CS 1675 Introduction to Machine Learning
Lecture 12

Multilayer neural networks

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

Midterm exam

Midterm Thursday, March 2, 2017

- **in-class (75 minutes)**
- **closed book**

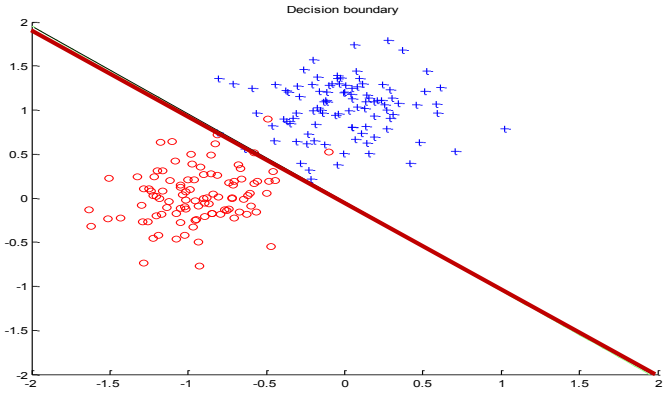
Multilayer neural networks

Or another way of modeling nonlinearities for regression and classification problems

Classification with the linear model.

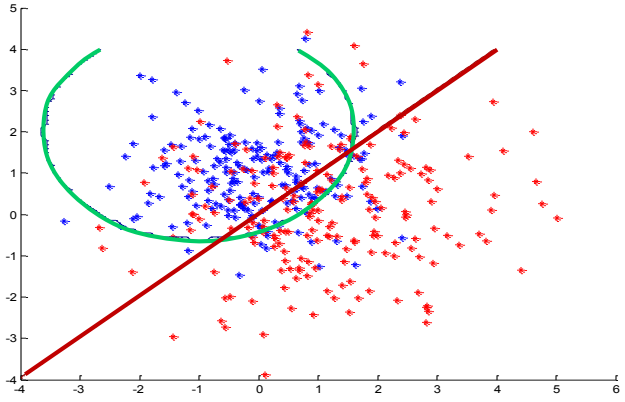
Logistic regression model defines a linear decision boundary

- Example: 2 classes (blue and red points)



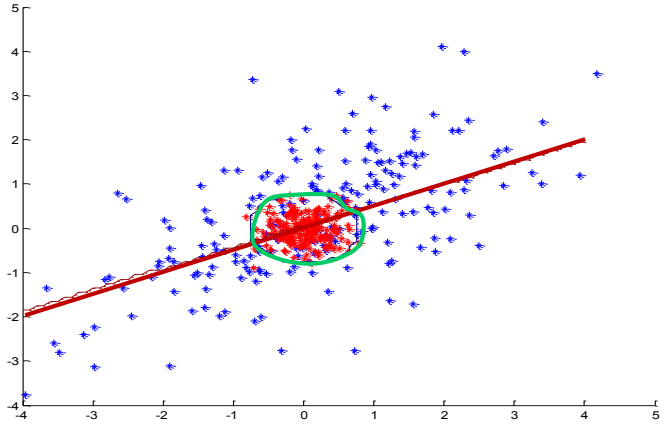
Linear decision boundary

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



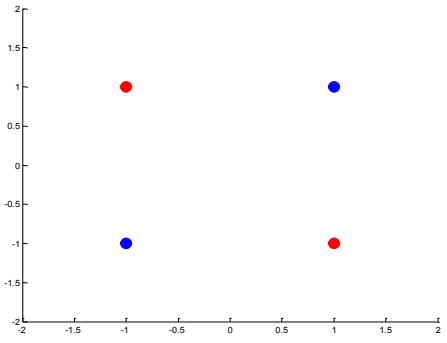
When logistic regression fails?

