DM534

INTRODUCTION TO COMPUTER SCIENCE

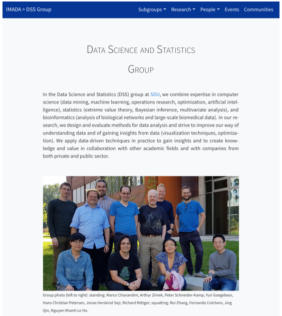
Machine Learning: Linear Regression and Neural Networks

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

About Me

- Marco Chiarandini, Asc. Prof. in CS at IMADA since 2011
  - Master in Electronic Engineering, University of Udine, Italy.
  - Ph.D. in Computer Science at the Darmstadt University of Technology, Germany.
  - Post-Doc researcher at IMADA
  - Visiting Researcher, Institute of Interdisciplinary Research and Development in Artifcial Intelligence, Université Libre de Bruxelles.



- Research Interests
  - Optimization (Operations Research) | Scheduling, Timetabling, Routing
  - Artificial Intelligence | Heuristics, Metaheuristics, Machine Learning
- Current Teaching in CS
  - Applications in Linear Algebra (Bachelor, 3rd semester)
  - Linear and Integer Programming (Master)
  - Mathematical Optimization at Work (Master)

Outline

Outline

Machine Learning

An agent is learning if it improves its performance on future tasks after making observations about the world.

Why learning instead of directly programming?

Three main situations:

- the designer cannot anticipate all possible solutions
- the designer cannot anticipate all changes over time
- the designer has no idea how to program a solution (see, for example, face recognition)

Forms of Machine Learning

- **Unsupervised learning (with Richard Röttger)**
  Correct responses are not provided, but instead the agent tries to identify similarities between the inputs so that inputs that have something in common are categorised together.
  Eg. Clustering

- **Supervised learning (this week)**
  the agent is provided with a series of examples and then it generalizes from those examples to develop an algorithm that applies to new cases.
  Eg: learning to recognize a person's handwriting or voice, to distinguish between junk and welcome email, or to identify a disease from a set of symptoms.

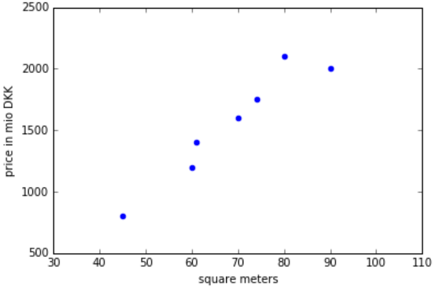
- **Reinforcement learning:**
  the agent is given a general rule to judge for itself when it has succeeded or failed at a task during trial and error. The agent acts autonomously and it learns to improve its behavior over time.
  Eg: learning how to play a game like backgammon (success or failure is easy to define)

Supervised Learning

- inputs that influence outputs
  inputs $\equiv$ independent variables, predictors, features
  outputs $\equiv$ dependent variables, responses
- goal: predict value of outputs
- **supervised**: we provide data set with exact answers

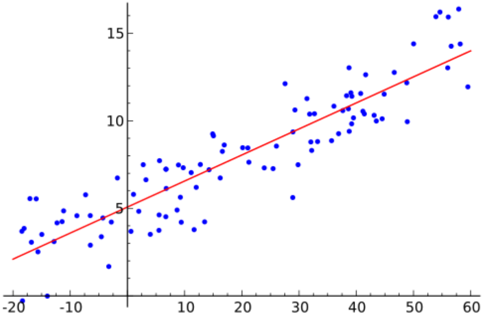
Example: House price prediction:

Size in m^2	Price in mio DKK
45	800
60	1200
61	1400
70	1600
74	1750
80	2100
90	2000

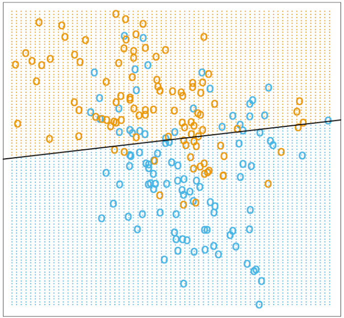


Types of Supervised Learning

Regression problem:
variable to predict is continuous/quantitative



Classification problem:
variable to predict is discrete/qualitative



Supervised Learning Problem

Given: m points (pairs of numbers) $\{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$

Task: determine a model, aka a function $g(x)$ of a simple form, such that

$$\begin{aligned} g(x_1) &\approx y_1, \\ g(x_2) &\approx y_2, \\ &\vdots \\ g(x_m) &\approx y_m. \end{aligned}$$

- We denote by $\hat{y} = g(x)$ the response value predicted by g on x.
- The type of function (linear, polynomial, exponential, logistic, blackbox) may be suggested by the nature of the problem (the underlying physical law, the type of response).
  It is a form of prior knowledge.

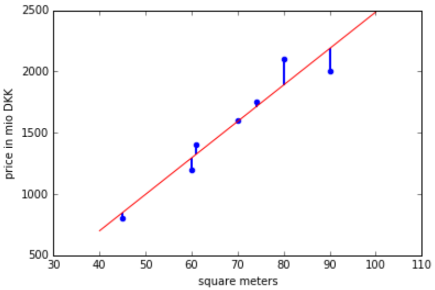
$\leadsto$ Corresponds to fitting a function to the data

House Price Example

Size in m^2	Price in mio DKK
45	800
60	1200
61	1400
70	1600
74	1750
80	2100
90	2000

Training data set

$$\begin{bmatrix} (x_1, y_1) \\ (x_2, y_2) \\ \vdots \\ \vdots \\ (x_m, y_m) \end{bmatrix} \rightsquigarrow \begin{bmatrix} (45, 800) \\ (60, 1200) \\ (61, 1400) \\ (70, 1600) \\ (74, 1750) \\ (80, 2100) \\ (90, 2000) \end{bmatrix}$$



