CIS501 – Lecture 16

Woon Wei Lee
Fall 2013, 10:00am-11:15pm,
Sundays and Wednesdays

For today:

- Neural Networks
  - Perceptron network
  - Multi-layer Perceptrons intro
- Presentations
  - Khawla Masood Aldhaheri
  - Tesfagabir Meharizghi

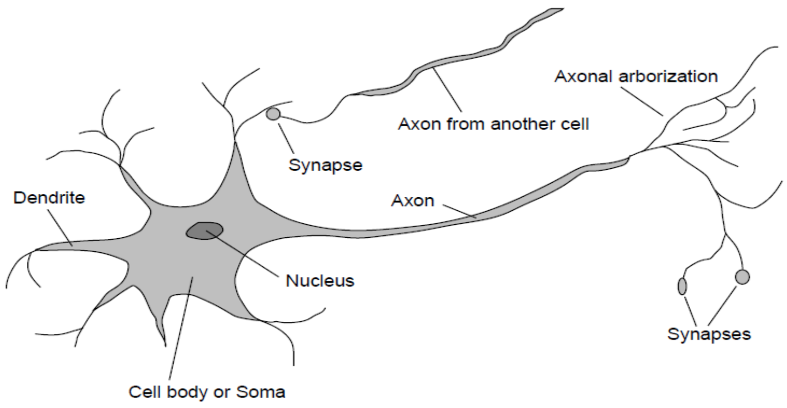
Neural Networks

- **Biologically inspired computing paradigm based on the functioning of the human brain.**
  - $\approx 10^{11}$ Neurons
  - $\approx 10^{14}$ inter-neural connections
- **Desirable properties / Motivations:**
  - It's awesome! - nothing approaches the flexibility and speed (in certain aspects) of the brain
  - Massively parallel computing architecture
    - easy for implementation on a large number of smaller/cheaper computing units
  - Redundancy and distributed computing – damage to parts of the brain are often either "repaired", or the brain works around it
    - Stories of patients surviving horrendous damage to the brain without serious cognitive defects

Masdar INSTITUTE

Neural Networks (Cont'd)

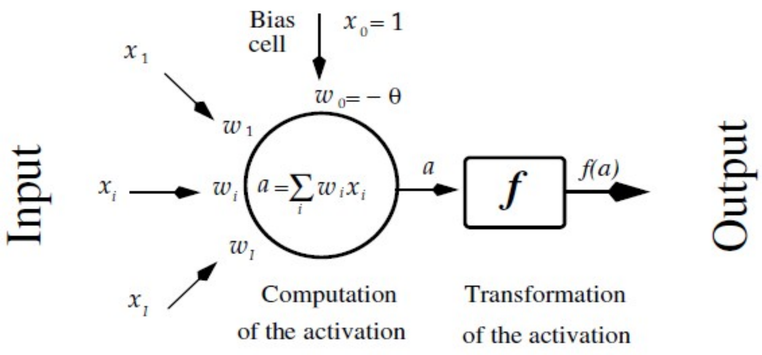
- Based on “actual” (Biological) neurons:



- Inputs from other neurons collected through “Dendrites” and aggregated
- Operates on “all or nothing” principle →
  - If aggregated inputs do not exceed a certain threshold, nothing happens
  - If threshold exceed, neuron “fires”
  - Transmitted to downstream neurons via “Axon”
- Reminiscent of a step function (recall from previous lecture..)
- Believed to be basic unit of pattern recognition

McCulloch-Pitts Neuron

- **Mathematical abstraction which aims to re-produce the operation of the neuron**



