DM534
INTRODUCTION TO COMPUTER SCIENCE

Machine Learning: Linear Regression and Neural Networks

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Outline

Numbers

- set of real numbers: $\mathbb{R}$
- the set $\mathbb{R}^2$ is the set of ordered pairs (x, y) of real numbers
  (eg, coordinates of a point wrt a pair of axes, the Cartesian plane)
- the set $\mathbb{R}^n$ is the set of ordered tuples $(x_1, x_2, \dots, x_n)$ of real numbers
  (Euclidean space)

Matrices and Vectors

- A matrix is a rectangular array of numbers or symbols. It can be written as

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

- An $n \times 1$ matrix is a column vector, or simply a vector:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

- the set $\mathbb{R}^n$ is the set of vectors $[x_1, x_2, \ldots, x_n]^T$ of real numbers (eg, coordinates of a point wrt an n-dimensional space, the Euclidean Space)

Scalar product of two vectors

- For two vectors

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

the scalar (or dot) product denoted $\mathbf{v} \cdot \mathbf{w}$, is the real number given by

$$\mathbf{v} \cdot \mathbf{w} = v_1 w_1 + v_2 w_2 + \ldots + v_n w_n$$

Functions

- a function f on a set $\mathcal{X}$ into a set $\mathcal{Y}$ is a rule that assigns a unique element $f(x)$ in S to each element x in $\mathcal{X}$.

$$y = f(x)$$

y dependent variable

x independent variable

- a linear function has only sums and scalar multiplications, that is, for variable $x \in \mathbb{R}$ and scalars $a, b \in \mathbb{R}$:

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

Equations

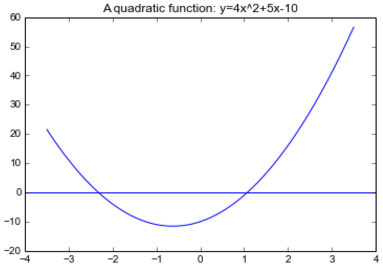
- Quadratic equation

$$ax^2 + bx + c = 0, \qquad a \neq 0.$$

- closed form or analytical solution:

$$x_1 = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

$$x_2 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$



Polynomial Equations

- A polynomial of degree n in x is an expression of the form:

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n,$$

where the a_i are real constants, $a_n \neq 0$, and x is a real variable.

- $P_n(x) = 0$ has at most n solutions, eg:

$$x^3 - 7x + 6 = (x-1)(x-2)(x+3) = 0,$$

which are called roots or zeros of $P_n(x)$

- No general (closed) formula

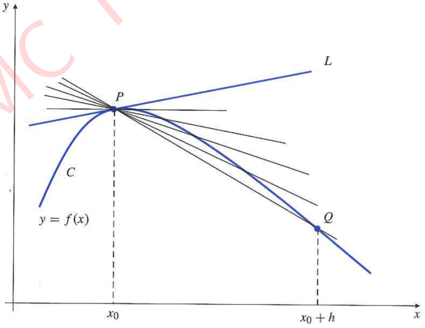
Differentiation

A line L through a point $(x_0, f(x_0))$ of f can be described by:

$$y = m(x - x_0) + f(x_0)$$

The derivative is the slope of the line that is tangent to the curve:

$$y = f'(x_0)(x - x_0) + f(x_0)$$



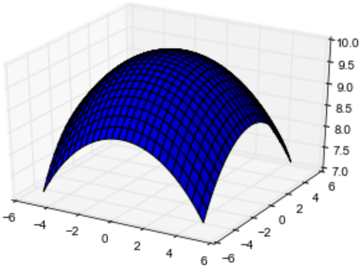
Functions of Several Variables

- A function f of n real variables is a rule that assigns a unique real number $f(x_1, x_2, \ldots, x_n)$ to each point $(x_1, x_2, \ldots, x_n)$

Example in $\mathbb{R}^2$:

$$f(x, y) = \sqrt{10^2 - x^2 - y^2}$$

$$x^2 + y^2 + z^2 = 10$$



Partial Derivatives

- The first partial derivative of the function $f(x, y)$ with respect to the variables x and y are:

$$f_1(x, y) = \lim_{h \to 0} \frac{f(x+h, y) - f(x, y)}{h} = \frac{\partial}{\partial x} f(x, y)$$

$$f_2(x, y) = \lim_{k \to 0} \frac{f(x, y+k) - f(x, y)}{k} = \frac{\partial}{\partial y} f(x, y)$$

- Their value in a point (x_0, y_0) is given by:

$$f_1(x_0, y_0) = \left(\frac{\partial}{\partial x} f(x, y) \right) \Bigg|_{(x_0, y_0)}$$

$$f_2(x_0, y_0) = \left(\frac{\partial}{\partial y} f(x, y) \right) \Bigg|_{(x_0, y_0)}$$

Outline

Forms of Machine Learning

- Supervised learning
  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, learning to distinguish between junk and welcome email, and learning how to identify a disease from a set of symptoms.