$f(x) = -489.76 + 29.75x$

x	$\hat{y}$	y
45	848.83	800
60	1295.03	1200
61	1324.78	1400
70	1592.5	1600
74	1711.48	1750
80	1889.96	2100
90	2187.43	2000

Example: k-Nearest Neighbors

Regression task

Given: $(x_1, y_1), \dots, (x_m, y_m)$
Task: predict the response value $\hat{y}$ for a new input x

$\rightsquigarrow$ Idea: Let $\hat{y}(x)$ be the average of the k closest points:

1. Rank the data points $(x_1, y_1), \dots, (x_m, y_m)$ in increasing order of distance from x in the input space, ie, $d(x_i, x) = |x_i - x|$.
2. Set the k best ranked points in $N_k(x)$.
3. Return the average of the y values of the k data points in $N_k(x)$.

In mathematical notation:

$$\hat{y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i = g(x)$$

Example: k-Nearest Neighbors

Classification task

Given: $(x_1, y_1), \ldots, (x_m, y_m)$
Task: predict the class $\hat{y}$ for a new input x.

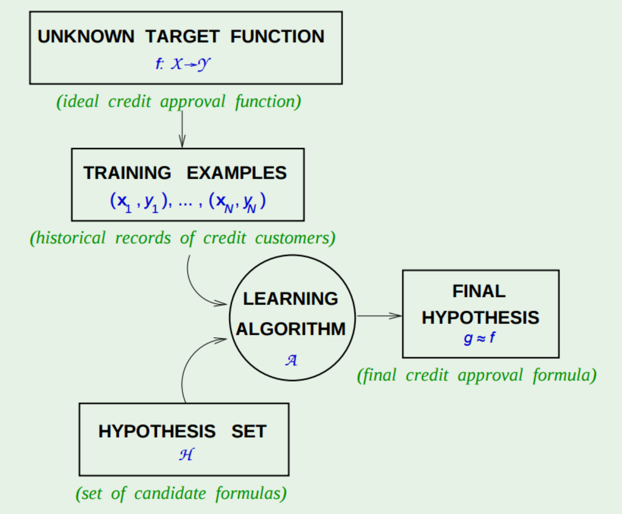
$\rightsquigarrow$ Idea: let the k closest points vote and majority decide

1. Rank the data points $(x_1, y_1), \ldots, (x_m, y_m)$ in increasing order of distance from $\vec{x}$ in the input space, ie, $d(\vec{x}_i, \vec{x}) = |x_i - x|$.
2. Set the k best ranked points in $N_k(x)$.
3. Return the class that is most represented in the k data points of $N_k(x)$.

In mathematical notation:

$$\hat{y} = \mathrm{argmax}_{G \in \mathcal{G}} \sum_{x_i \in N_k(x) | y_i = G} \frac{1}{k} = \hat{G}(x)$$

Learning model



Outline

Linear Regression with One Variable

- The hypothesis set $\mathcal{H}$ is made by linear functions $y = ax + b$ and we search in $\mathcal{H}$ the line that fits best the data:

  1. We evaluate each line by the distance of the points $(x_1, y_1), \ldots, (x_m, y_m)$ from the line in the vertical direction (the y-direction):
     Each point (x_i, y_i), $i = 1..m$ with abscissa x_i has the ordinate $ax_i + b$ in the fitted line.
     Hence, the distance for (x_i, y_i) is $|y_i - ax_i - b|$.

  2. We define as loss (or error, or cost) function the sum of the squares of the distances from the given points $(x_1, y_1), \ldots, (x_m, y_m)$:

     $$\hat{L}(a, b) = \sum_{i=1}^{m} (y_i - ax_i - b)^2 \qquad \text{sum of squared errors}$$

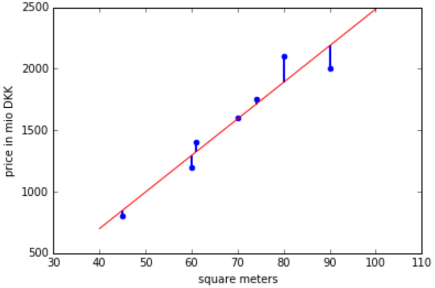
     $\rightsquigarrow$ $\hat{L}$ depends on a and b, while the values x_i and y_i are given by the data available.

  3. We look for the coefficients a and b that yield the line of minimal loss.

House Price Example

Training data set

$$\begin{bmatrix} (x_1, y_1) \\ (x_2, y_2) \\ \vdots \\ \vdots \\ (x_m, y_m) \end{bmatrix} \rightsquigarrow \begin{bmatrix} (45, 800) \\ (60, 1200) \\ (61, 1400) \\ (70, 1600) \\ (74, 1750) \\ (80, 2100) \\ (90, 2000) \end{bmatrix}$$



$f(x) = 29.75x - 489.76$

x	$\hat{y}$	y
45	848.83	800
60	1295.03	1200
61	1324.78	1400
70	1592.5	1600
74	1711.48	1750
80	1889.96	2100
90	2187.43	2000

