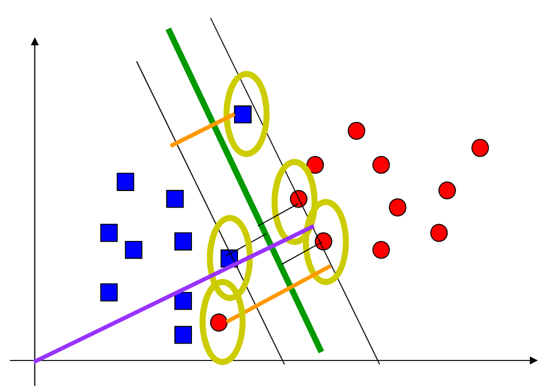
CS 2750 Machine Learning
Lecture 11

SVMs for regression
Multilayer neural networks

Milos Hauskrecht
milos@cs.pitt.edu
5329 Sennott Square

Linearly non-separable case

- Allow some flexibility on crossing the separating hyperplane



Linearly non-separable case

minimize $\|\mathbf{w}\|^2 / 2 + C\sum_{i=1}^{n} \xi_i$

$$\mathbf{w}^T \mathbf{x}_i + w_0 \geq 1 - \xi_i \quad \textbf{for} \quad y_i = +1$$

$$\mathbf{w}^T \mathbf{x}_i + w_0 \leq -1 + \xi_i \quad \textbf{for} \quad y_i = -1$$

$$\xi_i \geq 0$$

- Rewrite $\xi_i = \max\left[0, \quad 1 - y_i(\mathbf{w}^T \mathbf{x}_i + w_0)\right]$ in $\|\mathbf{w}\|^2 / 2 + C\sum_{i=1}^{n} \xi_i$

$$\|\mathbf{w}\|^2 / 2 + C\sum_{i=1}^{n} \max\left[0, \quad 1 - y_i(\mathbf{w}^T \mathbf{x}_i + w_0)\right]$$

Regularization penalty

Hinge loss

Support vector machines

- **The decision boundary:**

$$\hat{\mathbf{w}}^T \mathbf{x} + w_0 = \sum_{i \in SV} \hat{\alpha}_i y_i (\mathbf{x}_i^T \mathbf{x}) + w_0 = 0$$

- **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 $(\mathbf{x}_i^T \mathbf{x}_j)$

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

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

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

- It is possible to use SVM formalism on feature vectors

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

- **Kernel function**

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

- **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

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

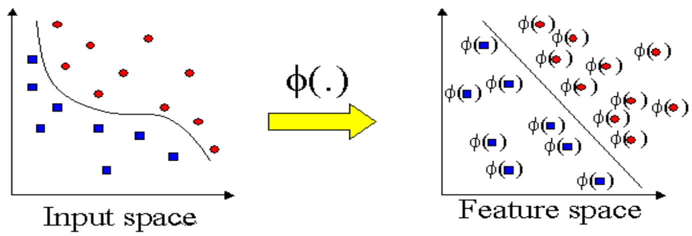
- Kernel function for the feature space:

$$\begin{aligned} K(\mathbf{x}', \mathbf{x}) &= \boldsymbol{\varphi}(\mathbf{x}')^T \boldsymbol{\varphi}(\mathbf{x}) \\ &= x_1^2 x_1'^2 + x_2^2 x_2'^2 + 2x_1 x_2 x'_1 x'_2 + 2x_1 x'_1 + 2x_2 x'_2 + 1 \\ &= (x_1 x'_1 + x_2 x'_2 + 1)^2 \\ &= (1 + (\mathbf{x}^T \mathbf{x}'))^2 \end{aligned}$$

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

CS 2750 Machine Learning

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**

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

- **Radial basis kernel**

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

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(x,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

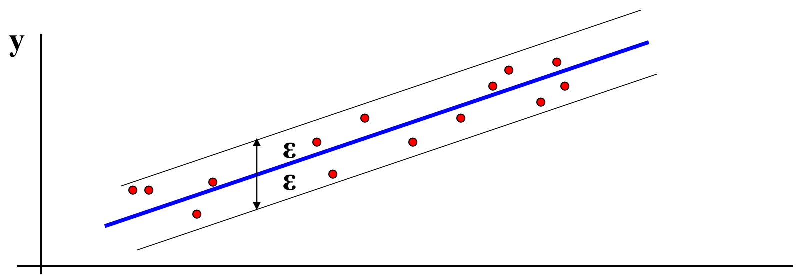
Support vector machine for regression

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.