- 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

- **Feature (basis) functions** to model **nonlinearities**

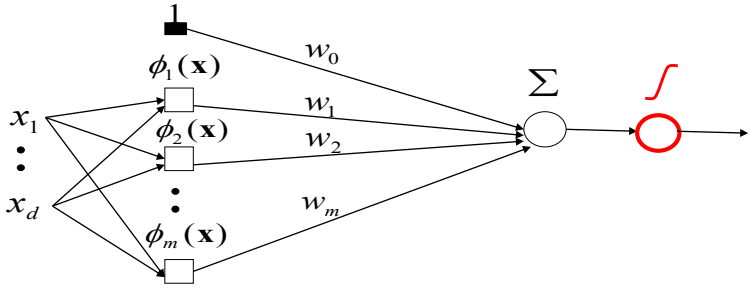
Linear regression

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

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

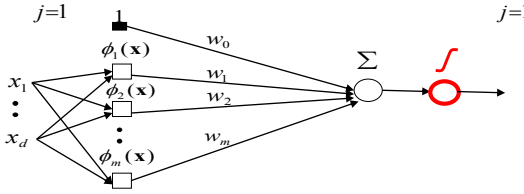
Feature (basis) functions model **nonlinearities**

Linear regression

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

Logistic regression

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



Good property:

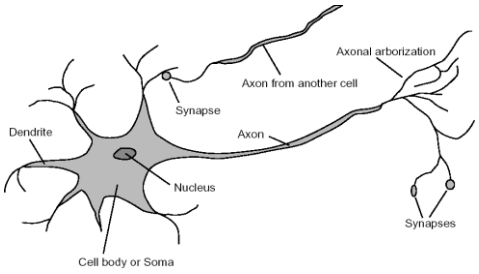
- The same problem as learning of the weights for linear units

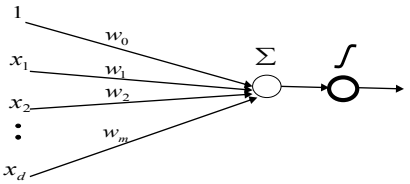
Problems:

- define the right set of basis functions
- Many basis functions → many weights to learn

Multi-layered neural networks

- An alternative way to introduce **nonlinearities to regression/classification models**
- **Key idea: Cascade several simple nonlinear switching models (e.g. logistic units) to model nonlinearities.** Much like neuron connections.



