3.3 Gradient Methods

Suppose this parameter has a non-linear relationship with the output:

$$\vec{\theta}$$

For example (gaussian function),

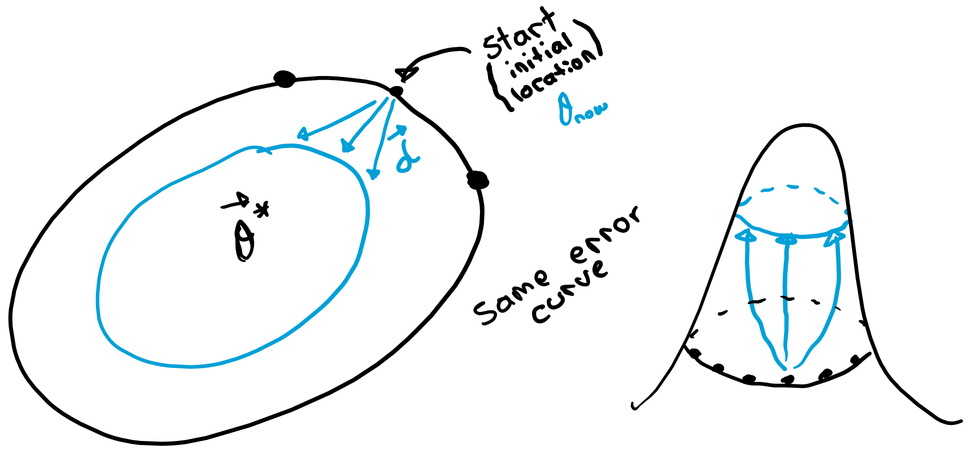
$$\mu_A(x) = e^{\frac{-(x-\mu)^2}{\sigma^2}}$$

For this function, the relationship between μ and σ and the MF μ_A is non-linear, and by association, the error function E.

Objective Function $\quad \vec{\theta} = [\theta_1, \theta_2, \dots, \theta_n]^T$

Error Function $\quad E(\vec{\theta})$

Looking for optimal $\vec{\theta}^*$



$$\vec{\theta}_{next} = \vec{\theta}_{now} + \eta\vec{d}$$

$\eta = \text{step}$
$\vec{d} = \text{direction vector}$

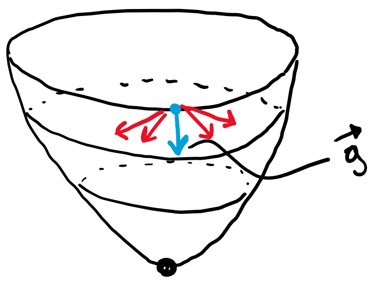
k^{th} step:

$$\vec{\theta}_k$$

$(k+1)^{th}$ step:

$$\vec{\theta}_k + \eta_k\vec{d}_k$$

Generally, $E(\vec{\theta}_{k+1}) \leq E(\vec{\theta}_k)$



(the optimal direction) ... the steepest direction

Steepest-gradient descent method:

$$\vec{g}(\vec{\theta}) = \begin{bmatrix} \frac{\partial E}{\partial \theta_1} & \frac{\partial E}{\partial \theta_2} & \cdots & \frac{\partial E}{\partial \theta_n} \end{bmatrix}^T$$

Let $\vec{g}(\vec{\theta}) = \frac{\partial E(\vec{\theta})}{\partial \hat{\vec{\theta}}}\Big|_{\hat{\vec{\theta}}} = 0$

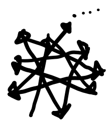
$$\vec{\theta}_{k+1} = \vec{\theta}_k - \eta_d \vec{g}$$

negative gradient direction

In general, this is a recursive (or repetitive) algorithm.

- Stopping criteria:

1) $E \leq$ threshold: 10^{-5}
2) # of iterations: ≤ 200

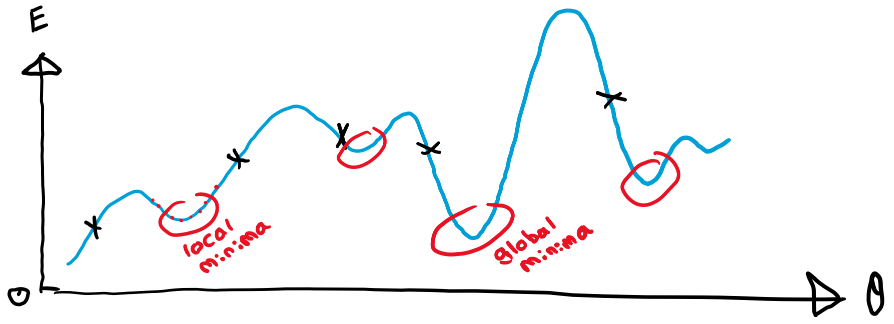


... no convergence (η too large)

3.4 Genetic Algorithms (GA)

Easy to read, does not involve gradient related operations, it is a derivative free method, but it takes a very long time to execute.

