CS 2750 Machine Learning
Lecture 10

Evaluation of classifiers
MLPs

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

Evaluation

For any data set we use to test the model we can build a **confusion matrix:**

- Counts of examples with:
- class label ω_j that are classified with a label α_i

target

predict		$\omega = 1$	$\omega = 0$
	$\alpha = 1$	140	17
	$\alpha = 0$	20	54

Evaluation

For any data set we use to test the model we can build a **confusion matrix:**

target

predict		$\omega = 1$	$\omega = 0$
	$\alpha = 1$	140	17
	$\alpha = 0$	20	54

agreement

Error: ?

Evaluation

For any data set we use to test the model we can build a confusion matrix:

target

predict		$\omega = 1$	$\omega = 0$
	$\alpha = 1$	140	17
	$\alpha = 0$	20	54

agreement

Error: = 37/231

Accuracy = 1- Error = 194/231

Evaluation for binary classification

Entries in the confusion matrix for binary classification have names:

target

predict	$\omega = 1$	$\omega = 0$
$\alpha = 1$	TP	FP
$\alpha = 0$	FN	TN

TP: True positive (hit)
FP: False positive (false alarm)
TN: True negative (correct rejection)
FN: False negative (a miss)

Additional statistics

- **Sensitivity (recall)**

$$SENS = \frac{TP}{TP + FN}$$

- **Specificity**

$$SPEC = \frac{TN}{TN + FP}$$

- **Positive predictive value (precision)**

$$PPT = \frac{TP}{TP + FP}$$

- **Negative predictive value**

$$NPV = \frac{TN}{TN + FN}$$

Binary classification: additional statistics

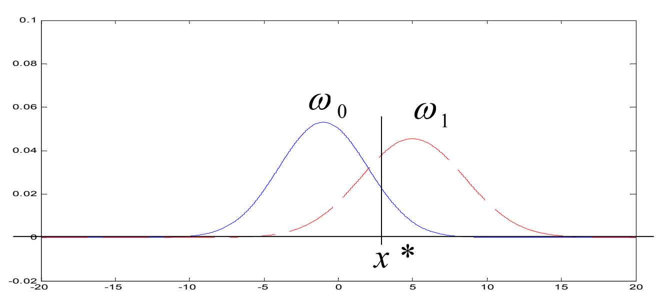
- **Confusion matrix**

		target		
		1	0	
predict	1	140	10	$PPV = 140/150$
	0	20	180	$NPV = 180/200$
		$SENS = 140/160$	$SPEC = 180/190$	

Row and column quantities:

- Sensitivity (SENS)
- Specificity (SPEC)
- Positive predictive value (PPV)
- Negative predictive value (NPV)

Binary decisions: ROC



- **Probabilities:**
  - *SENS* $p(x > x^* \mid \mathbf{x} \in \omega_1)$
  - *SPEC* $p(x < x^* \mid \mathbf{x} \in \omega_0)$

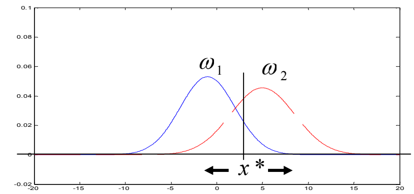
Receiver Operating Characteristic (ROC)

- **ROC curve plots :**

  SN= $p(x > x^* \mid \mathbf{x} \in \omega_1)$

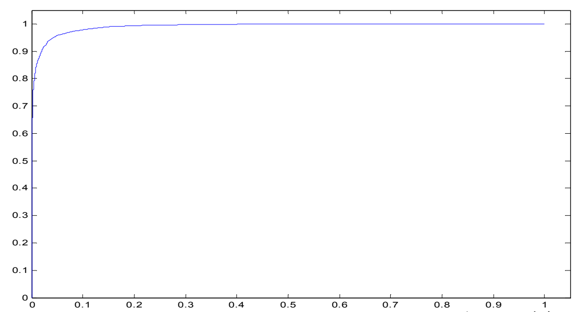
  1-SP= $p(x > x^* \mid \mathbf{x} \in \omega_0)$