- Unsupervised learning

  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.

- 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)

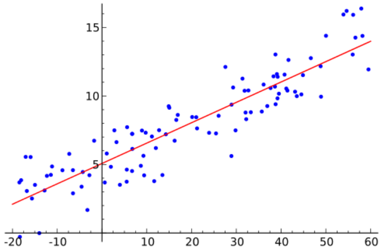
- Evolutionary learning:

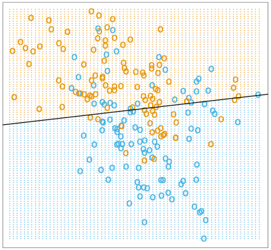
  the agent adapts to improve its success and chance of having offspring in the environment.

  Success assessed by fitness, which corresponds to a score for how good the current solution is.

Supervised Learning

- inputs that influence outputs
  inputs: independent variables, predictors, features
  outputs: dependent variables, responses
- goal: predict value of outputs
- **supervised**: we provide data set with exact answers
  - regression problem $\rightsquigarrow$ variable to predict is continuous/quantitative
  - classification problem $\rightsquigarrow$ variable to predict is discrete/qualitative





Supervised Learning Problem

Given m points (pairs of numbers) $(x_1, y_1), \ldots, (x_m, y_m)$

Task determine a function $f(x)$ of a simple form such that

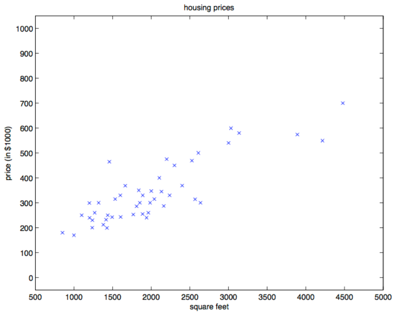
$$f(x_1) \approx y_1, \cdots, f(x_m) \approx y_m.$$

The type of function: polynomials, exponential functions, logistic may be suggested by the nature of the problem (the underlying physical law, the type of response)

$\rightsquigarrow$ Corresponds to fitting a model to the data

Supervised Learning Problem

Living area (feet^2)	Price (1000\$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮



Example: k-Nearest Neighbors

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

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

  $\hat{y}(\vec{x})$ is the average of the k closest points
  It requires the definition of a distance metric, eg, Euclidean distance

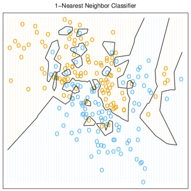
1. Rank the data points $(\vec{x}_1, y_1), \dots, (\vec{x}_m, y_m)$ in increasing order of distance from $\vec{x}$ in the input space, ie, $d(\vec{x}_j, \vec{x}) = \sqrt{\sum_i (x_{ij} - x_i)^2}$.
2. Set the k best ranked points in $N_k(\vec{x})$.
3. Return the average of the y variables of the k data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

Example: k-Nearest Neighbors

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

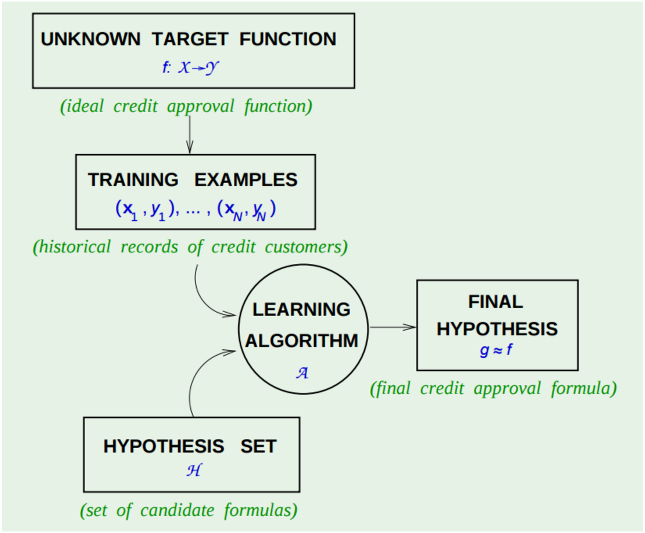
$$\hat{G}(\vec{x}) = \text{argmax}_{G \in \mathcal{G}} \sum_{x_i \in N_k(\vec{x}) | y_i = G} \frac{1}{k}$$

corresponds to a majority rule.



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, $d(\vec{x}_j, \vec{x}) = \sqrt{\sum_i (x_{ij} - x_i)^2}$.
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 $N_k(\vec{x})$.