It can reach the global minimum, but all other derivative-based methods reach a local minimum (depending on initial conditions):

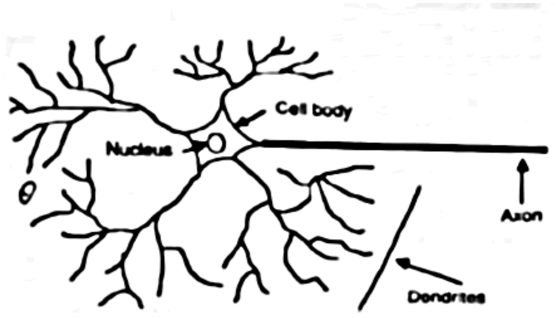


- Population-based search, so it returns the *best* results
- But it’s <u>very</u> time consuming (like, 8 hours)
- Compare with gradient-based method (15 seconds)
- Evolution operation
    - Reproduction, cross over, mutation

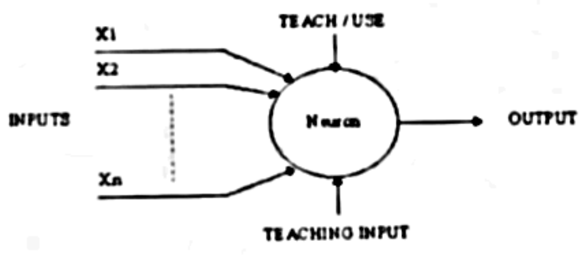
Chapter 4: Artificial Neural Networks

4.1 Introduction

Neuron networks: parallel and distributed neurons.



Artificial neural networks:

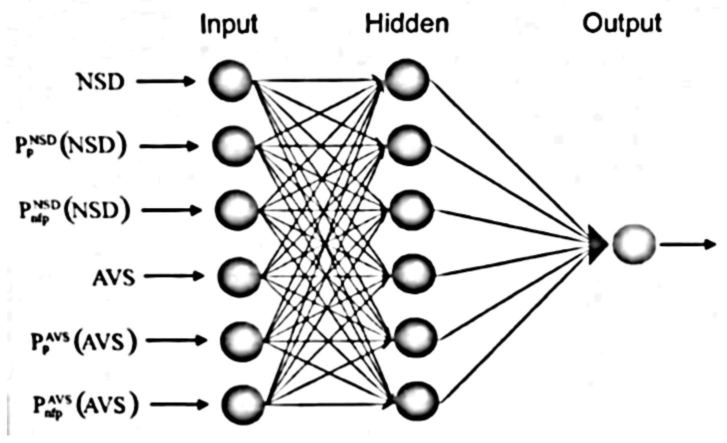


4.2 Features of Neural Networks

Layered neurons

Weighted links

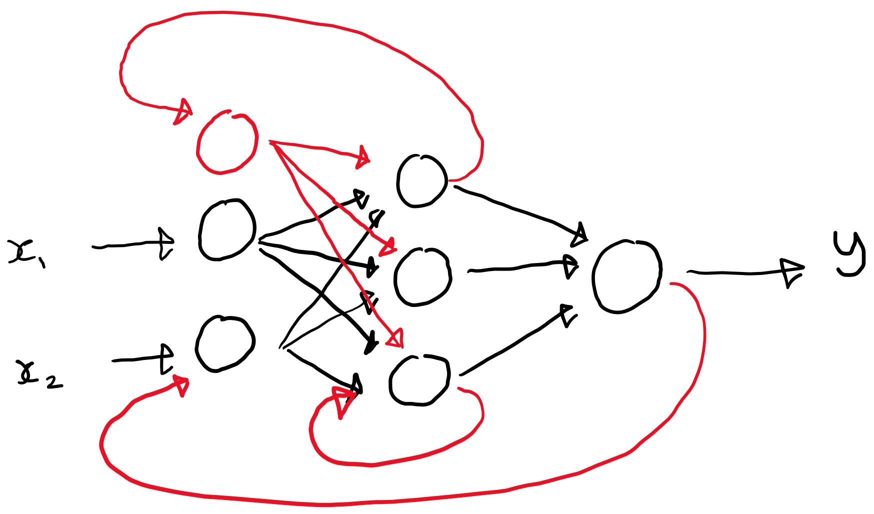
(link weights 0~1.0)



1) Neural network topologies

(a) Feed forward topology (static neural network)

unidirectional links (just move in one direction, in this case from input to hidden nodes)



(b) Recurrent topology (dynamic neural network)

Outputs can move to back into itself, can mode into a different node… they can move anywhere depending on requirements.