  for different x*



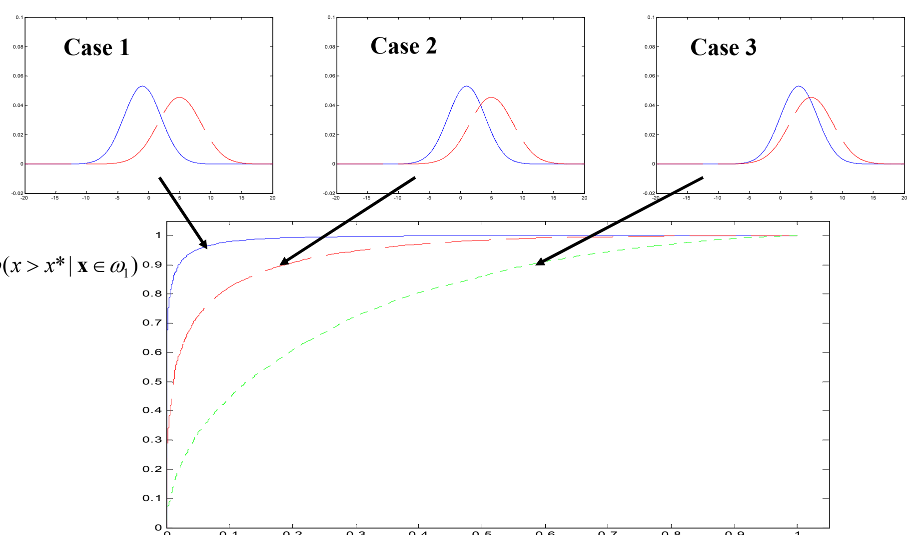
SN

$p(x > x^* \mid \mathbf{x} \in \omega_1)$



1-SPEC $p(x > x^* \mid \mathbf{x} \in \omega_0)$

ROC curve



$p(x > x^* \mid \mathbf{x} \in \omega_1)$

$p(x > x^* \mid \mathbf{x} \in \omega_0)$

Receiver operating characteristic

- **ROC**
  - shows the discriminability between the two classes under different decision biases
- **Decision bias**
  - can be changed using different loss function

Zero-one loss function

- **Misclassification error**
  - Based on the zero-one loss function
    - Any misclassified example counts as 1
    - Correctly classified example counts as 0

	$\omega = 0$	$\omega = 1$	$\omega = 2$
$\alpha = 0$	140	20	22
$\alpha = 1$	17	54	8
$\alpha = 2$	12	4	76

agreement

General loss function

- **Error function based on a more general loss function**
  - Different misclassifications have different weight (loss)
  - α_i our choice
  - ω_j true label
  - $\lambda(\alpha_i \mid \omega_j)$ loss for classification

Example:

$\lambda(\alpha_i \mid \omega_j)$

	$\omega = 0$	$\omega = 1$	$\omega = 2$
$\alpha = 0$	0	1	5
$\alpha = 1$	3	0	2
$\alpha = 2$	3	1	0

Bayesian decision theory

- **More general loss function**
  - Different misclassifications have different weight (loss) $\lambda(\alpha_i \mid \omega_j)$
- **Expected loss for the classification choice** α_i

$$R(\alpha_i \mid \mathbf{x}) = \sum_j \lambda(\alpha_i \mid \omega_j) P(y = \omega_j \mid \mathbf{x})$$

  - Also called conditional risk
- **Decision rule:** $\alpha(\mathbf{x})$
  - Chooses label (action) according to the input
- **The optimal decision rule**

$$\alpha^*(\mathbf{x}) = \arg\min_{\alpha_i} \sum_j \lambda(\alpha_i \mid \omega_j) P(y = \omega_j \mid \mathbf{x})$$

Bayesian decision theory

- **The optimal decision rule**

$$\alpha^*(\mathbf{x}) = \arg\min_{\alpha_i} \sum_j \lambda(\alpha_i \mid \omega_j) P(y = \omega_j \mid \mathbf{x})$$

How to modify classifiers to handle different loss?