Learning model


UNKNOWN TARGET FUNCTION
$f: \mathcal{X} \to \mathcal{Y}$
(ideal credit approval function)
TRAINING EXAMPLES
$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$
(historical records of credit customers)
LEARNING ALGORITHM
$\mathcal{A}$
FINAL HYPOTHESIS
$g \approx f$
(final credit approval formula)
HYPOTHESIS SET
$\mathcal{H}$
(set of candidate formulas)

Outline

Linear Regression

- The hypothesis set $\mathcal{H}$ is made by linear functions
- We search for the line $y = ax + b$ that fits best the data:
  - we measure the distance of the points $(x_1, y_1), \dots, (x_m, y_m)$ from the straight line by the vertical direction (the y-direction).
  - we look for the line that minimizes the sum of the squares of the distances from the points $(x_1, y_1), \dots, (x_m, y_m)$
- each point (x_j, y_j), $j = 1..m$ with abscissa x_j has the ordinate $ax_j + b$ in the fitted line. The distance from the actual data (x_j, y_j) is $|y_j - ax_j - b|$
- the sum of square errors is

$$\hat{L} = \sum_{j=1}^{m} (y_j - ax_j - b)^2$$

Partial Derivative

- A necessary condition for E to be minimum is

$$\frac{\partial}{\partial a}\hat{L} = -2\sum_{j=1}^{m}(y_j - ax_j - b) = 0$$

$$\frac{\partial}{\partial b}\hat{L} = -2\sum_{j=1}^{m}x_j(y_j - ax_j - b) = 0$$

- We can rewrite as:

$$\begin{cases} bm + a\sum x_j = \sum y_j \\ b\sum x_j + a\sum x_j^2 = \sum x_j y_j \end{cases}$$

which is a system of linear equations in the variables $[a, b]$.

- The solution to this systems gives the values of a and b that minimize the square distance. They can be calculated in closed form.

Analytical Solution

Closed form solution:

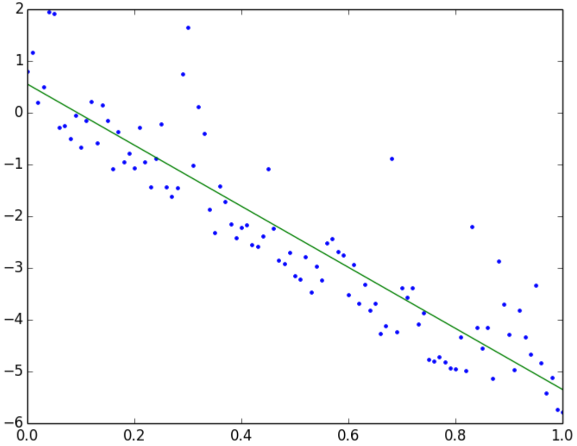
$$a = \frac{\sum (x_j - \bar{x})(y_j - \bar{y})}{\sum (x_j - \bar{x})^2}$$

$$b = \bar{y} - a\bar{x}$$

where:

$$\bar{x} = \frac{1}{m} \sum x_j$$

$$\bar{y} = \frac{1}{m} \sum y_j$$


2
1
0
−1
−2
−3
−4
−5
−6
0.0
0.2
0.4
0.6
0.8
1.0

Learning Task: Overview

Learning = Representation + Evaluation + Optimization

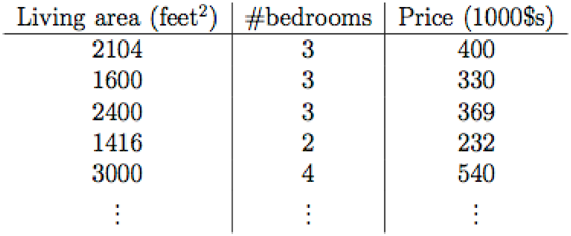
- Representation: formal language that the computer can handle. Corresponds to choosing the set of functions that can be learned, ie. the hypothesis space of the learner. How to represent the input, that is, what features to use.

- Evaluation: an evaluation function (aka objective function or scoring function)

- Optimization. a method to search among the learners in the language for the highest-scoring one. Efficiency issues. Common for new learners to start out using off-the-shelf optimizers, which are later replaced by custom-designed ones.

Outline

Linear Regression with Several Variables



Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
$\vdots$	$\vdots$	$\vdots$

Representation of hypothesis space:

$$h(x) = \theta_0 + \theta_1 x \qquad \text{linear function}$$

if we know another feature:

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

for conciseness, defining $x_0 = 1$

$$h(\vec{\theta}, x) = \vec{\theta} \cdot \vec{x} = \sum_{i=0}^{2} \theta_i x_i$$

Notation:

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

Evaluation
loss function $L(y, h(\vec{x}))$ for penalizing errors in prediction.
Most common is squared error loss:

$$L(y, h(\vec{\theta}, \vec{x})) = (y - h(\vec{\theta}, \vec{x}))^2$$

this leads to minimize:

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

Optimization

$$\hat{L}(\vec{\theta}) = \sum_{j=1}^{p} \left(y_j - h(\vec{\theta}, \vec{x}_j) \right)^2 \qquad \text{cost function}$$

