CS559 Lecture 12: Gaussion Process

Reading: Chapter 6, Bishop book

Gaussian Process

- Extension of kernels to probabilistic discriminant models
- Define a prior probability distribution over the functions directly.
- Only need to consider function values at training and test data sets – work in finite space.
- Kriging, ARMA, Kalman filters, RBFs are all Gaussian Processes.
- http://www.gaussianprocess.org/

Gaussian Stochastic Process

Consider linear model

$$y(x) = w^T \phi(x)$$

with prior over w

$$p(w) = \mathcal{N}(w|0, \alpha^{-1} I)$$

Now, if $\mathbf{y}$ is vector of samples from a stochastic process, then it is a Gaussian stochastic process with

$$\begin{aligned} E[\mathbf{y}] &= \Phi E[w] = 0 \\ E[\mathbf{y}] &= \Phi E[ww^T]\Phi^T = \frac{1}{\alpha}\Phi\Phi^T = K \end{aligned}$$

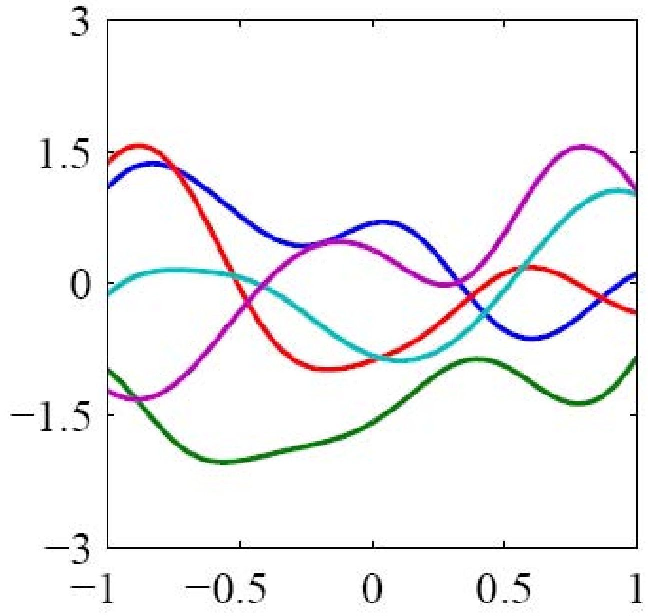
where K is the Gram matrix

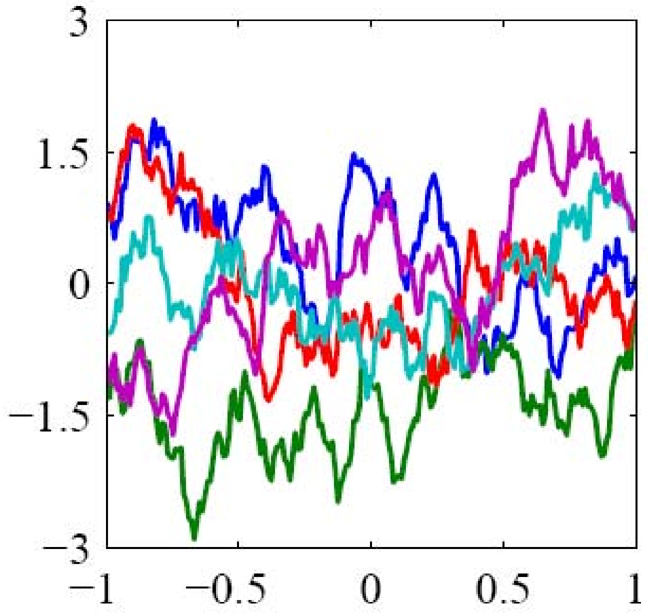
$$K_{nm} = k(x_n, x_m) = \frac{1}{\alpha}\phi(x_n)^T \phi(x_m)$$

Gaussion process

- Gaussion process is a probability distribution over function $y(x)$ such that the values of y’s at points x1,…x_N have a joint Gaussion distribution.

- 2D -> Gaussian random field.





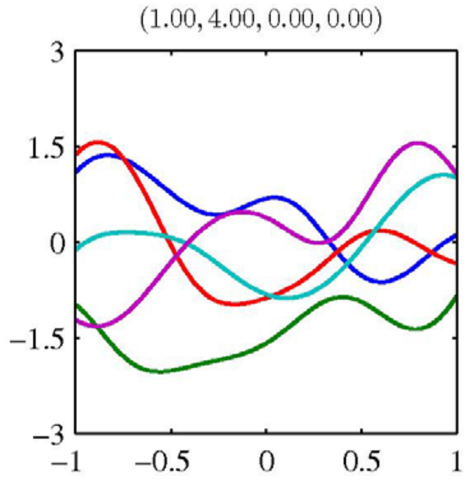
$$\begin{aligned} k_1(x_n, x_m) &= \exp(-||x_n - x_m||^2/2\sigma^2) \\ k_2(x_n, x_m) &= \exp(-\theta|x_n - x_m|) \end{aligned}$$

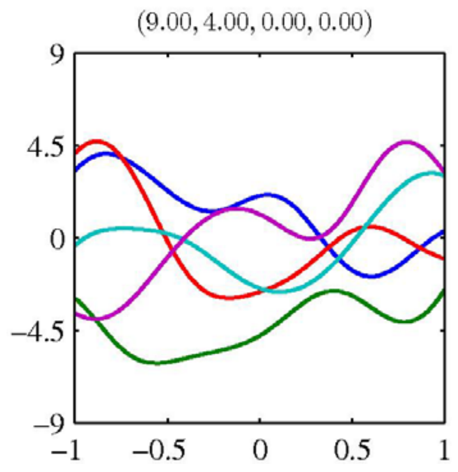
Gaussian stochastic processes are completely defined by the second order statistics!

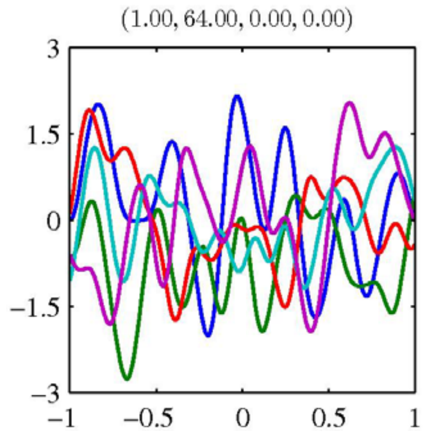
Gaussian Process

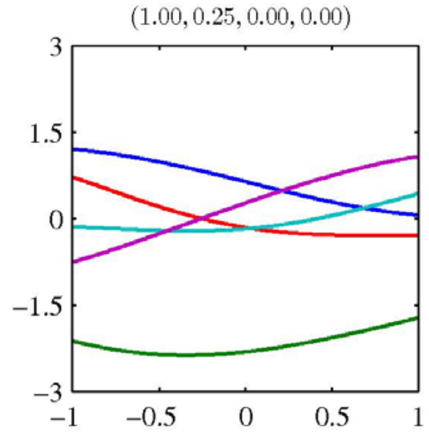
One popular choice of kernel in this case is

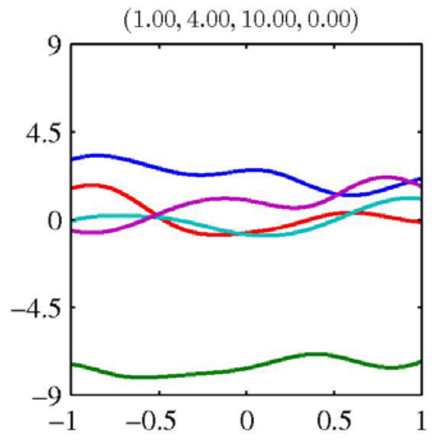
$$k(x_n, x_m) = \theta_0 \exp\left(-\frac{\theta_1}{2}||x_n - x_m||^2\right) + \theta_2 + \theta_3 x_n^T x_m$$

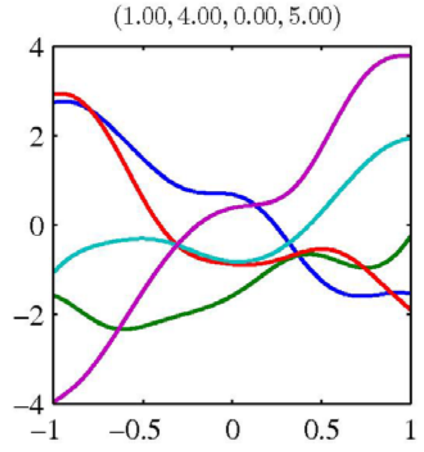












Gaussian Process for Regression

$$t_n = y_n + \epsilon_n, \quad \epsilon_n \sim \mathcal{N}(0, \beta^{-1})$$

Since y_n and ϵ_n are independent,

$$p(\mathbf{t}|\mathbf{y}) = \mathcal{N}(\mathbf{t}|\mathbf{y}, \beta^{-1}\mathbf{I})$$

and we know

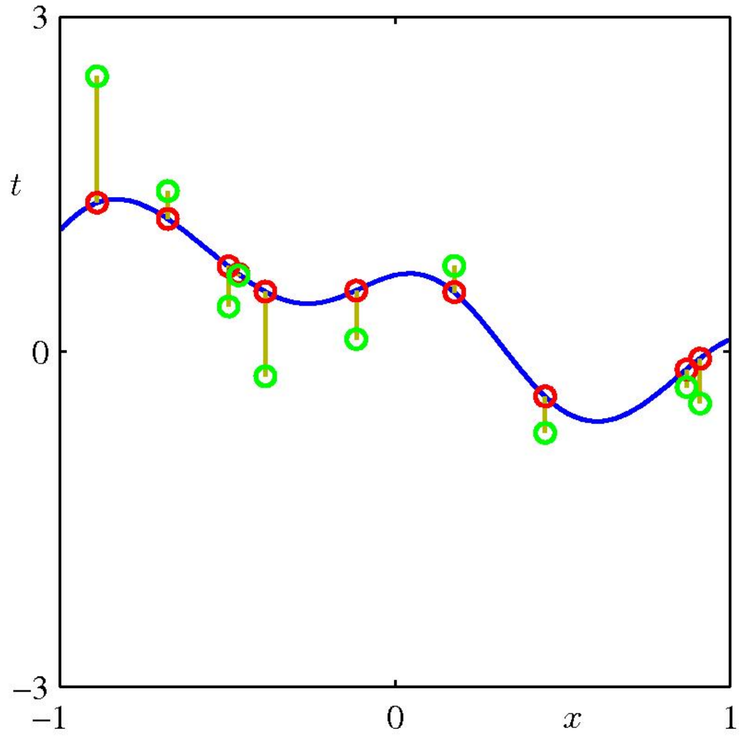
$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|0, K)$$