- **Discriminative models:**
  – Directly optimize the parameters according to the new loss function
- **Generative models:**
  – Learn probabilities as before
  – Decisions about classes are biased to minimize the empirical loss (as seen above)

Calculating the loss for data

- **Confusion matrix:**
  – Counts of examples with:
  – class label ω_j that are classified with a label α_i

	$\omega = 0$	$\omega = 1$	$\omega = 2$
$\alpha = 0$	140	20	22
$\alpha = 1$	17	54	8
$\alpha = 2$	12	4	76

agreement

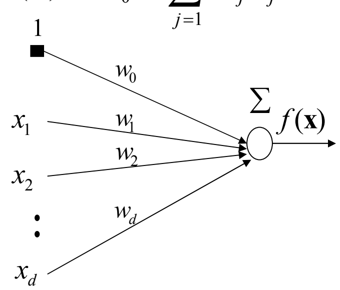
- **Loss** $\frac{1}{N} \sum_i \sum_j \lambda(\alpha_i \mid \omega_j) N(\alpha_j \mid \omega_j)$

Multilayer neural networks

Linear units

Linear regression

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



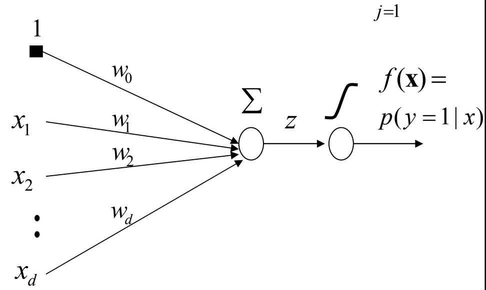
On-line gradient update:

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

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

Logistic regression

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



On-line gradient update:

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

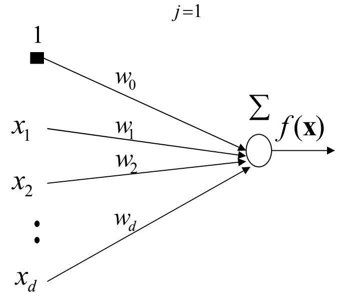
$$w_j \leftarrow w_j + \alpha(y - f(\mathbf{x}))x_j$$

The same

Limitations of basic linear units

Linear regression

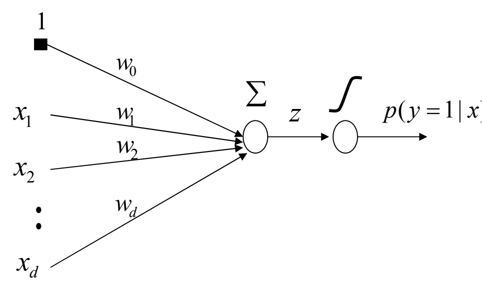
$$f(\mathbf{x}) = w_0 + \sum_{j=1}^{d} w_j x_j$$



Function linear in inputs !!

Logistic regression

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

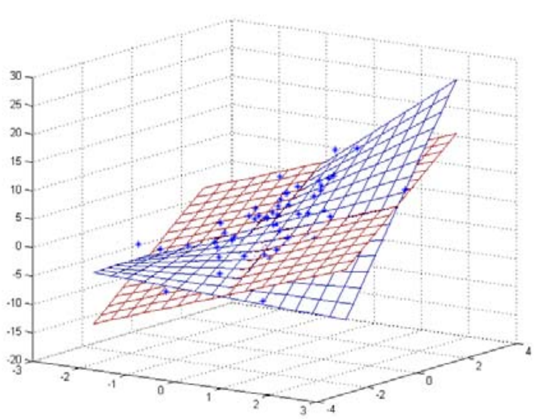


Linear decision boundary!!