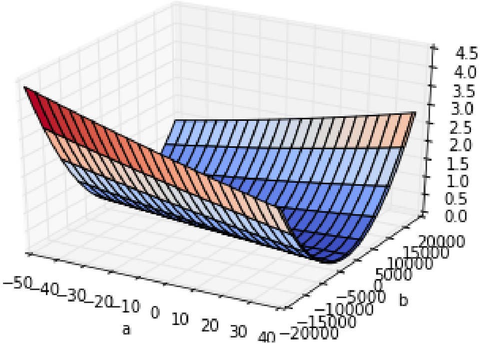
$$\begin{aligned} \hat{L} &= \textstyle\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 = \\ &= (800 - 848.83)^2 \\ &\quad +(1200 - 1295.03)^2 \\ &\quad +(1400 - 1324.78)^2 \\ &\quad +(1600 - 1592.5)^2 \\ &\quad +(1750 - 1711.48)^2 \\ &\quad +(2100 - 1889.96)^2 \\ &\quad +(2000 - 2187.43)^2 = 97858.86 \end{aligned}$$

House Price Example

For

$$f(x) = b + ax$$

$$\begin{aligned}\hat{L}(a,b) = \textstyle\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \\ = \quad & (800 - b - 45 \cdot a)^2 \\ & +(1200 - b - 60 \cdot a)^2 \\ & +(1400 - b - 61 \cdot a)^2 \\ & +(1600 - b - 70 \cdot a)^2 \\ & +(1750 - b - 74 \cdot a)^2 \\ & +(2100 - b - 80 \cdot a)^2 \\ & +(2000 - b - 90 \cdot a)^2\end{aligned}$$



Analytical Solution

Theorem (Closed form solution)

The value of the coefficients of the line that minimizes the sum of squared errors for the given points can be expressed in closed form as a function of the input data:

$$a = \frac{\sum_{i=1}^{m}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{m}(x_i - \bar{x})^2} \qquad b = \bar{y} - a\bar{x}$$

where:

$$\bar{x} = \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \bar{y} = \frac{1}{m}\sum_{i=1}^{m} y_i$$

Proof: (not in the curriculum of DM534)
[Idea: use partial derivaties to obtain a linear system of equations that can be solved analytically]

Learning Task: Framework

Learning = Representation + Evaluation + Optimization

- Representation: formal language that the computer can handle. Corresponds to choosing the set of functions that can be learned, ie. the hypothesis set of the learner. How to represent the input, that is, which input variables to use.
- Evaluation: definition of a loss function
- Optimization: a method to search among the learners in the language for the one minimizing the loss.

Outline

Linear Regression with Multiple Variables

There can be several input variables (aka features). In practice, they improve prediction.

Size in m^2	# of rooms	$\cdots$	Price in mio DKK
45	2	$\cdots$	800
60	3	$\cdots$	1200
61	2	$\cdots$	1400
70	3	$\cdots$	1600
74	3	$\cdots$	1750
80	3	$\cdots$	2100
90	4	$\cdots$	2000
$\vdots$	$\vdots$	$\vdots$	

In vector notation:

$$\begin{bmatrix} (\vec{x}_1, y_1) \\ (\vec{x}_2, y_2) \\ \vdots \\ (\vec{x}_m, y_m) \end{bmatrix}$$

$$\vec{x}_i = \begin{bmatrix} x_{i1} \ x_{i2} \ \ldots \ x_{ip} \end{bmatrix}$$

$$i = 1, 2, \ldots, m$$

k-Nearest Neighbors Revisited

Case with multiple input variables

Regression task

Given: $(\vec{x}_1, y_1), \dots, (\vec{x}_m, y_m)$
Task: predict the response value $\hat{y}$ for a new input $\vec{x}$

⇝ Idea: Let $\hat{y}(\vec{x})$ be the average of the k closest points:

1. Rank the data points $(\vec{x}_1, y_1), \dots, (\vec{x}_m, y_m)$ in increasing order of distance from x in the input space, ie, $d(\vec{x}_i, \vec{x}) = \sqrt{\sum_j (x_{ij} - x_j)^2}$.
2. Set the k best ranked points in $N_k(\vec{x})$.
3. Return the average of the y values of the k data points in $N_k(\vec{x})$.

In mathematical notation:

$$\hat{y}(\vec{x}) = \frac{1}{k} \sum_{\vec{x}_i \in N_k(\vec{x})} y_i = g(\vec{x})$$

⇝ It requires the redefinition of the distance metric, eg, Euclidean distance

k-Nearest Neighbors Revisited

Case with multiple input variables

Classification task

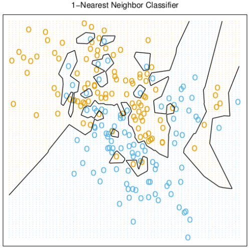
Given: $(\vec{x}_1, y_1), \ldots, (\vec{x}_m, y_m)$
Task: predict the class $\hat{y}$ for a new input $\vec{x}$.

$\rightsquigarrow$ Idea: let the k closest points vote and majority decide

1. Rank the data points $(\vec{x}_1, y_1), \ldots, (\vec{x}_m, y_m)$ in increasing order of distance from $\vec{x}$ in the input space, ie, $d(\vec{x}_i, \vec{x}) = \sqrt{\sum_j (x_{ij} - x_j)^2}$.
2. Set the k best ranked points in $N_k(\vec{x})$.
3. Return the class that is most represented in the k data points of $N_k(\vec{x})$

In mathematical notation:

$$\hat{G}(\vec{x}) = \text{argmax}_{G \in \mathcal{G}} \sum_{\vec{x}_i \in N_k(\vec{x}) | y_i = G} \frac{1}{k}$$



Linear Regression Revisited

Representation of hypothesis space if only one variable (feature):

$h(x) = \theta_0 + \theta_1 x$ linear function

if there is another input variable (feature):

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = h(\vec{\theta}, \vec{x})$$

for conciseness, defining $x_0 = 1$

$$h(\vec{\theta}, \vec{x}) = \vec{\theta} \cdot \vec{x} = \sum_{j=0}^{2} \theta_j x_j$$

$$h(\vec{\theta}, \vec{x}_i) = \vec{\theta} \cdot \vec{x}_i = \sum_{j=0}^{p} \theta_j x_{ij}$$

