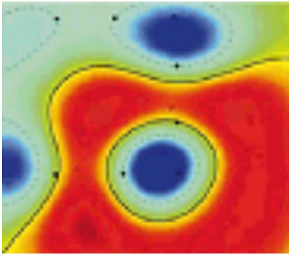
Technische Universität Berlin



Lecture 8

Neural Networks 1

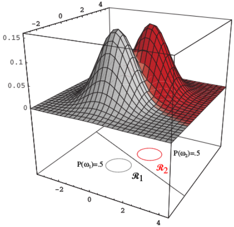
Outline

- Recap:
  - Bayes Optimal Classifier
  - Maximum Mean Separation
  - Fisher Discriminant
- Artificial Neural Network
  - The Perceptron
  - Neurons
  - Forward propagation
  - Optimizing neural networks
  - Error backpropagation

Recap: Bayes Optimal Classifier

- Assume our data is generated for each class ω_j according to the multivariate Gaussian distribution $p(\mathbf{x}|\omega_j) = \mathcal{N}(\boldsymbol{\mu}_j, \Sigma)$ and with class priors $P(\omega_j)$. The Bayes optimal classifier is derived as

$$
\begin{aligned}
&\arg\max_j \{P(\omega_j|\mathbf{x})\} \\
&= \arg\max_j \{\log p(\mathbf{x}|\omega_j) + \log P(\omega_j)\} \\
&= \arg\max_j \left\{\mathbf{x}^\top \Sigma^{-1} \boldsymbol{\mu}_j^\top - \frac{1}{2}\boldsymbol{\mu}_j \Sigma^{-1} \boldsymbol{\mu}_j + \log P(\omega_j)\right\}
\end{aligned}
$$

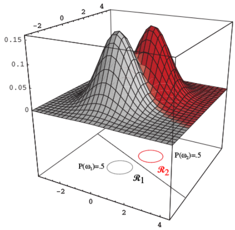


- Given our generative assumptions, there is no more accurate classifier than the one above.

Recap: Bayes Optimal Classifier

Limitations:

- In practice, we don't know the data generating distributions and only have the data.
- Estimating the data-generating distribution from a limited number of observations is difficult (e.g. it is hard to estimate the covariance of a Gaussian distribution in a way that the covariance remains invertible).



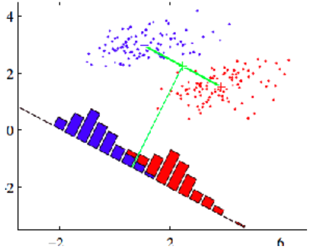
Recap: Mean Separation Criterion

- We want to learn a projection of the data $z_k = \mathbf{w}^\top \mathbf{x}_k$ with $\|\mathbf{w}\| = 1$ such that the means of classes in projected space are as distant as possible.
- First, we compute the means in projected space for the two classes

$$\mu_1 = \frac{1}{N_1} \sum_{k \in \mathcal{C}_1} z_k \qquad \mu_2 = \frac{1}{N_2} \sum_{k \in \mathcal{C}_2} z_k$$

- Then we find $\mathbf{w}$ that maximizes the difference of means, i.e. we express the means as a function of $\mathbf{w}$ and pose the optimization problem:

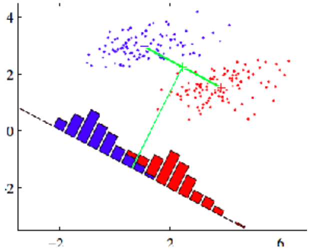
$$\arg\max_{\mathrm{w}} |\mu_2(\mathbf{w}) - \mu_1(\mathbf{w})| \qquad \text{with} \quad \|\mathbf{w}\| = 1$$



Recap: Mean Separation Criterion

Limitations:

- There is a significant class overlap in projected space.
- A better classifier seems achievable if we rotate the projection by a few degrees clockwise.
- Making means distant may not be sufficient to induce class separability in projected space.



Recap: Fisher Discriminant

R.A. Fisher (1890 - 1962)

Idea:

- In addition to maximizing the separation between class means in projected space, also consider to reduce the within-class variance.