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

Polynomial Regression

Generalize the linear function $y = a + bx$ to a polynomial of degree k

Representation

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

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

$$\vec{x} = \left[1 \; x \; x^2 \; \ldots \; x^n\right]^T$$

Evaluation Then L takes the form

$$\hat{L}(\vec{\theta}) = \sum_{j=1}^{m} (y_j - p(\vec{\theta}, \vec{x}_j))^2$$

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

Optimization

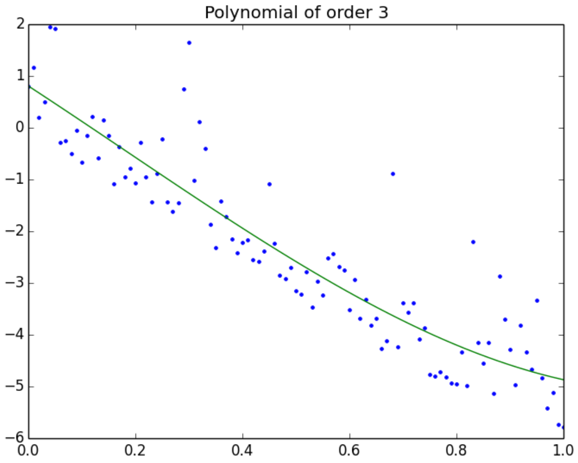
The necessary condition for $\hat{L}$ to be minimum gives a system of $k+1$ linear equations:

$$\frac{\partial}{\partial\theta_0}\hat{L}(\vec{\theta}) = 0$$
$$\frac{\partial}{\partial\theta_1}\hat{L}(\vec{\theta}) = 0$$
$$\vdots$$
$$\frac{\partial}{\partial\theta_m}\hat{L}(\vec{\theta}) = 0$$

For $k=3$ we obtain (summations are all from 1 to m):

$$\begin{cases} \theta_0 m + \theta_1 \sum x_j + \theta_2 \sum x_j^2 + \theta_3 \sum x_j^3 = \sum y_j \\ \theta_0 \sum x_j + \theta_1 \sum x_j^2 + \theta_2 \sum x_j^3 + \theta_3 \sum x_j^4 = \sum x_j y_j \\ \theta_0 \sum x_j^2 + \theta_1 \sum x_j^3 + \theta_2 \sum x_j^4 + \theta_3 \sum x_j^5 = \sum x_j^2 y_j \\ \theta_0 \sum x_j^3 + \theta_1 \sum x_j^4 + \theta_2 \sum x_j^5 + \theta_3 \sum x_j^6 = \sum x_j^3 y_j \end{cases}$$

which can be solved in closed form.


Polynomial of order 3
2
1
0
−1
−2
−3
−4
−5
−6
0.0
0.2
0.4
0.6
0.8
1.0

Basis Functions

Combining several variables with a fixed set of nonlinear functions known as basis functions.

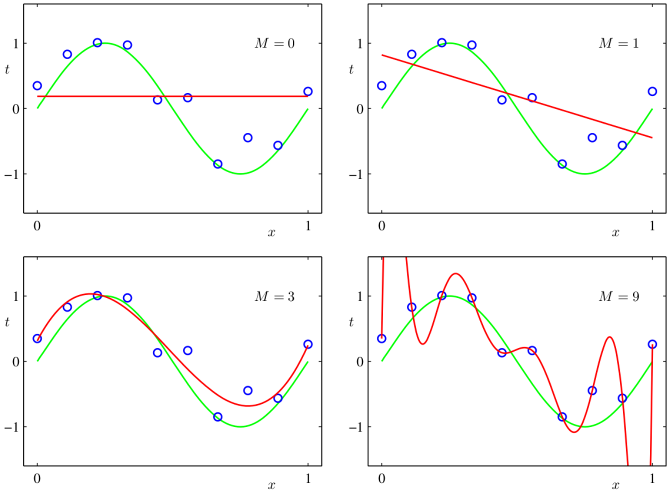
Representation

$$h(\vec{\theta}, \vec{x}) = \theta_0 + \sum_{i=1}^{p} \theta_i x_i + \sum_{i=1}^{p} \sum_{k=1}^{p} \theta_{ik} x_i x_k + \sum_{i=1}^{p} \sum_{k=1}^{p} \sum_{\ell=1}^{p} \theta_{ik\ell} x_i x_k x_\ell$$

$$h(\vec{\theta}, \vec{x}) = \theta_0 + \sum_{j=1}^{p} \theta_j \phi_j(x) = \vec{\theta} \cdot \vec{\phi}(\vec{x})$$

h is now a nonlinear function of input vector $\vec{x}$ but h is linear in $\vec{\theta}$

Overfitting



Training and Assessment

Use different data for different tasks:

