

# CS 2750 Machine Learning

## Lecture 10

### SVMs for regression

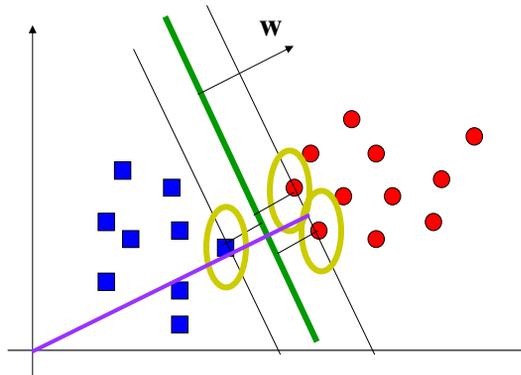
### Multilayer neural networks

Milos Hauskrecht  
[milos@cs.pitt.edu](mailto:milos@cs.pitt.edu)  
5329 Sennott Square

CS 2750 Machine Learning

### Support vector machine (SVM)

- SVM maximize the margin around the separating hyperplane.
- The decision function is fully specified by a subset of the training data, **the support vectors**.



CS 2750 Machine Learning

## Support vector machines

- **The decision boundary:**

$$\hat{\mathbf{w}}^T \mathbf{x} + w_0 = \sum_{i \in SV} \hat{\alpha}_i y_i (\mathbf{x}_i^T \mathbf{x}) + w_0$$

- **The decision:**

$$\hat{y} = \text{sign} \left[ \sum_{i \in SV} \hat{\alpha}_i y_i (\mathbf{x}_i^T \mathbf{x}) + w_0 \right]$$

- **(!!):**

- Decision on a new  $\mathbf{x}$  requires to compute the inner product between the examples  $(\mathbf{x}_i^T \mathbf{x})$
- Similarly, the optimization depends on  $(\mathbf{x}_i^T \mathbf{x}_j)$

$$J(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j (\mathbf{x}_i^T \mathbf{x}_j)$$

CS 2750 Machine Learning

## Nonlinear case

- The linear case requires to compute  $(\mathbf{x}_i^T \mathbf{x})$
- The non-linear case can be handled by using a set of features. Essentially we map input vectors to (larger) feature vectors

$$\mathbf{x} \rightarrow \boldsymbol{\phi}(\mathbf{x})$$

- It is possible to use SVM formalism on feature vectors

$$\boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{x}')$$

- **Kernel function**

$$K(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^T \boldsymbol{\phi}(\mathbf{x}')$$

- **Crucial idea:** If we choose the kernel function wisely we can compute linear separation in the feature space implicitly such that we keep working in the original input space !!!!

CS 2750 Machine Learning

## Kernel function example

- Assume  $\mathbf{x} = [x_1, x_2]^T$  and a feature mapping that maps the input into a quadratic feature set

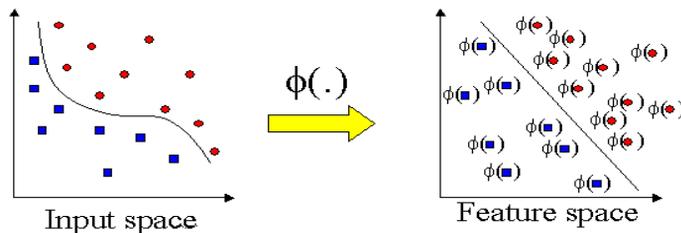
$$\mathbf{x} \rightarrow \boldsymbol{\varphi}(\mathbf{x}) = [x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1]^T$$

- Kernel function for the feature space:

$$\begin{aligned} K(\mathbf{x}', \mathbf{x}) &= \boldsymbol{\varphi}(\mathbf{x}')^T \boldsymbol{\varphi}(\mathbf{x}) \\ &= x_1^2 x_1'^2 + x_2^2 x_2'^2 + 2x_1 x_2 x_1' x_2' + 2x_1 x_1' + 2x_2 x_2' + 1 \\ &= (x_1 x_1' + x_2 x_2' + 1)^2 \\ &= (1 + (\mathbf{x}^T \mathbf{x}'))^2 \end{aligned}$$

- The computation of the linear separation in the higher dimensional space is performed implicitly in the original input space