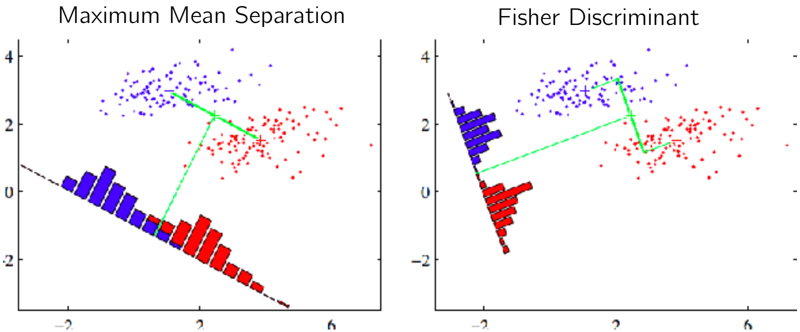
$$\mu_1 = \frac{1}{|\mathcal{C}_1|} \sum_{k \in \mathcal{C}_1} z_k \qquad \mu_2 = \frac{1}{|\mathcal{C}_2|} \sum_{k \in \mathcal{C}_2} z_k$$

$$s_1 = \sum_{k \in \mathcal{C}_1} (z_k - \mu_1)^2 \qquad s_2 = \sum_{k \in \mathcal{C}_2} (z_k - \mu_2)^2$$

- Maximizing distance between means while minimizing within-class variance can be formulated as:

$$\arg\max_{w} \frac{(\mu_2(\mathbf{w}) - \mu_1(\mathbf{w}))^2}{s_1(\mathbf{w}) + s_2(\mathbf{w})}$$

Recap: Means vs. Fisher

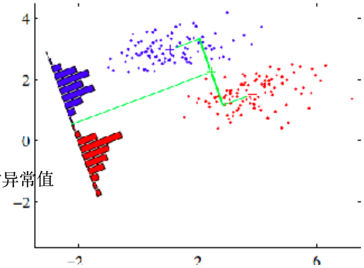


- Fisher Discriminant leads (in general) to better class separability, and therefore, better classification accuracy.
- Fisher Discriminant requires inversion of a covariance matrix (only tractable for low-dimensional data).

Recap: Fisher Discriminant

Limitations:

- The resulting decision boundary can become suboptimal when the data is not Gaussian.
- In particular, like principal component analysis, Fisher Discriminant is not robust to outliers. 对异常值不鲁棒，也就是说，该方法对异常值很敏感，容易受到异常值的影响。
- When the distribution is non-Gaussian, the model does not focus on optimizing the classification error directly.



ML1 Roadmap

The Perceptron

F. Rosenblatt (1928–1971)

- Proposed by F. Rosenblatt in 1958.
- Classifier that prefectly separates training data (if the data is linearly separable).
- Trained using an simple and cheap iterative procedure.
- The perceptron gave rise to artificial neural networks.

The Perceptron Algorithm

- Consider our linear model

$$z_k = \mathbf{w}^\top \mathbf{x}_k + b \qquad y_k = \text{sign}(z_k)$$

  and let t_k be 1 and -1 when the true class of $\mathbf{x}_k$ is ω_1 and ω_2 respectively.

Algorithm

- Iterate over $k = 1 \dots, N$ (multiple times).
  - If $\mathbf{x}_k$ is correctly classified($y_k = t_k$), continue.
  - If $\mathbf{x}_k$ is wrongly classified ($y_k \neq t_k$), apply:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta \cdot \mathbf{x}_k t_k$$
$$b \leftarrow b + \eta \cdot t_k$$