Much more complicated, but has some distinct advantages, specifically in terms of access to historical data:

Consider:

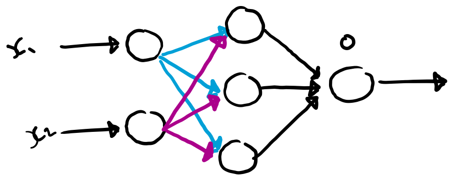
- k^{th} step: $x_1(k),\ x_2(k)$
  Output: $y(k)$
- $(k+1)^{th}$ step: $x_1(k+1),\ x_2(k+1)$
  Historical information $\rightarrow y(k)$
  Output: $y(k+1)$

4.2 Features of Neural Networks (Cont'd)

1) NN Topologies

- FF NNs

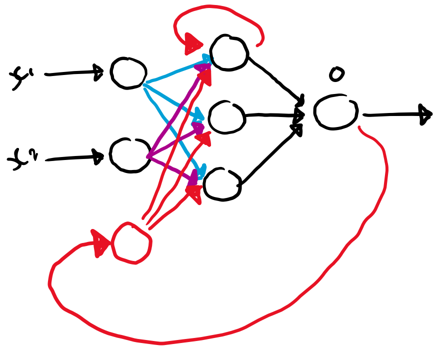
Simple in structure, consider two inputs and one output below:



Connections between intermediate neurons and output neurons are unidirectional.
Static modeling method (given a set of inputs, an output is generated).

- Recurrent NNs

Similar in structure, but can have feedback links.

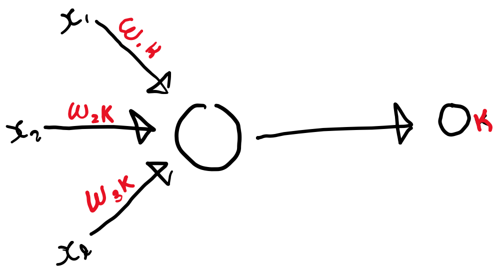


Gives access to historical information, making the network a 'dynamic' network. However, training complexity increases.

$x(k) = O(k)$
Has access to inputs x_1 and x_2

$x(k+1) = O(k+1)$
Has access to inputs x_1, x_2, and historical data $x(k)$

2) Activation functions



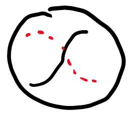
Input: $x_1 w_{1k} + x_2 w_{2k} + \cdots + x_l w_{lk}$

$$\sum_{i=1}^{l} x_i w_{ik}$$

$$O_k = f\left(\sum_{i=1}^{l} x_i w_{ik} - \theta_k\right)$$

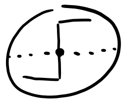
Where:
f = activation function
θ_k = threshold of the kth neuron (bias)

Sigmoid function:



$$sig(x) = \frac{1}{1+e^{-x}}$$

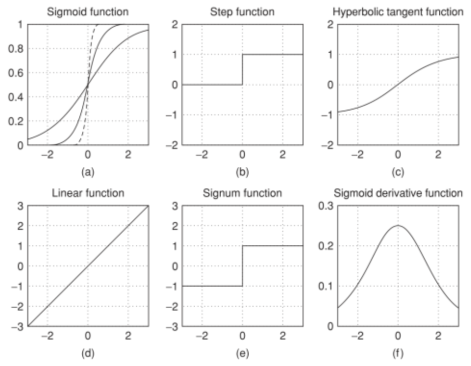
Signum function:



$$sgm(x) = \begin{cases} 1 & ; \quad x > 0 \\ 0 & ; \quad x = 0 \\ -1 & ; \quad x < 0 \end{cases}$$

Step function:

$$step(x) = \begin{cases} 1 & ; \quad x > 0 \\ 0 & ; \quad x \leq 0 \end{cases}$$