- **Diagram above shows single "neuron". Operates as follows:**
  - As before, can take a number of inputs $x_1 \ldots x_n$
  - Inputs are aggregated (summed) via the weight parameters, $w_1 \ldots w_n$
  - Bias term b can adjust the "threshold"
  - Result is the activation term, a
  - Activation is transformed via a "transfer function" $f(.)$, to produce the neuron output:

$$Output = f(a) = f\left(\sum_{i=1}^{n} w_i \cdot x_i + b\right)$$

Broader perspective

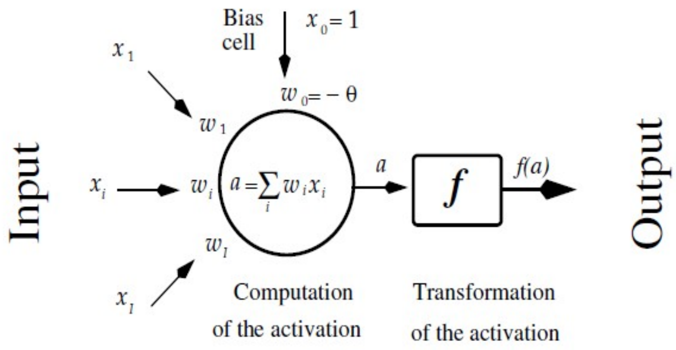
- **In general, the term “network” can be misleading..**
  - A neural network can be as simple as a single neuron
  - Alternative term *neural computing* possibly more representative
- Two general classes:
  - Feed-forward neural networks
    - Perceptrons
    - Multi-Layer Perceptrons
    - RBFs
    - SOMs
    - Hebbian Networks

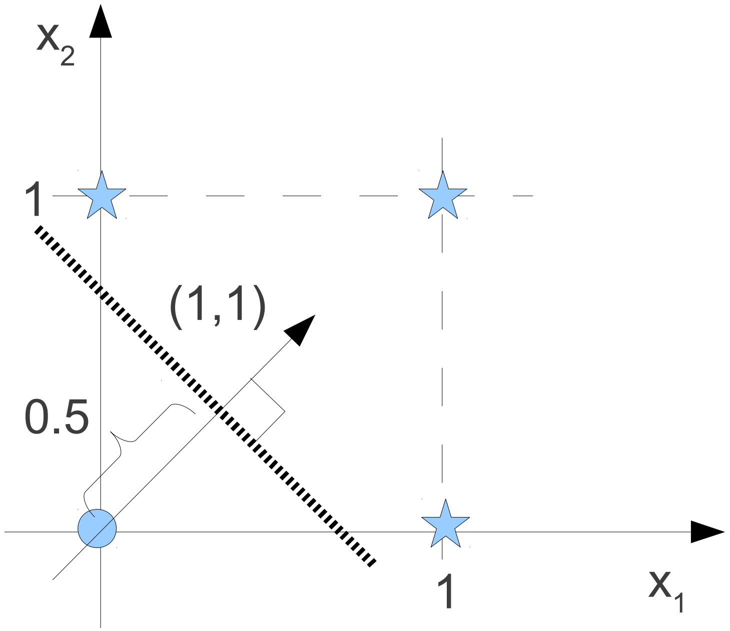
    Supervised, Classification/Regression (Perceptrons, Multi-Layer Perceptrons, RBFs)

    Un-supervised, Clustering/Visualization (SOMs, Hebbian Networks)
  - Recurrent neural networks
    - Hopfield Networks
    - Boltzmann Networks
  - “Modern” approaches
    - Support Vector Machines
    - Gaussian Processes