Notation:

- p num. of features, $\vec{\theta}$ vector of $p+1$ coefficients, θ_0 is the bias
- x_{ij} is the value of feature j in sample i, for $i = 1..m, j = 1..p$
- y_i is the value of the response in sample i

Linear Regression Revisited

Evaluation

loss function for penalizing errors in prediction.
Most common is squared error loss:

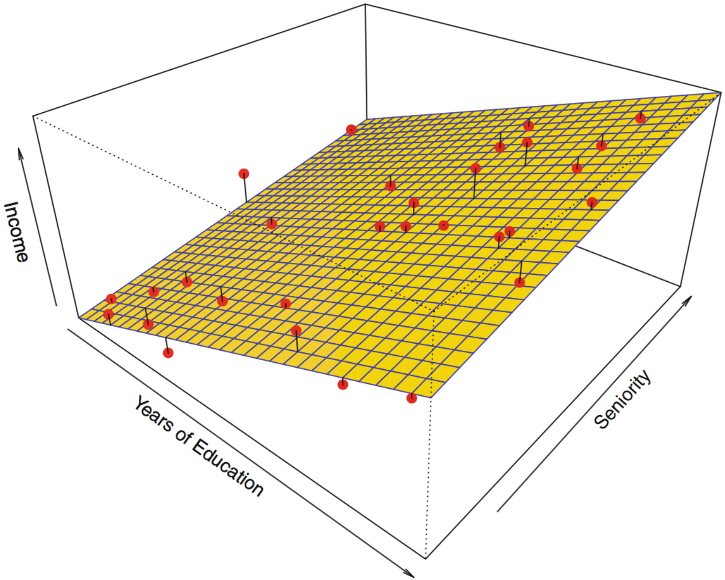
$$\hat{L}(\vec{\theta}) = \sum_{i=1}^{m} \left(y_i - h(\vec{\theta}, \vec{x}_i) \right)^2 = \sum_{i=1}^{m} \left(y_i - \sum_{j=0}^{p} \theta_j x_{ij} \right)^2 \qquad \text{loss function}$$

Optimization

$$\min_{\vec{\theta}} \hat{L}(\vec{\theta})$$

⇝ Although not shown here, the optimization problem can be solved analytically and the solution can be expressed in closed form.

Multiple Variables: Example



Polynomial Regression

It generalizes the linear function $h(x) = ax + b$ to a polynomial of degree k

Representation

$$h(x) = \text{poly}(\vec{\theta}, x) = \theta_0 + \theta_1 x + \cdots + \theta_k x^k$$

where $k \leq m - 1$.
$\rightsquigarrow$ Each term acts like a different variable in the previous case.

$$\vec{x} = \begin{bmatrix} 1 \ x \ x^2 \ \ldots \ x^k \end{bmatrix}$$

Evaluation Again, we use the loss function defined as the sum of squared errors loss:

$$\hat{L}(\vec{\theta}) = \sum_{i=1}^{m} \left(y_i - \text{poly}(\vec{\theta}, \vec{x}_i)\right)^2 = \sum_{i=1}^{m} \left(y_i - \theta_0 - \theta_1 x_i - \cdots - \theta_k x_i^k\right)^2$$

Polynomial Regression

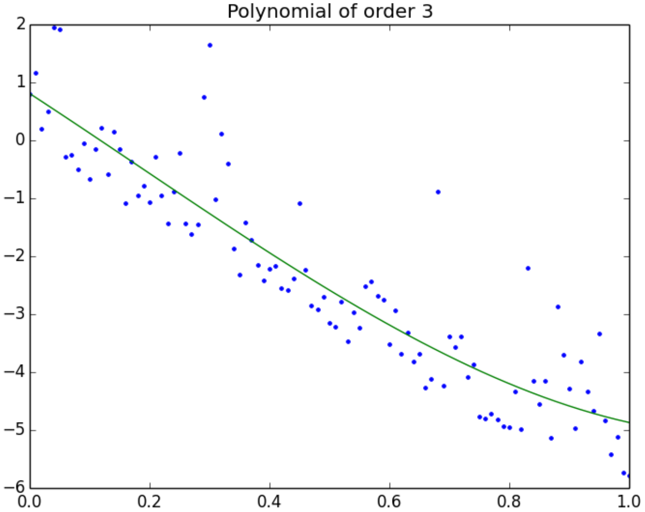
Optimization:

$$\begin{aligned}\min_{\vec{\theta}} L(\vec{\theta}) &= \min \sum_{i=1}^{m} \Big(y_i - \text{poly}(\vec{\theta}, \vec{x}_i)\Big)^2 \\ &= \min \sum_{i=1}^{m} \left(y_i - \theta_0 - \theta_1 x_i - \cdots - \theta_k x_i^k\right)^2\end{aligned}$$

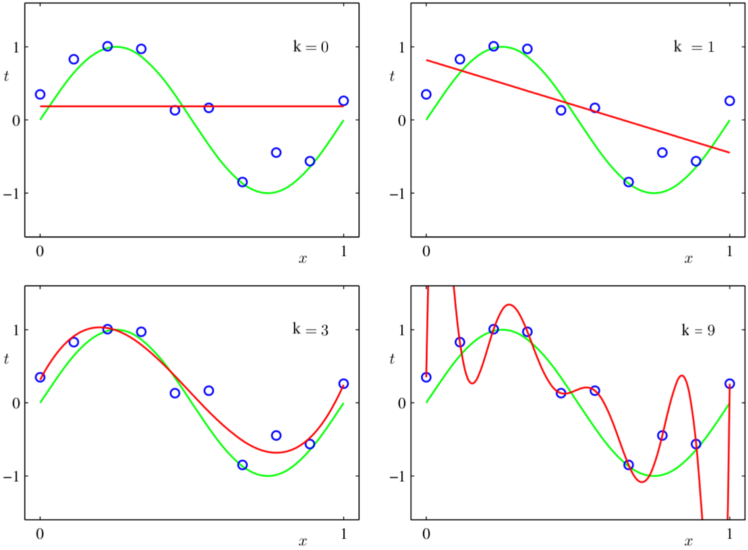
this is a function of $k+1$ coefficients $\theta_0, \cdots, \theta_k$.