- Training and Test data: holdout cross validation
- If small data: k-fold cross validation

Avoid peeking:

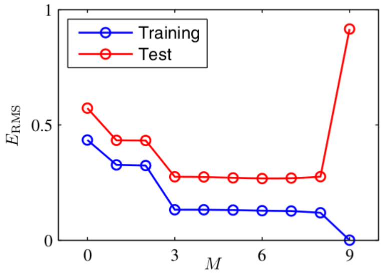
- Weights learned on training data.
- Parameters and model compared on validation data
- Final assessment on test data

Model Comparison

$$L(\vec{\theta}) = \frac{1}{2}\left[h(\vec{\theta},\vec{x}) - y\right]^2$$ on training data
on test data

$$E[L(\vec{\theta})] = \frac{1}{2}\sum_{j=1}^{m}\left[h(\vec{\theta}\vec{x}_j) - y_j\right]^2$$ average loss

$$E_{RMS} = \sqrt{2E[L(\vec{\theta})]/m}$$ root mean square



Outline

Classification and Logistic Regression

Binary classification problem: $y = \{0, 1\}$ or $y = \{-1, 1\}$ (labels)

- We could use the linear regression algorithms that we saw and round.
- Or: we change our hypothesis:

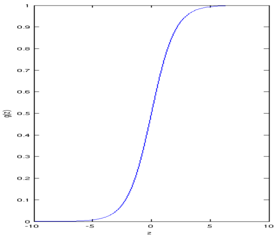
$$h(\vec{\theta}, \vec{x}) = g(\vec{\theta} \cdot \vec{x}) \qquad g : \mathbb{R} \to [0, 1], h : \mathbb{R}^p \to [0, 1].$$

In ML $g(\cdot)$ is called activation function

A common choice for g is the logistic function or sigmoid function:

$$g(z) = \frac{1}{1 + e^{-z}}, \qquad \text{hence}$$

$$h(\vec{\theta}, \vec{x}) = \frac{1}{1 + e^{-\vec{\theta} \cdot \vec{x}}}$$



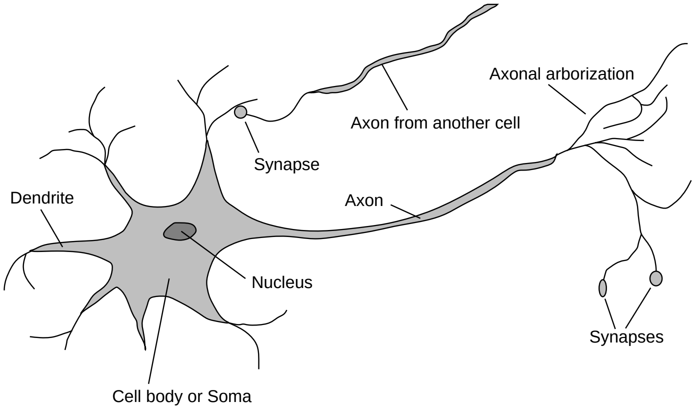
- Note that g is nonlinear in both the parameters and the inputs
- However, the decision surface corresponds to $h(\vec{x}) = \text{constant}$ and hence to a linear function of $\vec{x}$:

  $$\vec{\theta} \cdot \vec{x} = g^{-1}(\text{constant}) = \text{constant}$$

  in one dimension (y, x_1) it is a vertical line
  in two dimensions $(y, (x_1, x_2))$ it is a line
  in three dimensions $(y, (x_1, x_2, x_3))$ it is a plane
  in more dimensions $(y, (x_1, \ldots, x_p))$ it is an hyperplane
- how do we determine $\vec{\theta}$? (see Appendix)

Outline

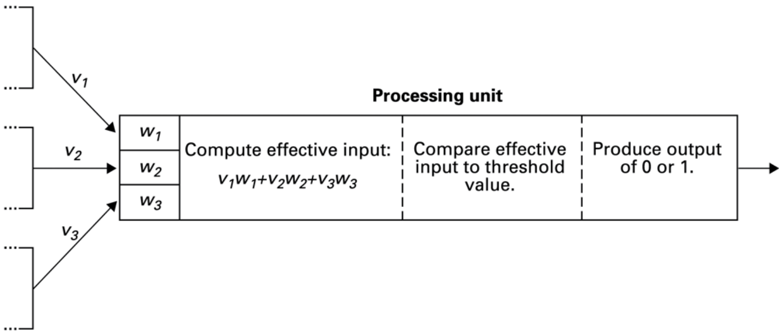
A neuron in a living biological system

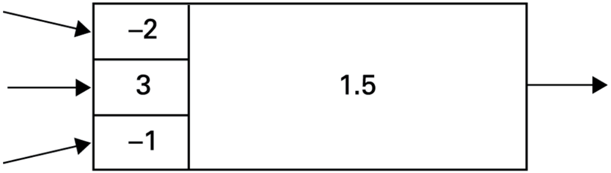


Signals are noisy “spike trains” of electrical potential

McCulloch–Pitts “unit” (1943)