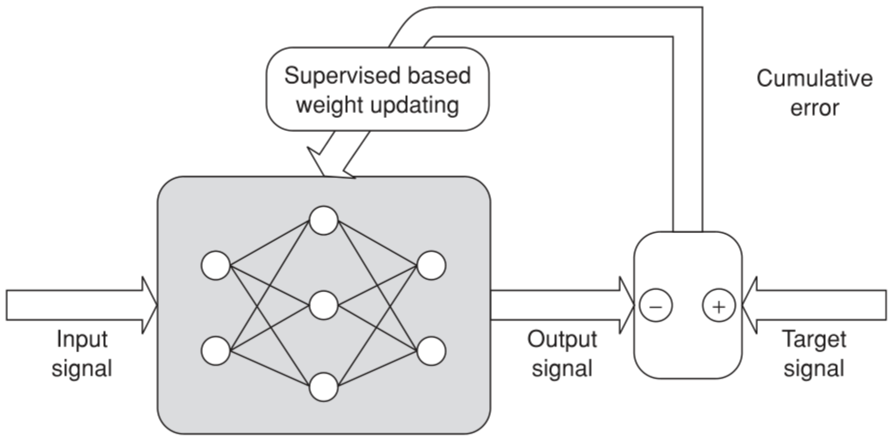
3) Neural Network Learning

- Supervised learning

We have a desired output, which can be considered a teacher.

Teacher $(\vec{x}, t)$

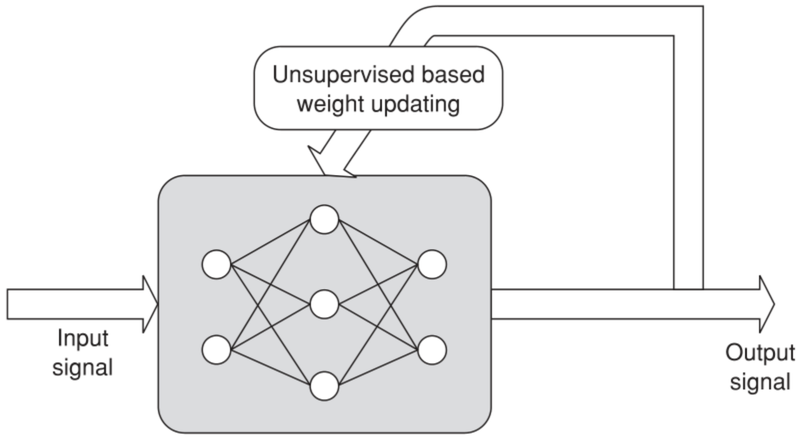
We compare the desired output and the calculated output and feed the error information back into the system to train it. This is the general approach for engineering applications.



- Unsupervised learning

In applications where we can’t get a target, or we can’t find the desired output, we have no teacher.

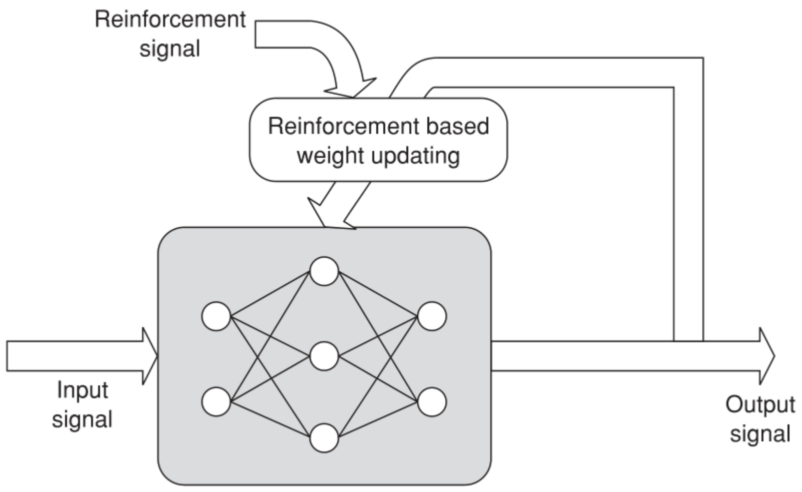
Thus, we cannot do supervised learning – this is usually the case for ‘big data’.



- Reinforcement learning

Feedback information provides a guide for training, but not a target.