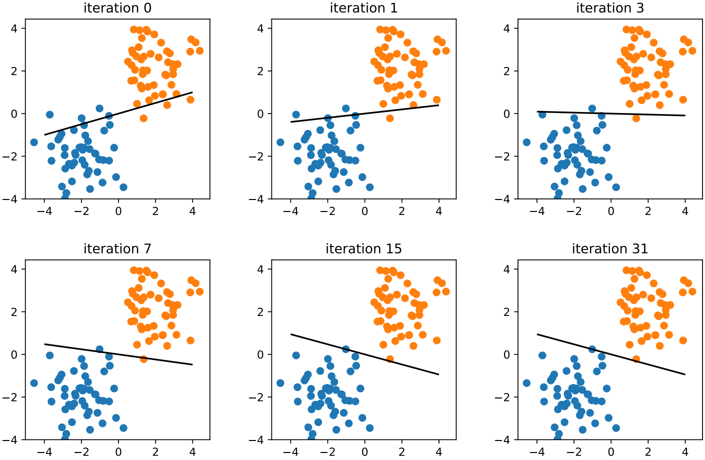
    where η is a learning rate.
- Stop once all examples are correctly classified.

The Perceptron: Optimization View

The perceptron can be seen as a gradient descent of the error function

$$\mathcal{E}(\mathbf{w}, b) = \frac{1}{N} \sum_{k=1}^{N} \underbrace{\max(0, -z_k t_k)}_{\mathcal{E}_k(\mathbf{w}, b)}$$

Perceptron at Work



Nonlinear Classification

Observation:

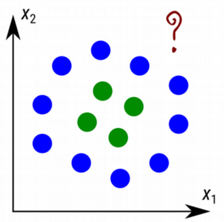
- Mean separation, Fisher discriminant, and the perceptron, build a decision function which is linear in input space. In practice, the data may not be linearly separable.

Key Idea:

- Transform the data nonlinearly through some function Φ before applying the linear decision function.

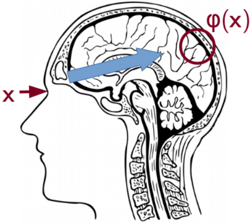
$$f(\mathbf{x}) = \mathbf{w}^\top \Phi(\mathbf{x}) + b$$

- *Example:* $\Phi(\mathbf{x}) = [x_1, x_2, x_1^2, x_2^2, x_1 x_2]$ and $\mathbf{w} \in \mathbb{R}^5$.

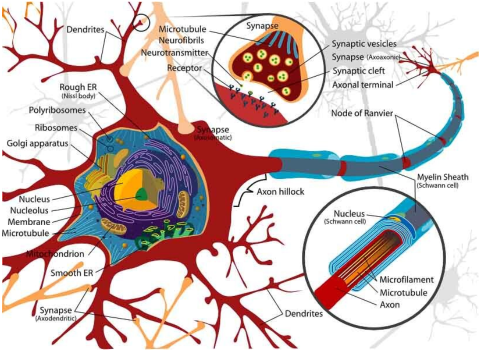


Artificial Neural Networks

- Models that are inspired by the way the brain represents sensory input and learn from repeated stimuli.
- Neuron activations can be seen as a nonlinear transformation of the sensory input (similar to $\Phi(\mathbf{x})$).
- The neural representation adapts itself after repeated exposure to certain stimuli (plasticity).

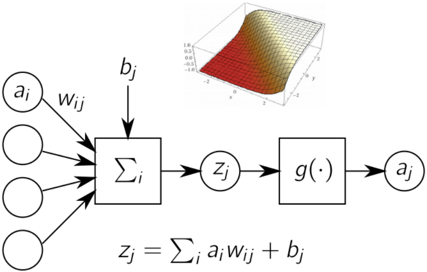


The Biological Neuron



- Highly sophisticated physical system with complex spatio-temporal dynamics that transfers signal received by dendrites to the axon.
- Human brain is composed of a very large number of neurons (approx. 86 billions) that are interconnected (150 trillions synapses).

The Artificial Neuron

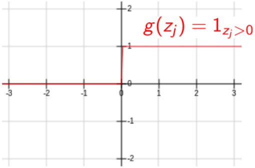


$$z_j = \sum_i a_i w_{ij} + b_j$$
$$a_j = g(z_j)$$

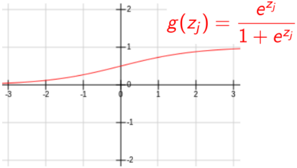
- Simple multivariate, nonlinear and differentiable function.
- Ultra-simplification of the biological neuron that retains two key properties: (1) ability to produce complex nonlinear representations when many neurons are interconnected (2) ability to learn from the data.