Perceptron

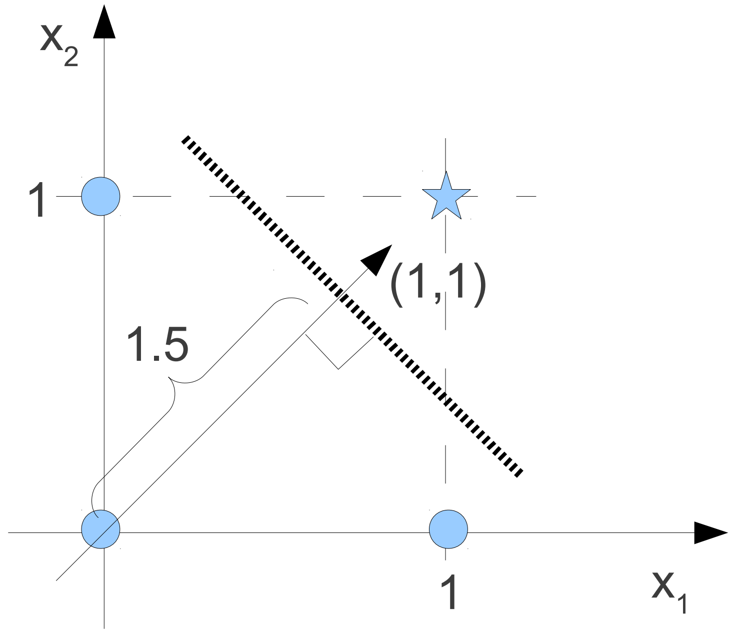
- **Simplest class of feedforward network**
  - As in figure above, but *f(.)* is a step function
  - Input weights $w_1 \ldots w_n$ define a direction in the input space, bias b defines the location of the decision threshold.



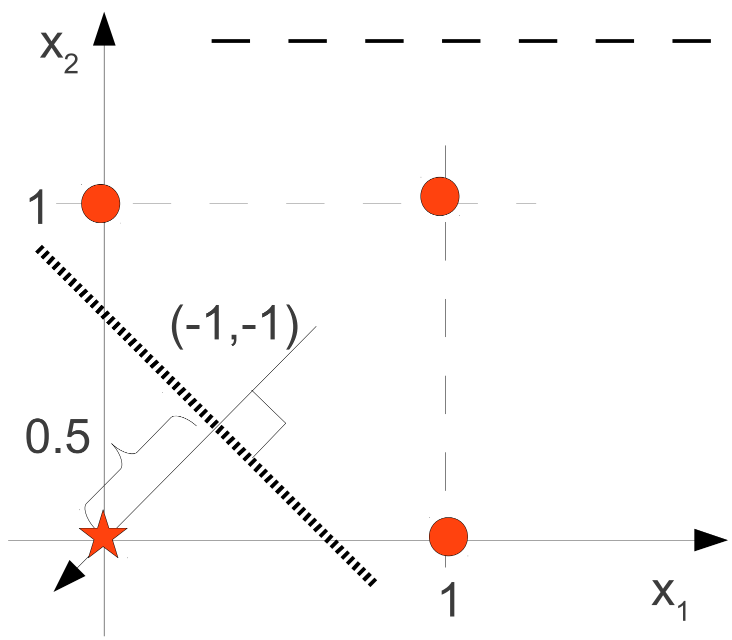


- **Example: OR boolean function**
  - Consider the following mapping:
    - {[(1,1),1],[(0,1),1],[(1,0),1],[(0,0),0]}
  - This can be approximated using a perceptron network with the following parameters:
    - w_1=1; w_2=1; b=0.5
  - The corresponding decision boundary is denoted as shown on left:

Perceptron



- **Example 2: AND boolean function**
  - Consider the following mapping:
    - {[(1,1),1],[(0,1),0],[(1,0),0],[(0,0),0]}
  - This can be approximated using a perceptron network with the following parameters:
    - w_1=1; w_2=1; b=1.5
  - Decision boundary is denoted as shown on left
    - note that the weight vectors remain the same, only the bias term has changed, resulting in a shift in the decision boundary along the direction of w