Regression with the quadratic model.

Limitation: linear hyper-plane only

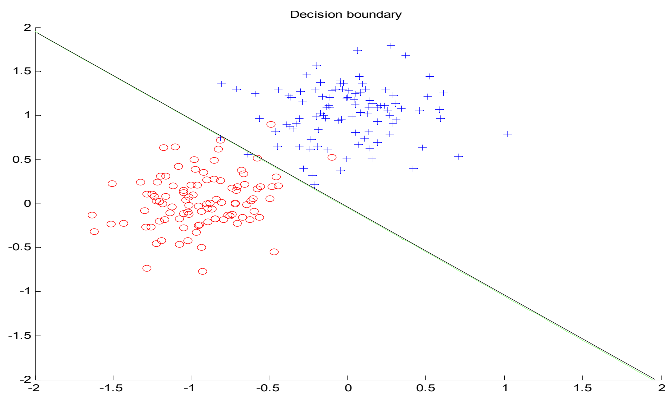
- a non-linear surface can be better



Classification with the linear model.

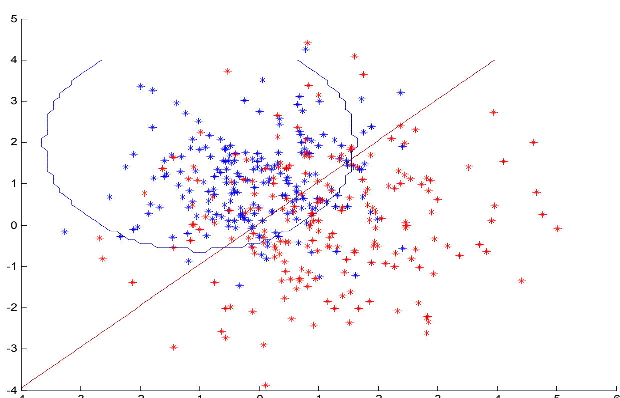
Logistic regression model defines a linear decision boundary

- Example: 2 classes (blue and red points)



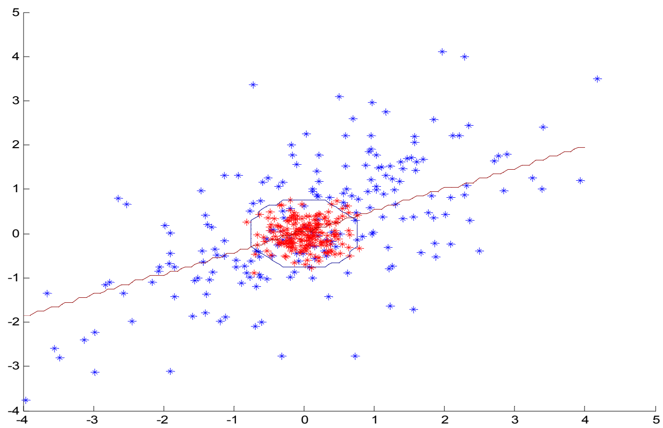
Linear decision boundary

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



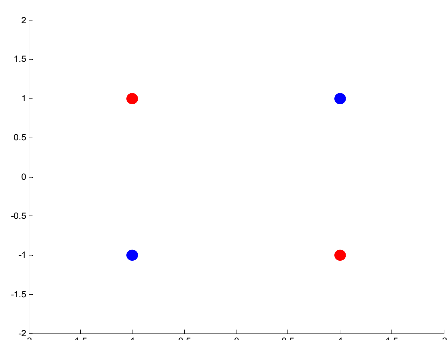
When logistic regression fails?

- Example in which the logistic regression model fails



Limitations of linear units.