Therefore

$$p(\mathbf{t}) = \int p(\mathbf{t}|\mathbf{y})p(\mathbf{y})d\mathbf{y} = \mathcal{N}(\mathbf{t}|0, \mathbf{C})$$

where

$$C(x_n, x_m) = k(x_n, x_m) + \beta^{-1}\delta_{nm}$$


3
t
0
−3
−1
0
x
1

Gaussian Process for Regression

For regression, however, we need $p(t_{n+1}|\mathbf{t})$. We start with

$$p(\mathbf{t}_{n+1}) = \mathcal{N}(\mathbf{t}_{n+1}|0, \mathbf{C}_{n+1})$$

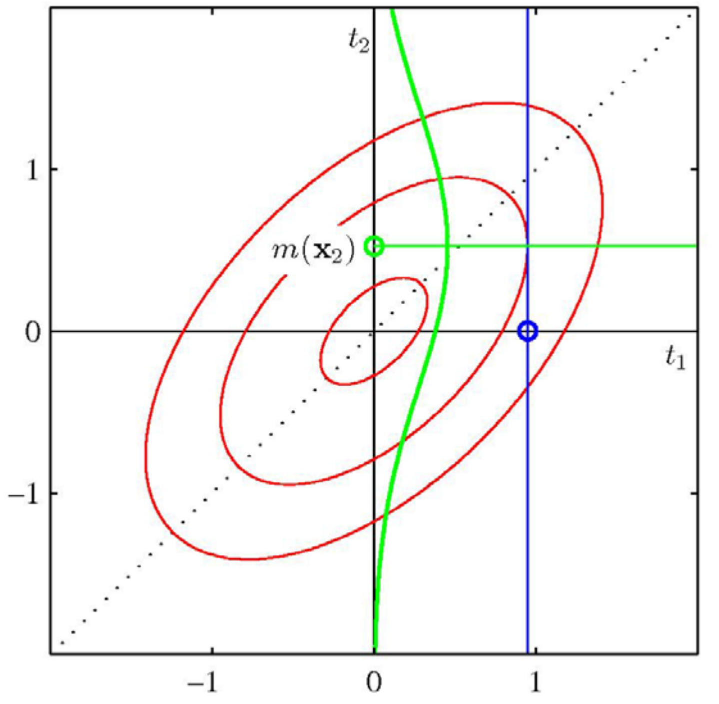
where $\mathbf{C}_{n+1}$ is a $(N+1) \times (N+1)$ covariance matrix.

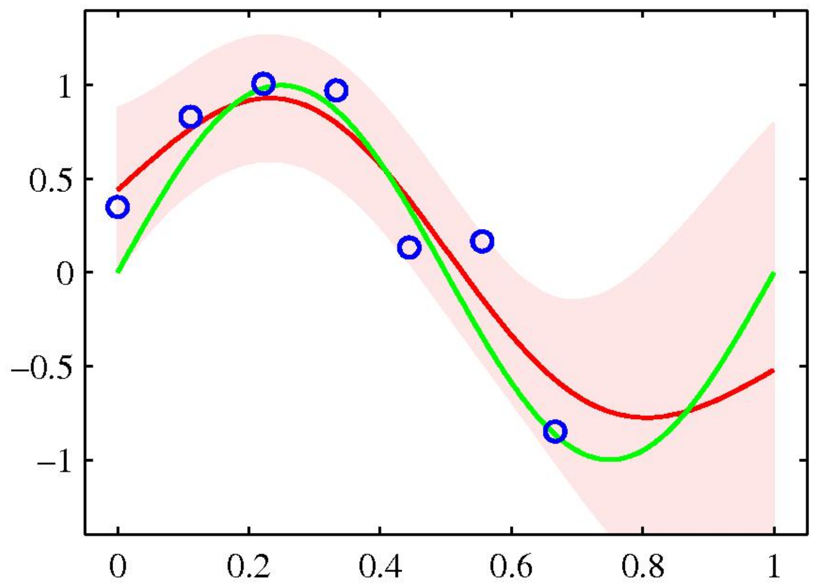
Since the underlying process is a Gaussian stochastic process, you can marginalize the distribution by partitioning the covariance (2.81,2.82), thus

$$\begin{aligned} E[t_{n+1}] &= m(x_{n+1}) = \mathbf{k}^T \mathbf{C}_n^{-1} \mathbf{t} \\ E[t_{n+1} t_{n+1}] &= \sigma^2(x_{n+1}) = c - \mathbf{k}^T \mathbf{C}_n^{-1} \mathbf{k} \end{aligned}$$

$m(x_{n+1})$ can also be written as

$$m(x_{n+1}) = \sum_{n=1}^{N} a_n k(x_n, x_{n+1})$$





If the kernel function is chose as a specific finite set of basis functions, it will lead to the Linear Regression case we talked about before.

Parametric space + linear regresss ←→ functional space + Gaussion Process

Multilayer neural networks

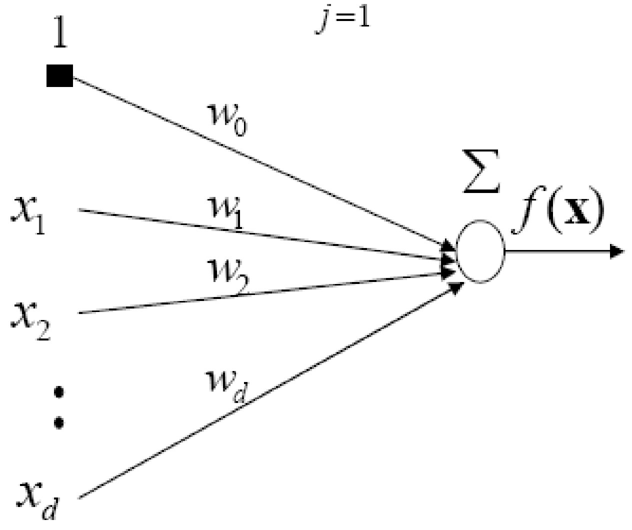
Or another way of modeling nonlinearities for regression and classification problems

Reading: Chapter 5, Bishop book.

Linear units

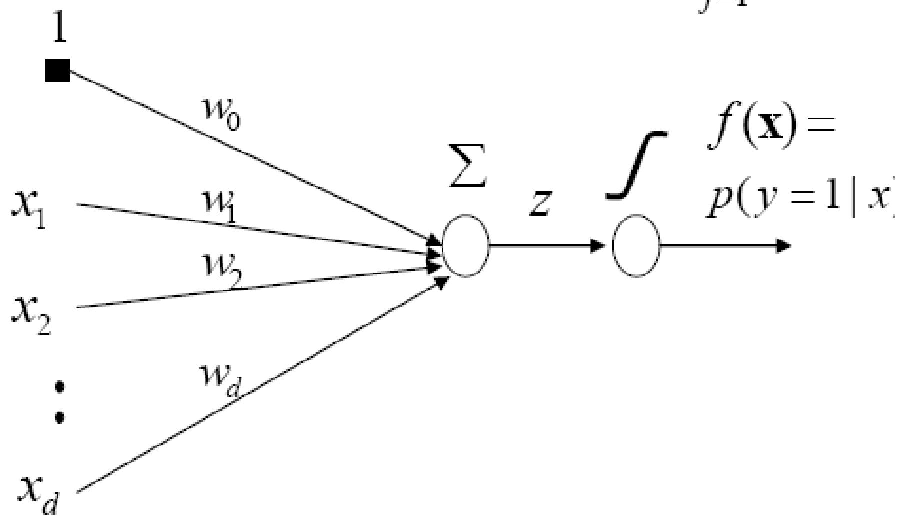
Linear regression

$$f(\mathbf{x}) = w_0 + \sum_{j=1}^{d} w_j x_j$$



Logistic regression

$$f(\mathbf{x}) = p(y=1 \mid \mathbf{x}, \mathbf{w}) = g(w_0 + \sum_{j=1}^{d} w_j x_j)$$



On-line gradient update:

$$w_0 \leftarrow w_0 + \alpha(y - f(\mathbf{x}))$$
$$\vdots$$
$$w_j \leftarrow w_j + \alpha(y - f(\mathbf{x}))x_j$$

The same

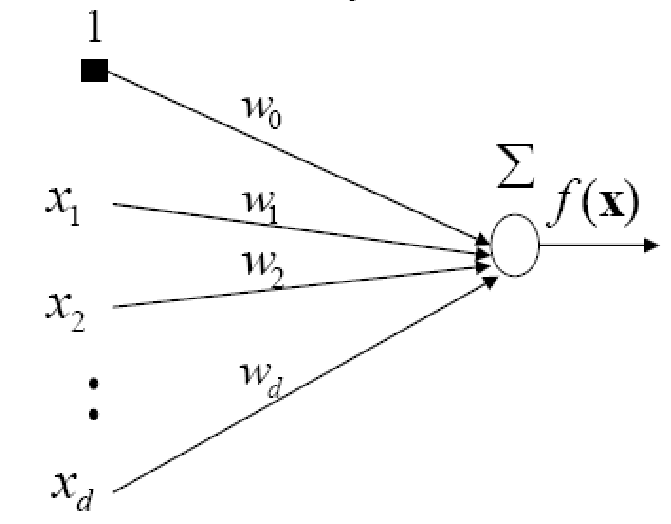
On-line gradient update:

$$w_0 \leftarrow w_0 + \alpha(y - f(\mathbf{x}))$$
$$\vdots$$
$$w_j \leftarrow w_j + \alpha(y - f(\mathbf{x}))x_j$$