Examples of Nonlinear Functions

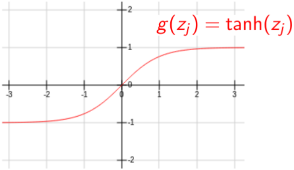
threshold function



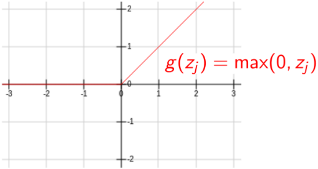
logistic sigmoid



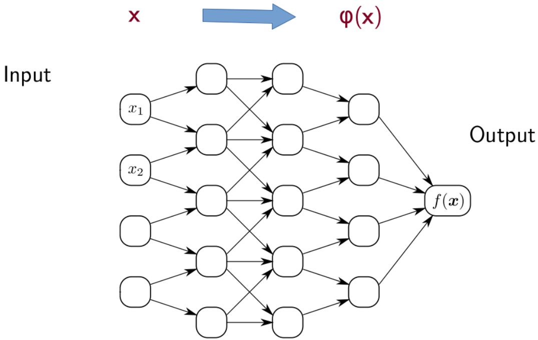
hyperbolic tangent



rectified linear unit



Example of a Neural Network



- Artificial neural networks are typically connected in some regular manner, e.g. sequences of layers.
- Number of neurons in an neural networks varies from a few neurons for simple tasks up to millions of neurons for image classifiers.

The Forward Pass

Universal Approximation Theorem (1)

Theorem (simplified): With sufficiently many neurons, neural networks can approximate any nonlinear functions.

"Visual Proof":

Universal Approximation Theorem (2)

Theorem (simplified): With sufficiently many neurons, neural networks can approximate any nonlinear functions.

Sketch proof taken from the book Bishop'95 Neural Network for Pattern Recognition, p. 130–131, (after Jones'90 and Blum&Li'91):

- Consider the special class of functions $y : \mathbb{R}^2 \to \mathbb{R}$ where input variables are called x_1, x_2.
- We will show that any two-layer network with threshold functions as nonlinearity can approximate $y(x_1, x_2)$ up to arbitrary accuracy.
- We first observe that any function of x_2 (with x_1 fixed) can be approximated as an infinite Fourier series.

$$y(x_1, x_2) \simeq \sum_s A_s(x_1) \cos(s x_2)$$

Universal Approximation Theorem (3)

- We first observe that any function of x_2 (with x_1 fixed) can be approximated as an infinite Fourier series.

$$y(x_1, x_2) \simeq \sum_s A_s(x_1) \cos(sx_2)$$

- Similarly, the coefficients themselves can be expressed as an infinite Fourier series:

$$y(x_1, x_2) \simeq \sum_s \sum_l A_{sl} \cos(lx_1) \cos(sx_2)$$

- We now make use of a trigonometric identity to write the function above as a sum of cosines:

$$\cos(\alpha) \cos(\beta) = \frac{1}{2} \cos(\alpha + \beta) + \frac{1}{2} \cos(\alpha - \beta)$$

- Thus, the function to approximate can be written as a sum of cosines, where each of them receives a linear combination of the input variables:

$$y(x_1, x_2) \simeq \sum_{j=1}^{\infty} v_j \cos(x_1 w_{1j} + x_2 v_{2j})$$

Universal Approximation Theorem (4)

- Thus, the function to approximate can be written as a sum of cosines, where each of them receives a linear combination of the input variables:

$$y(x_1, x_2) \simeq \sum_{j=1}^{\infty} v_j \cos(x_1 w_{1j} + x_2 v_{2j})$$

- This is a two-layer neural network, except for the cosine nonlinearity. The latter can however be approximated by a superposition of a large number of step functions.