$\leadsto$ Although not shown here, also this optimization problem can be solved analytically and the solution can be expressed in closed form.

Polynomial Regression: Example



Overfitting



Training and Assessment

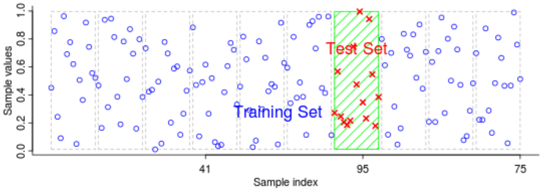
Avoid peeking: use different data for different tasks:

Training and Test data

- Coefficients learned on Training data
- Coefficients and models compared on Validation data
- Final assessment on Test data

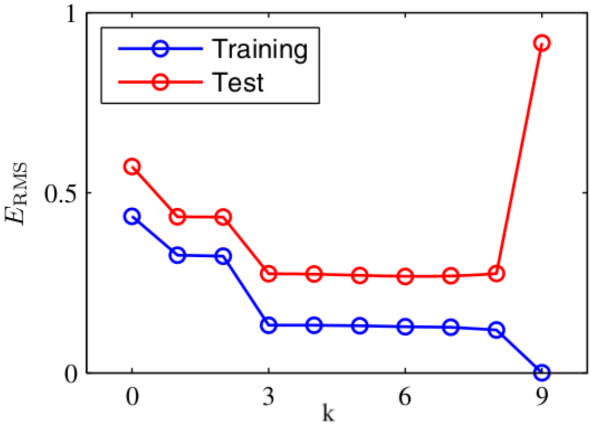
Techniques:

- Holdout cross validation
- If small data:
  k-fold cross validation



Model Comparison

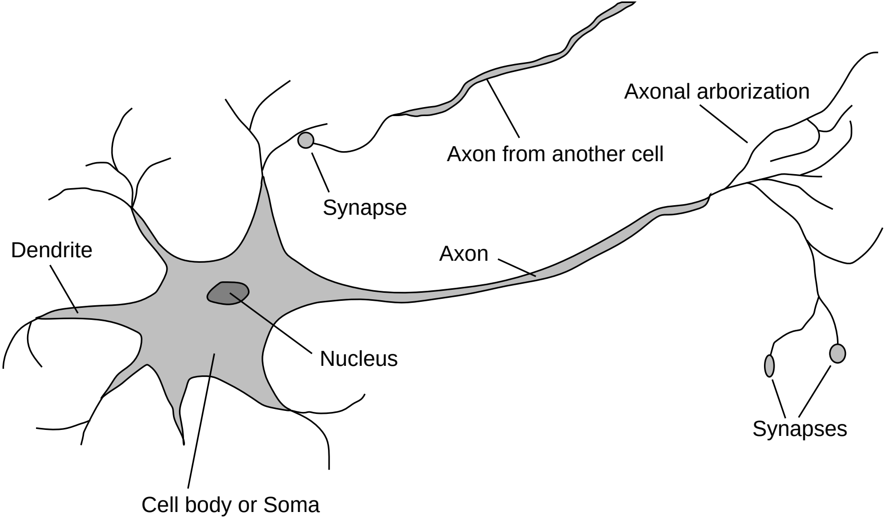
k number of coefficients, eg, in polynomial regression the order of the polynomial
E_{RMS} root mean square of loss



Outline

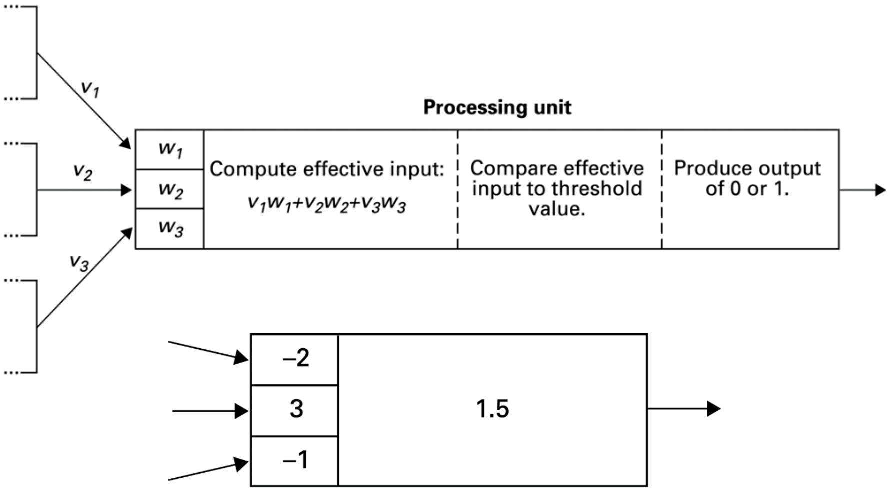
The Biological Neuron

A neuron in a living biological system



McCulloch–Pitts "unit" (1943)

Activities within a processing unit



Artificial Neural Networks

Basic idea:

- Artificial Neuron
  - Each input is multiplied by a weighting factor.
  - Output is 1 if sum of weighted inputs exceeds the threshold value; 0 otherwise.
- Network is programmed by adjusting weights using feedback from examples.