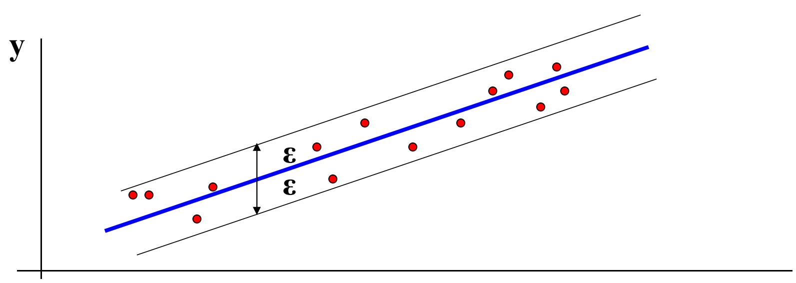
Linear model

- **Training data:**

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

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

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



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 **w**
- all data points are within its ε neighborhood

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

$$\text{minimize} \quad \frac{1}{2}\|w\|^2$$

$$\text{subject to} \quad \begin{cases} y_i - \langle w_i, x_i \rangle - b \le \varepsilon \\ \langle w_i, x_i \rangle + b - y_i \le \varepsilon \end{cases}$$

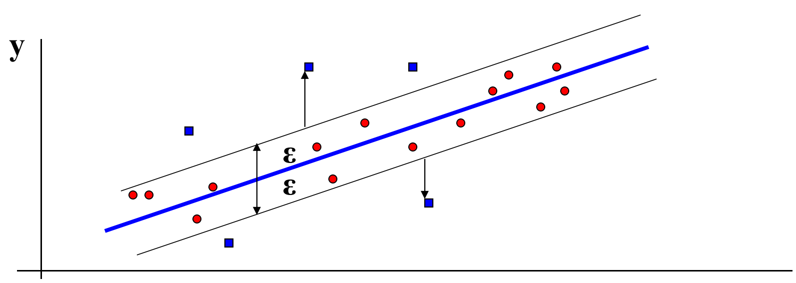
All data points are assumed to be in the ε neighborhood

Linear model

- **Real data: not all data points always fall into the** ε neighborhood

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

- **Idea:** penalize points that fall outside the ε neighborhood



Linear model

Linear function:

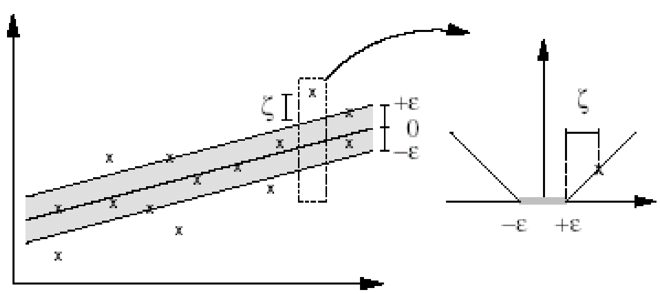
$$f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b = \langle \mathbf{w}, \mathbf{x} \rangle + b$$

Idea: penalize points that fall outside the ε neighborhood

$$\text{minimize} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*)$$

$$\text{subject to} \quad \begin{cases} \mathrm{y}_{\mathrm{i}} - \langle w_i, x_i \rangle - b \le \varepsilon + \xi_i \\ \langle w_i, x_i \rangle + b - \mathrm{y}_{\mathrm{i}} \le \varepsilon + \xi_i^* \\ \xi_i, \xi_i^* \ge 0 \end{cases}$$

Linear model



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

ε-intensive loss function

Optimization

Lagrangian that solves the optimization problem

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

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

Primal variables w, b, ξ_i, ξ_i^*

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

$$L = \frac{1}{2}\langle w, w\rangle + \sum_{i=1}^{l} C\xi_i + \sum_{i=1}^{l} C\xi_i^*$$

$$- \sum_{i=1}^{l} a_i\varepsilon - \sum_{i=1}^{l} a_i\xi_i - \sum_{i=1}^{l} a_i y_i - \sum_{i=1}^{l} a_i\langle \omega, x_i\rangle + \sum_{i=1}^{l} a_i b$$