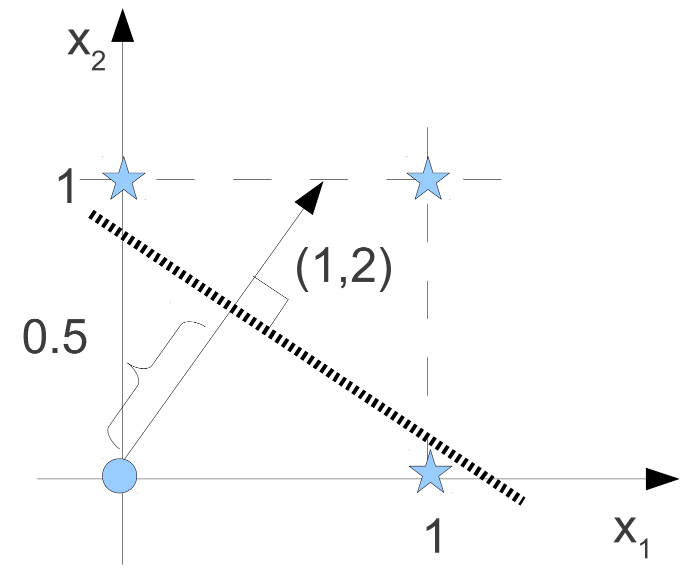
- **Example 3: NOT boolean function**
  - NOT logic function:
    - {[(1,1),0],[(0,1),0],[(1,0),0],[(0,0),1]}
  - Corresponding perceptron network has the following parameters:
    - w_1=-1; w_2=-1; b=0.5
  - Decision boundary shown on left. Note the reversal in the direction of w

Perceptron training

- (OR function revisited)
- Two possible situations:
  - There are no errors:
    - Perceptron is trained → terminate learning
  - There are errors, initiate “perceptron learning”:
    - Situation is as depicted on right – point (1,0) is misclassified; $w_1=2$, $w_2=1$
  - Basic intuition:
    - For the points that are correctly classified, do nothing
    - For points that are wrongly classified, need to update these points
    - Update should be based on the misclassified point
- General form which satisfies these requirements:

$$\boldsymbol{w}(n+1)=\boldsymbol{w}(n)+\eta(t-y)\boldsymbol{x}$$

- This expression is known as the ***Perceptron Learning Rule***
- It can be seen that:
  - when the output matches the target, no update is performed
  - Let's examine what happens in event of mismatch..



Perceptron training (cont'd)

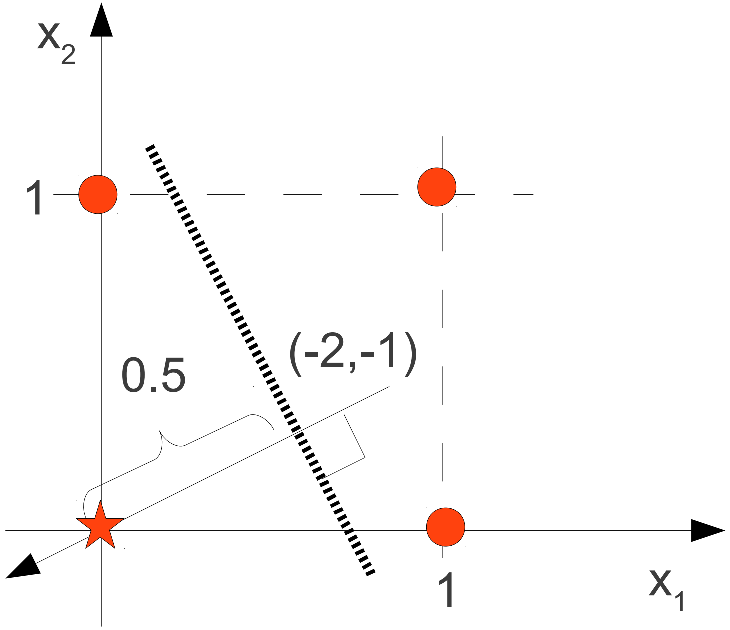
- **In this case, this expression gives us:**

$$\boldsymbol{w}(n+1)=\boldsymbol{w}(n)+\eta(t-y)\boldsymbol{x}$$
$$=\begin{pmatrix}1\\2\end{pmatrix}+\eta\cdot(1-0)\cdot\begin{pmatrix}1\\0\end{pmatrix}$$