- Logistic regression does not work for **parity function**s
  - 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**

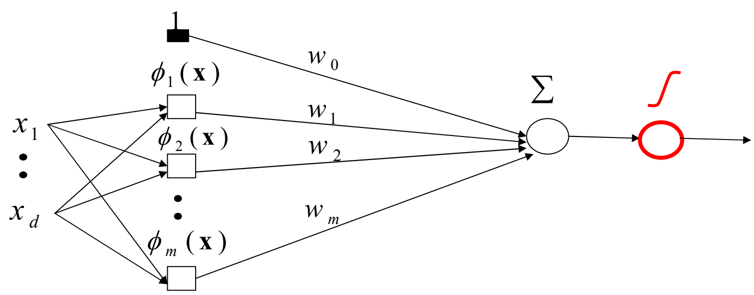
Linear regression

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

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

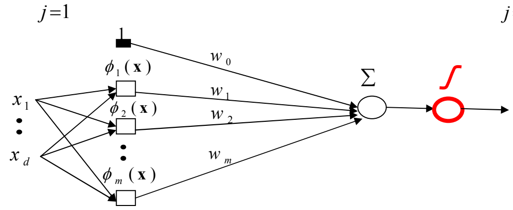
Feature (basis) functions model **nonlinearities**

Linear regression

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

Logistic regression

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



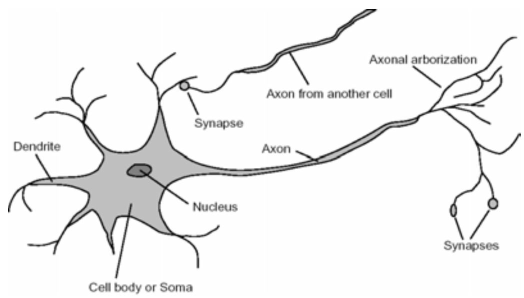
Important property:

- 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

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

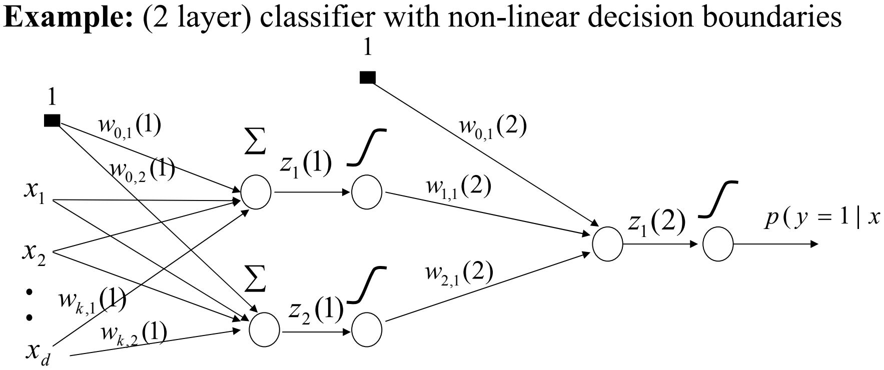


Multilayer neural network

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

Cascades multiple logistic regression units

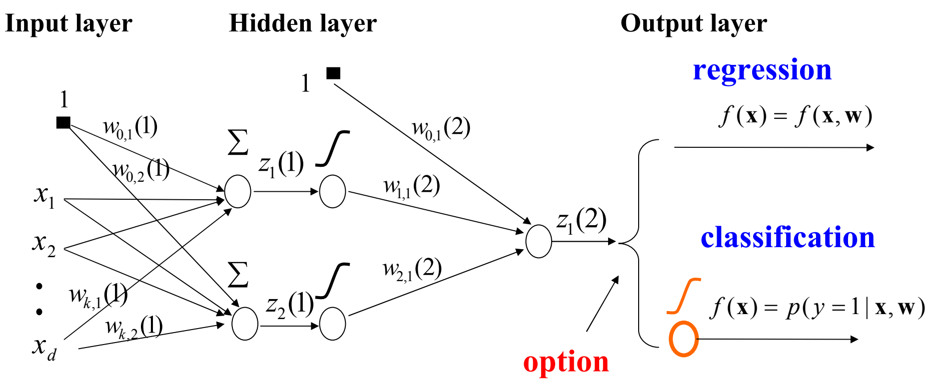
Example: (2 layer) classifier with non-linear decision boundaries