Limitations of basic linear units

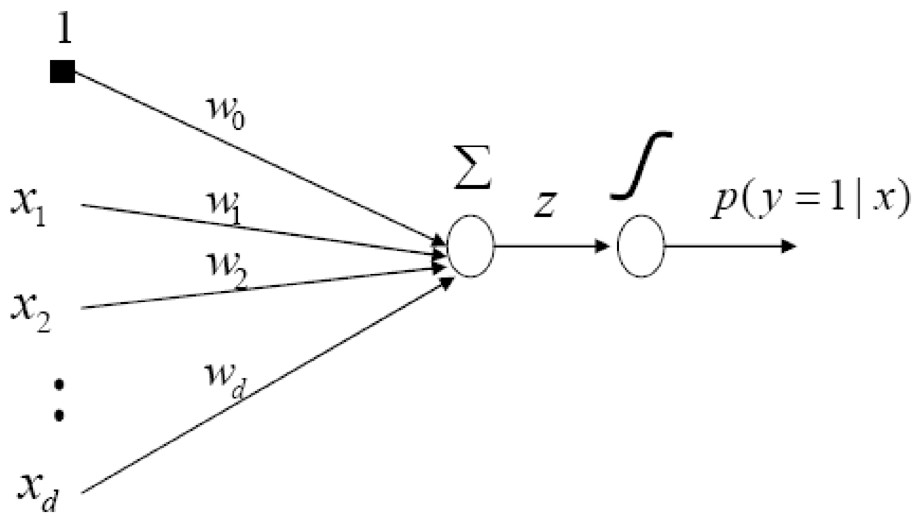
Linear regression

$$f(\mathbf{x}) = w_0 + \sum_{j=1}^{d} w_j x_j$$



Logistic regression

$$f(\mathbf{x}) = p(y=1 \mid \mathbf{x}, \mathbf{w}) = g(w_0 + \sum_{j=1}^{d} w_j x_j)$$



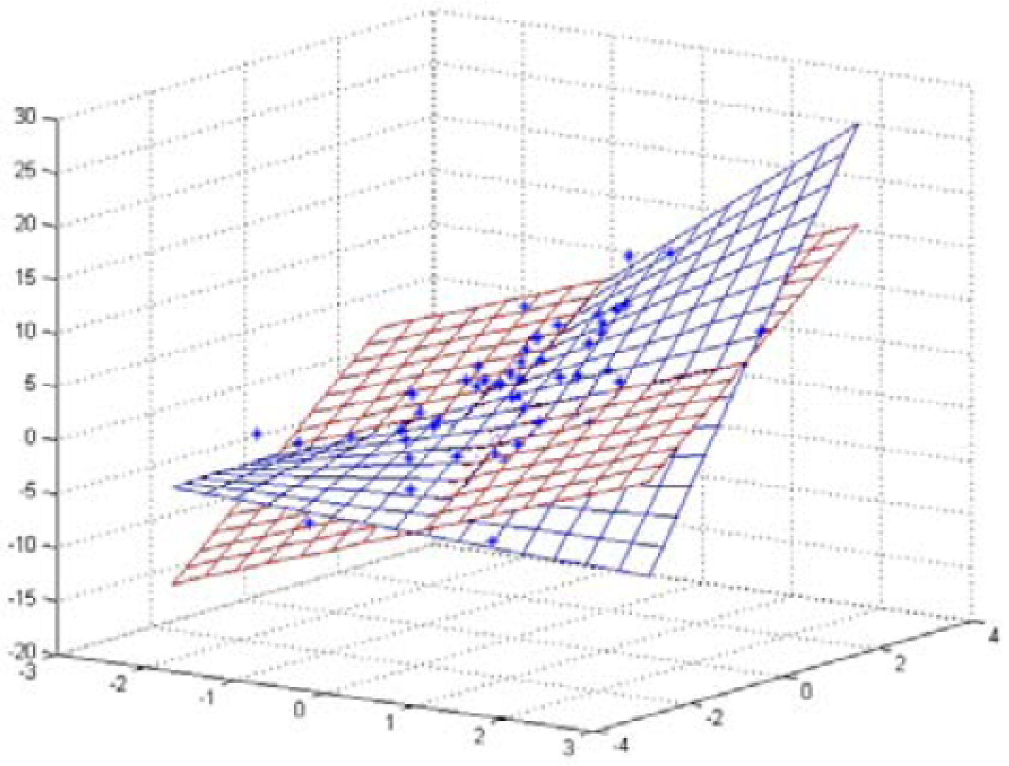
Function linear in inputs !!

Linear decision boundary!!

Regression with the quadratic model.

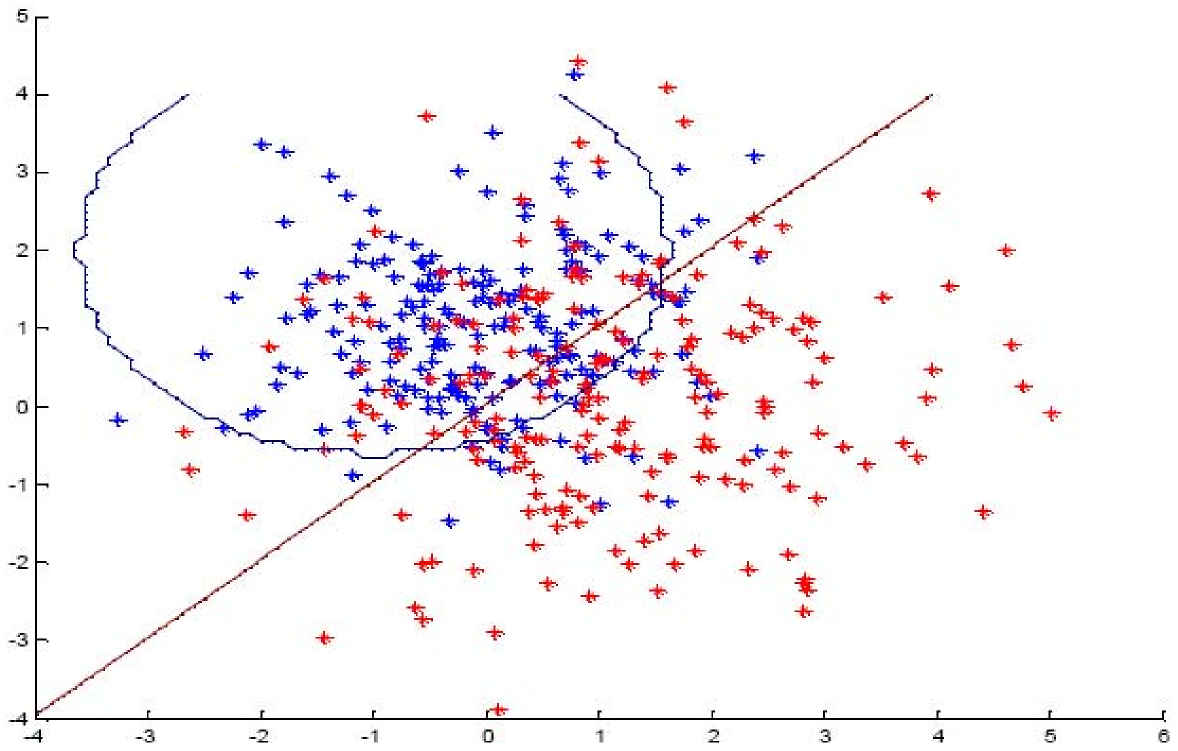
Limitation: linear hyper-plane only

- a non-linear surface can be better



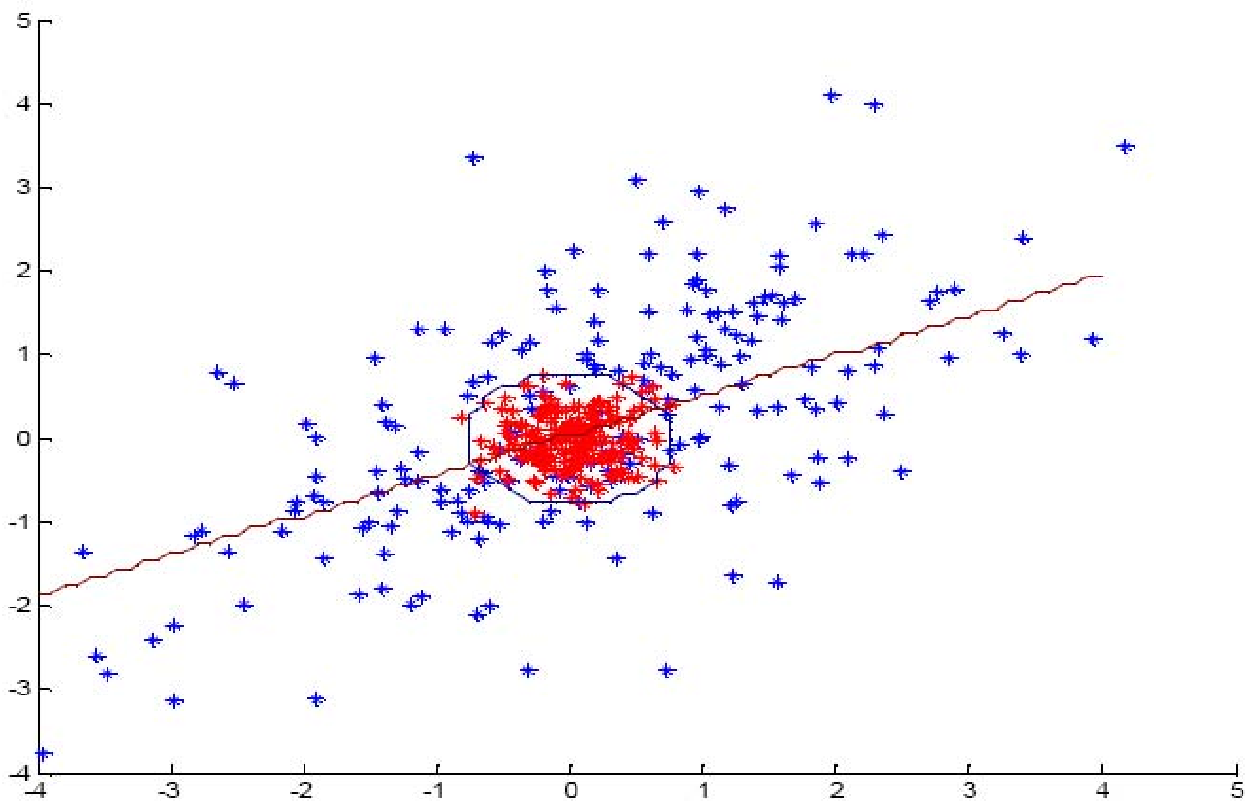
Linear decision boundary

- logistic regression model is not optimal, but not that bad



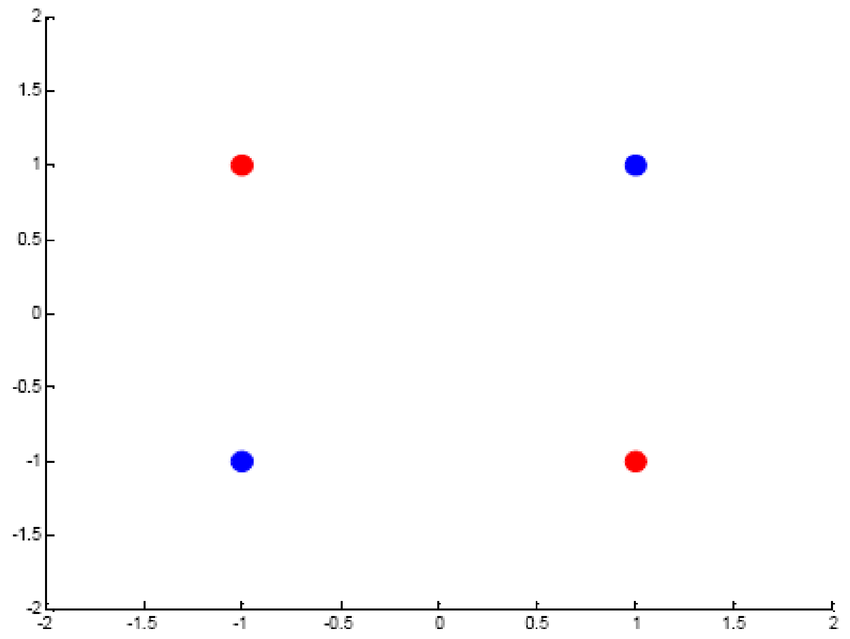
When logistic regression fails?

