Machine Learning/Advanced Machine Learning

Lecture 6.1: Multilayer Perceptrons

Sami S. Brandt

Department of Computer Science
IT University of Copenhagen

Based on slides originally made by Jes Frellsen

30 September 2019

IT UNIVERSITY OF COPENHAGEN

Learning Objectives for Week 6

- Reflect the structure of MLP networks
- Apply MLP networks and reflect their role in deep learning
- Use Tensorflow to build simple models and train them
- Explain backpropagation
- Use simple regularisation methods with neural networks
- Explain radial basis function networks
- Explain the principle of convolutional neural networks

Outline of lecture

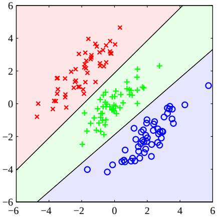
Feed-forward neural networks

Training neural networks

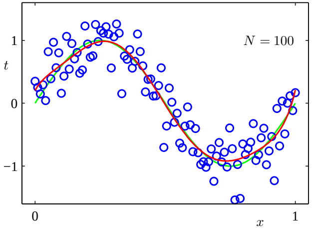
Re-cap: Categories of Machine Learning

Supervised learning: $\mathcal{D} = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_n, t_n)\}$

Classification

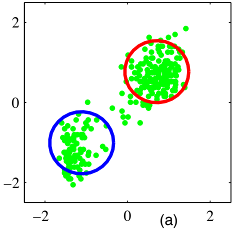


Regression

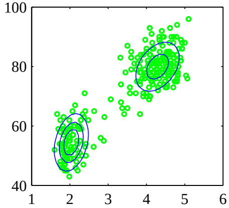


Unsupervised learning: $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$

Clustering

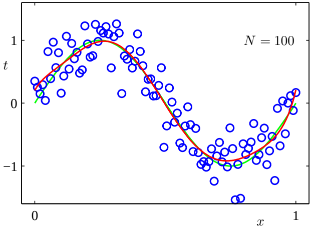


Density estimation



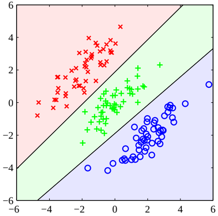
Linear models for regression and classification

Regression



$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=1}^{M} w_j \phi_i(\mathbf{x}) = \mathbf{w}^\mathsf{T} \boldsymbol{\phi}(\mathbf{x})$$

Classification



$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_i(\mathbf{x})\right) = f(\mathbf{w}^\mathsf{T} \boldsymbol{\phi}(\mathbf{x}))$$

where f is an **activation function** such as the **sigmoid** or **softmax**.

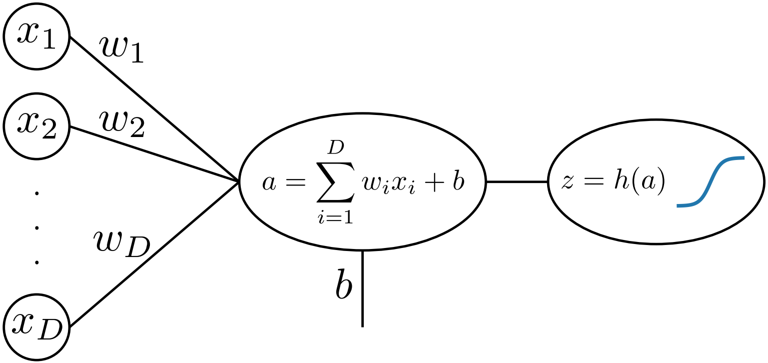
Advantages:

- Useful computational properties: have **closed form expressions**
- Can model “arbitrary” complicated functions.

Disadvantages:

- We have to **choose** the basis functions $\phi_i(\cdot)$, and these are **not adapted to data**.

