CS 1571 Introduction to AI
Lecture 26

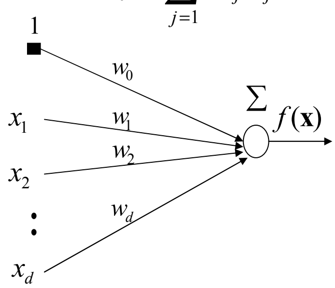
Multi-layer neural networks

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

Linear units

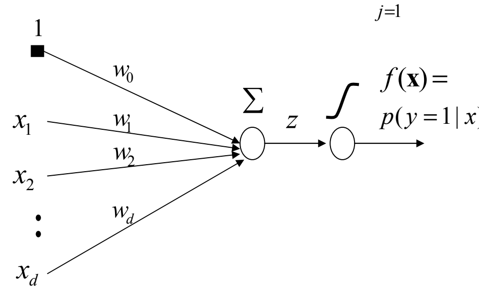
Linear regression

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



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:

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

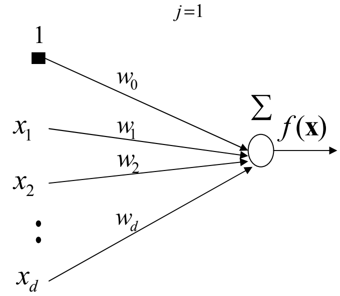
$\vdots$

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

Limitations of basic linear units

Linear regression

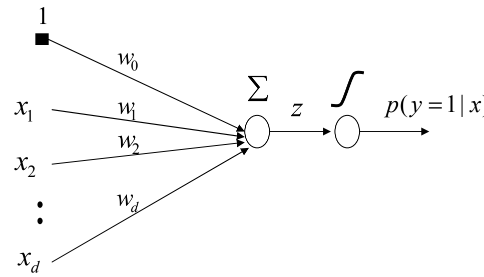
$$f(\mathbf{x}) = w_0 + \sum_{j=1}^{d} w_j x_j$$



Function linear in inputs !!

Logistic regression

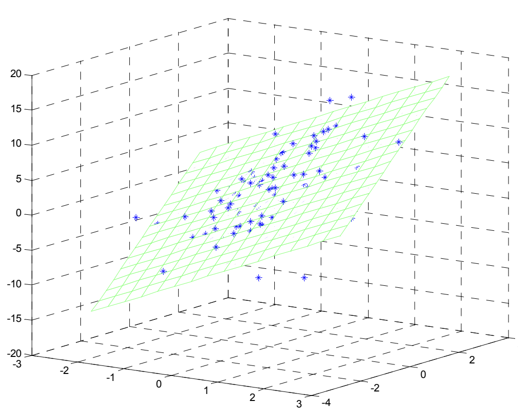
$$f(\mathbf{x}) = p(y=1 \mid \mathbf{x}, \mathbf{w}) = g(w_0 + \sum_{j=1}^{d} w_j x_j)$$



Linear decision boundary!!

Regression with the linear model.

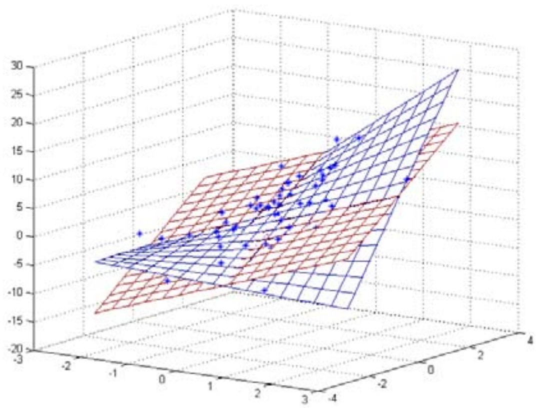
Limitation: linear hyper-plane only



Regression with the linear model.

Limitation: linear hyper-plane only

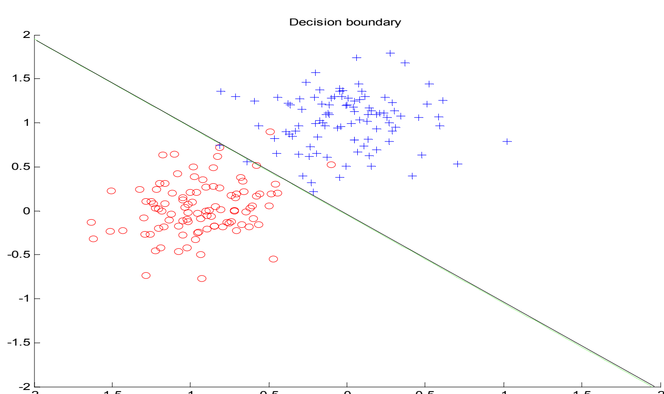
a non-linear surface can be better



Classification with the linear model.