$\leadsto$ "**The** neural network" does not exist. There are different paradigms for neural networks, how they are trained and where they are used.

Generalization of McCulloch–Pitts unit

Let a_j be the j input to node i.
Then, the output of the unit is 1 when:

$$-2a_1 + 3a_2 - 1a_3 \geq 1.5$$

or equivalently when:

$$-1.5 - 2a_1 + 3a_2 - 1a_3 \geq 0$$

and, defining $a_0 = -1$, when:

$$1.5a_0 - 2a_1 + 3a_2 - 1a_3 \geq 0$$

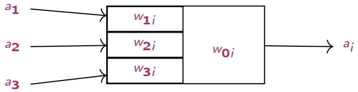
In general, for weights w_{ji} on arcs ji a neuron outputs 1 when:

$$\sum_{j=0}^{p} w_{ji} a_j \geq 0,$$

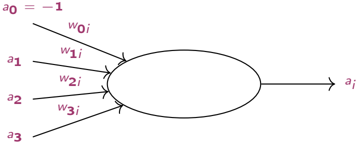
and 0 otherwise. (We will assume the zeroth input a_0 to be always -1.)

Generalization of McCulloch–Pitts unit

Hence, we can draw the artificial neuron unit i:



also in the following way:



where now the output a_i is 1 when the linear combination of the inputs:

$$in_i = \sum_{j=0}^{p} w_{ji} a_j = \vec{w}_i \cdot \vec{a} \qquad \vec{a}^{\intercal} = \begin{bmatrix} -1 & a_1 & a_2 & \cdots & a_p \end{bmatrix}$$

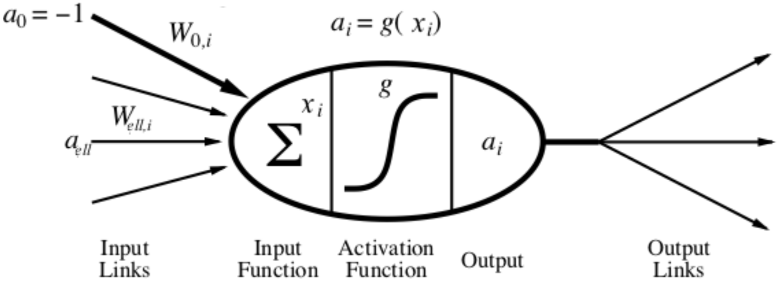
is > 0.

Generalization of McCulloch–Pitts unit

Output is a function of weighted inputs. At unit i

$$a_i = g(x_i) = g\left(\sum_{j=0}^{p} w_{ji} a_j\right)$$

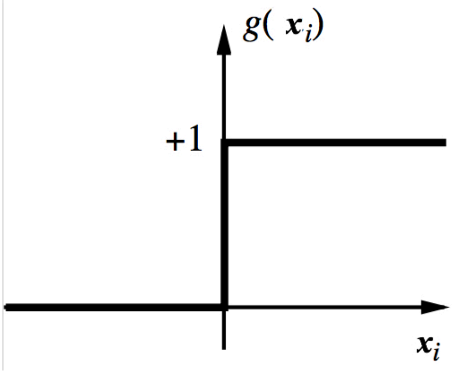
a_i for activation values;
w_{ji} for weight parameters

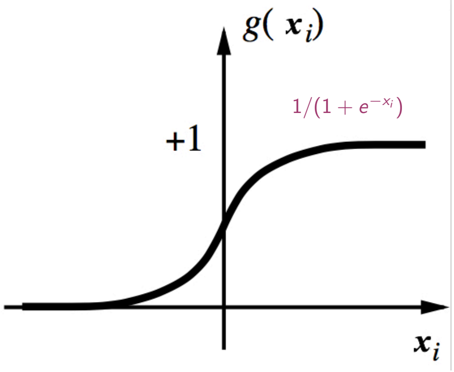


Changing the weight w_{0i} moves the threshold location

Activation functions

Non linear activation functions

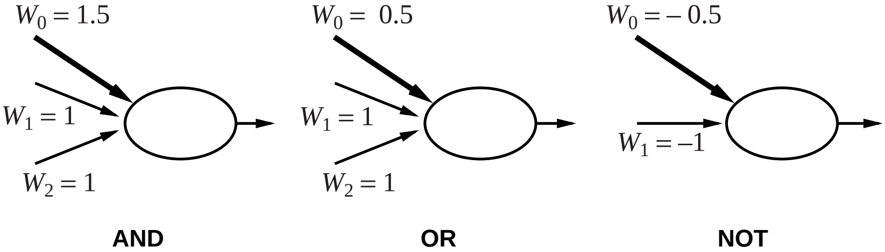




step function or threshold function
(mostly used in theoretical studies)

continuous activation function, e.g., sigmoid function $1/(1+e^{-z})$
(mostly used in practical applications)

Implementing logical functions



McCulloch and Pitts: every Boolean function can be implemented by combining this type of units

Rosenblatt (1958) and Minsky and Papert (1969) showed that this is not true for a basic neuron alone. Exclusive-or circuit cannot be processed.

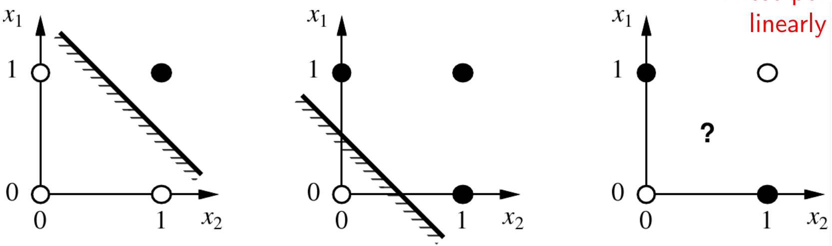
Minsky and Papert (1969): true for networks of neurons.