- Example in which the logistic regression model fails



Limitations of linear units.

- Logistic regression does not work for **parity functions**
  - no linear decision boundary exists



Solution: a model of a non-linear decision boundary

Extensions of simple linear units

- use **feature (basis) functions** to model **nonlinearities**

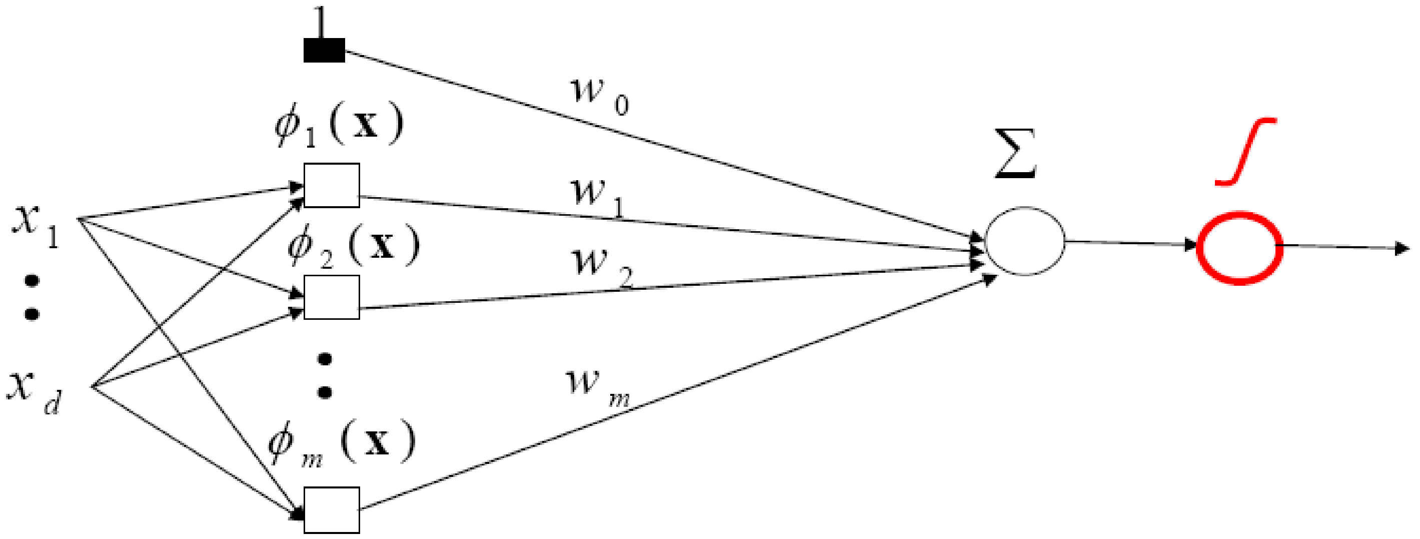
Linear regression

$$f(\mathbf{x}) = w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x})$$

Logistic regression

$$f(\mathbf{x}) = g(w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x}))$$

$\phi_j(\mathbf{x})$ - an arbitrary function of **x**



Learning with extended linear units

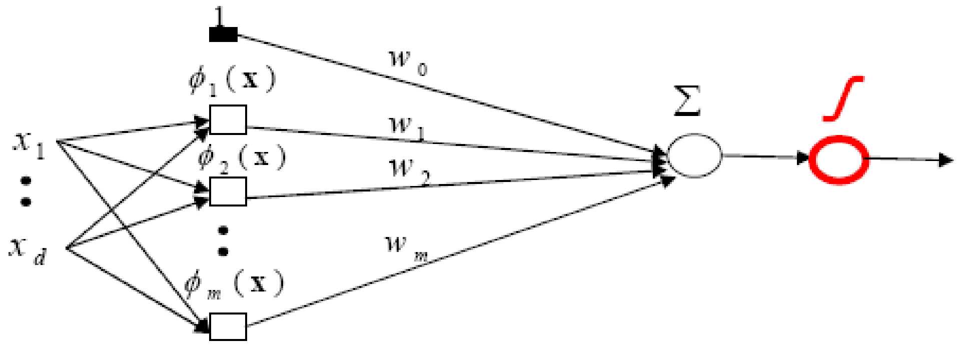
Feature (basis) functions model **nonlinearities**

Linear regression

$$f(\mathbf{x}) = w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x})$$

Logistic regression

$$f(\mathbf{x}) = g(w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x}))$$



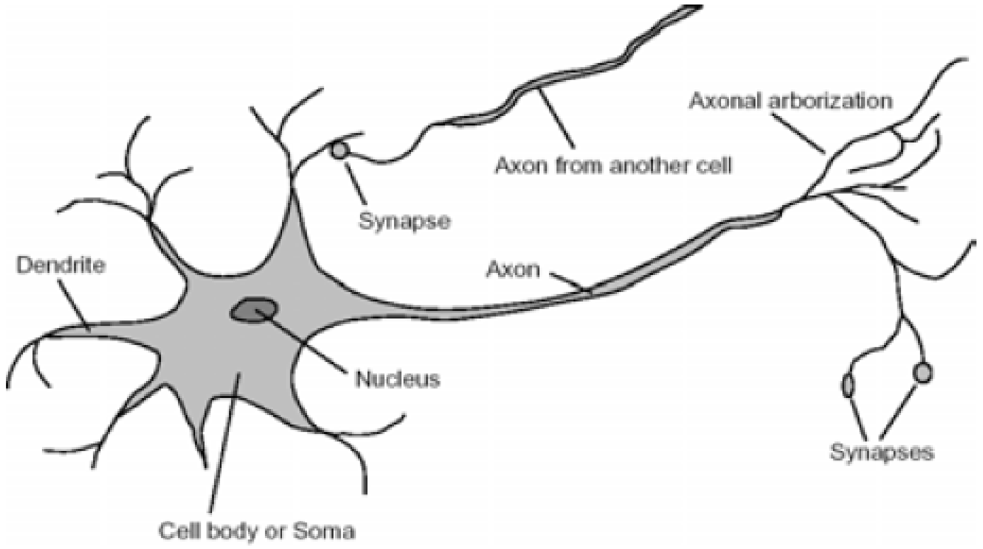
Important property:

- The same problem as learning of the weights for linear units , the input has changed– but the weights are linear in the new input

Problem: too many weights to learn

Multi-layered neural networks

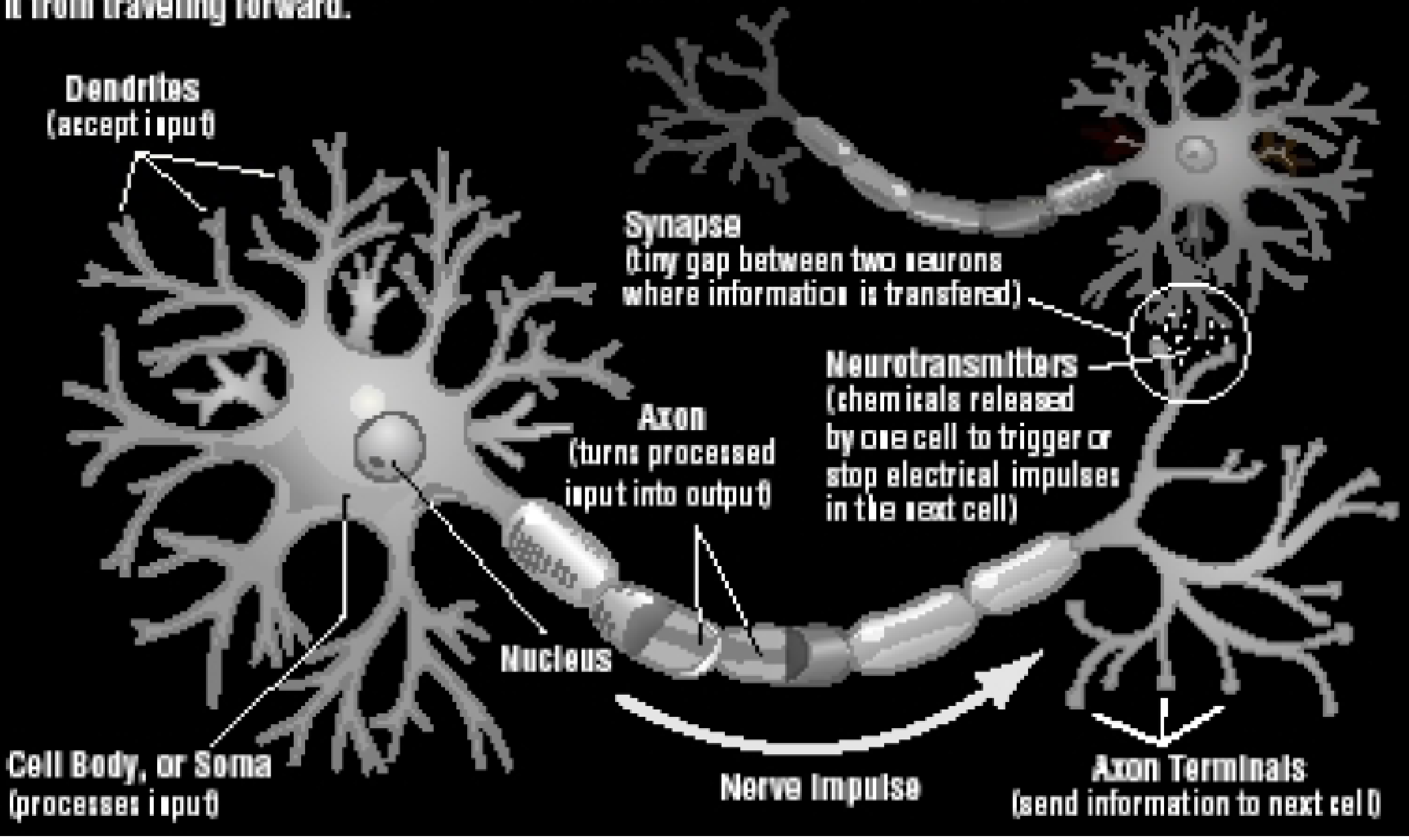
- An alternative way to introduce **nonlinearities to regression/classification models**
- **Key idea: Cascade several simple neural models with logistic units.** Much like neuron connections.