## Nonlinear extension



### Kernel trick

- Replace the inner product with a kernel
- A well chosen kernel leads to an efficient computation

## Kernel functions

- **Linear kernel**

$$K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$$

- **Polynomial kernel**

$$K(\mathbf{x}, \mathbf{x}') = [1 + \mathbf{x}^T \mathbf{x}']^k$$

- **Radial basis kernel**

$$K(\mathbf{x}, \mathbf{x}') = \exp\left[-\frac{1}{2}\|\mathbf{x} - \mathbf{x}'\|^2\right]$$

## Kernels

- **Kernels define a similarity measure** :
  - define a distance in between two objects
- **Design criteria**: we want kernels to be
  - **valid** – Satisfy **Mercer condition** of positive semi-definiteness
  - **good** – embody the “true similarity” between objects
  - **appropriate** – generalize well
  - **efficient** – the computation of  $K(\mathbf{x}, \mathbf{x}')$  is feasible
    - NP-hard problems abound with graphs

## Kernels

- Research have proposed kernels for comparison of variety of objects:
  - Strings
  - Trees
  - Graphs
- **Cool thing:**
  - SVM algorithm can be now applied to classify a variety of objects

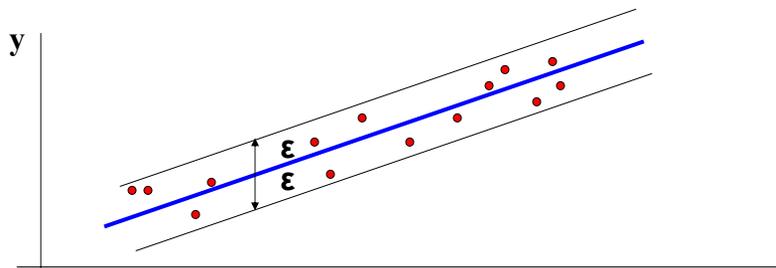
## Support vector machine for regression

**Regression** = find a function that fits the data.

- A data point may be wrong due to the noise

**Idea:** Error from points which are close **should count as a valid noise**

- Line should be influenced by the real data not the noise.



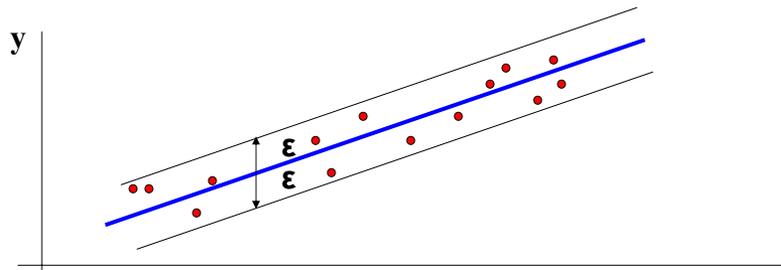
## Linear model

- **Training data:**

$$\{(x_1, y_1), \dots, (x_l, y_l)\}, x \in R^n, y \in R$$

- Our goal is to find a function  $f(x)$  that has at most  $\epsilon$  deviation from the actually obtained target for all the training data.

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \langle \mathbf{w}, \mathbf{x} \rangle + b$$



CS 2750 Machine Learning

## Linear model

**Linear function:**

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

We want a function that is:

- **flat:** means that one seeks small  $\mathbf{w}$
- all data points are within its  $\epsilon$  neighborhood

The problem can be formulated as a **convex optimization problem:**

$$\begin{aligned} & \text{minimize} && \frac{1}{2} \|\mathbf{w}\|^2 \\ & \text{subject to} && \begin{cases} y_i - \langle \mathbf{w}_i, \mathbf{x}_i \rangle - b \leq \epsilon \\ \langle \mathbf{w}_i, \mathbf{x}_i \rangle + b - y_i \leq \epsilon \end{cases} \end{aligned}$$

All data points are assumed to be in the  $\epsilon$  neighborhood

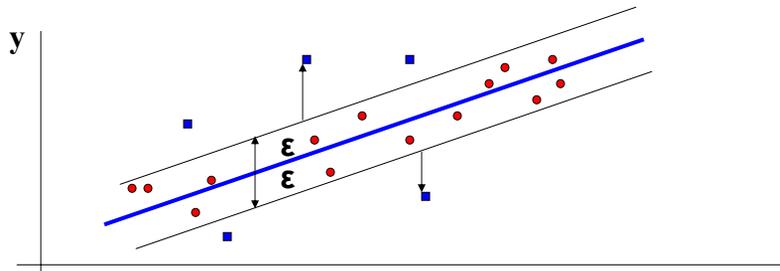
CS 2750 Machine Learning

## Linear model

- **Real data:** not all data points always fall into the  $\epsilon$  neighborhood