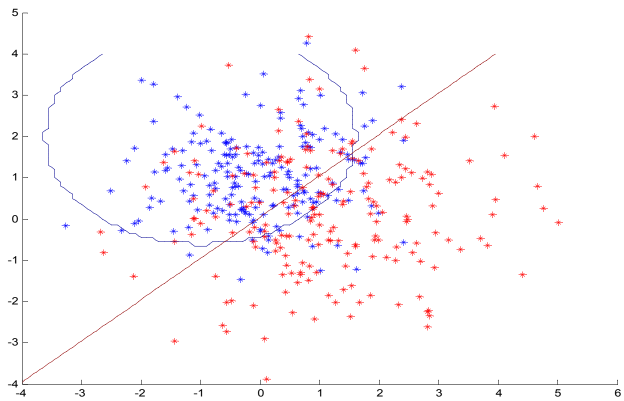
Logistic regression model defines a linear decision boundary

- Example: 2 classes (blue and red points)



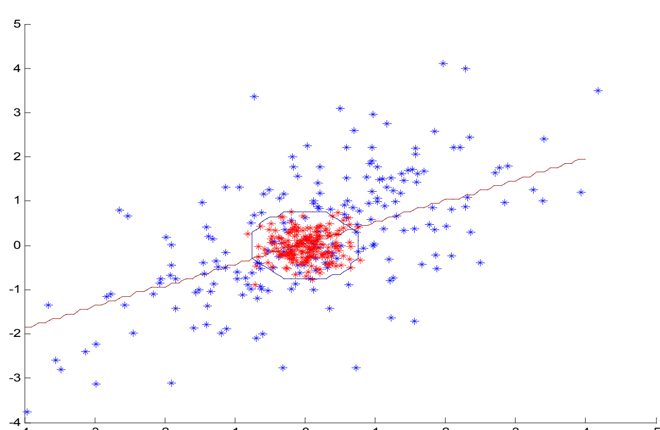
Linear decision boundary

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



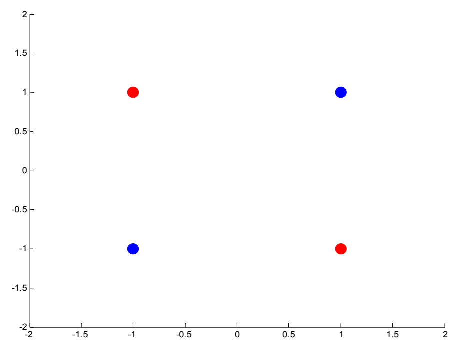
When logistic regression fails?

- 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**

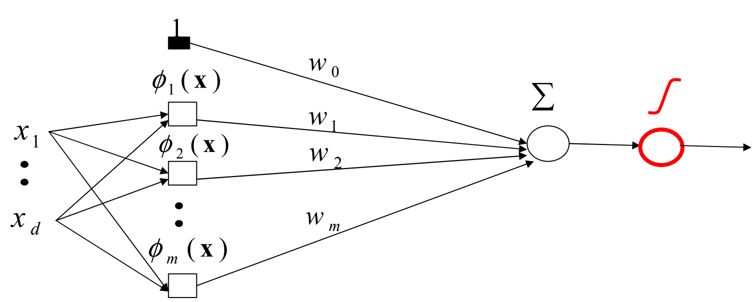
Linear regression

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

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 $\mathbf{x}$



Example. Regression with polynomials.

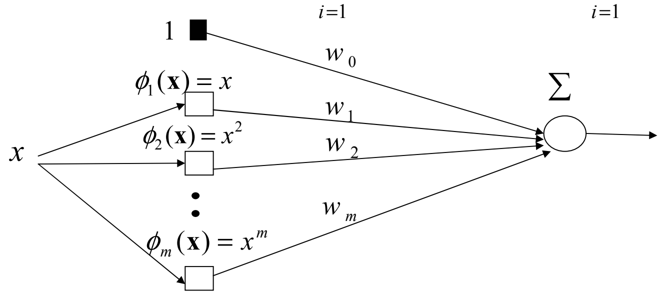
Regression with polynomials of degree m

- **Data points:** **pairs of** $< x, y >$
- **Feature functions:** **m feature functions**

$$\phi_i(x) = x^i \qquad i = 1, 2, \ldots, m$$

- **Function to learn:**

$$f(x, \mathbf{w}) = w_0 + \sum_{i=1}^{m} w_i \phi_i(x) = w_0 + \sum_{i=1}^{m} w_i x^i$$