$$- \sum_{i=1}^{l} a_i^*\varepsilon - \sum_{i=1}^{l} a_i^*\xi_i^* - \sum_{i=1}^{l} a_i^* y_i + \sum_{i=1}^{l} a_i^*\langle \omega, x_i\rangle + \sum_{i=1}^{l} a_i^* b$$

$$- \sum_{i=1}^{l} \eta_i\xi_i - \sum_{i=1}^{l} \eta_i^*\xi_i^*$$

Optimization

$$L = \frac{1}{2}\langle w, w\rangle + \sum_{i=1}^{l} \xi_i \underbrace{(C - \eta_i - a_i)}_{=0\,(C-\eta_i - a_i = 0)} +$$

$$\sum_{i=1}^{l} \xi_i^* \underbrace{(C - \eta_i^* - a_i^*)}_{=0\,(C-\eta_i^{(*)} - a_i^{(*)} = 0)} - \sum_{i=1}^{l} (a_i + a_i^*)\varepsilon - \sum_{i=1}^{l} (a_i + a_i^*) y_i$$

$$- \sum_{i=1}^{l} \underbrace{(a_i - a_i^*)\langle \omega, x_i\rangle}_{=\langle w, w\rangle\,(\omega = \sum_{i=1}^{l}(a_i + a_i^*) x_i)} + \sum_{i=1}^{l} \underbrace{(a_i^* - a_i) b}_{=0\,(\sum_{i=1}^{l}(a_i^* - a_i) = 0)}$$

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

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

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

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 ε band.**

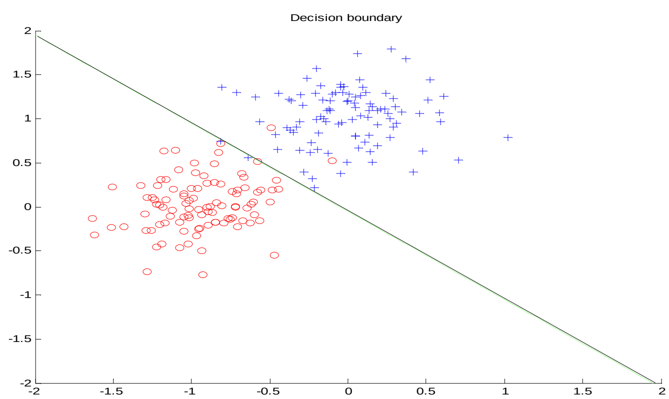
Multilayer neural networks

Or another way of modeling nonlinearities for regression and classification problems

Classification with the linear model.

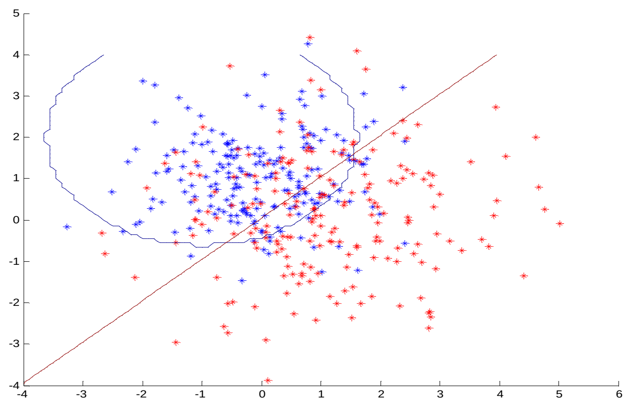
Logistic regression model defines a linear decision boundary

- Example: 2 classes (blue and red points)