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

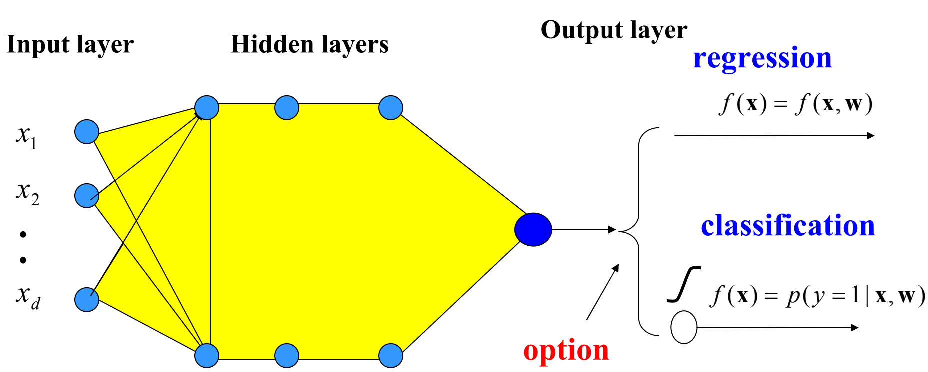
Multilayer neural network

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



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**

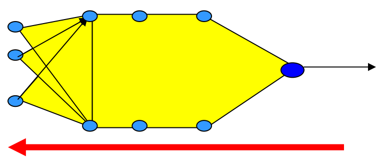


Learning with MLP

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

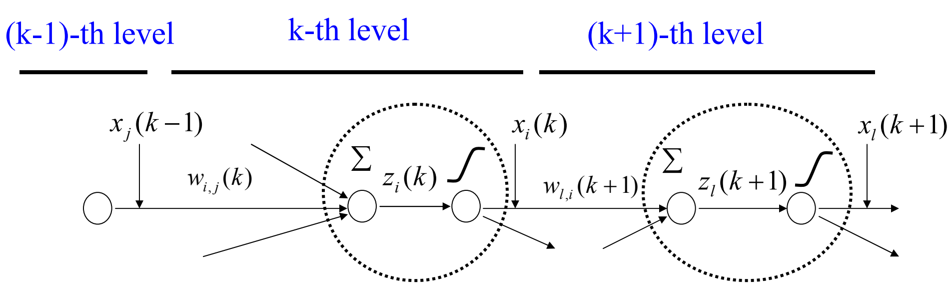
$$\mathbf{w} \leftarrow \mathbf{w} - \alpha \nabla_{\mathbf{w}} J(D, \mathbf{w})$$

- 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 backpropagation

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=<$\mathbf{x}$,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)

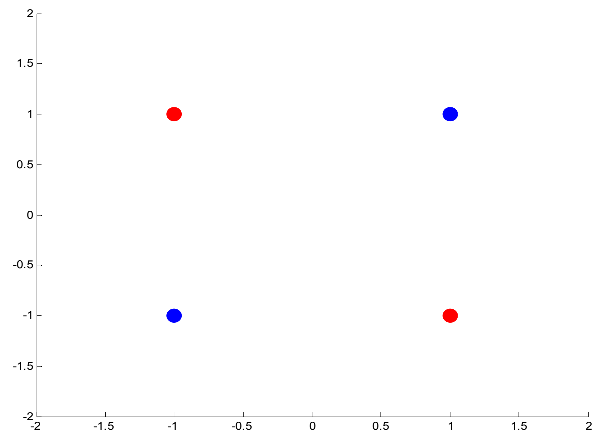
$$w_{i,j}(k) \leftarrow w_{i,j}(k) - \alpha \delta_i(k) x_j(k-1)$$