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

- **Idea:** penalize points that fall outside the  $\epsilon$  neighborhood



CS 2750 Machine Learning

## Linear model

**Linear function:**

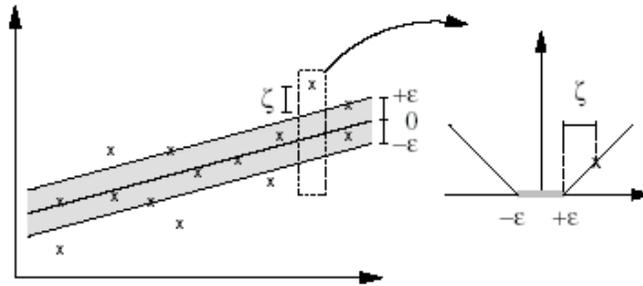
$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

**Idea:** penalize points that fall outside the  $\epsilon$  neighborhood

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^l (\xi_i + \xi_i^*) \\ \text{subject to} \quad & \begin{cases} y_i - \langle \mathbf{w}_i, \mathbf{x}_i \rangle - b \leq \epsilon + \xi_i \\ \langle \mathbf{w}_i, \mathbf{x}_i \rangle + b - y_i \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* \geq 0 \end{cases} \end{aligned}$$

CS 2750 Machine Learning

## Linear model



$$|\xi|_{\varepsilon} = \begin{cases} 0 & \text{for } |\xi| \leq \varepsilon \\ |\xi| - \varepsilon & \text{otherwise} \end{cases}$$

**$\varepsilon$ -insensitive loss function**

CS 2750 Machine Learning

## Optimization

**Lagrangian that solves the optimization problem**

$$\begin{aligned} L = & \frac{1}{2} \langle w, w \rangle + C \sum_{i=1}^l (\xi_i + \xi_i^*) \\ & - \sum_{i=1}^l a_i (\varepsilon - \xi_i - y_i + \langle w, x_i \rangle + b) - \sum_{i=1}^l a_i^* (\varepsilon + \xi_i^* + y_i - \langle w, x_i \rangle - b) \\ & - \sum_{i=1}^l (\eta_i \xi_i + \eta_i^* \xi_i^*) \end{aligned}$$

**Subject to**  $a_i, a_i^*, \eta_i, \eta_i^* \geq 0$

**Primal variables**  $w, b, \xi_i, \xi_i^*$

CS 2750 Machine Learning

## Optimization

### Derivatives with respect to primal variables

$$\frac{\partial L}{\partial b} = \sum_{i=1}^l (a_i^* - a_i) = 0$$

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^l (a_i^* - a_i) \mathbf{x}_i = \mathbf{0}$$

$$\frac{\partial L}{\partial \xi_i^{(*)}} = C - a_i^{(*)} - \eta_i^{(*)} = 0$$

$$\frac{\partial L}{\partial \xi_i} = C - a_i - \eta_i = 0$$

## Optimization

$$\begin{aligned} L &= \frac{1}{2} \langle w, w \rangle + \sum_{i=1}^l C \xi_i + \sum_{i=1}^l C \xi_i^* \\ &- \sum_{i=1}^l a_i \varepsilon - \sum_{i=1}^l a_i \xi_i - \sum_{i=1}^l a_i y_i - \sum_{i=1}^l a_i \langle \omega, x_i \rangle + \sum_{i=1}^l a_i b \\ &- \sum_{i=1}^l a_i^* \varepsilon - \sum_{i=1}^l a_i^* \xi_i^* - \sum_{i=1}^l a_i^* y_i + \sum_{i=1}^l a_i^* \langle \omega, x_i \rangle + \sum_{i=1}^l a_i^* b \\ &- \sum_{i=1}^l \eta_i \xi_i - \sum_{i=1}^l \eta_i^* \xi_i^* \end{aligned}$$