Learning with extended linear units

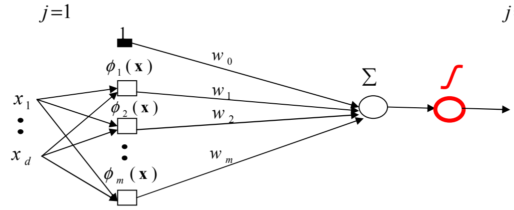
Feature (basis) functions model **nonlinearities**

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:

- Learning of weights problem is the same as it was for the linear units
- **Trick:** we have changed the inputs – the weights are still linear in the new input

Learning with feature functions.

Function to learn:

$$f(x, \mathbf{w}) = w_0 + \sum_{i=1}^{k} w_i \phi_i(x)$$

On line gradient update for the <*x,y*> pair

$$w_0 = w_0 + \alpha(y - f(\mathbf{x}, \mathbf{w}))$$

$$\vdots$$

$$w_j = w_j + \alpha(y - f(\mathbf{x}, \mathbf{w}))\phi_j(\mathbf{x})$$

Gradient updates are of the same form as in the linear and logistic regression models

Example. Regression with polynomials.

Example: Regression with polynomials of degree m

$$f(x, \mathbf{w}) = w_0 + \sum_{i=1}^{m} w_i \phi_i(x) = w_0 + \sum_{i=1}^{m} w_i x^i$$

- **On line update** for <*x,y*> pair

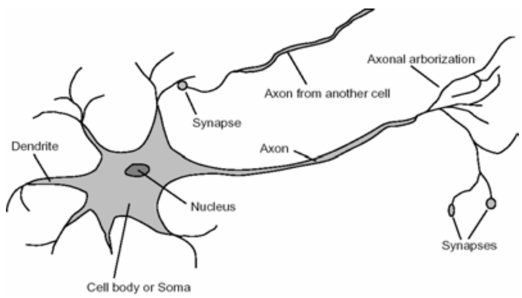
$$w_0 = w_0 + \alpha(y - f(\mathbf{x}, \mathbf{w}))$$

$$\vdots$$

$$w_j = w_j + \alpha(y - f(\mathbf{x}, \mathbf{w}))x^j$$

Multi-layered neural networks

- Alternative way to introduce nonlinearities to regression/classification models
- **Idea:** Cascade several simple neural models with logistic units. Much like neuron connections.