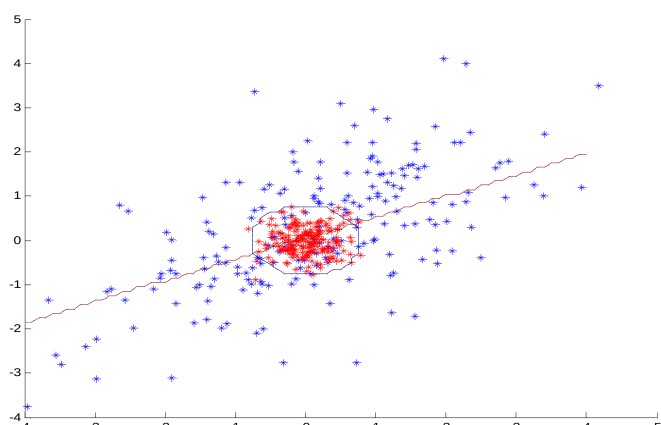
Linear decision boundary

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



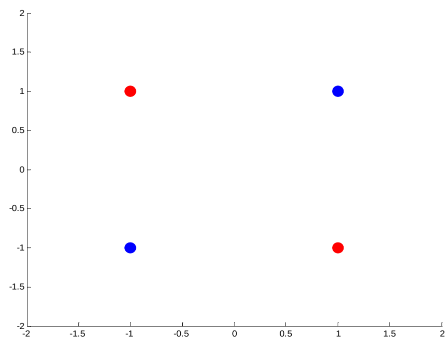
When logistic regression fails?

- 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**

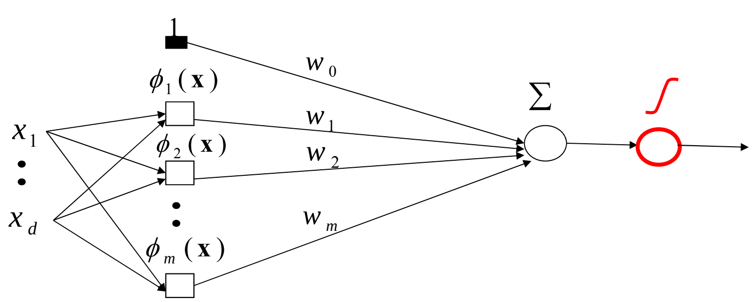
Linear regression

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

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 $\mathbf{x}$



Learning with extended linear units

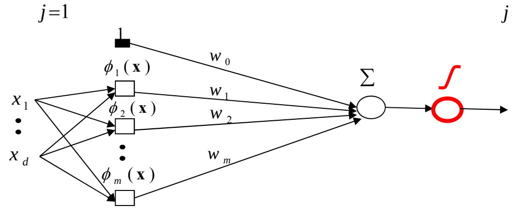
Feature (basis) functions model **nonlinearities**

Linear regression

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

Logistic regression

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



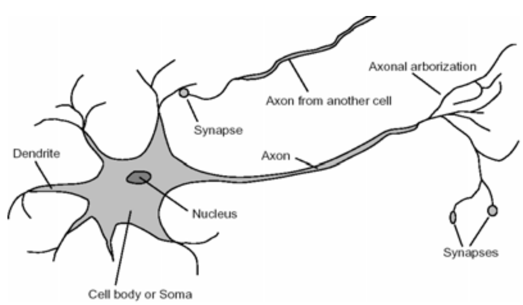
Important property:

- 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.