Activities within a processing unit





Artificial Neural Networks

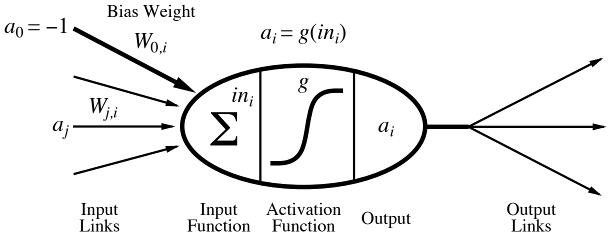
$\rightsquigarrow$ "*The* neural network" does not exist. There are different paradigms for neural networks, how they are trained and where they are used.

- 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.

McCulloch–Pitts “unit” (1943)

Output is a function of weighted inputs:

$$a_i = g(in_i) = g\left(\sum_j w_{ji} a_j\right)$$

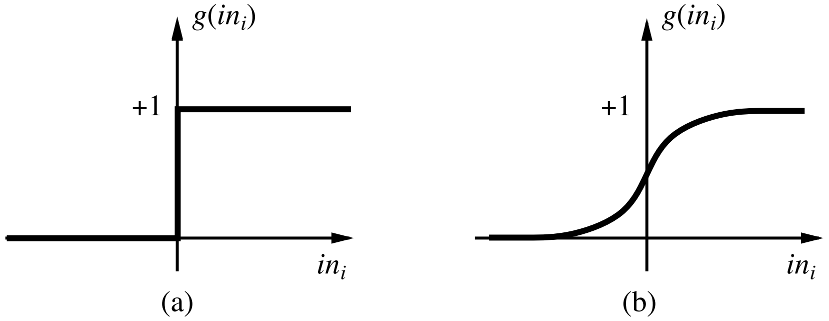


Changing the bias weight $w_{0,j}$ moves the threshold location

A gross oversimplification of real neurons, but its purpose is to develop understanding of what networks of simple units can do

Activation functions

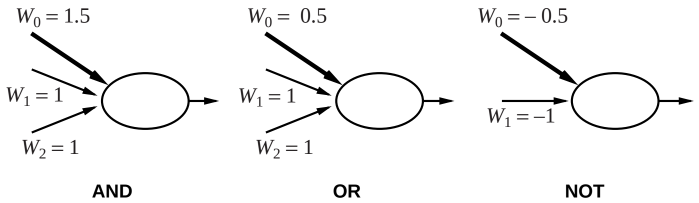
Non linear activation functions



(a) is a step function or threshold function
(mostly used in theoretical studies) $\rightsquigarrow$ perceptron

(b) is a continuous activation function, e.g., sigmoid function $1/(1 + e^{-x})$
(mostly used in practical applications) $\rightsquigarrow$ sigmoid neuron

Implementing logical functions



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

Network structures

Architecture: definition of number of nodes and interconnection structures and activation functions σ but not weights.

- Feed-forward networks:
  no cycles in the connection graph
  - (no hidden layer)
  - (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)
  – for *associative memory*

Feed-Forward Networks – Use

Neural Networks are used in classification and regression

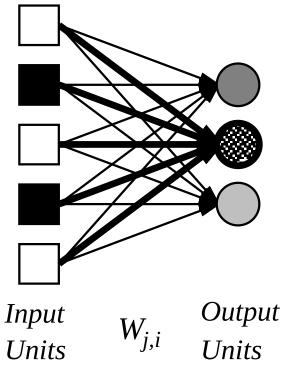
- Boolean classification:
  - value over 0.5 one class
  - value below 0.5 other class

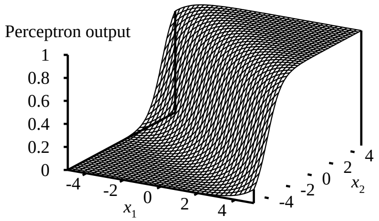
- k-way classification
  - divide single output into k portions
  - k separate output unit

- continuous output
  - identity activation function in output unit

Outline

Single-layer NN



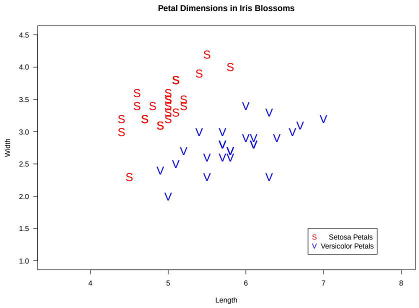


Output units all operate separately—no shared weights
Adjusting weights moves the location, orientation, and steepness of cliff