end for

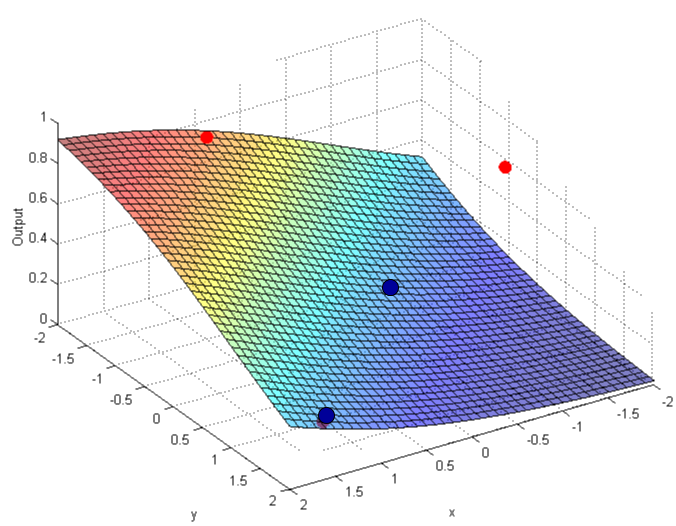
return weights **w**

Xor Example.

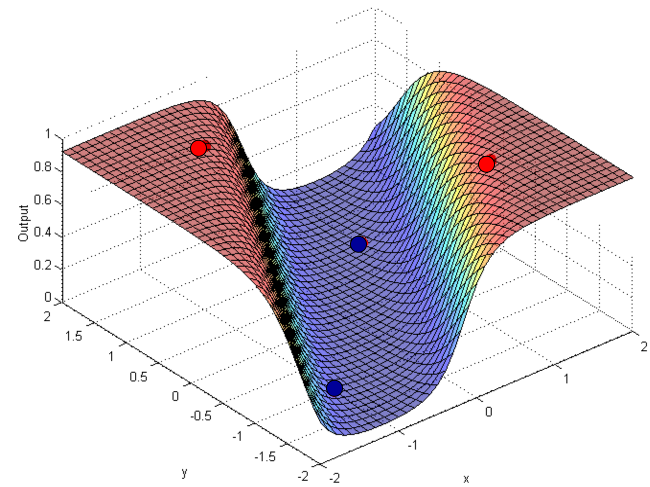
• linear decision boundary does not exist



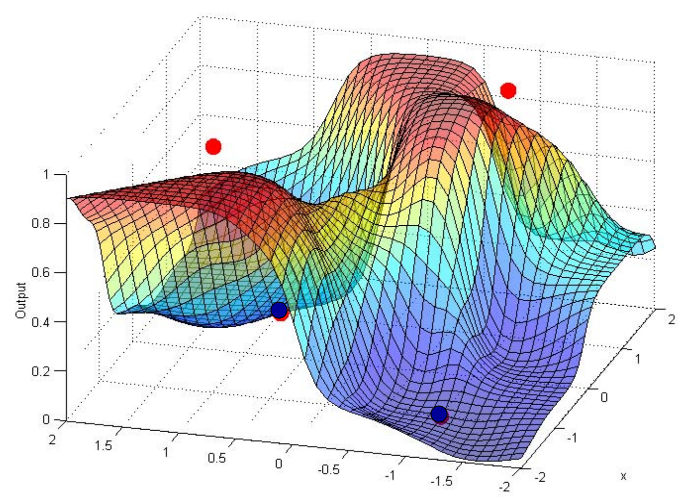
Xor example. Linear unit



Xor example.
Neural network with 2 hidden units

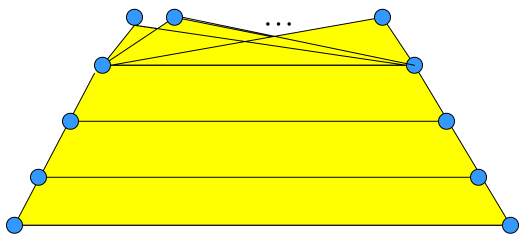


Xor example.
Neural network with 10 hidden units



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