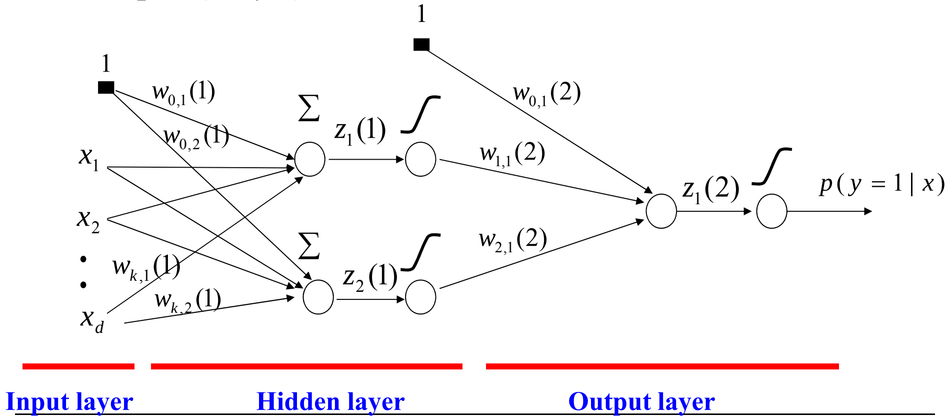


Multilayer neural network

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

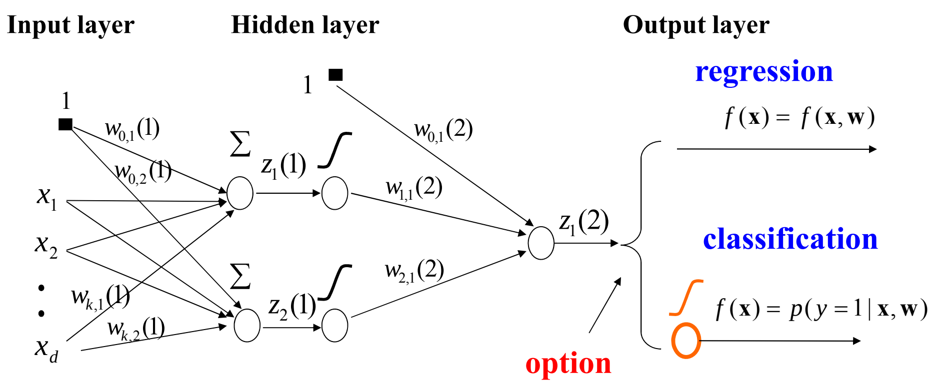
Cascades multiple logistic regression units

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



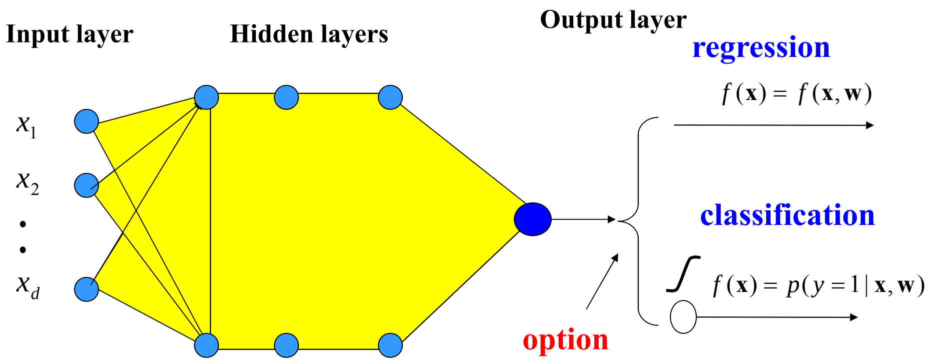
Multilayer neural network

- 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**

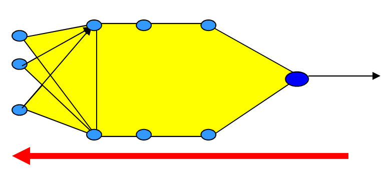


Learning with MLP

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

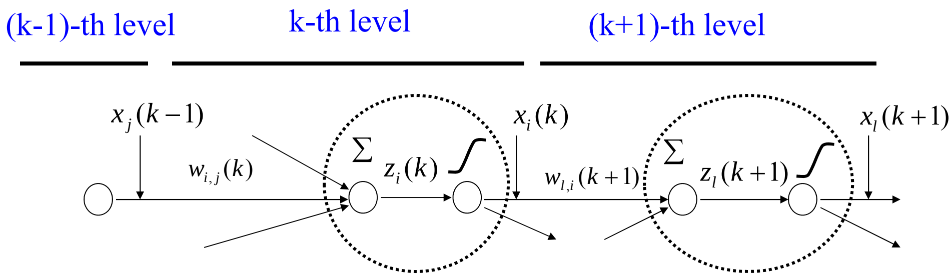
$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} J(D, \mathbf{w})$$

- 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



$x_i(k)$ - 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

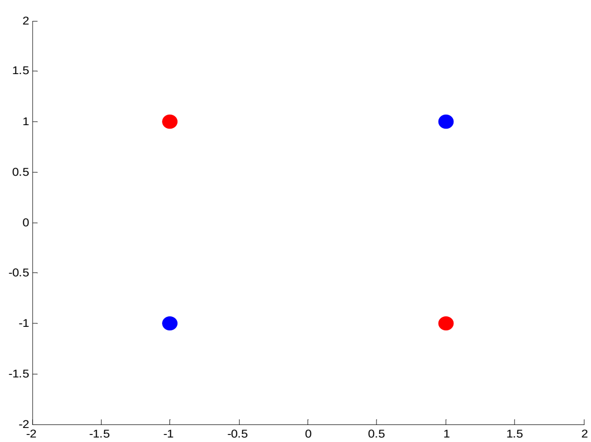
Online-gradient-descent (*D, number of iterations*)
Initialize all weights $w_{i,j}(k)$
for i=1:1: *number of iterations*
do **select** a data point D_u=<**x**,*y*> from *D*
set learning rate α
compute outputs $x_j(k)$ for each unit
compute derivatives $\delta_i(k)$ via **backpropagation**
update all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