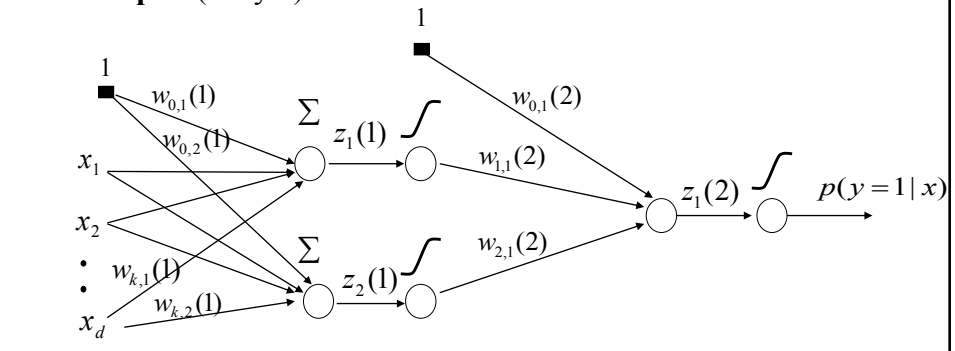


Multilayer neural network

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

Cascades multiple **logistic regression** units

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

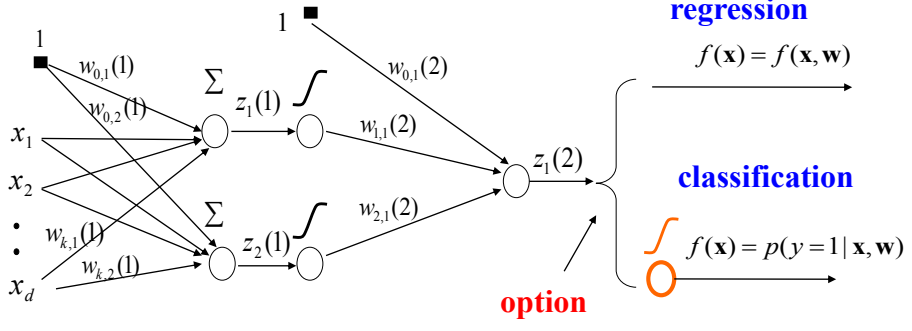


Input layer **Hidden layer** **Output layer**

Multilayer neural network

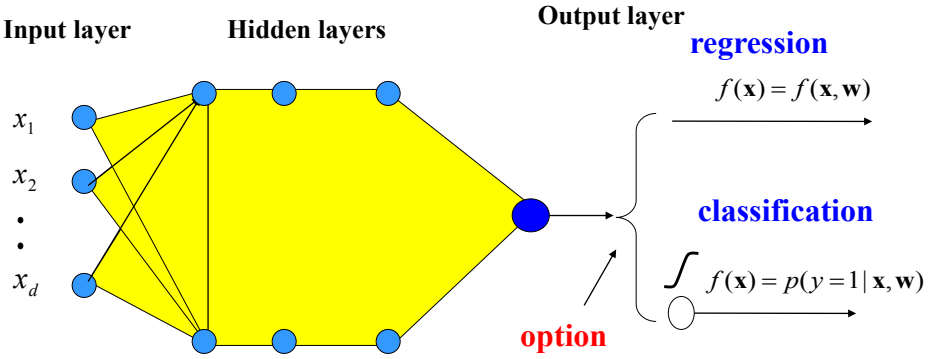
- Models **non-linearity through nonlinear switching units**
- Can be applied to both **regression and binary classification problems**

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



Multilayer neural network

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

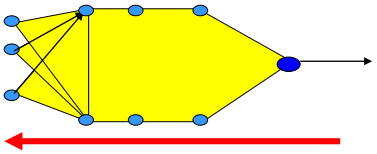


Learning with MLP

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

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

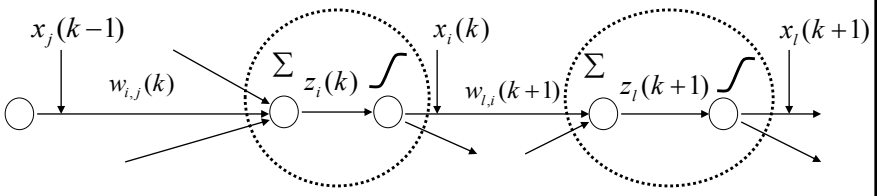
- We need to **compute gradients for weights in all units**
- **Can be computed in one backward sweep through the net !!!**



- 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

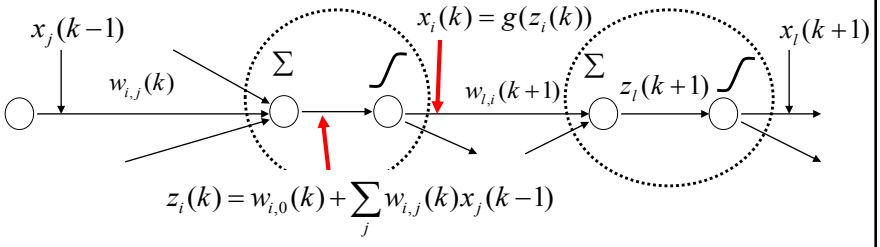
$w_{i,j}(k)$ - weight between units j and i on levels (k-1) and k

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

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

Backpropagation

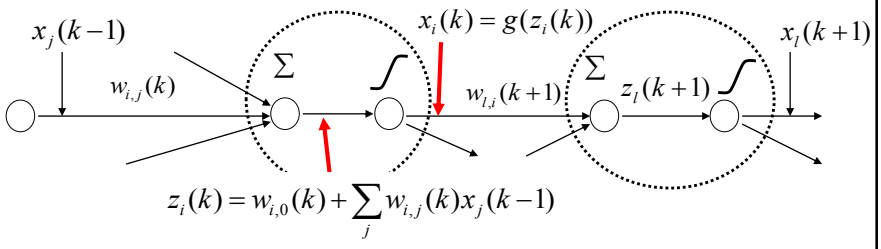
(k-1)-th level k-th level (k+1)-th level



- **Error function**: $J(D, \mathbf{w})$ (online) error where D is a data point
  - **Regression** $J(D, \mathbf{w}) = (y_u - f(\mathbf{x}_u))^2$
  - **Classification** $J(D, \mathbf{w}) = -\log p(y_u \mid \mathbf{x}_u)$
- **Gradient descent:** $w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J(D, \mathbf{w})$