Can be used for special circumstances like scenarios in video games (i.e. beating a bad guy in fewer moves to achieve a higher bonus)



Additional info:

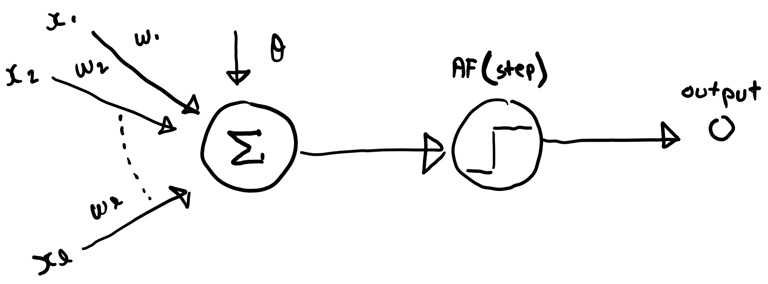
ANFIS – adaptive neuro-fuzzy inference system

Our course will focus on the following engineering applications:

- Control
- Classification (diagnosis)
- Modeling (forecasting)

4.4 Connectionist Modeling

1) McCulloch-Pitts (MP) Modeling

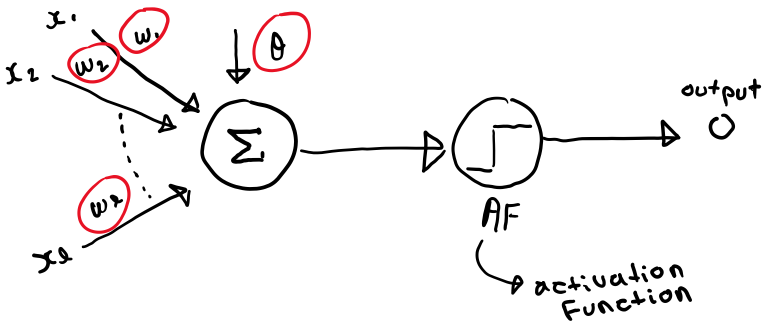


$w_1, w_2, \ldots, w_l$ are fixed

$$O = f(x_1 w_1 + x_2 w_2 + \cdots + x_l w_l - \theta)$$
$$= f\left(\underbrace{\sum_{i=1}^{l} x_i w_i - \theta}_{\text{threshold}}\right)$$

Step: $\sum x_i w_i = 0.001 \quad ; \quad 0 \rightarrow 1$

2) Perceptron Modeling

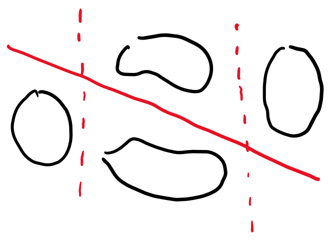


Train link weights $(w_1, w_2, \ldots, w_l)$ and the bias (θ), but we don't train the activation function parameters, it is fixed – we simply choose one.

$$O = f\left(\sum_{i=1}^{l} x_i w_i - \theta\right)$$

If the training data pairs are linearly separable (or separable by hyper planes) then the training process can converge.

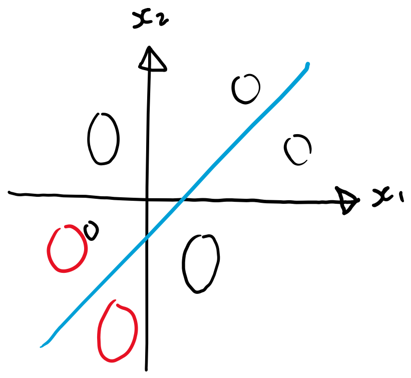
Meaning we can get optimal parameters by a finite number of training operations.



Consider 2-D data sets:

x_1, x_2

$w_1 x_1 + w_2 x_2 - \theta = 0$



Summary of Perceptron training algorithm:

1. Initialize weights and thresholds to small random values
2. Choose an input-output pattern $(x^{(k)}, t^{(k)})$ from the training data.
3. Compute the network's actual output $o^{(k)} = f\left(\sum_{i=1}^{j} w_i x_i^{(k)} - \theta\right)$.
4. Adjust the weights and bias according to the Perceptron learning rule:
   $\Delta w_i = \eta\left[t^{(k)} - o^{(k)}\right] x_j^{(k)}$, and $\Delta\theta = -\eta\left[t^{(k)} - o^{(k)}\right]$, where $\eta \in [0, 1]$ is the Perceptron's learning rate.
   If f is the signum function, this becomes equivalent to:

$$\Delta w_i = \begin{cases} 2\eta t^{(k)} x_j^{(k)} & ; \quad \text{if } \mathrm{t}^{(k)} \neq o^{(k)} \\ 0 & ; \quad \text{otherwise} \end{cases}$$

$$\Delta\theta = \begin{cases} -2\eta t^{(k)} & ; \quad \text{if } \mathrm{t}^{(k)} \neq o^{(k)} \\ 0 & ; \quad \text{otherwise} \end{cases}$$

5. If a whole epoch is complete, then pass to the following stepl otherwise go to Step 2
6. If the weights (and bias) reached steady state ($\Delta w_i \approx 0$) through the whole epoch, then stop the learning; otherwise go through one more epoch starting from Step 2.s