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

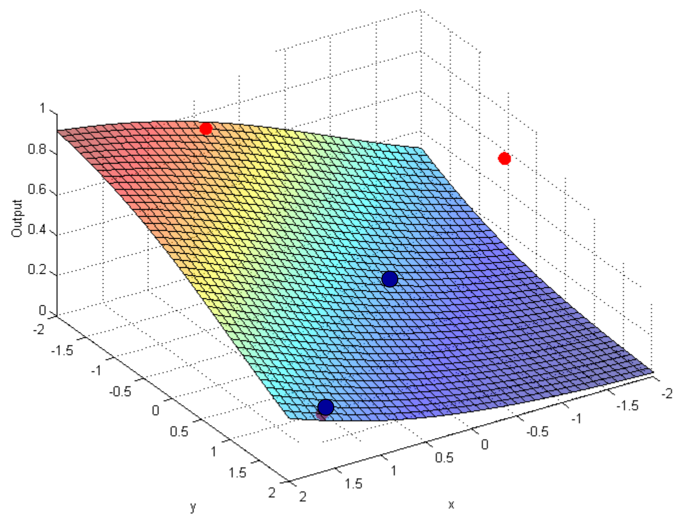
end for
return weights **w**

Xor Example.

- linear decision boundary does not exist

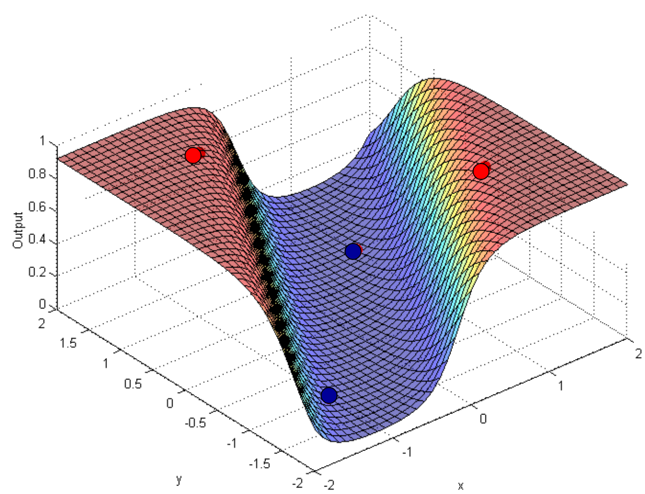


Xor example. Linear unit



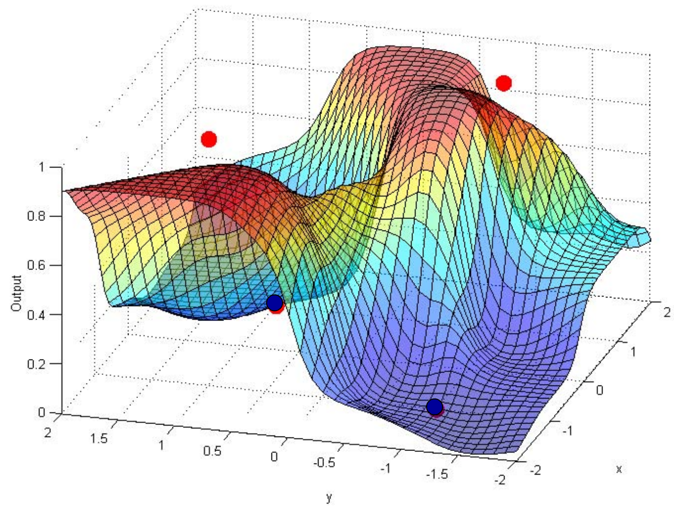
CS 2750 Machine Learning

Xor example. Neural network with 2 hidden units



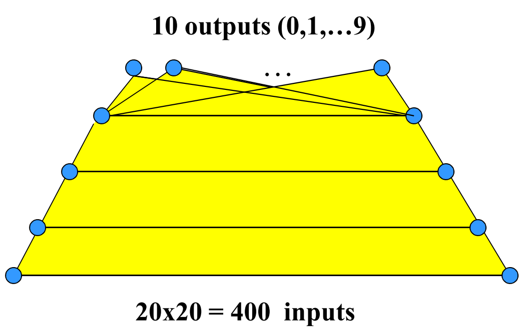
CS 2750 Machine Learning

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