CS 1571 Introduction to AI
Lecture 10
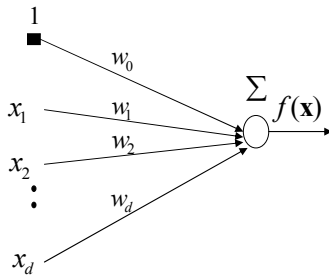
Multi-layer neural networks

Milos Hauskrecht

milos@cs.pitt.edu

5329 Sennott Square

Linear units

Linear regression

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}$$

1
w_0
x_1 w_1
w_2
x_2
$\vdots$ w_d
x_d

Σ $f(\mathbf{x})$

Logistic regression

$$f(\mathbf{x}) = p(y = 1 \mid \mathbf{x}, \mathbf{w}) = g(\mathbf{w}^T \mathbf{x})$$

1
w_0
x_1 w_1
w_2
x_2
$\vdots$ w_d
x_d

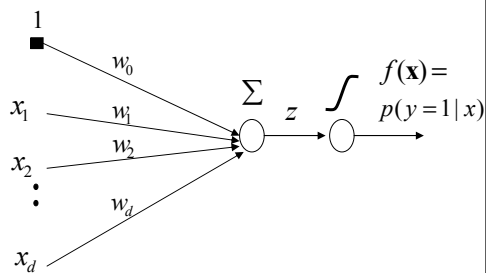Σ z $\int$ $f(\mathbf{x}) = p(y = 1 \mid x)$

Gradient update:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i)) \mathbf{x}_i$$

Online: $\mathbf{w} \leftarrow \mathbf{w} + \alpha (y - f(\mathbf{x})) \mathbf{x}$

The same

Gradient update:

$$\mathbf{w} \leftarrow \mathbf{w} + \alpha \sum_{i=1}^{n} (y_i - f(\mathbf{x}_i)) \mathbf{x}_i$$
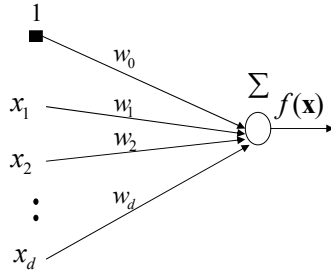
Online: $\mathbf{w} \leftarrow \mathbf{w} + \alpha (y - f(\mathbf{x})) \mathbf{x}$

Limitations of basic linear units
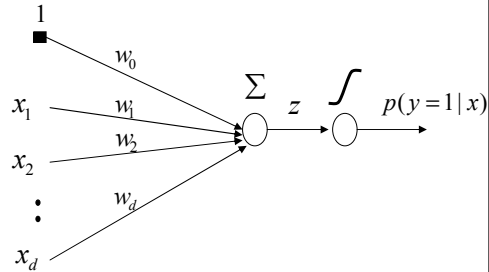
Linear regression

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

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!!

Extensions of simple linear units

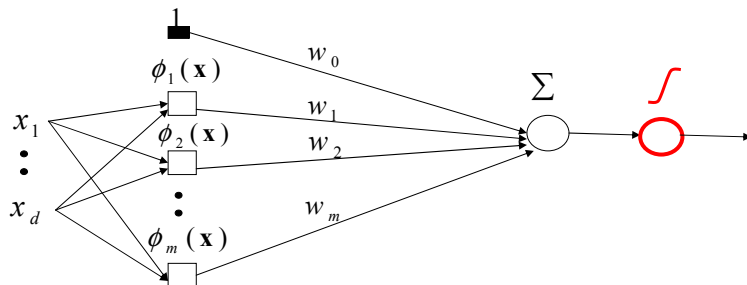- use **feature (basis) functions** to model **nonlinearities**