Backpropagation

(k-1)-th level | k-th level | (k+1)-th level



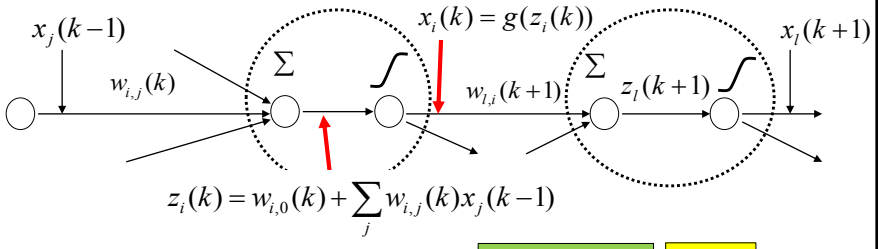
- **Gradient descent:** $w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \frac{\partial}{\partial w_{i,j}(k)} J(D, \mathbf{w})$

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

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

Backpropagation

(k-1)-th level | k-th level | (k+1)-th level

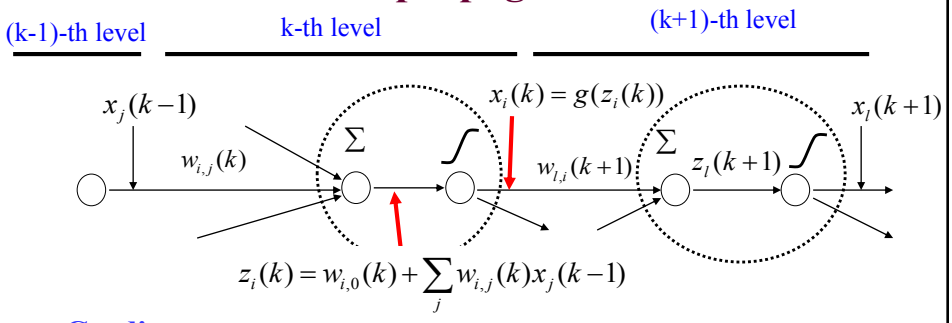


- **Derivation:** $\delta_i(k) = \frac{\partial}{\partial z_i(k)} J(D, \mathbf{w}) = \frac{\partial}{\partial x_i(k)} J(D, \mathbf{w}) * \frac{\partial x_i(k)}{\partial z_i(k)}$

$$\frac{\partial}{\partial x_i(k)} J(D, \mathbf{w}) = \sum_l \underbrace{\frac{\partial}{\partial z_l(k+1)} J(D, \mathbf{w})}_{\delta_l(k+1)} * \underbrace{\frac{\partial z_l(k+1)}{\partial x_l(k+1)}}_{w_{l,i}(k+1)} \qquad \frac{\partial x_i(k)}{\partial z_i(k)} = x_i(k)(1 - x_i(k))$$

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

Backpropagation



- **Gradient:**

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

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

- **Last unit** (is the same as for the regular linear units),

  E.g. for regression:

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

Backpropagation

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

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

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

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

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):

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

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

Learning with MLP

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

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

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

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

$x_j(k-1)$ - 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

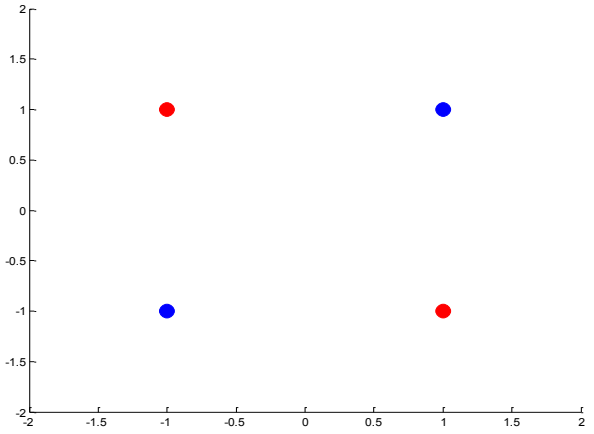
Online-gradient-descent (*D, number of iterations*)
  Initialize all weights $w_{i,j}(k)$
  for *i*=1:1*: number of iterations*
    do **select** a data point D_u=<**x**,*y*> from *D*
      set learning rate α
      compute outputs $x_j(k)$ for each unit
      compute derivatives $\delta_i(k)$ via **backpropagation**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