## Optimization

$$\begin{aligned}
 L &= \frac{1}{2} \langle w, w \rangle + \sum_{i=1}^l \xi_i \underbrace{(C - \eta_i - a_i)}_{=0(C-\eta_i-a_i=0)} + \\
 &\sum_{i=1}^l \xi_i^* \underbrace{(C - \eta_i^* - a_i^*)}_{=0(C-\eta_i^*-a_i^*=0)} - \sum_{i=1}^l (a_i + a_i^*) \varepsilon - \sum_{i=1}^l (a_i + a_i^*) y_i \\
 &- \sum_{i=1}^l \underbrace{(a_i - a_i^*) \langle \omega, x_i \rangle}_{=\langle w, w \rangle (\omega = \sum_{i=1}^l (a_i + a_i^*) x_i)} + \sum_{i=1}^l \underbrace{(a_i^* - a_i) b}_{=0(\sum_{i=1}^l (a_i^* - a_i) = 0)}
 \end{aligned}$$

---

CS 2750 Machine Learning

## Optimization

$$L = -\frac{1}{2} \langle w, w \rangle - \sum_{i=1}^l (a_i + a_i^*) \varepsilon - \sum_{i=1}^l (a_i + a_i^*) y_i$$

**Maximize the dual**

$$\begin{aligned}
 L(a, a^*) &= -\frac{1}{2} \sum_{i=1}^l (a_i - a_i^*) (a_j - a_j^*) \langle x_i, x_j \rangle \\
 &\quad - \sum_{i=1}^l (a_i + a_i^*) \varepsilon - \sum_{i=1}^l (a_i + a_i^*) y_i
 \end{aligned}$$

$$\text{subject to } : \begin{cases} \sum_{i=1}^l (a_i - a_i^*) = 0 \\ a_i, a_i^* \in [0, C] \end{cases}$$

---

CS 2750 Machine Learning

## SVM solution

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^l (a_i^* - a_i) \mathbf{x}_i = \mathbf{0}$$

$$\mathbf{w} = \sum_{i=1}^l (a_i - a_i^*) \mathbf{x}_i$$

We can get:

$$f(\mathbf{x}) = \sum_{i=1}^l (a_i - a_i^*) \langle \mathbf{x}_i, \mathbf{x} \rangle + b$$

at the optimal solution the Lagrange multipliers are non-zero only **for points outside the  $\epsilon$  band.**

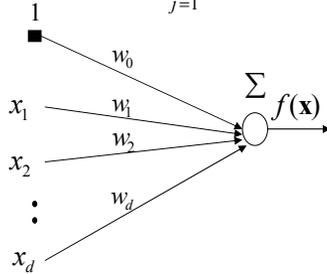
## Multilayer neural networks

**Or another way of modeling nonlinearities  
for regression and classification problems**

## Linear units

### Linear regression

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



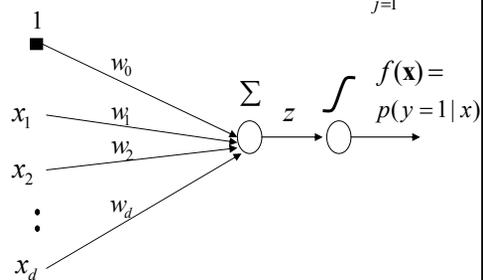
#### On-line gradient update:

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

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

### Logistic regression

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



#### On-line gradient update:

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

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

The same

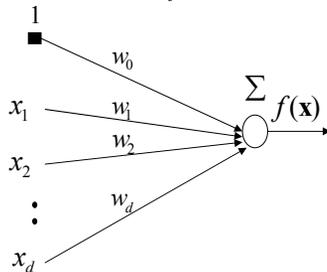
↔

CS 2750 Machine Learning

## Limitations of basic linear units

### Linear regression

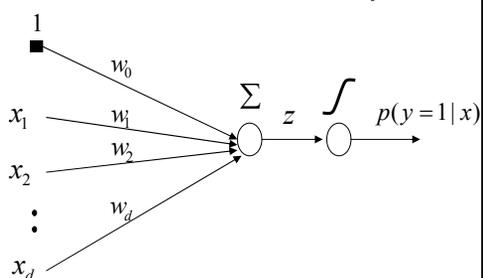
$$f(\mathbf{x}) = w_0 + \sum_{j=1}^d w_j x_j$$



Function linear in inputs !!