Expressiveness of single perceptrons

Consider a perceptron with g = step function
At unit i the output is 1 when:

$$\sum_{j=0}^{p} w_{ji} x_j > 0 \quad \text{or} \quad \vec{w}_i \cdot \vec{x} > 0$$

Hence, it represents a linear separator in input space:
- line in 2 dimensions
- plane in 3 dimensions
- hyperplane in multidimensional space



These points are not linearly separable

Network structures

Structure (or architecture): definition of number of nodes, interconnections and activation functions g (but not weights).

- Feed-forward networks:
  no cycles in the connection graph
  - single-layer perceptrons (no hidden layer)
  - multi-layer perceptrons (one or more hidden layer)

  Feed-forward networks implement functions, have no internal state

- Recurrent networks:
  connections between units form a directed cycle.
  – internal state of the network
  exhibit dynamic temporal behavior (memory, apriori knowledge)
  – Hopfield networks for *associative memory*

Feed-Forward Networks – Use

Neural Networks are used in classification and regression

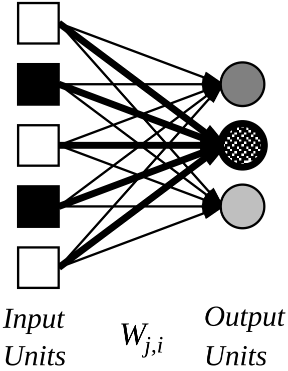
- Boolean classification:
  - value over 0.5 one class
  - value below 0.5 other class

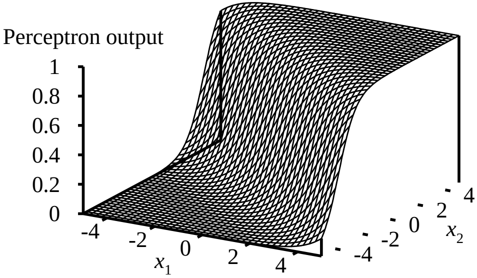
- k-way classification
  - divide single output into k portions
  - k separate output units

- continuous output
  - identity or linear activation function in output unit

Outline

Single-layer NN



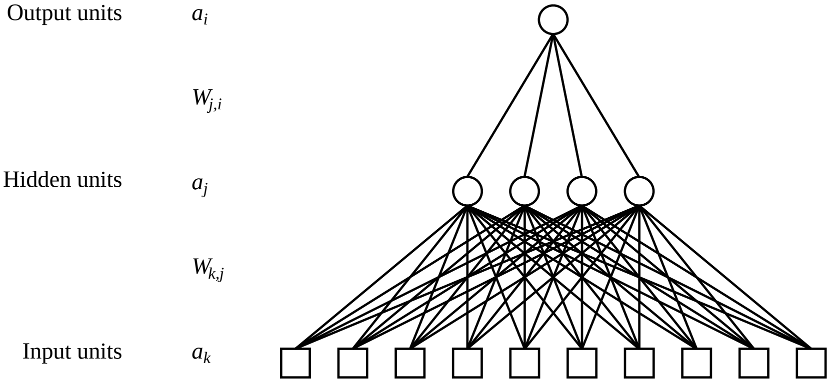


Output units all operate separately—no shared weights
Adjusting weights moves the location, orientation, and steepness of cliff

Outline

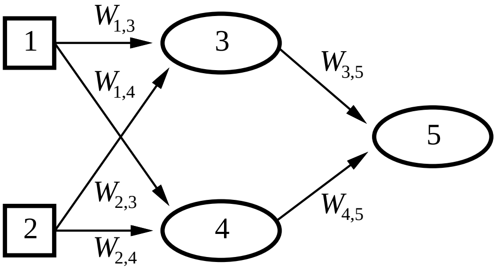
Multilayer perceptrons

Layers are usually fully connected;
number of hidden units typically chosen by hand



(a for activation values; W for weight parameters)

Multilayer Feed-forward



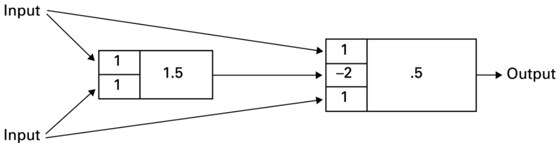
Feed-forward network = a parametrized family of nonlinear functions:

$$\begin{aligned} a_5 &= g(w_{3,5} \cdot a_3 + w_{4,5} \cdot a_4) \\ &= g(w_{3,5} \cdot g(w_{1,3} \cdot a_1 + w_{2,3} \cdot a_2) + w_{4,5} \cdot g(w_{1,4} \cdot a_1 + w_{2,4} \cdot a_2)) \end{aligned}$$

Adjusting weights changes the function: do learning this way!

Neural Network with two layers

What is the output of this two-layer network on the input $a_1 = 1, a_2 = 0$ using step-functions as activation functions?