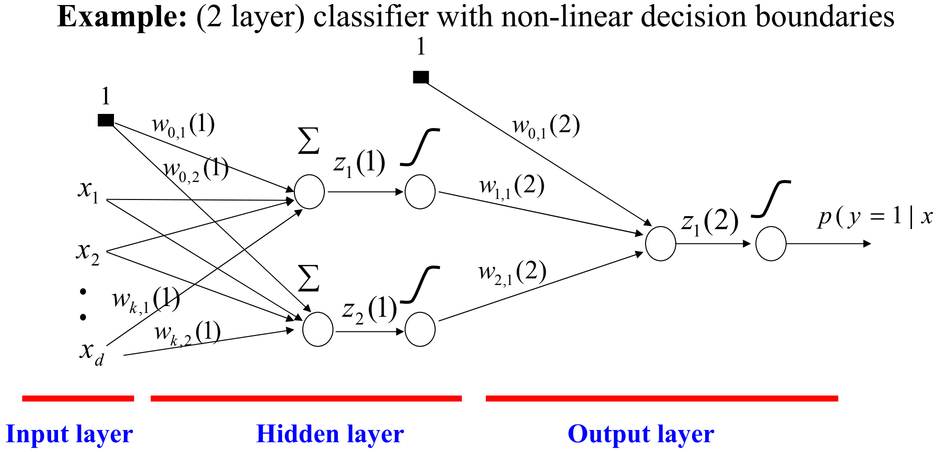


Multilayer neural network

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

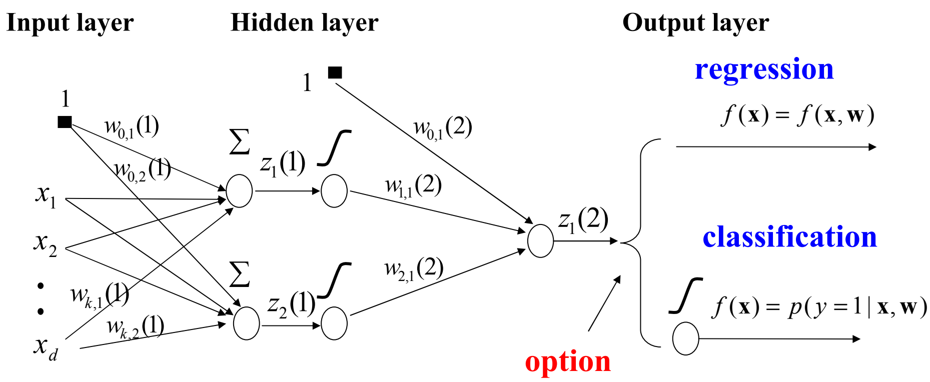
Cascades multiple logistic regression units

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



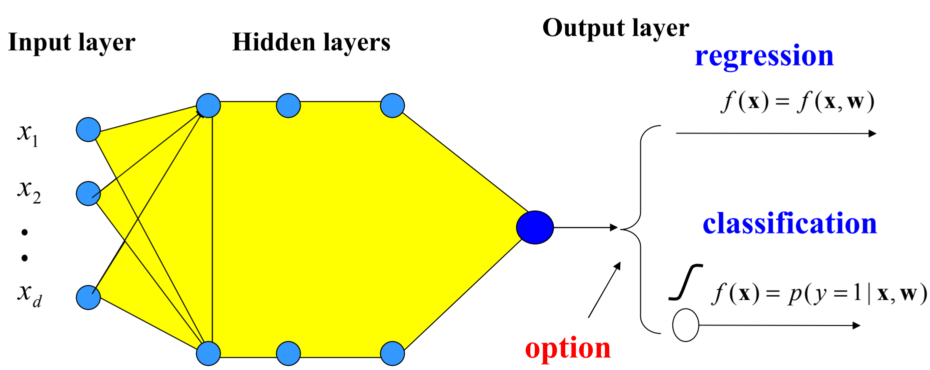
Multilayer neural network

- Models **non-linearities 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)**
- Output layer determines whether it is a **regression and binary classification problem**

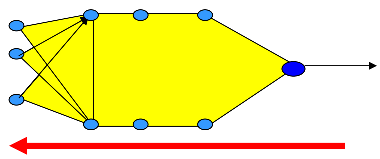


Learning with MLP

- How to learn the parameters of the neural network?
- **Online gradient descent algorithm**
  - Weight updates based on $J_{\text{online}}(D_i, \mathbf{w})$

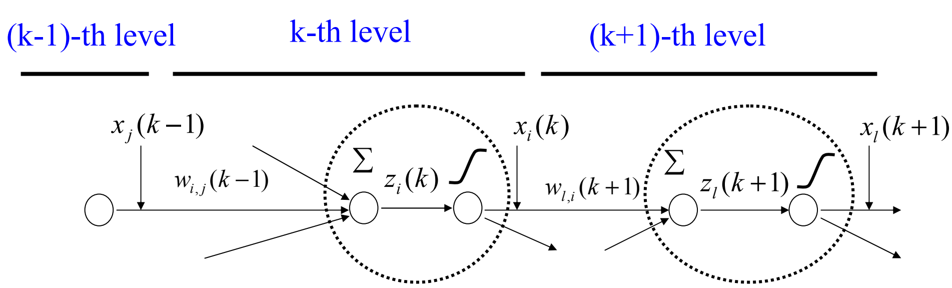
$$w_j \leftarrow w_j - \alpha \frac{\partial}{\partial w_j} J_{\text{online}}(D_i, \mathbf{w})$$

- 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



$x_i(k)$ - 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 $\delta_i(k) = \frac{\partial}{\partial z_i(k)} J_{online}(D_u, \mathbf{w})$

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

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

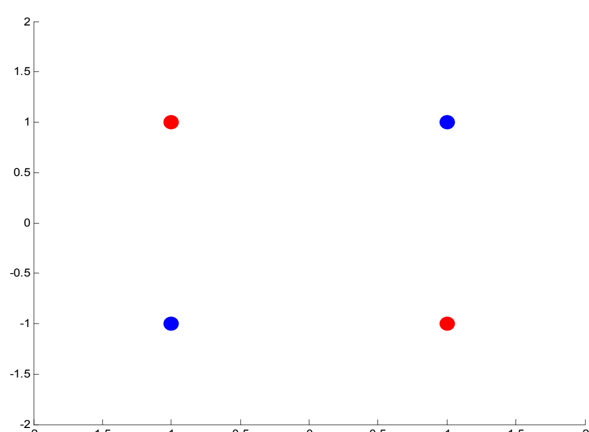
Online-gradient-descent (*D, number of iterations*)
Initialize all weights $w_{i,j}(k)$
for i=1:1*: number of iterations*
do **select** a data point D_u=<$\boldsymbol{x}$,y> from D
set $\alpha = 1/i$
compute outputs $x_j(k)$ for each unit
compute derivatives $\delta_i(k)$ via backpropagation
update all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