Numerical Example

Boolean Classification

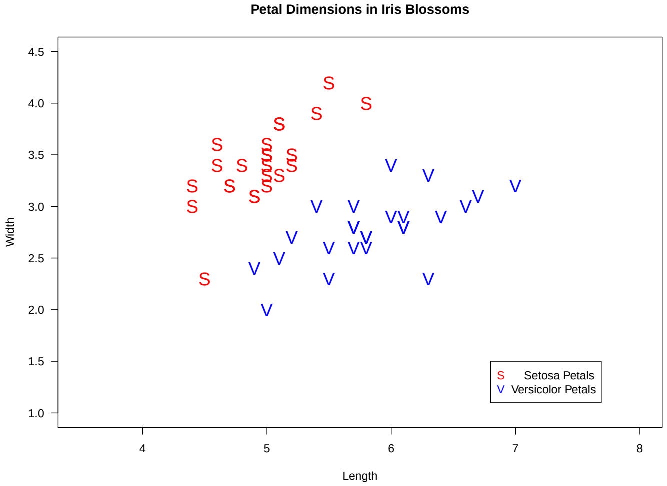
The (Fisher's or Anderson's) iris data set gives measurements in centimeters of the variables: sepal length and petal length and petal width for 50 flowers from 2 species of iris: Iris setosa, and versicolor.