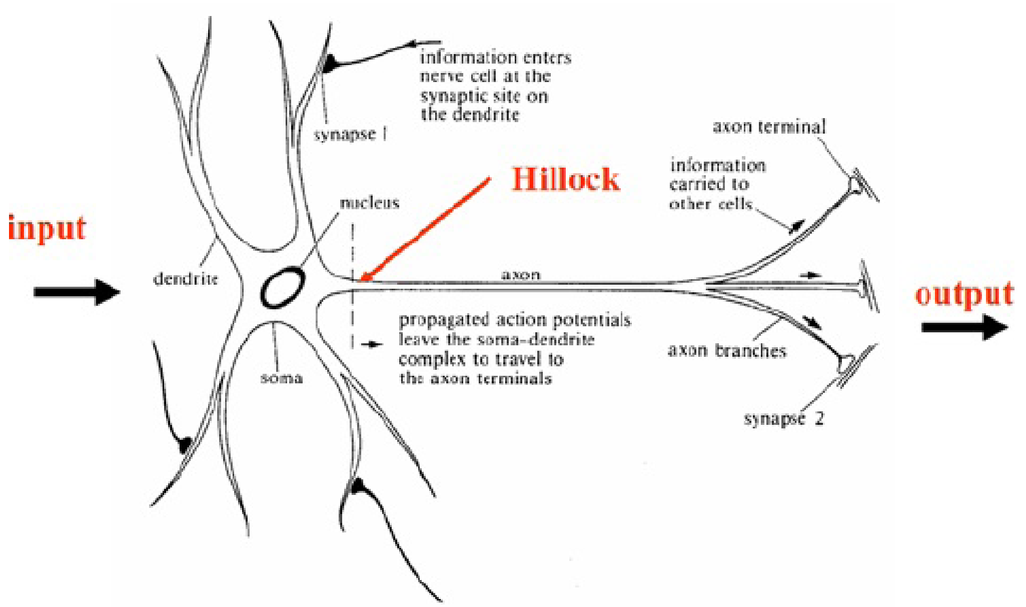
- Perceptrons ≡ nerve cells or neurons
- Network of perceptrons ≡ connections between neurons

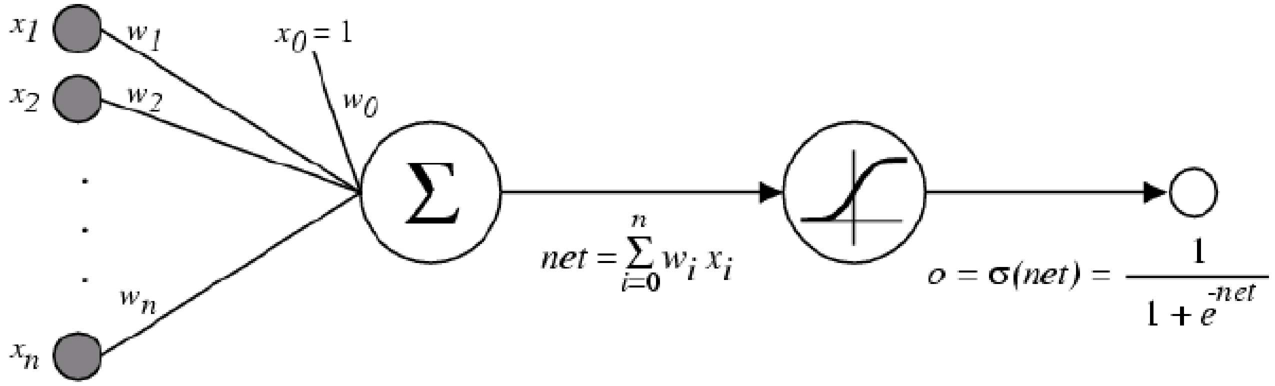
Biological Neural Network

The human brain has approximately 100 billion nerve cells, called neurons, each connected to thousands of others. Senses and thoughts trigger electrical impulses that quickly travel through the neural network. When a neuron receives information, it can send the message on or stop it from traveling forward.


Dendrites (accept input)
Synapse (tiny gap between two neurons where information is transfered)
Neurotransmitters (chemicals released by one cell to trigger or stop electrical impulses in the next cell)
Axon (turns processed input into output)
Nucleus
Cell Body, or Soma (processes input)
Nerve Impulse
Axon Terminals (send information to next cell)

Neurons vs. Perceptrons


information enters nerve cell at the synaptic site on the dendrite
synapse 1
axon terminal
Hillock
information carried to other cells
input
nucleus
dendrite
axon
output
propagated action potentials leave the soma-dendrite complex to travel to the axon terminals
axon branches
soma
synapse 2


x_1
w_1
$x_0 = 1$
x_2
w_2
w_0
Σ
w_n
x_n
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1 + e^{-net}}$

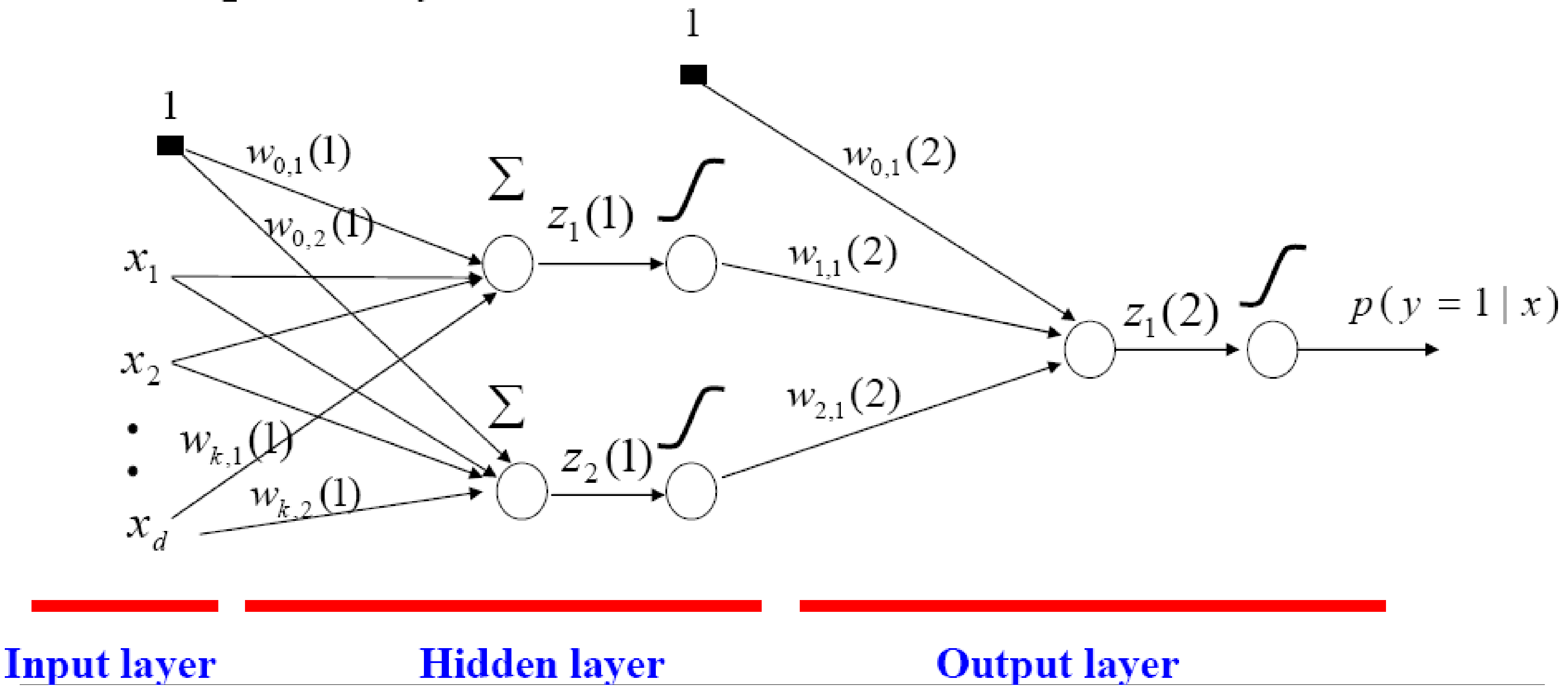
But, there are real differences, firing rate vs. potential, etc

Multilayer neural network

Also called a **multilayer perceptron (MLP)**

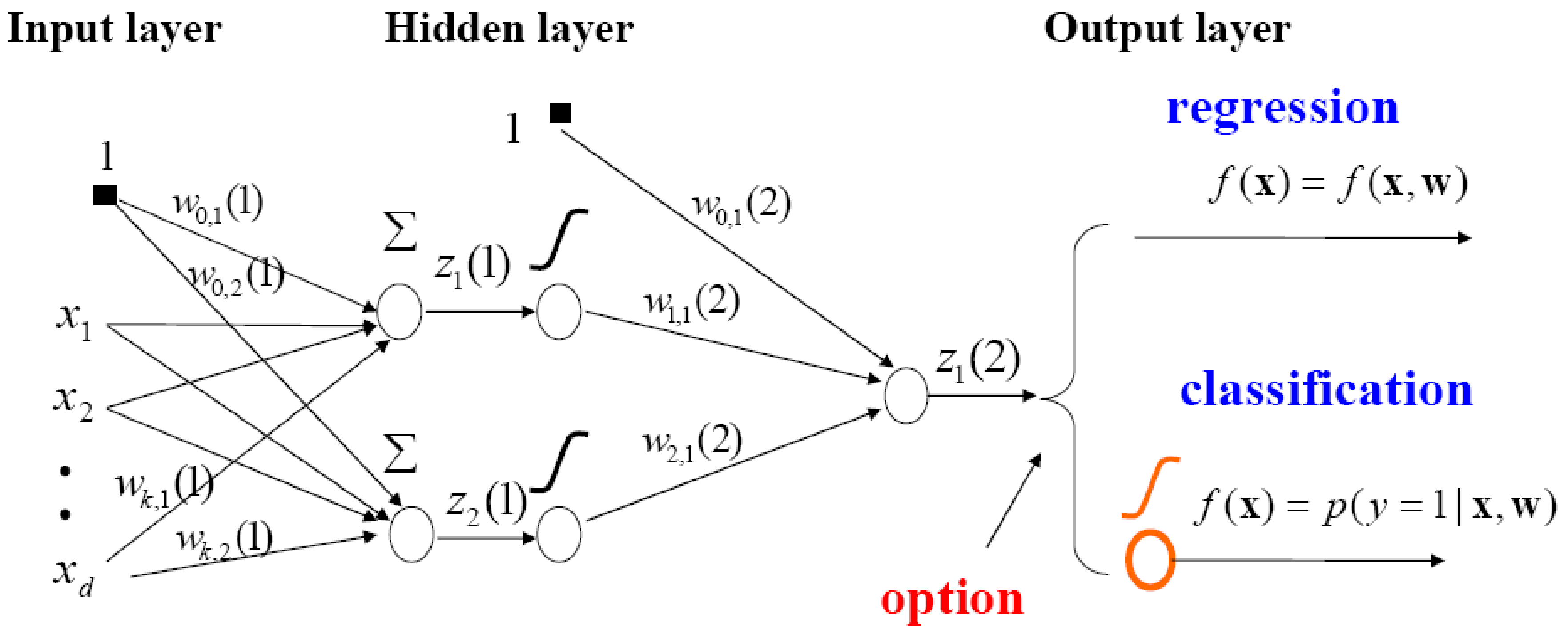
Cascades multiple logistic regression units