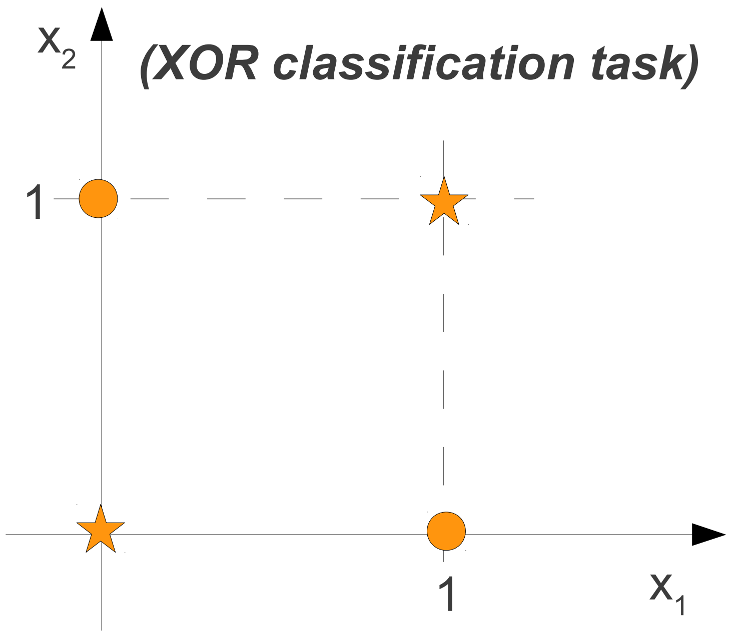
- **η is a *learning rate* constant**
  - (not necessarily constant but let's assume it is!)
  - Determines the speed at which the weights are adjusted – typically set heuristically
  - Regardless of the actual value, this update clearly moves *w* towards the desired value
- **NOT operation example →**
  - misclassified point is (0,1)
  - Update term is:

$$\boldsymbol{w}(n+1)=\boldsymbol{w}(n)+\eta(t-y)\boldsymbol{x}$$
$$=\begin{pmatrix}-2\\-1\end{pmatrix}+\eta\cdot(0-1)\cdot\begin{pmatrix}0\\1\end{pmatrix}$$

  - i.e. result is weight get's closer to desired value of (-2,-2)

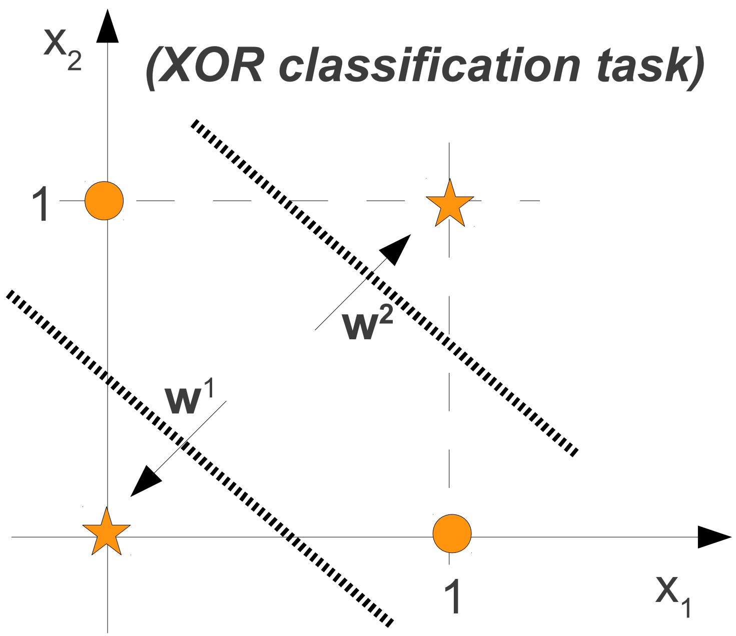


Perceptron training (cont'd)