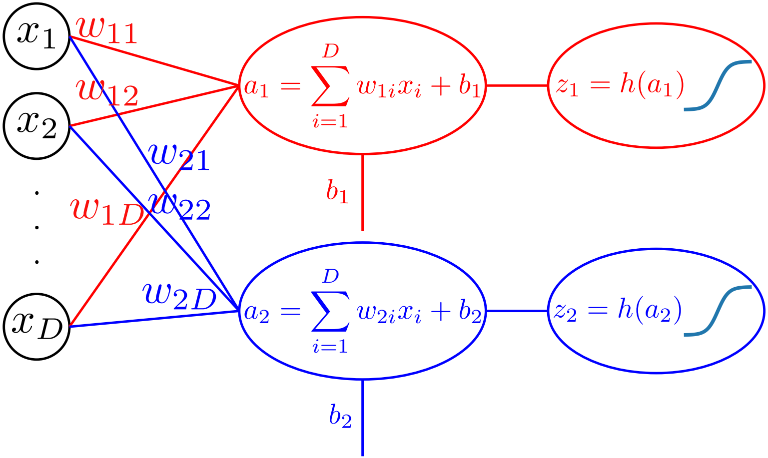
Artificial neuron



Input: $\mathbf{x} = (x_1, \ldots, x_D)^\intercal$
Weights: $\mathbf{w} = (w_1, \ldots, w_D)^\intercal$
Bias: b

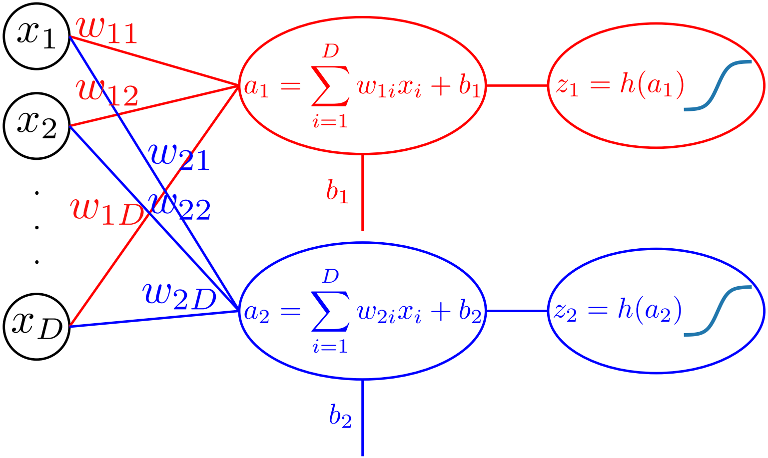
Activation: $a = \sum_{i=1}^{D} w_i x_i + b = \mathbf{w}^\intercal \mathbf{x} + b$
Output: $z = h(a)$
Where $h(\cdot)$ is a **nonlinear activation function**.

Combining artificial neurons



$\mathbf{x}$ $\quad \mathbf{a} = W\mathbf{x} + \mathbf{b} \quad \mathbf{z} = h(\mathbf{a})$

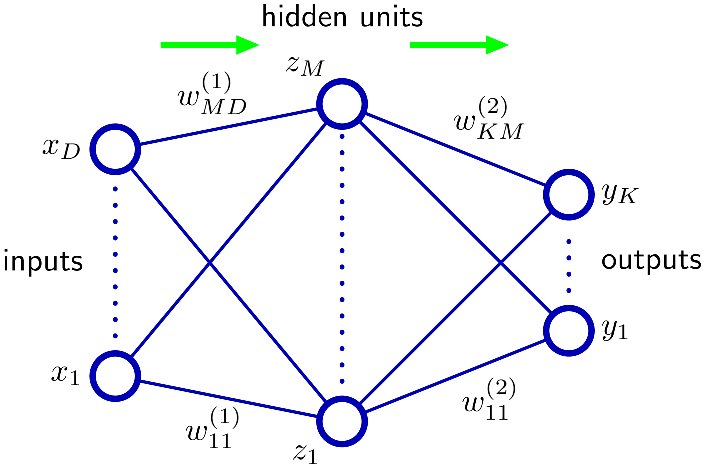
Combining artificial neurons



$\mathbf{x}$ $\quad$ $\mathbf{a} = W\mathbf{x} + \mathbf{b}$ $\quad$ $\mathbf{z} = h(\mathbf{a})$