### Logistic regression

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

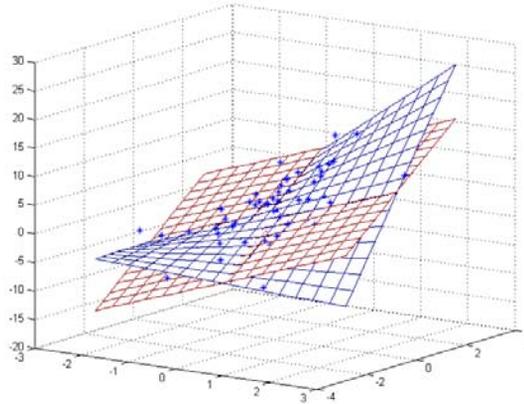


Linear decision boundary !!

CS 2750 Machine Learning

## Regression with the quadratic model.

- Limitation:** linear hyper-plane only
- a non-linear surface can be better

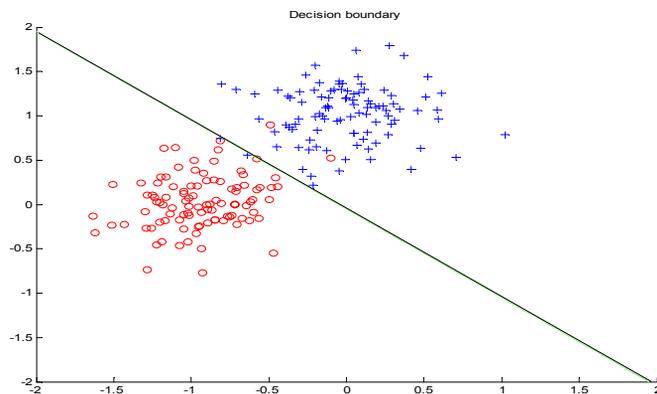


CS 2750 Machine Learning

## Classification with the linear model.

**Logistic regression model defines a linear decision boundary**

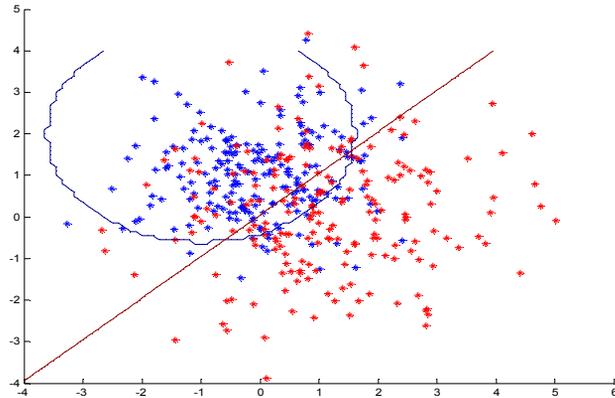
- Example: 2 classes (blue and red points)



CS 2750 Machine Learning

## Linear decision boundary

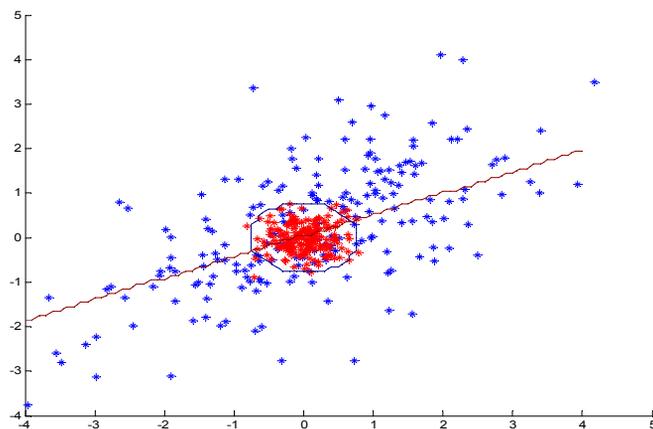
- logistic regression model is not optimal, but not that bad