Linear regression

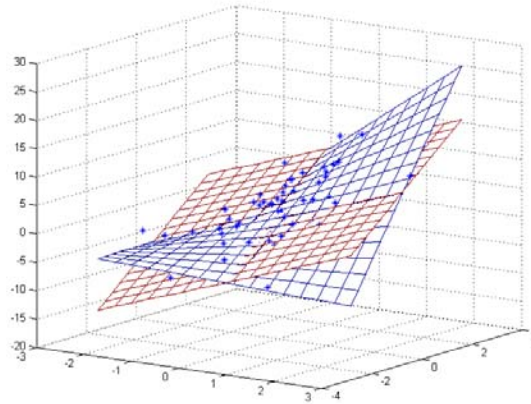$$f(\mathbf{x}) = w_0 + \sum_{j=1}^{m} w_j \phi_j(\mathbf{x})$$

Logistic regression

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

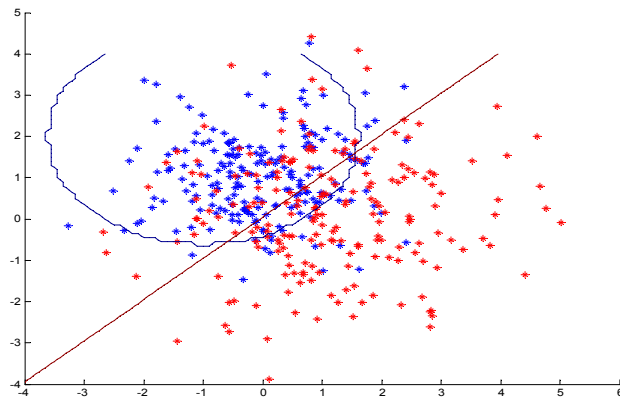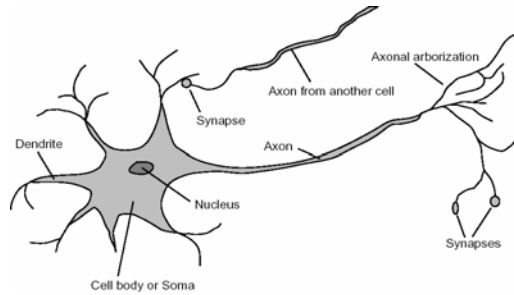$\phi_j(\mathbf{x})$ - an arbitrary function of $\mathbf{x}$

Regression with a quadratic model.
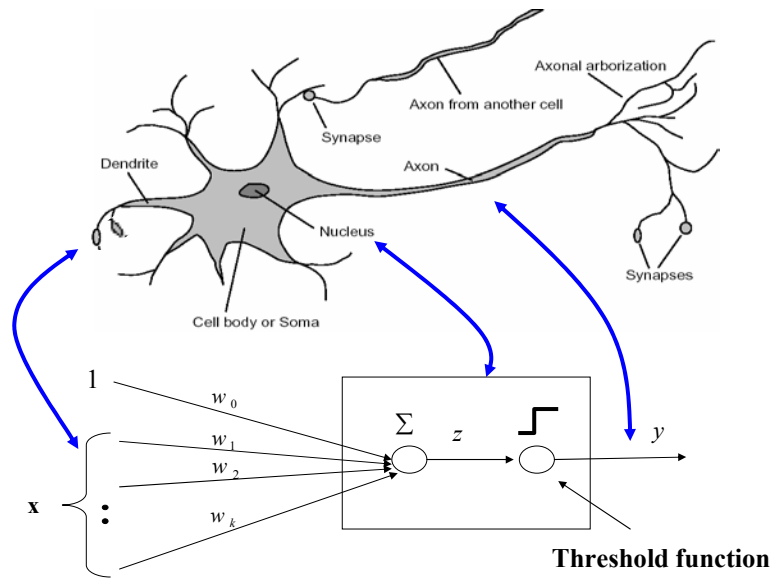
Quadratic decision boundary