Example: (2 layer) classifier with non-linear decision boundaries



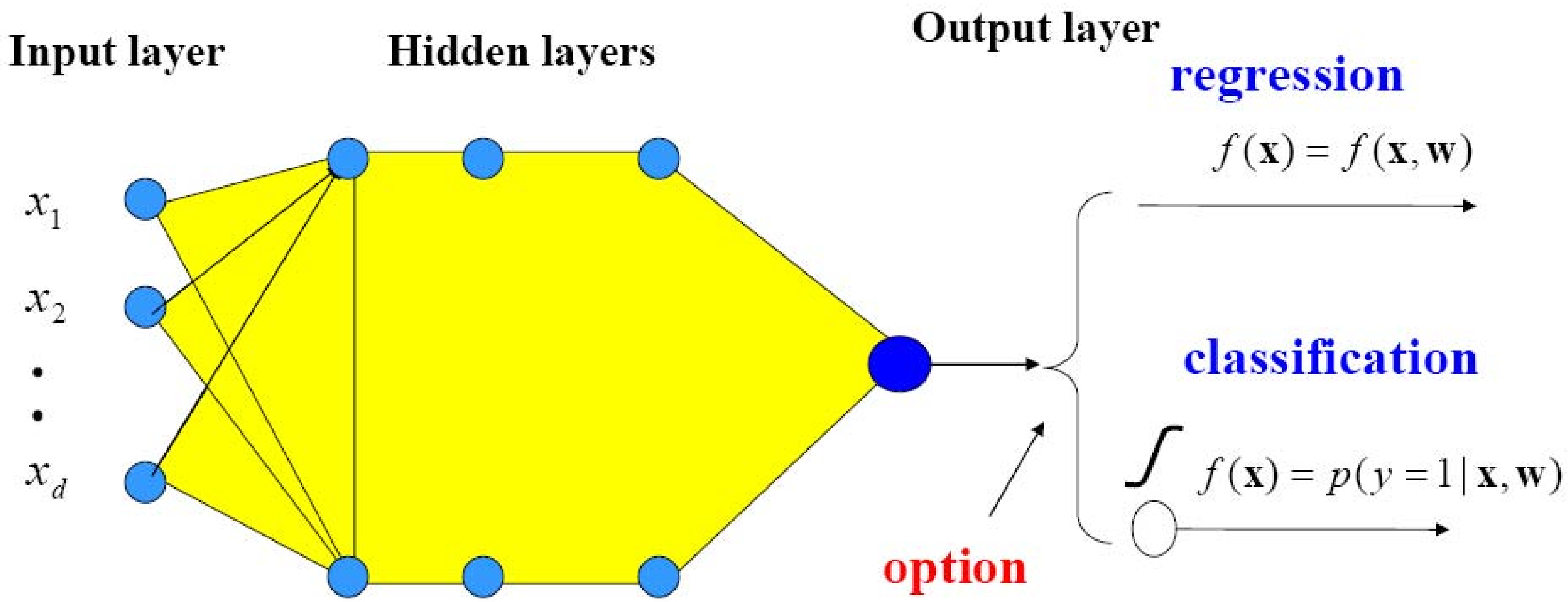
Multilayer neural network

- Models **non-linearity through logistic regression units**
- Can be applied to both **regression and binary classification problems**



Multilayer neural network

- **Non-linearities are modeled using multiple hidden logistic regression units (organized in layers)**
- The output layer determines whether it is a **regression or a binary classification problem**


Input layer
Hidden layers
Output layer
regression
$f(\mathbf{x}) = f(\mathbf{x}, \mathbf{w})$
x_1
x_2
x_d
classification
$f(\mathbf{x}) = p(y = 1 \mid \mathbf{x}, \mathbf{w})$
option

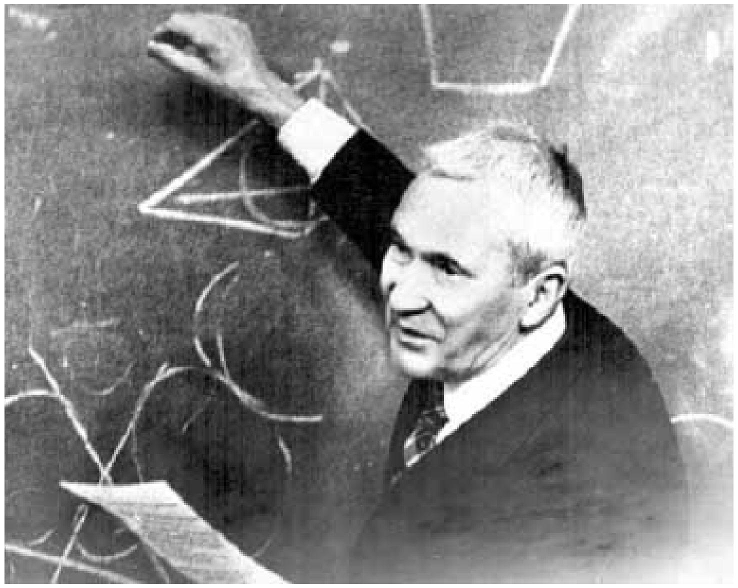
Expressive Power of NNs

- Can NNs express any function $\mathcal{F} : \Re^d \longrightarrow \Re$?
- Yes, approximately!

Kolmogorov's Superposition Theorem (1957)

For all $n \geq 2$ and for any continuous real function $\mathcal{F}$ of n variables in the domain $[0,1]$, $\mathcal{F} : [0,1]^n \mapsto \Re$, there exists $(2n+1)n$ continuous, monotone increasing univariate functions on $[0,1]$, by which $\mathcal{F}$ can be reconstructed as:

$$\mathcal{F}(x_1, \ldots, x_n) = \sum_{q=1}^{2n+1} \Phi_q \left(\sum_{p=1}^{n} \Psi_{pq}(x_q) \right)$$



Universal Approximation Theorem (Hornik 1991)

Any continuous, nonconstant, bounded and monotonically increasing function, $\mathcal{F}$, can be approximated arbitrarily closely, $\epsilon > 0$, by a feed-forward network with one semilinear hidden units using a threshold function and one linear output unit, formally,

$$\max \left| \mathcal{F}(x_1, \ldots, x_n) - \sum_{k=1}^{K} w_k^{(2)} \Phi \left(\sum_{j=1}^{n} w_{kj}^{(1)} x_j - w_{0j}^{(1)} \right) \right| < \epsilon$$

- Semilinear units: $g(L(x) - b)$, where $L(x)$ is linear in x and g (hard limiter) is a monotone real function with the limits

$$\lim_{x \to -\infty} g(x) = 0 \quad \text{and} \quad \lim_{x \to \infty} g(x) = 1$$

- This approximation is lenient about activation functions

Learning in NNs

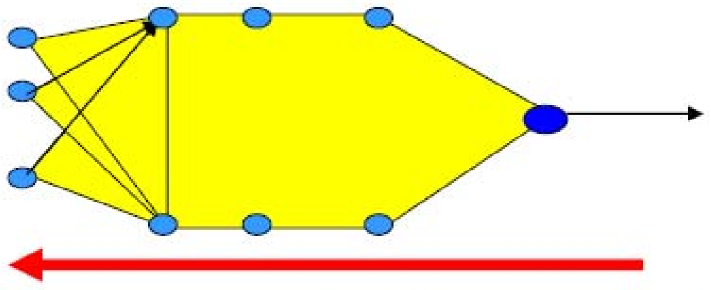
- Okay, so NNs have tremendous expressive power
- But, is there a way to learn the function for a given problem?
- Yes, the back propogation algorithm!