$$\cos(z) = \lim_{\tau \to 0} \sum_i \underbrace{[\cos(\tau \cdot (i+1)) - \cos(\tau \cdot i)]}_{\text{constant}} \cdot \underbrace{1_{z > \tau \cdot (i+1)}}_{\text{step function}} + \text{const.}$$

Training a Neural Network

Idea:

- Use the same error function as the perceptron, but replace the perceptron output z by the neural network output z_{out}:

$$\mathcal{E}(\theta) = \frac{1}{N} \sum_{k=1}^{N} \underbrace{\max(0, -z_{\text{out}}^{(k)} t^{(k)})}_{\mathcal{E}^{(k)}(\theta)}$$

  and compute the gradient of the error function w.r.t. the parameters θ of the neural network.

Question:

- How to compute the gradient of the error function?

Error Backpropagation

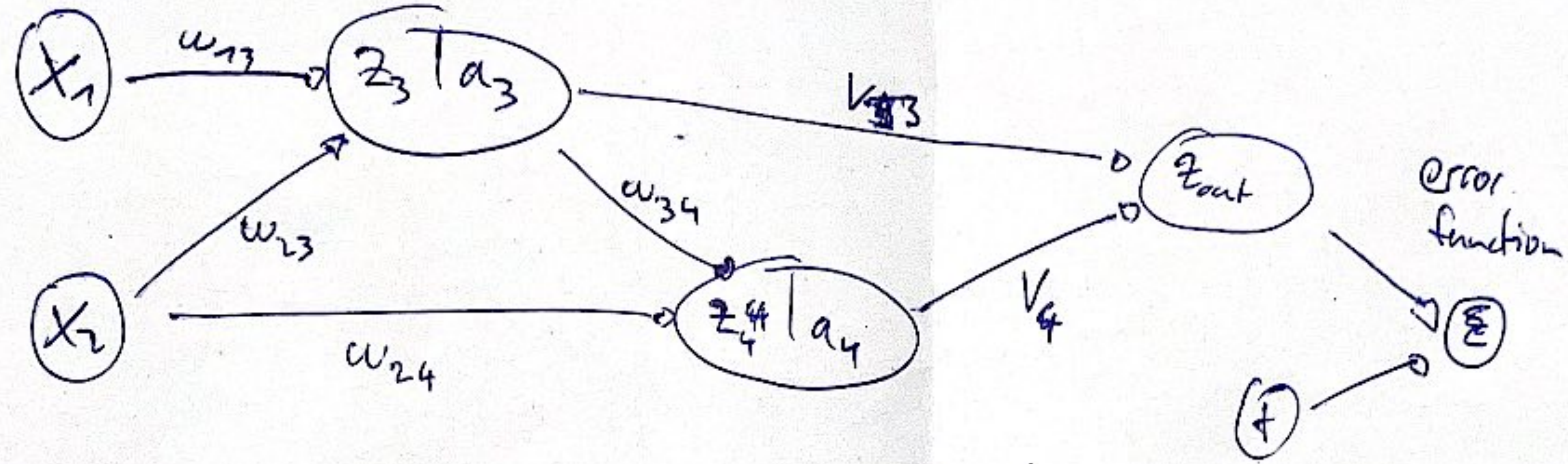
Idea:

- The gradient can be expressed in terms of gradient in the higher layers using the multivariate chain rule.

$$\frac{\partial \mathcal{E}}{\partial z_i} = \sum_j \frac{\partial z_j}{\partial z_i} \frac{\partial \mathcal{E}}{\partial z_j}$$

- Similarly, one can then extract the gradient w.r.t. the parameters of the model as:

$$\frac{\partial \mathcal{E}}{\partial w_{ij}} = \frac{\partial z_j}{\partial w_{ij}} \frac{\partial \mathcal{E}}{\partial z_j}$$



$\mathcal{E} = \max(0, -z_{out} \cdot t)$

"Forward"

$z_3 = x_1 w_{13} + x_2 w_{23} + b_3 \qquad a_3 = g(z_3)$

$z_4 = x_2 w_{24} + a_3 w_{34} + b_4 \qquad a_4 = g(z_4)$

$z_{out} = v_3 a_3 + v_4 a_4 + b_{out}$ (Perceptron)