Multi-layered neural networks

- Offer an alternative way to introduce nonlinearities to regression/classification models
- **Idea:** Cascade several simple logistic regression units.
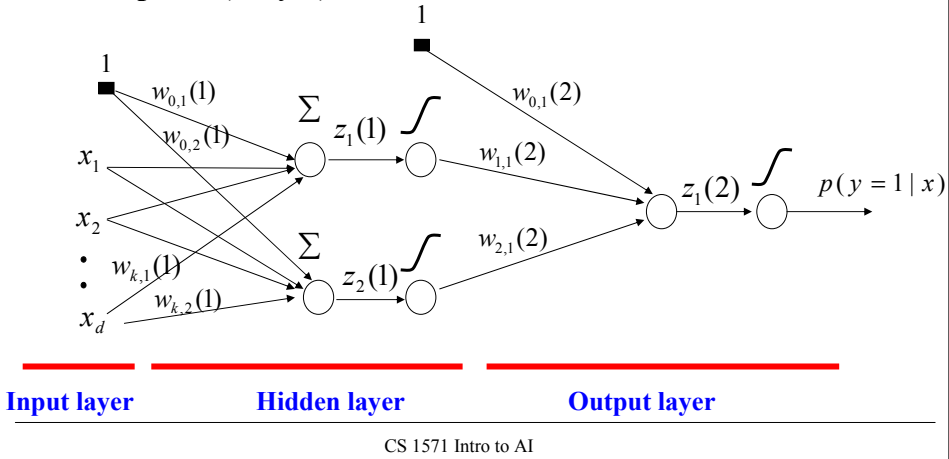- **Motivation:** from a neuron and synaptic connections.

Model of a neuron

Σ z y

1 w_0

w_1

w_2

$\mathbf{x}$ w_k

Threshold function

Multilayer neural network

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

Cascades multiple logistic regression units

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

1

$w_{0,1}(1)$

1

$w_{0,1}(2)$

Σ $z_1(1)$ $\int$

x_1 $w_{0,2}(1)$ $w_{1,1}(2)$

$z_1(2)$ $\int$ $p(y = 1 \mid x)$

x_2

Σ $w_{2,1}(2)$

$w_{k,1}(1)$ $z_2(1)$ $\int$

$w_{k,2}(1)$

x_d

Input layer   **Hidden layer**   **Output layer**
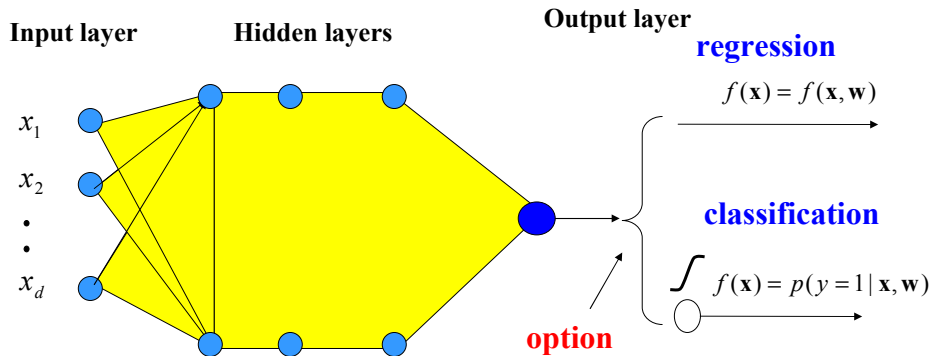

Multilayer neural network

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

Input layer   **Hidden layer**   **Output layer**

regression

1

$w_{0,1}(1)$

1

$w_{0,1}(2)$

$f(\mathbf{x}) = f(\mathbf{x}, \mathbf{w})$

Σ $z_1(1)$ $\int$

x_1 $w_{0,2}(1)$ $w_{1,1}(2)$

$z_1(2)$

x_2

Σ $w_{2,1}(2)$

classification

$w_{k,1}(1)$ $z_2(1)$ $\int$

$w_{k,2}(1)$

$\int$ $f(\mathbf{x}) = p(y = 1 \mid \mathbf{x}, \mathbf{w})$

x_d

option

Multilayer neural network

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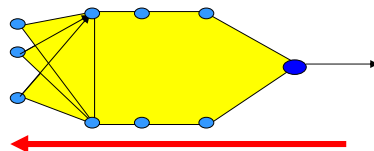- Output layer determines whether it is a **regression and binary classification problem**