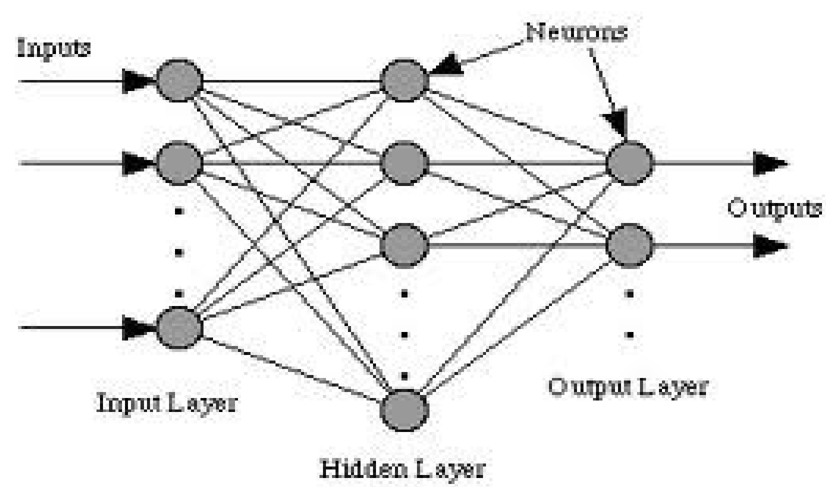
- **Perceptron "neural network" is plagued by a number of shortcomings:**
  - Really only suitable for very simple problems
  - Training algorithm does not lead to good separation of patterns
    - Resulting from binary output
  - Perceptron learning algorithm is based on ad-hoc intuition
    - difficult to improve!
  - Only works with problems that are *linearly separable*
    - AND, OR and NOT are ok
    - XOR (depicted left) is a simple classification problem, but not soluble using perceptrons

Multi-Layer Perceptrons



- XOR can be solved by combining two separate decision boundaries as shown (**w**^{1} and **w**^{2})
- The outputs of the individual perceptrons then become an "OR" problem (linearly separable).
- The resulting architecture is known as a "multi-layer" perceptron (for obvious reasons)
- Frequently referred to as an "MLP"

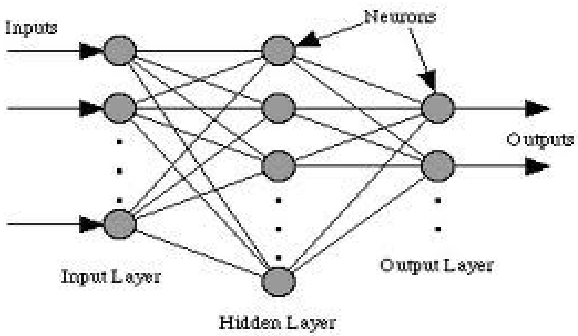
- MLP structure shown on right →
- In present situation, we would require
  - Single *hidden layer*
  - Two *hidden units*
- In general, MLPs can have as many hidden layers as required, however..
  - A single layer is capable of approximating arbitrary functions



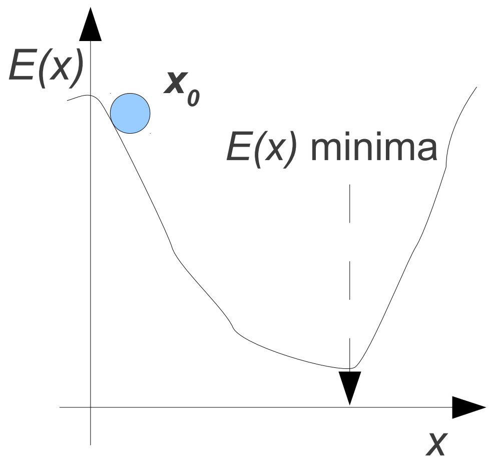
Multi-Layer Perceptrons (Cont'd)

- **Next problem – how to train an MLP**
- **Perceptron learning rule is not suitable → no way to determine the errors at the internal layers to perform updates**
  - What is needed is a transfer function which is “smooth” (i.e. differentiable)
    - allows errors at the output layer to be propagated back to the internal layers
  - Also, probabilistic interpretation for output function would be nice!