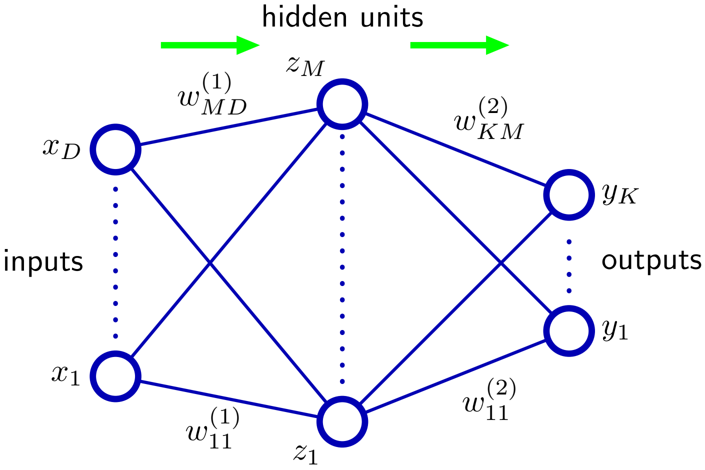
Where $W = \begin{bmatrix} w_{11} & w_{12} & \dots & w_{1D} \\ w_{21} & w_{22} & \dots & w_{2D} \end{bmatrix}$ and h is applied element wise.

A two-layer neural network



$\mathbf{x}$ $\quad$ $\mathbf{z} = h(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$ $\quad$ $\mathbf{y} = \sigma(W^{(2)}\mathbf{z} + \mathbf{b}^{(2)})$

A two-layer neural network

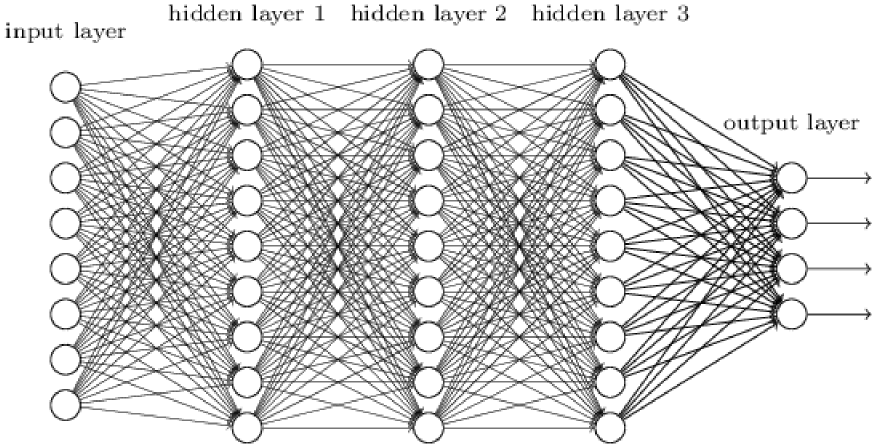


$\mathbf{x}$ $\quad$ $\mathbf{z} = h(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)})$ $\quad$ $\mathbf{y} = \sigma(W^{(2)}\mathbf{z} + \mathbf{b}^{(2)})$

$$\mathbf{y}(\mathbf{x}, \mathbf{W}, \mathbf{b}) = \sigma(W^{(2)}h(W^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)})$$

Figure(s) from Bishop.

A deep neural network



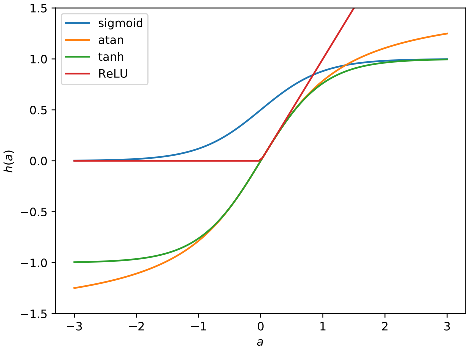
$$\mathbf{y}(\mathbf{x}, \mathbf{W}, \mathbf{b}) = f^{(4)}(f^{(3)}(f^{(2)}(f^{(1)}(\mathbf{x}))))$$

where

$$f^{(\ell)}(\mathbf{z}) = h^{(\ell)}(W^{(\ell)}\mathbf{z} + b^{(\ell)})$$

Figure from neuralnetworksanddeeplearning.com/chap6.html

Activations functions



For hidden layers:

- Sigmoid: $h(a) = \frac{1}{1+\exp(-a)}$
- Arctangent[1]: $h(a) = \operatorname{atan}(a)$
- Hyperbolic tangent: $h(a) = \tanh(a)$
- **Rectified linear**[2]: $h(a) = \max(0, a)$

For output layer:

- Regression: identity, $h(a) = a$
- Binary classification: sigmoid, $h(a) = \frac{1}{1+\exp(-a)}$
- Multiclass classification: softmax, $h(a_k) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$

---

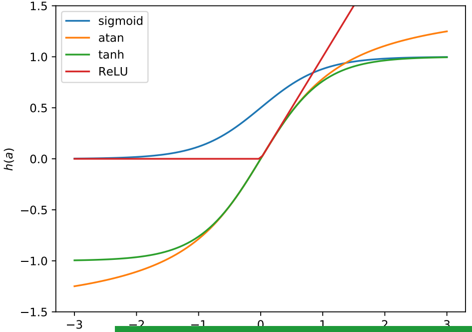
[2] The inverse of $\tan$.

