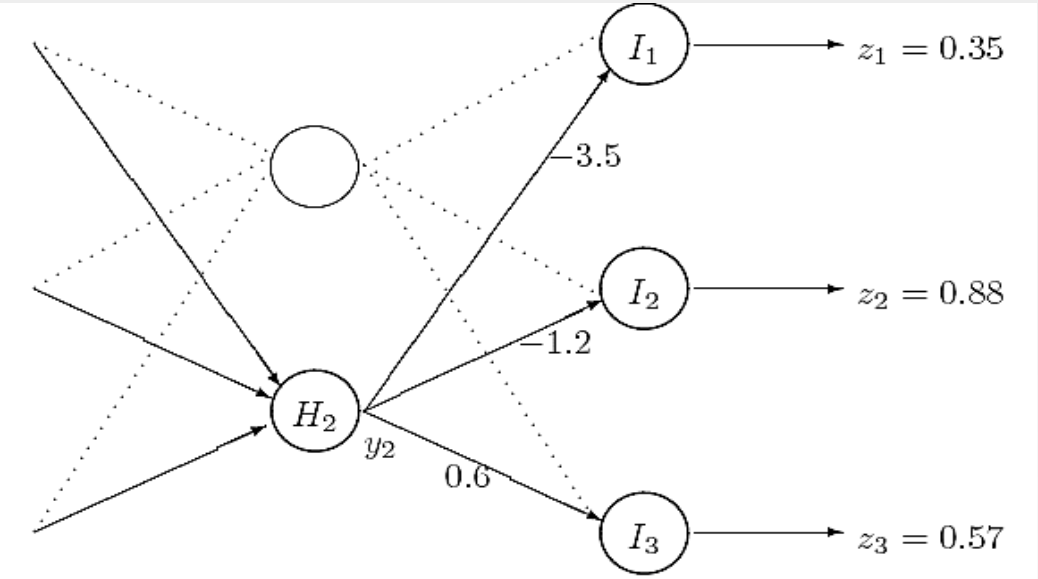
where
newValue is the computed value falling in the [0,1] interval range
originalValue is the value to be converted
minimumValue is the smallest possible value for the attribute
maximumValue is the largest possible attribute value

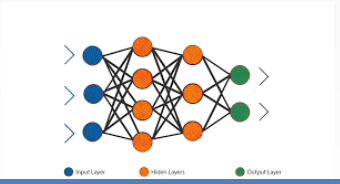


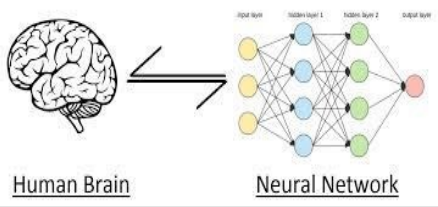


Homework

The following figure shows part of a neural network. For an input pattern, the actual outputs of the network are given by $z = [0.35, 0.88, 0.57]^T$ and the corresponding target outputs are given by $t = [1.00, 0.00, 1.00]^T$. The weights w_{12}, w_{22} and w_{32} are also shown below.







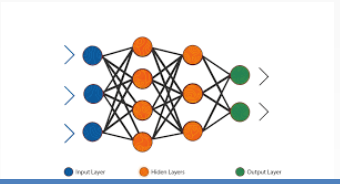
Homework, cont.

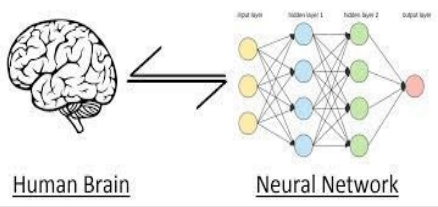
Assume that all units have sigmoid activation functions given by

$$f(x) = \frac{1}{1 + \exp(-x)}$$

1- What is the error signal (local gradient) for each of the output units?

2 - For the hidden units of the previous network, what is the error signal (local gradient) for hidden unit 2 given that its activation for the pattern being processed is currently $y_2 = 0.74$?





Next Lecture

Deep Learning

Thanks