    → Logistic function!
- **Note: MLPs are suitable for both regression and classification**
  - Difference is in the transfer function
  - Logistic function is the choice method for classification, but for regression, MLPs with linear outputs can be used
  - (the main requirement is that the function is differentiable)
- **Basic principle for training:**
  - Start with a randomly chosen initial weight value
  - Determine suitable change in weight value which would result in reduction in the network error
  - Update weights, iterate until convergence...



Gradient Descent

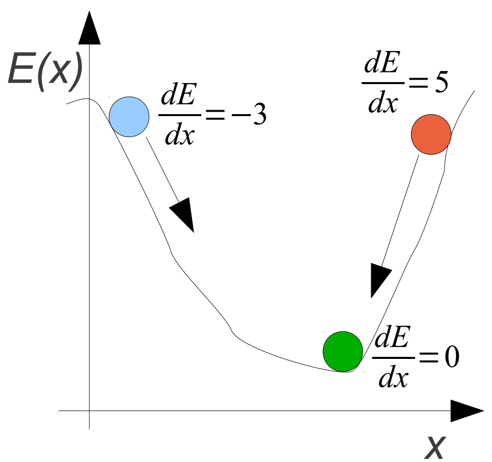


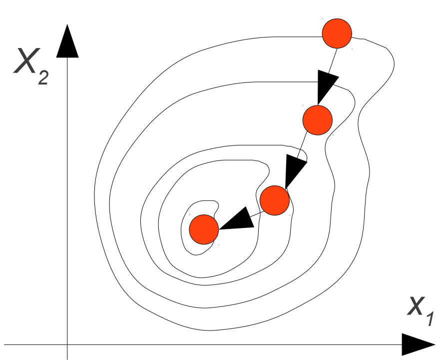
- **Gradient descent is an Iterative parameter optimization technique**
  - Start with initial value, $x_0 \rightarrow$
  - The goal is to find the minimum in the function *E(x)* (the “error” or “objective” function)
  - At each iteration, need an update term which “improves” on current value of *x*

- Suitable *direction* and *magnitude* of update can be found via the gradient of the error function at a given point.
- Update is of the form:

$$x(n+1)=x(n)-\frac{\eta \cdot dE(x)}{dx}$$

- Termination of the learning can be found when gradient equals zero





- For multivariate ***x***, same procedure can be applied but by using the *vector gradient*:

$$\frac{dE(\boldsymbol{x})}{d\,\boldsymbol{x}}=\left(\frac{\partial E}{\partial x_1},\frac{\partial E}{\partial x_2}\right)$$

- As before, η is a learning rate parameter which is set heuristically

Masdar INSTITUTE

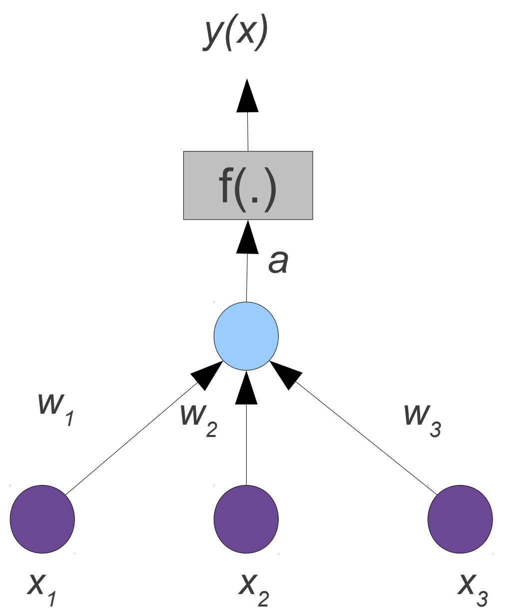
Gradient Descent (single layer)