Input layer　　　**Hidden layers**　　　**Output layer**

regression

$$f(\mathbf{x}) = f(\mathbf{x}, \mathbf{w})$$

x_1

x_2

$\vdots$

x_d

classification

$$\int f(\mathbf{x}) = p(y = 1 \mid \mathbf{x}, \mathbf{w})$$

option

Learning with MLP

- How to learn the parameters of the neural network?
- **Gradient descent algorithm.**
- **On-line version:** Weight updates are based on $J_{\text{online}}(D_i, \mathbf{w})$

$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} J_{\text{online}}(D_i, \mathbf{w})$$

- We need to **compute gradients for weights in all units**
- **Can be computed in one backward sweep through the net !!!**
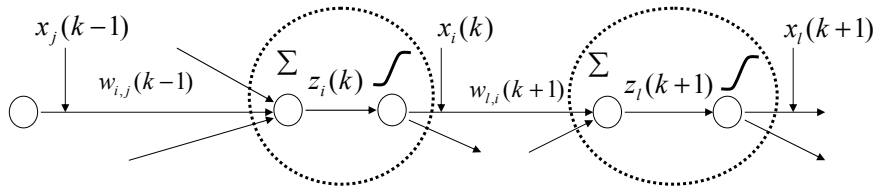
- The process is called **back-propagation**

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 $D_u = <\mathbf{x}, y>$

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

Let $\delta_i(k) = \dfrac{\partial}{\partial z_i(k)} J_{online}(D_u, \mathbf{w})$

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

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):

$$\delta_i(K) = -(y - f(\mathbf{x}, \mathbf{w}))$$

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

Learning with MLP

- **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 $D_u = <\boldsymbol{x}, y>$ from *D*

        set $\alpha = 1/i$

        compute outputs $x_j(k)$ for each unit

        compute derivatives $\delta_i(k)$ via backpropagation

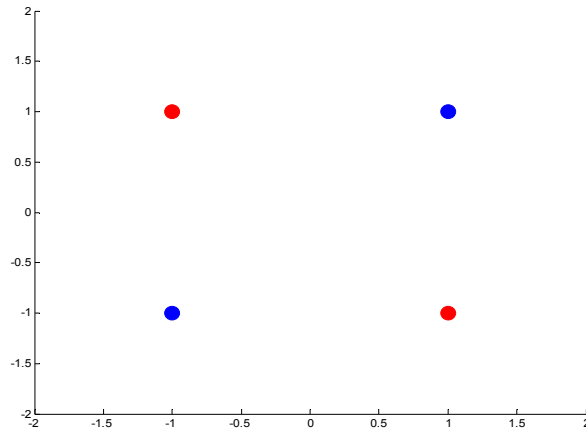   **update** all weights (in parallel)

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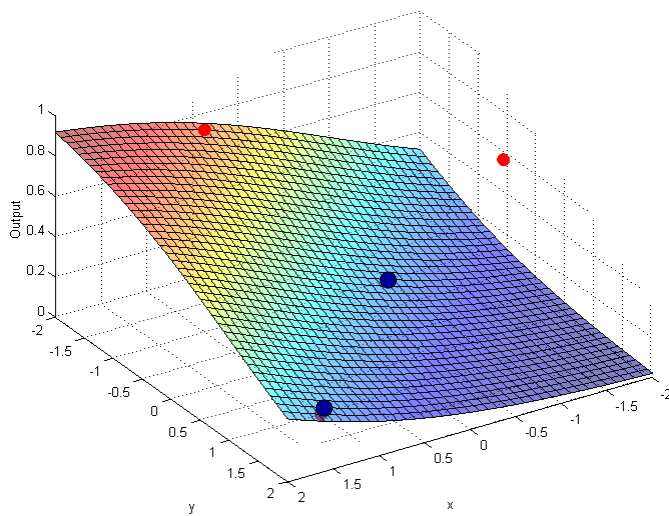
  end for