[2] Often called ReLU, but this is a misnomer as it means *rectified linear unit*

Activations functions

For hidden layers:

- Sigmoid: $h(a) = \frac{1}{1+\exp(-a)}$
- Arctangent[1]: $h(a) = \operatorname{atan}(a)$
- Hyperbolic tangent: $h(a) = \tanh(a)$
- **Rectified linear**[2]: $h(a) = \max(0, a)$



For output layer:

- Regression: identity, $h(a) = a$
- Binary classification: sigmoid, $h(a) = \frac{1}{1+\exp(-a)}$
- Multiclass classification: softmax, $h(a_k) = \frac{\exp(a_k)}{\sum_j \exp(a_j)}$

Exercise

Why do we need activation functions in the **hidden layers**?

Why do we need activation functions in the **output layer for classification**?

[2] The inverse of $\tan$.
[2] Often called ReLU, but this is a misnomer as it means *rectified linear unit*

Universal approximation theorem

Neural networks are universal approximators (Bishop):

"A two-layer network with linear outputs can uniformly **approximate any continuous function** on a compact input domain (compact subset of $\mathbb{R}^N$) to **arbitrary accuracy** provided the network has **sufficiently large number of hidden units**"

How do we program neural networks?

We could in principle program them directly in Python / NumPy

However, we need **derivatives / gradients** for training (later)

We will use **TensorFlow**, which has automatic differentiations.

TensorFlow works in a **declarative** style:

- First you declare/define a **dataflow graph**.
- Then you use a session to run/evaluate operations in the graph.

Example: Notebook 1

TensorFlow

PyTorch

Outline of lecture

Feed-forward neural networks

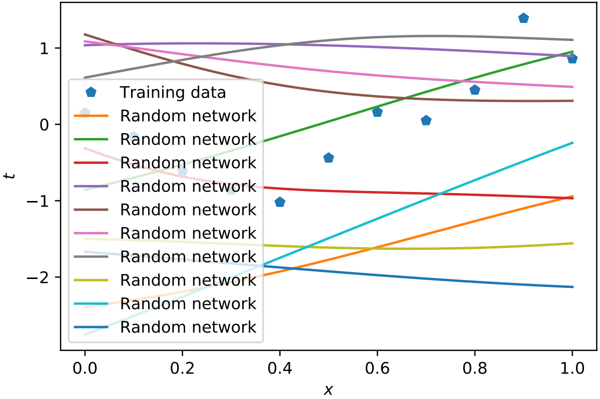
Training neural networks

Training neural networks

We looked a the definition of a neural network, or a **neural network function** $\mathbf{y}(\mathbf{x}, \mathbf{W}, \mathbf{b})$.

If we are given some data as **input vectors X** $= \{\mathbf{x}_n\}_{n=1}^N$ **and target vectors t** $= \{\mathbf{t}_n\}_{n=1}^N$:

- How can we find $\mathbf{W}$ and $\mathbf{b}$?



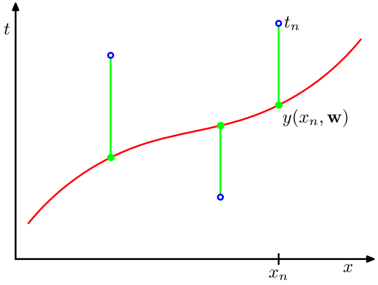
Example: Notebook 2

Regression with neural networks

We find $\mathbf{W}$ and $\mathbf{b}$ by minimizing an **error function** that **measures the misfit** between $\mathbf{y}(\mathbf{x}, \mathbf{W}, \mathbf{b})$ and $\mathbf{t} = \{\mathbf{t}_n\}_{n=1}^N$.