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

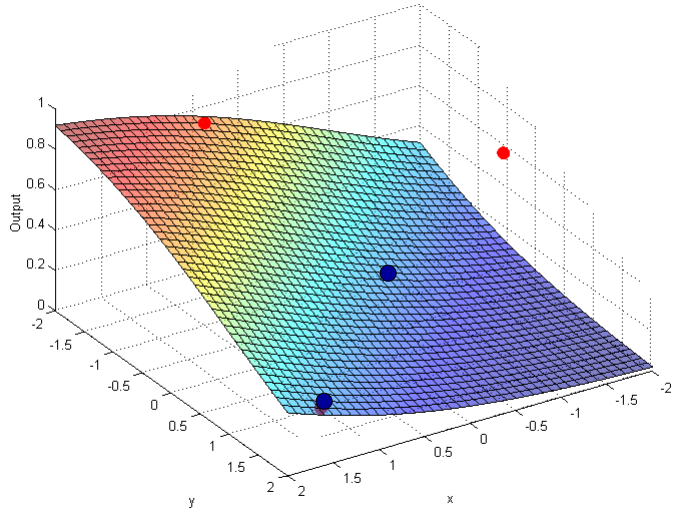
  end for
  return weights **w**

Xor Example.

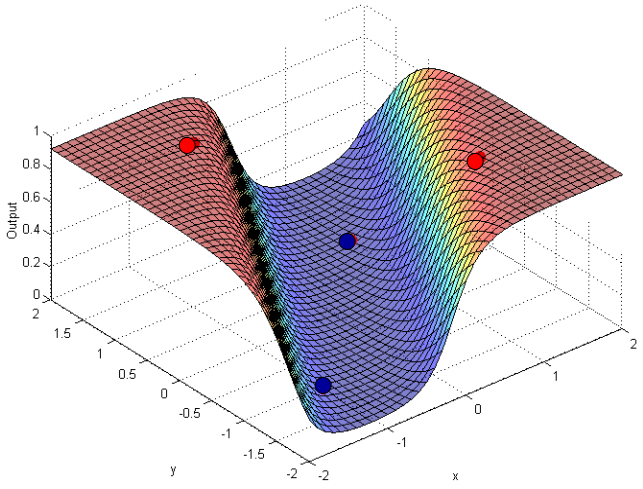
• linear decision boundary does not exist



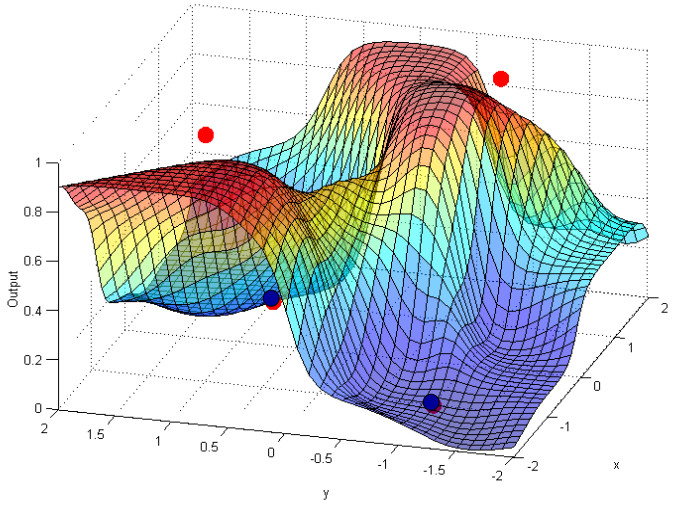
Xor example. Linear unit



Xor example.
Neural network with 2 hidden units

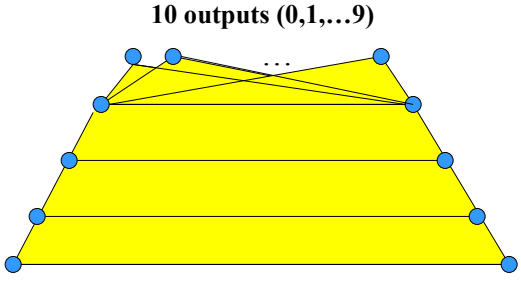


Xor example.
Neural network with 10 hidden units



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