CS 2750 Machine Learning

## When logistic regression fails?

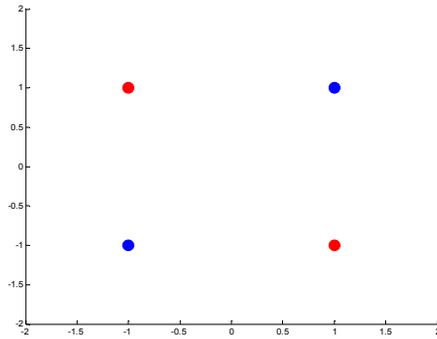
- Example in which the logistic regression model fails



CS 2750 Machine Learning

## 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

CS 2750 Machine Learning

## Extensions of simple linear units

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

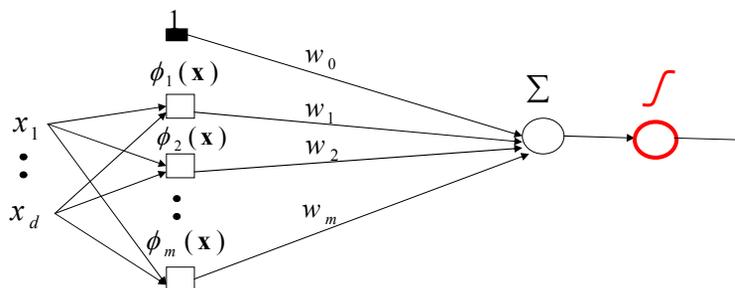
**Linear regression**

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

**Logistic regression**

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

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



CS 2750 Machine Learning

## Learning with extended linear units

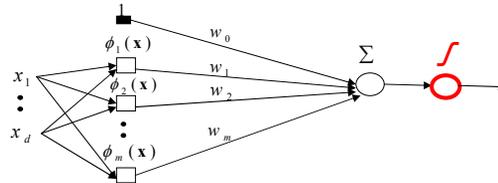
Feature (basis) functions model **nonlinearities**

**Linear regression**

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

**Logistic regression**

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



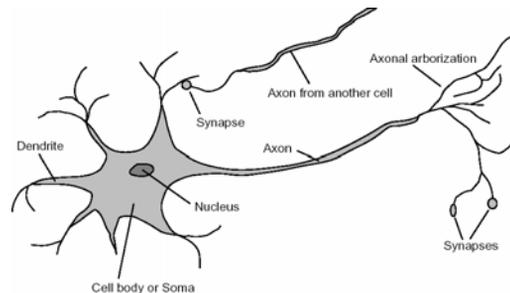
**Important property:**

- 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.