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

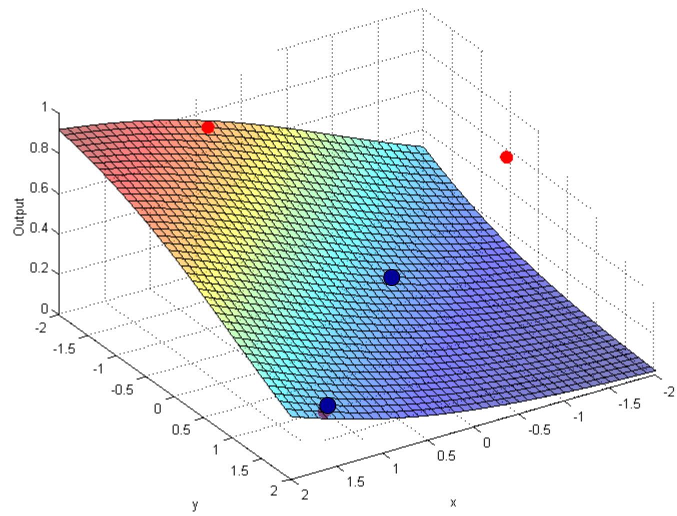
end for
return weights **w**

Xor Example.

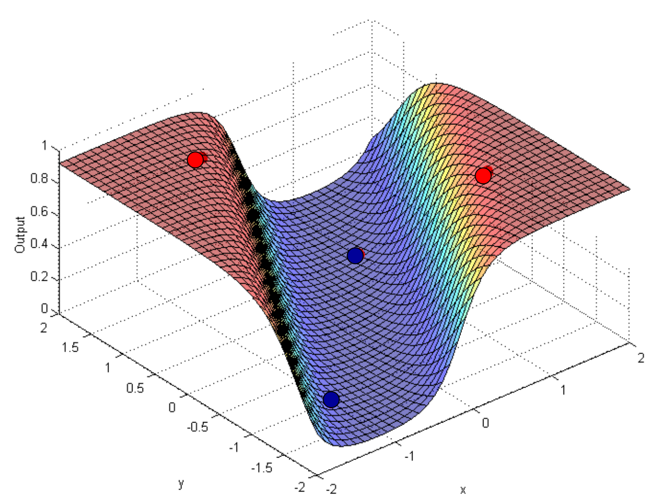
- No linear decision boundary



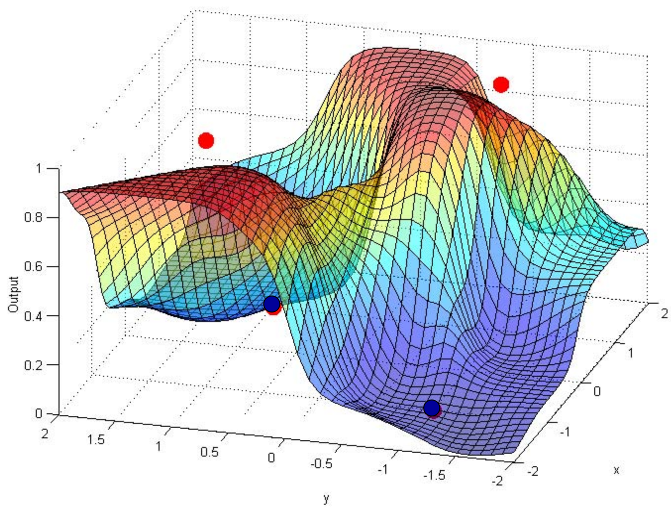
Xor example. Linear unit



Xor example.
Neural network with 2 hidden units



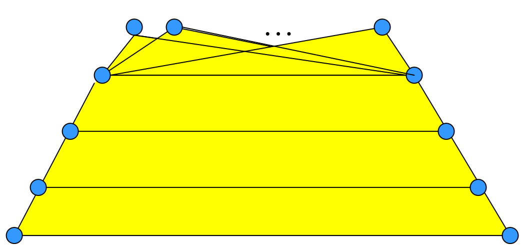
Xor example.
Neural network with 10 hidden units



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**