"Backprop"

$$\frac{\partial \mathcal{E}}{\partial z_{out}} = 1_{\{z_{out} \cdot t \leq 0\}} (-t)$$

$$\frac{\partial \mathcal{E}}{\partial z_4} = \frac{\partial z_{out}}{\partial z_4} \frac{\partial \mathcal{E}}{\partial z_{out}} \qquad \frac{\partial z_{out}}{\partial z_4} = v_4 \cdot g'(z_4)$$

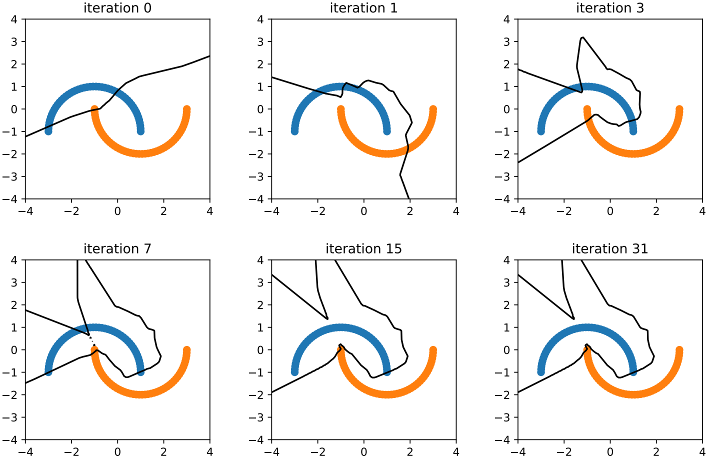
$$\frac{\partial \mathcal{E}}{\partial z_3} = \frac{\partial z_{out}}{\partial z_3} \frac{\partial \mathcal{E}}{\partial z_{out}} + \frac{\partial z_4}{\partial z_3} \frac{\partial \mathcal{E}}{\partial z_4}$$

$$= v_3 g'(z_3) \frac{\partial \mathcal{E}}{\partial z_{out}} + w_{34} g'(z_3) \cdot \frac{\partial \mathcal{E}}{\partial z_4}$$

$$\frac{\partial \mathcal{E}}{\partial w_{13}} = \frac{\partial z_3}{\partial w_{13}} \cdot \frac{\partial \mathcal{E}}{\partial z_3} = x_1 \frac{\partial \mathcal{E}}{\partial z_3}$$

Error Backpropagation

Neural Network at Work

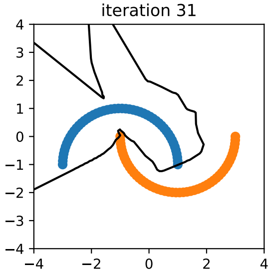


Neural Networks

Remaining questions:

- How to ensure the perceptron and the neural network learn solutions that are simple and generalize well to new data?
- How hard it is to optimize a neural network. Are we guaranteed to converge to a local minima?
- How to learn multiclass classifiers?
- How to implement neural networks?

(These questions will be addressed in the next lectures.)



Summary

- The **perceptron** and the **neural network** enable training classifiers on more complex distributions by focusing on what is critical for classification, i.e. the boundary between classes.
- The **neural network** enables learning **nonlinear** decision boundaries. This is useful when the problem is complex (most practical problems are nonlinear).
- The gradient of a **neural network** required for learning can be easily and quickly computed using the method of **error backpropagation**.
- The perceptron and the neural network do not have closed form solutions but can be trained iteratively using **gradient descent**.

Summary

- The **perceptron** and the **neural network** enable training classifiers on more complex distributions by focusing on what is critical for classification, i.e. the boundary between classes.
- The **neural network** enables learning **nonlinear** decision boundaries. This is useful when the problem is complex (most practical problems are nonlinear).
- The gradient of a **neural network** required for learning can be easily and quickly computed using the method of **error backpropagation**.
- The perceptron and the neural network do not have closed form solutions but can be trained iteratively using **gradient descent**.