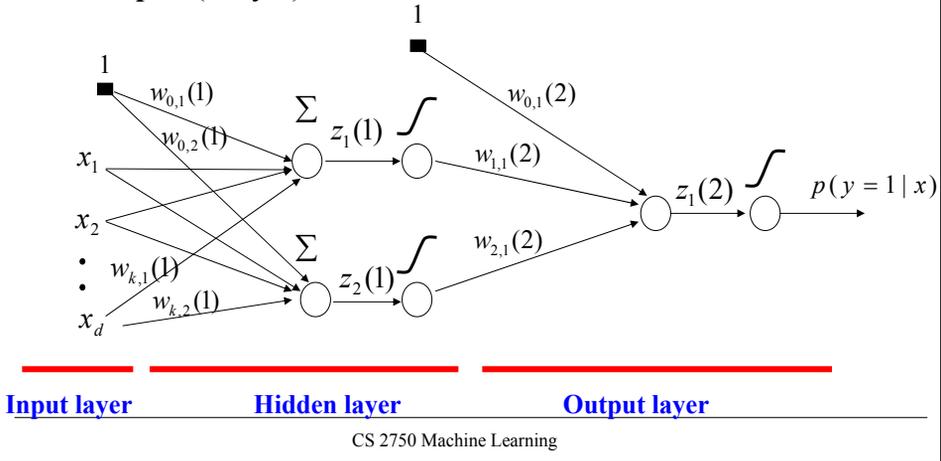


## Multilayer neural network

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

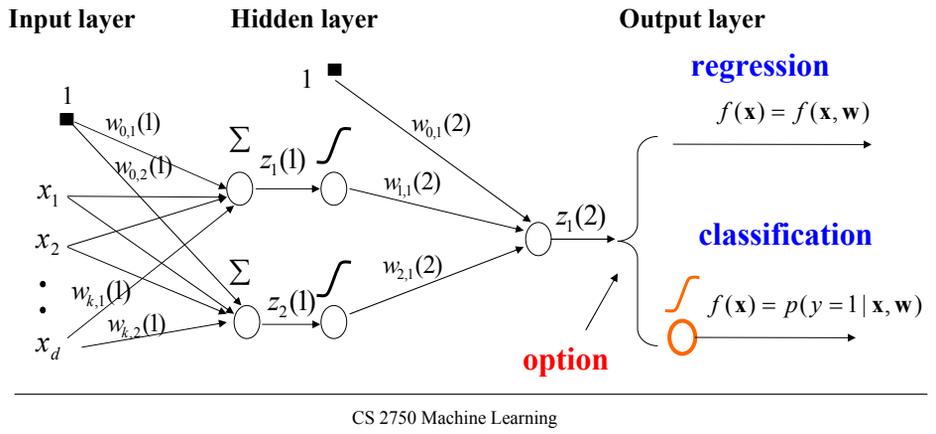
Cascades multiple logistic regression units

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



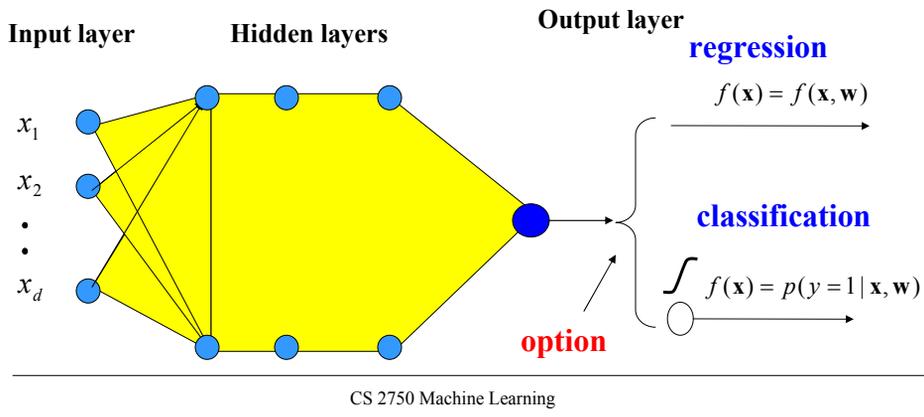
## Multilayer neural network

- 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**

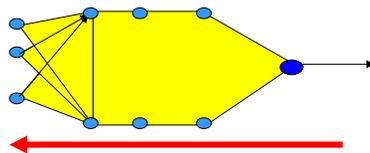


## Learning with MLP

- 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 !!!**



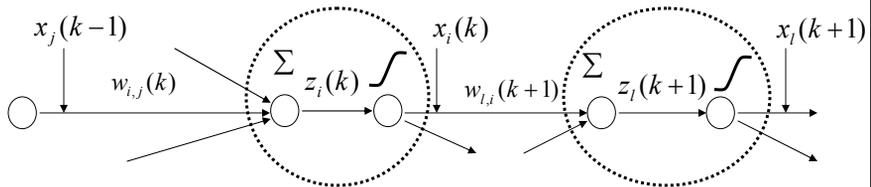
- The process is called **back-propagation**

# Backpropagation

(k-1)-th level

k-th level

(k+1)-th level



$x_i(k)$  - 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  $D = \{< \mathbf{x}, y >\}$

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

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

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

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

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

**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:

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

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

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

$x_j(k-1)$  - j-th output of the (k-1) layer

$\delta_i(k)$  - derivative computed via back-propagation

$\alpha$  - a learning rate

## Learning with MLP

- **Online gradient descent algorithm**

– Weight update:

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

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

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

$x_j(k-1)$  - j-th output of the (k-1) layer

$\delta_i(k)$  - derivative computed via backpropagation

$\alpha$  - a learning rate

CS 2750 Machine Learning

## 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  $D_u = \langle \mathbf{x}, y \rangle$  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)