Learning with MLP

- How to learn the parameters of the neural network?
- **Gradient descent algorithm**
  - Weight updates based on the error: $J(D, \mathbf{w})$

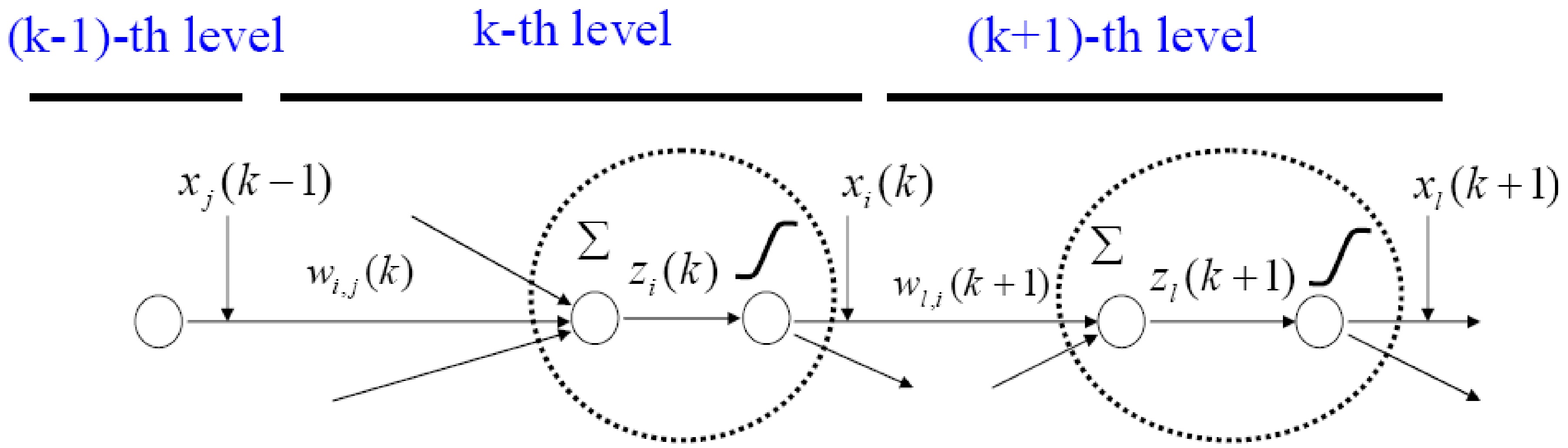
$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} J(D, \mathbf{w})$$

- We need to **compute gradients for weights in all units**
- **Can be computed in one backward sweep through the net !!!**



- The process is called **back-propagation**

Backpropagation



$x_i(k)$ - output of the unit i on level k

$z_i(k)$ - input to the sigmoid function on level k

$w_{i,j}(k)$ - weight between units j and i on levels (k-1) and k

$$z_i(k) = w_{i,0}(k) + \sum_j w_{i,j}(k) x_j(k-1)$$

$$x_i(k) = g(z_i(k))$$

Backpropagation

Update weight $w_{i,j}(k)$ using a data point $D = \{< \mathbf{x}, y >\}$

$$w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J(D, \mathbf{w})$$

Let $\delta_i(k) = \frac{\partial}{\partial z_i(k)} J(D, \mathbf{w})$

Then: $\frac{\partial}{\partial w_{i,j}(k)} J(D, \mathbf{w}) = \frac{\partial J(D, \mathbf{w})}{\partial z_i(k)} \frac{\partial z_i(k)}{\partial w_{i,j}(k)} = \delta_i(k) x_j(k-1)$

S.t. $\delta_i(k)$ is computed from $x_i(k)$ and the next layer $\delta_l(k+1)$

$$\delta_i(k) = \left[\sum_l \delta_l(k+1) w_{l,i}(k+1) \right] x_i(k)(1 - x_i(k))$$

Last unit (is the same as for the regular linear units):

$$\delta_i(K) = -\sum_{u=1}^{n} (y_u - f(\mathbf{x}_u, \mathbf{w}))$$

It is the same for the classification with the log-likelihood measure of fit and linear regression with least-squares error!!!

Learning with MLP

- **Gradient descent algorithm**
  - Weight update:

$$w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J(D, \mathbf{w})$$

$$\frac{\partial}{\partial w_{i,j}(k)} J(D\ ,\mathbf{w}) = \frac{\partial J(D,\mathbf{w})}{\partial z_i(k)} \frac{\partial z_i(k)}{\partial w_{i,j}(k)} = \delta_i(k) x_j(k-1)$$

$$\boxed{w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \delta_i(k) x_j(k-1)}$$

$x_j(k-1)$	- j-th output of the (k-1) layer
$\delta_i(k)$	- derivative computed via back-propagation
α	- a learning rate

Learning with MLP

- **Online gradient descent algorithm**
  - Weight update:

$$w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J_{\text{online}}(D_u, \mathbf{w})$$

$$\frac{\partial}{\partial w_{i,j}(k)} J_{online}(D_u, \mathbf{w}) = \frac{\partial J_{online}(D_u, \mathbf{w})}{\partial z_i(k)} \frac{\partial z_i(k)}{\partial w_{i,j}(k)} = \delta_i(k) x_j(k-1)$$

$$\boxed{w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \delta_i(k) x_j(k-1)}$$

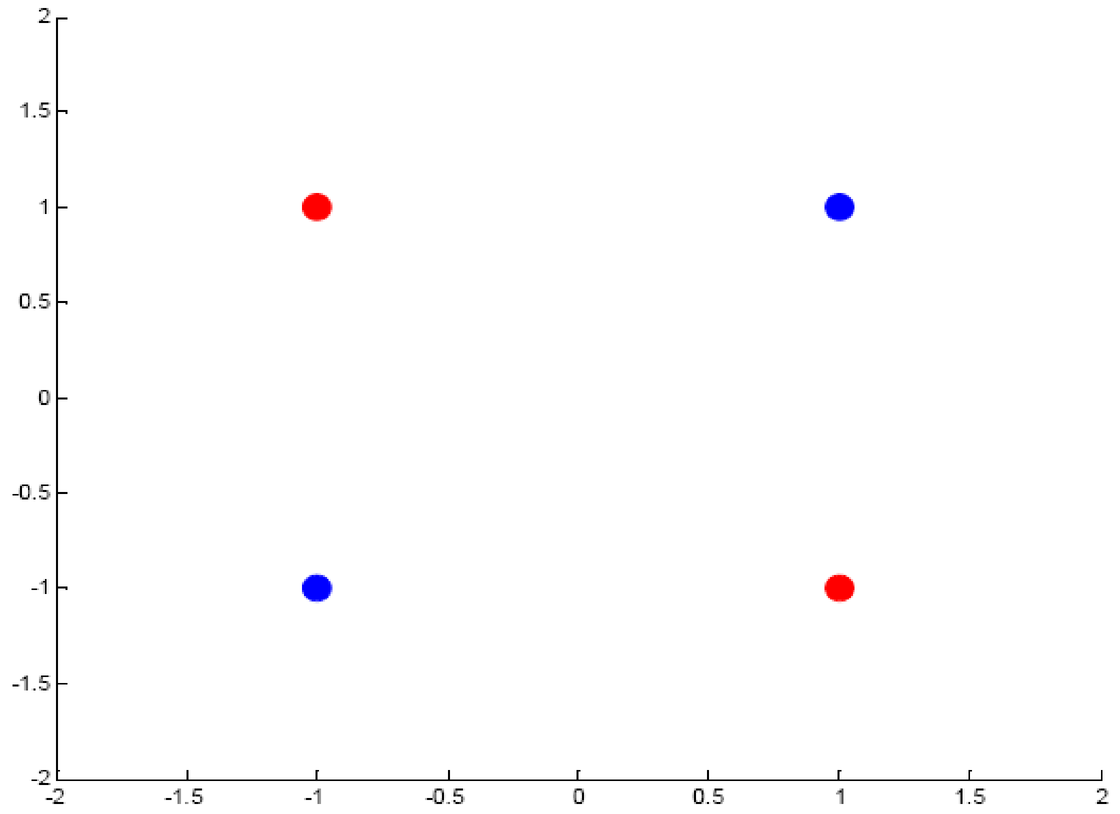
$x_j(k-1)$	- j-th output of the (k-1) layer
$\delta_i(k)$	- derivative computed via backpropagation
α	- a learning rate

Online gradient descent algorithm for MLP

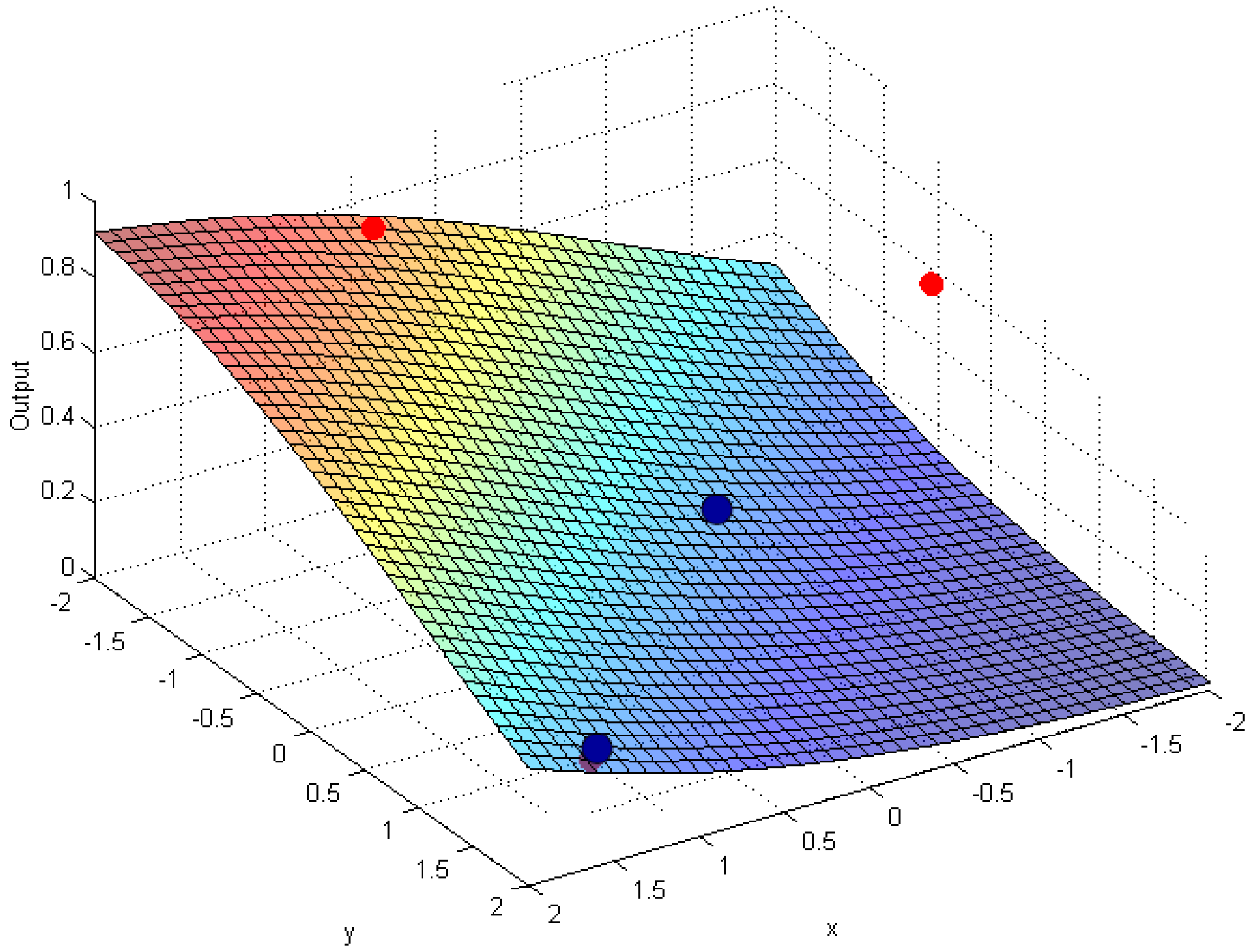
```
**Online-gradient-descent** (D, number of iterations)
    Initialize all weights $w_{i,j}(k)$
    for $i$=1:1: number of iterations
      do      select a data point $D_u$=<$\mathbf{x}$,y> from D
               set learning rate $\alpha$
               compute outputs $x_j(k)$ for each unit
               compute derivatives $\delta_i(k)$ via backpropagation
               update all weights (in parallel)
                  $w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha\delta_i(k)x_j(k-1)$
    end for
    return weights w
```

Xor Example.