The input of the first neuron (node 3) is:

$$\sum_j w_{j3} a_j = w_{13} \cdot a_1 + w_{23} \cdot a_2 = 1 \cdot 1 + 1 \cdot 0 = 1$$

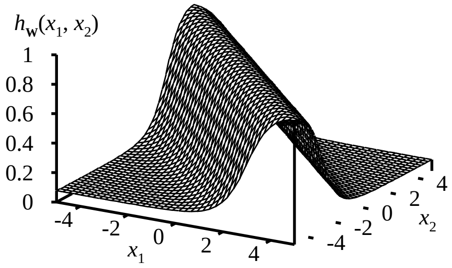
which is < 1.5, hence the output of node A is $a_3 = g(\sum_j w_{j3} a_j) = 0$.
The input to the second neuron (node 4) is:

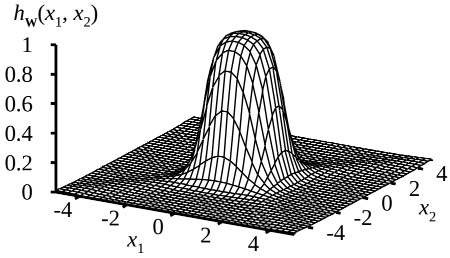
$$\sum_j w_{j4} a_j = w_{14} \cdot a_1 + w_{34} \cdot a_3 + w_{24} \cdot a_{24} = 1 \cdot 1 - 2 \cdot 0 + 1 \cdot 0 = 1$$

which is > 0.5, hence the output of the node 4 is $a_3 = g(\sum_j w_{j4} a_j) = 1$.

Expressiveness of MLPs

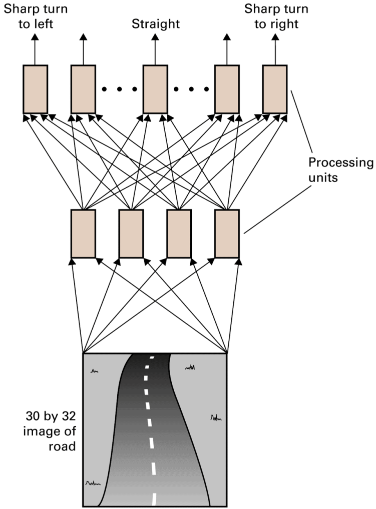
All continuous functions with 2 layers, all functions with 3 layers





Combine two opposite-facing threshold functions to make a ridge
Combine two perpendicular ridges to make a bump
Add bumps of various sizes and locations to fit any surface
Proof requires exponentially many hidden units (Minsky & Papert, 1969)

A Practical Example



Deep learning $\equiv$
convolutional neural networks $\equiv$
multilayer neural network with structure on the arcs

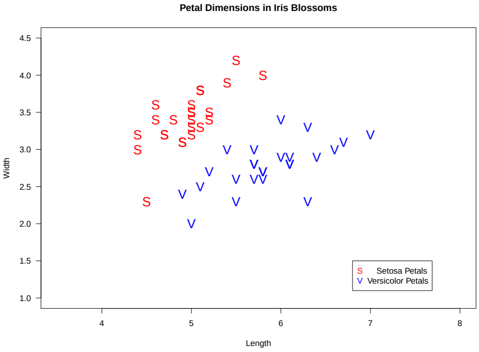
Example: one layer only for image recognition, another for action decision.

The image can be subdivided in regions and each region linked only to a subset of nodes of the first layer.

Numerical Example

Binary Classification

The Fisher's iris data set gives measurements in centimeters of the variables:
petal length and petal width for 50 flowers from 2 species of iris: iris setosa, and iris versicolor.



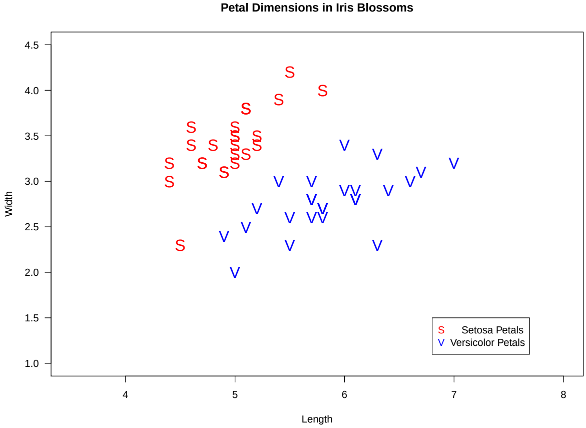
iris.data:

Petal.Length	Petal.Width	Species	id
4.9	3.1	setosa	0
5.5	2.6	versicolor	1
5.4	3.0	versicolor	1
6.0	3.4	versicolor	1
5.2	3.4	setosa	0
5.8	2.7	versicolor	1

Two classes encoded as $0/1$

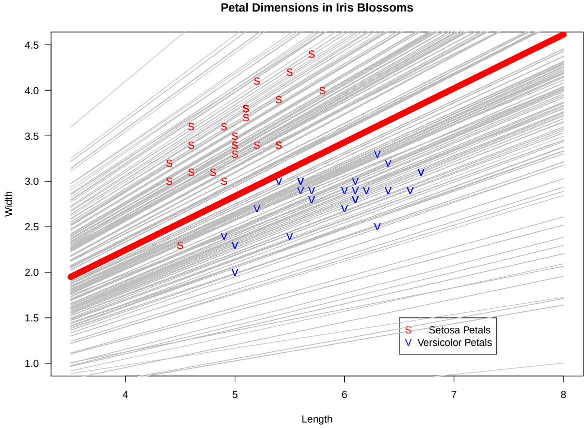
Perceptron Learning

In 2D, the decision surface of a linear combination of inputs gives: $\vec{w} \cdot \vec{x} = \mathtt{constant}$, a line!
Training the perceptron $\equiv$ searching the line that separates the points at best.



Perceptron Learning

We try different weight values moving towards the values that minimize the misprediction of the training data: the red line.
(Gradient descent algorithm) (Rosenblatt, 1958: the algorithm converges)



Summary

1. Machine Learning
2. Linear Regression
   - Extensions
3. Artificial Neural Networks
   - Single-layer Networks
   - Multi-layer perceptrons