The **sum-of-squares** error function is given by

$$E(\mathbf{W}, \mathbf{b}) = \frac{1}{2} \sum_{n=1}^{N} ||\mathbf{y}(x_n, \mathbf{W}, \mathbf{b}) - \mathbf{t}_n||^2.$$

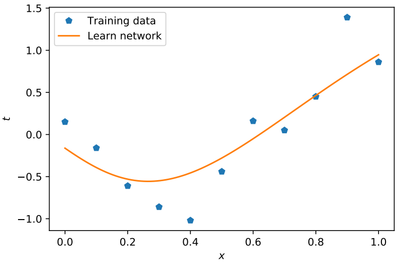


Recall, this corresponds to **finding the MLE** of $\mathbf{W}$ and $\mathbf{b}$ with the likelihood function

$$p(\mathbf{t}|\mathbf{X}, \mathbf{W}, \mathbf{b}, \beta) = \prod_{n=1}^{N} p(\mathbf{t}_n|\mathbf{x}_n, \mathbf{W}, \mathbf{b}, \beta)$$

where

$$p(\mathbf{t}_n|\mathbf{x}_n, \mathbf{W}, \mathbf{b}, \beta) = \mathcal{N}(\mathbf{t}_n|\mathbf{y}(x_n, \mathbf{W}, \mathbf{b}), \beta^{-1}I)$$

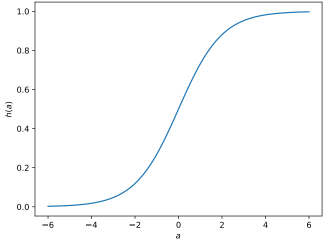


Binary classification with neural networks

Now consider a classification problem, where $t_n \in \{0, 1\}$.

In this cases we use the **sigmoid activation** function

$$y(\mathbf{x}, \mathbf{W}, \mathbf{b}) = \sigma(a^{(L)}) = \frac{1}{1 + \exp(-a^{(L)})}$$

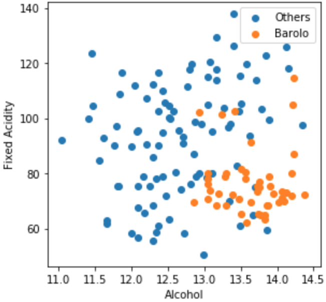


We then use the binomial distribution

$$p(t|\mathbf{x}, \mathbf{W}, \mathbf{b}) = y(\mathbf{x}, \mathbf{W}, \mathbf{b})^t (1 - y(\mathbf{x}, \mathbf{W}, \mathbf{b}))^{1-t} \quad (1)$$

Assuming i.i.d. training data we get the likelihood function

$$p(\mathbf{t}|\mathbf{X}, \mathbf{W}, \mathbf{b}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{W}, \mathbf{b}) \quad (2)$$



We then set the error function to the negative log-likelihood

$$E(\mathbf{W}, \mathbf{b}) = -\sum_{n=1}^{N} t_n \ln y(\mathbf{x}_n, \mathbf{W}, \mathbf{b}) + (1 - t_n) \ln(1 - y(\mathbf{x}_n, \mathbf{W}, \mathbf{b})) \quad (3)$$

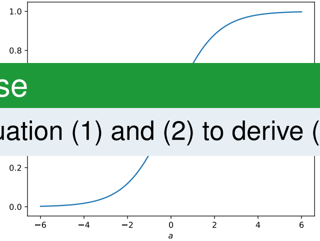
Which is also called the **cross-entropy error function**

Binary classification with neural networks

Now consider a classification problem, where $t_n \in \{0, 1\}$.

In this cases we use the **sigmoid activation** function

$$y(\mathbf{x}, \mathbf{W}, \mathbf{b}) = \sigma(a^{(L)}) = \frac{1}{1 + \exp(-a^{(L)})}$$



Exercise

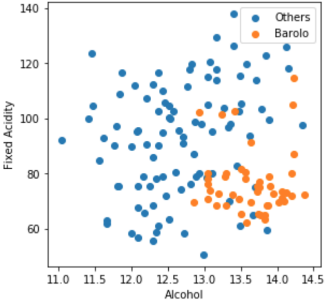
Use equation (1) and (2) to derive (3).