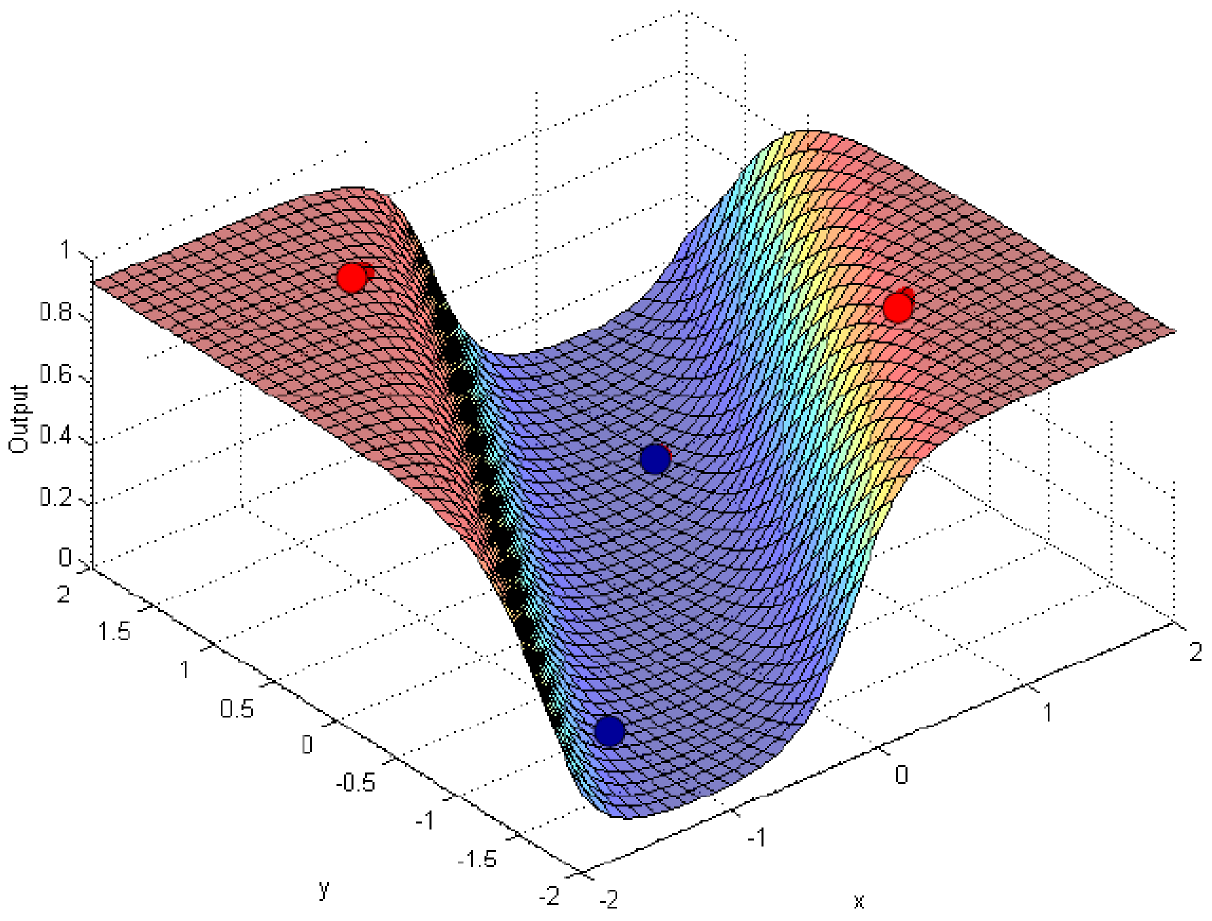
- linear decision boundary does not exist



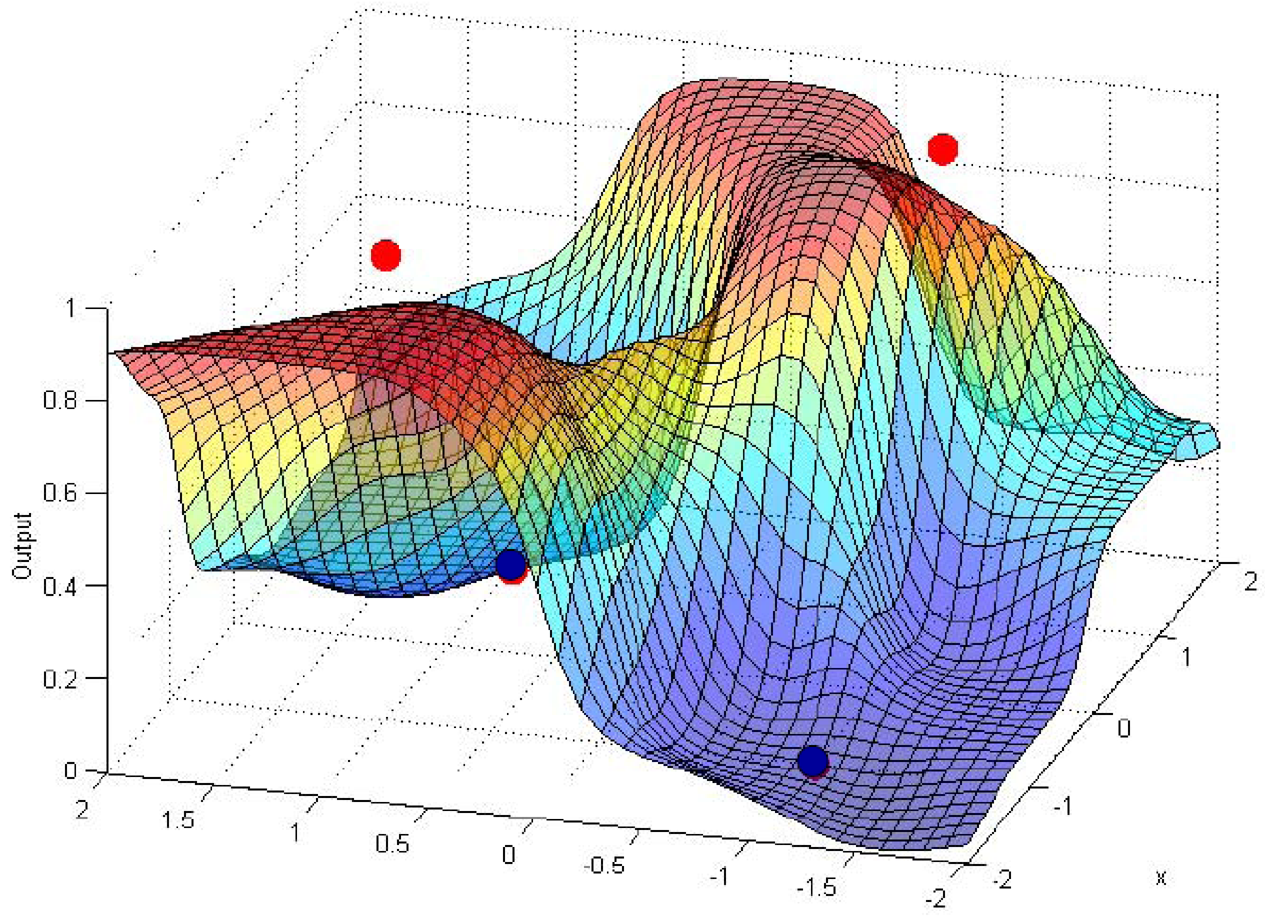
Xor example. Linear unit



Xor example.
Neural network with 2 hidden units

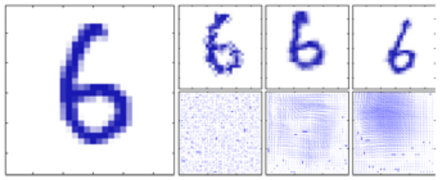


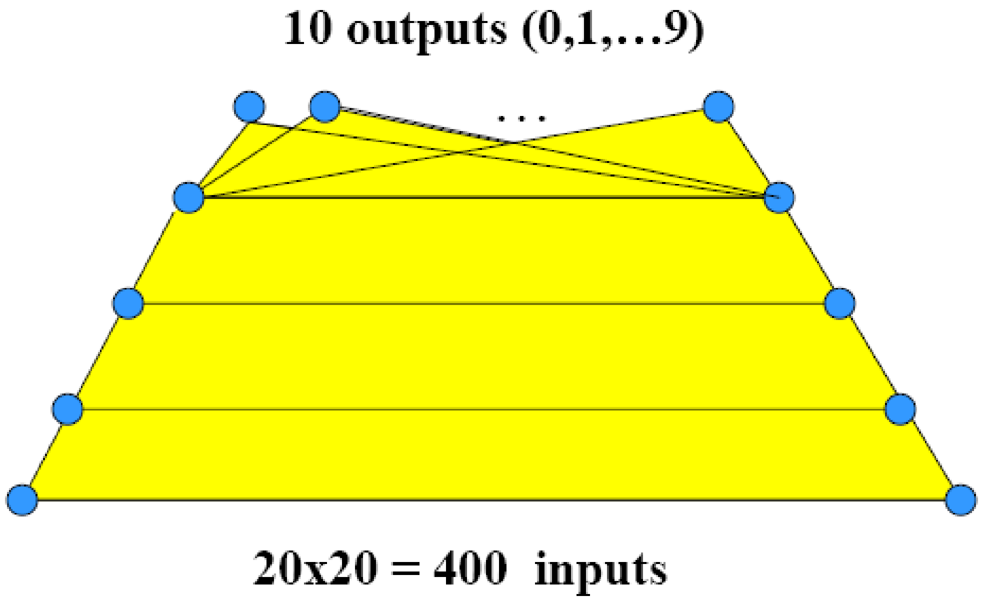
Xor example.
Neural network with 10 hidden units



MLP in practice

- **Optical character recognition** – digits 20x20
  - Automatic sorting of mails
  - 5 layer network with multiple output functions





layer	Neurons	Weights
5	10	3000
4	300	1200
3	1200	50000
2	784	3136
1	3136	78400