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

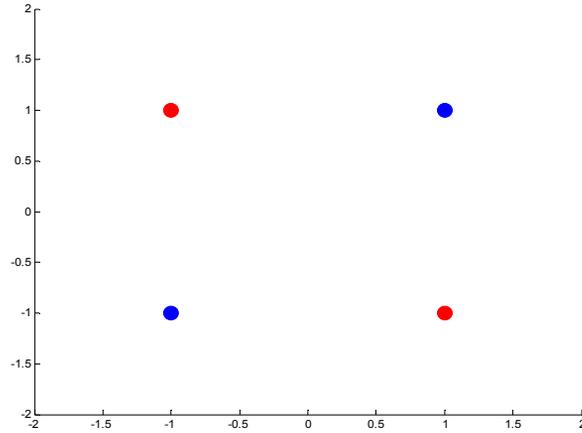
**end for**

**return** weights  $\mathbf{w}$

CS 2750 Machine Learning

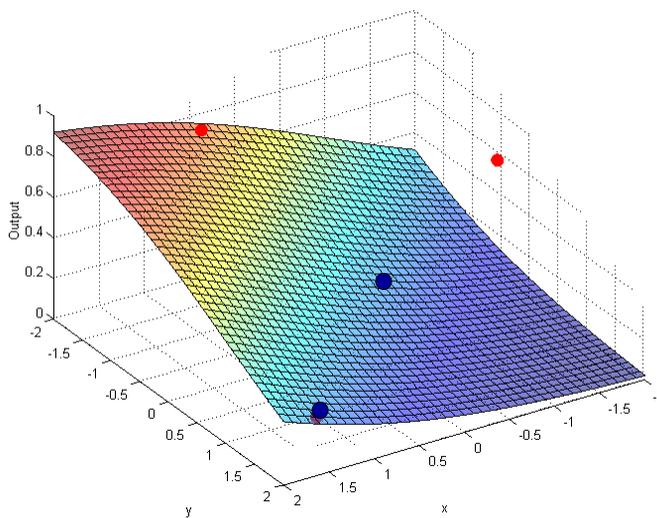
## Xor Example.

- linear decision boundary does not exist



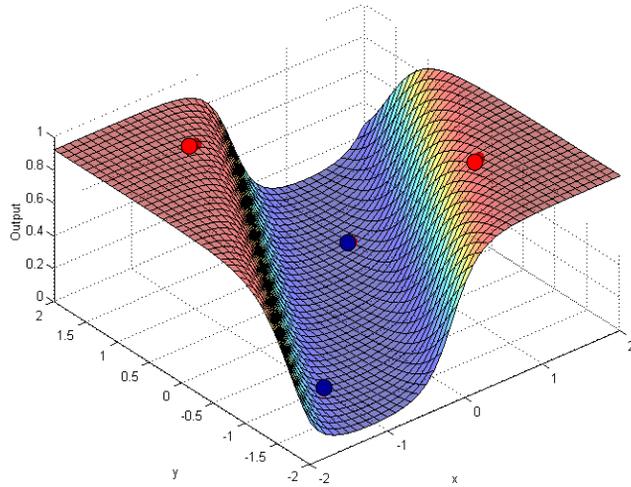
CS 2750 Machine Learning

## Xor example. Linear unit



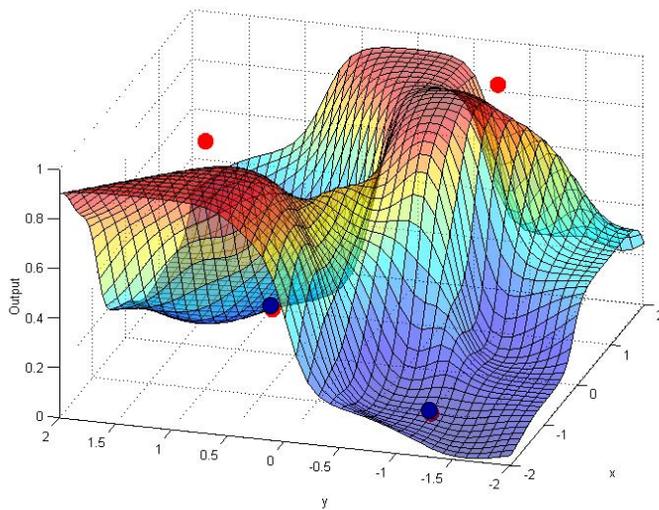
CS 2750 Machine Learning

## Xor example. Neural network with 2 hidden units



CS 2750 Machine Learning

## Xor example. Neural network with 10 hidden units



CS 2750 Machine Learning

## MLP in practice

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