  return weights **w**

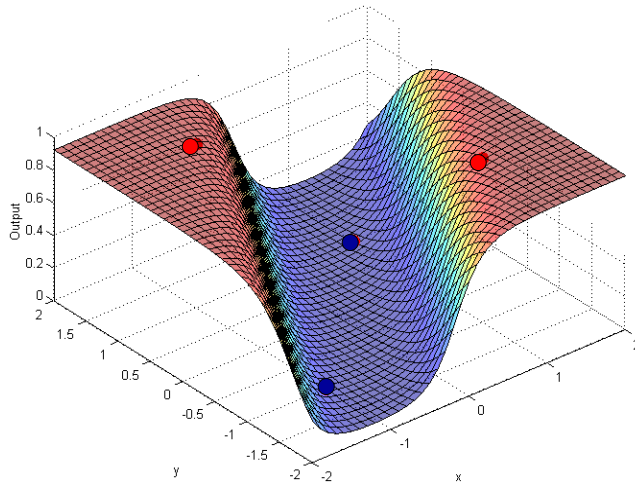# Xor Example.
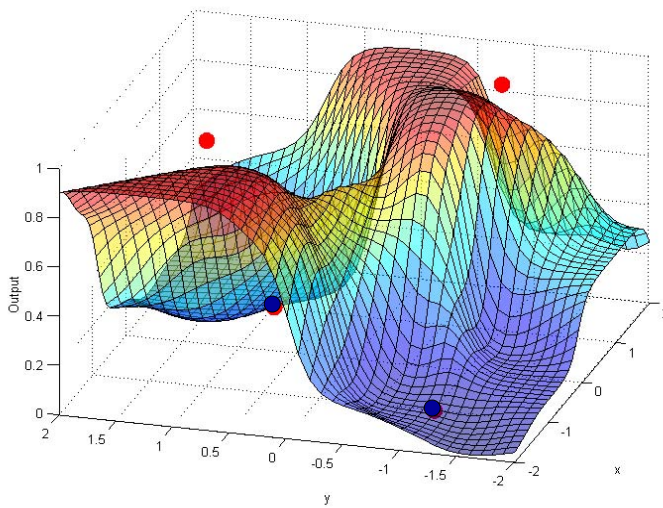
- No linear decision boundary

Xor example. Linear unit

Xor example.
Neural network with 2 hidden units

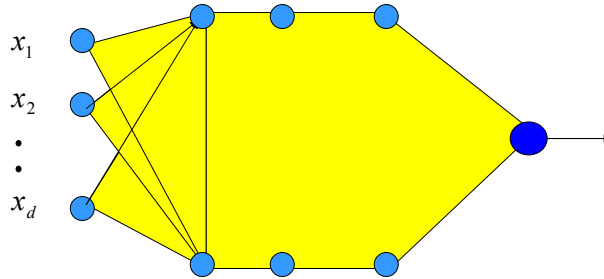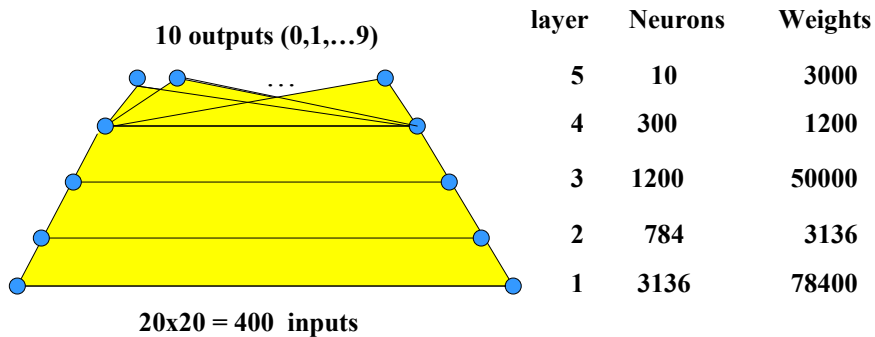# Xor example.
Neural network with 10 hidden units

Problems with learning MLPs

- Decision about the number of units must be made in advance
- Converges to local optima
- Sensitive to initial set of weights

x_1

x_2

$\vdots$

x_d

MLP in practice

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

10 outputs (0,1,…9)

…

20x20 = 400 inputs

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