- **The gradient descent procedure describe previously translates directly to NN context**
  - For single layer case:

$$E(w) = \sum_{i=1}^{n} \left[y(x(i)) - t_i \right]^2$$
$$= \sum_{i=1}^{n} \left[f\left(\sum_{j=1}^{3} w_j x_j(i) \right) - t_i \right]^2$$

  - For each w_i, we can find the partial derivative w.r.t. the error function thus:

$$\Rightarrow \frac{\partial E(w)}{\partial w_k} = \sum_{i=1}^{n} 2 \cdot \left[y(x(i)) - t_i \right] \cdot f'(a) \cdot x_k$$



- **Note the similarity between this and the perceptron rule**
  - Only difference is the *f'(.)* term
  - General principle: if there is an error, and this input is large, *change the weight!*
  - Known as **Hebbian Learning**

Backpropagation

- **For the MLP, the error is simply “propagated” back through the network layers:**

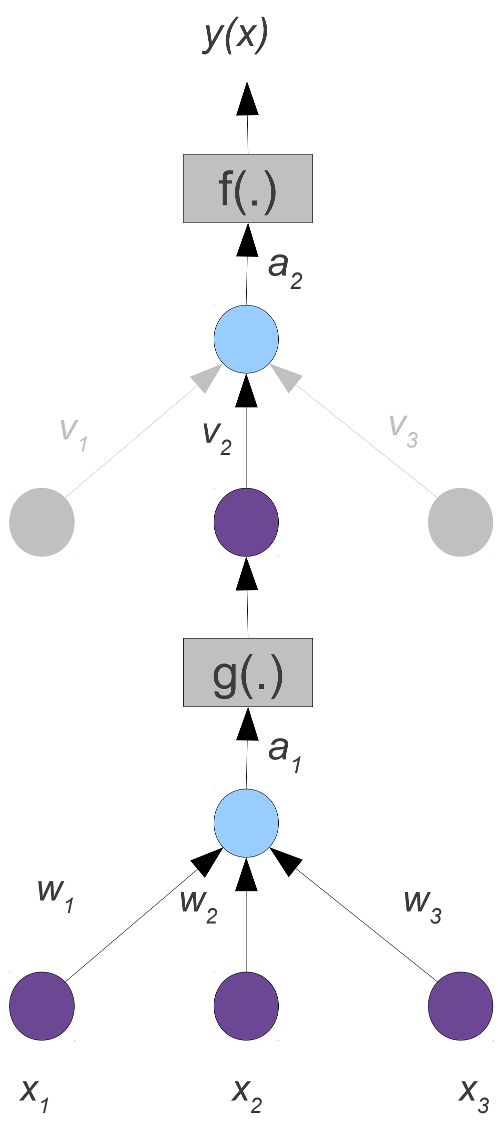
$$E(w)=\sum_{i=1}^{n}\left[y(x(i))-t_i\right]^2$$
$$=\sum_{i=1}^{n}\left[f\left(\sum_{j=1}^{3}v_j g\left(\sum_{k=1}^{3}w_{jk}x_k(i)\right)\right)-t_i\right]^2$$

- As before, gradient calculations through chain rule:

$$\Rightarrow\frac{\partial E(w)}{\partial w_k}=\sum_{i=1}^{n}2\cdot\left[y(x(i))-t_i\right]\cdot f'(a_2)\cdot v_2\cdot g'(a_1)\cdot x_k$$

- Just one thing left, finding those pesky *f'(.)* and *g'(.)*'s?
  - Depending on transfer function.. most commonly used ones are linear (*f'(.)*=1) and logistic.
  - For logistic function:

$$\frac{df}{dx}=\frac{d\left[\frac{1}{1+e^{-x}}\right]}{dx}=\frac{e^{-x}}{(1+e^{-x})^2}$$



Masdar INSTITUTE