We then use the binomial distribution

$$p(t|\mathbf{x}, \mathbf{W}, \mathbf{b}) = y(\mathbf{x}, \mathbf{W}, \mathbf{b})^t (1 - y(\mathbf{x}, \mathbf{W}, \mathbf{b}))^{1-t} \qquad (1)$$

Assuming i.i.d. training data we get the likelihood function

$$p(\mathbf{t}|\mathbf{X}, \mathbf{W}, \mathbf{b}, \beta) = \prod_{n=1}^{N} p(t_n|\mathbf{x}_n, \mathbf{W}, \mathbf{b}) \qquad (2)$$



We then set the error function to the negative log-likelihood

$$E(\mathbf{W}, \mathbf{b}) = -\sum_{n=1}^{N} t_n \ln y(\mathbf{x}_n, \mathbf{W}, \mathbf{b}) + (1 - t_n) \ln(1 - y(\mathbf{x}_n, \mathbf{W}, \mathbf{b})) \qquad (3)$$

Which is also called the **cross-entropy error function**

Multi-class classification with neural networks

Now consider a classification problem, where $t_n \in \{0, 1\}^K$.

In this cases we use the **softmax activation** function

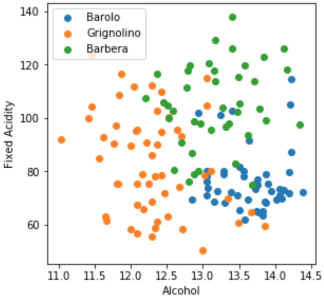
$$y_k(\mathbf{x}, \mathbf{W}, \mathbf{b}) = \sigma(\mathbf{a}^{(L)}) = \frac{\exp(a_k)}{\sum_{j=1}^{K} \exp(a_j)}$$

In this case we obtain the error function

$$E(\mathbf{W}, \mathbf{b}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln y_k(\mathbf{x}_n, \mathbf{W}, \mathbf{b}). \qquad (4)$$

from the log-likelihood function.

This is also called the **cross-entropy error function**



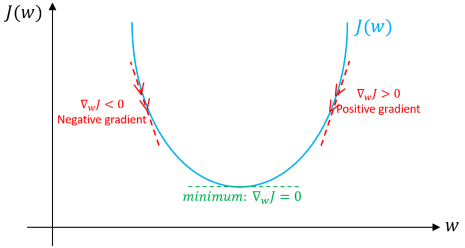
Parameter optimization and gradient descent

We want to find $\mathbf{W} = (\mathbf{W}, \mathbf{b})$ that **minimizes** $E(\mathbf{W})$, i.e. find $\mathbf{W}$ such that $\nabla E(\mathbf{W}) = 0$.

Gradient descent starts with an initial random point $\mathbf{W}^{(0)}$, and iteratively refines it by following the steepest descent direction:

$$\mathbf{W}^{(\tau+1)} = \mathbf{W}^{(\tau)} - \eta \nabla E(\mathbf{W}^{(\tau)})$$

where η is the called the **learning rate**.



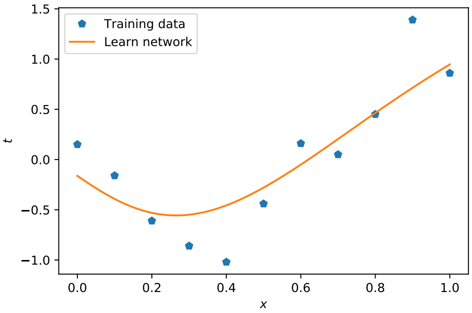
Normally the gradient is calculated on the full dataset (*batch* optimization).

To avoid getting stuck in local minima, we can calculate the gradient on **mini-batches**:

$$\mathbf{W}^{(\tau+1)} = \mathbf{W}^{(\tau)} - \eta \nabla E_s(\mathbf{W}^{(\tau)})$$

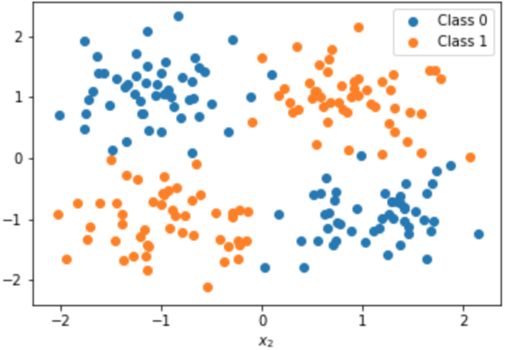
where E_s is the error on a subset of the data.

Examples: regression



Example: Notebook 3

Examples: binary classification



Example: Notebook 4

Next lecture

- Backpropagation
- Regularisation of neural networks
- Radial basis functions
- Brief introduction to CNNs

References I

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.