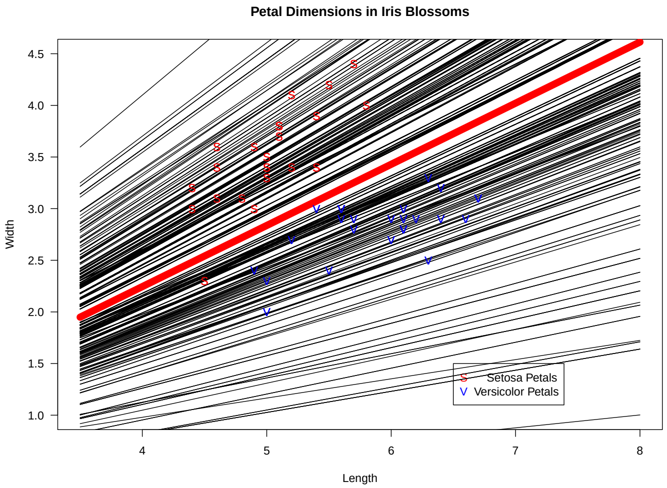
```
iris.data:

Sepal.Length Sepal.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 $0/1$

In two dimensions, training the perceptron corresponds to looking for the line that separates the points at best. (Remember, the decision surface of a linear combination of inputs gives: $\vec{\theta} \cdot \vec{x} = \texttt{constant}$, which in 2D is a line)



Basically, we try different values for the weights moving towards the values that minimize the misprediction of the training data: the red line.
(see Appendix for the algorithm)



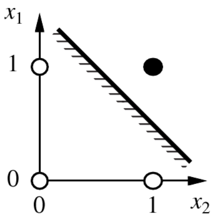
Expressiveness of single perceptrons

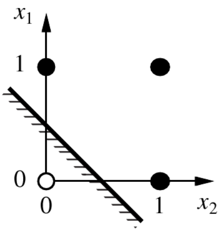
Consider a perceptron with g = step function (Rosenblatt, 1957, 1960)
The output is 1 when:

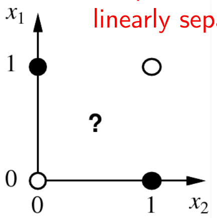
$$\sum_i w_i x_{ij} > 0 \quad \text{or} \quad \vec{w} \cdot \vec{x}_j > 0$$

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



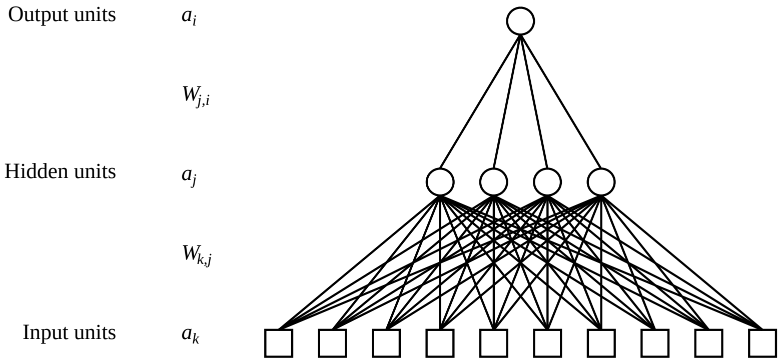




Outline

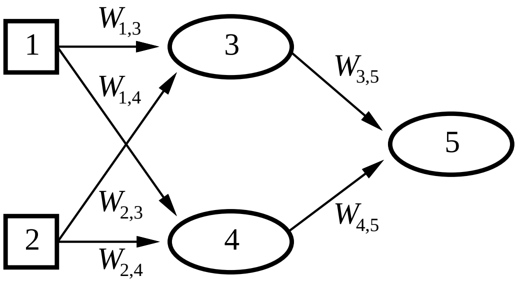
Multilayer perceptrons

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



(a for activation values; W for weight parameters)

Multilayer Feed-forward



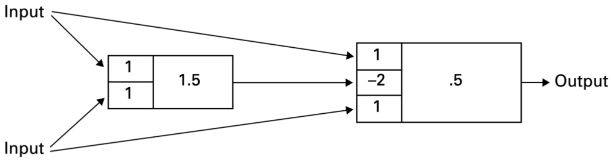
Feed-forward network = a parametrized family of nonlinear functions:

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

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 node (node 3) is:

$$\sum_i w_{i3} a_i = w_{13} \cdot a_1 + w_{23} \cdot a_2 = 1 \cdot 1 + 1 \cdot 0 = 1$$

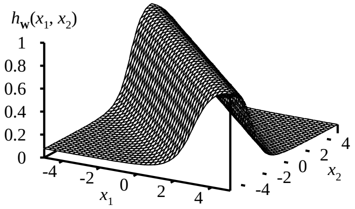
which is < 1.5, hence the output of node 3 is $a_3 = g(\sum_i w_{i3} a_i) = 0$.
The input to the second node (node 4) is:

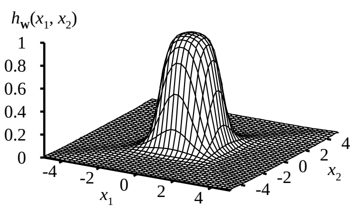
$$\sum_i w_{i4} a_i = w_{14} \cdot a_1 + w_{34} \cdot a_3 + w_{24} \cdot a_{24} = 1 \cdot 1 - 2 \cdot 0 + 1 \cdot 0 = 1$$

which is > 0.5, hence the output of the node 4 is $a_3 = g(\sum_i w_{i4} a_i) = 1$.

Expressiveness of MLPs

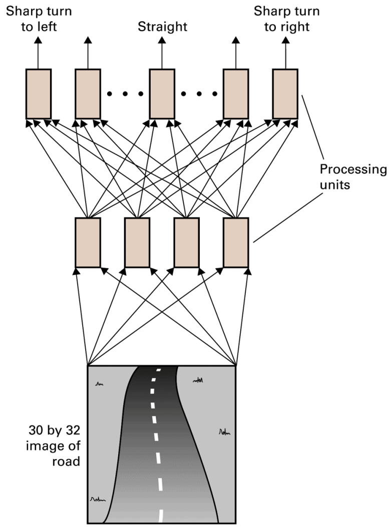
All continuous functions with 2 layers, all functions with 3 layers





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

A Practical Example



Deep learning ≡
convolutional neural networks ≡
multilayer neural network with structure on the arcs
for 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.

Summary

1. Mathematical Concepts
2. Machine Learning
3. Linear Regression
   - Extensions
4. Logistic Regression
5. Artificial Neural Networks
   - Single-layer Networks
   - Multi-layer perceptrons

Part I

Appendix

Outline

Outline

Perceptron learning

Learn by adjusting weights to reduce error on training set.

The squared error for an example with input $\vec{x}$ and true output y is

$$L = \frac{1}{2}(y - h(\vec{w}, \vec{x}))^2 ,$$

Find local optima for the minimization of the function $\hat{L}(\vec{w})$ in the vector of variables $\vec{\theta}$ by gradient methods.

Note, the function $\hat{L}$ depends on constant values $\vec{x}$ that are the inputs to the perceptron.

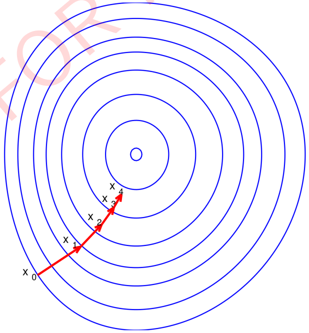
The function $\hat{L}$ depends on h which is non-convex, hence the optimization problem cannot be solved just by solving $\nabla \hat{L}(\mathbf{W}) = 0$

Digression: Gradient methods

Gradient methods are iterative approaches:

- find a descent direction with respect to the objective function $\hat{L}$
- move $\vec{w}$ in that direction by a step size

The descent direction can be computed by various methods, such as gradient descent, Newton-Raphson method and others. The step size can be computed either exactly or loosely by solving a line search problem.



Digression: Gradient methods

Example: gradient descent

```
1. Set iteration counter $t = 0$, and make an initial guess $\vec{w}_0$ for the
   minimum
2. Repeat:
3.     Compute a descent direction $\mathbf{p}_t = \nabla(\hat{L}(\vec{w}_t))$
4.     Choose $\alpha$ to minimize $f(\alpha) = \hat{L}(\vec{w}_t - \alpha\vec{p}_t)$ over $\alpha \in \mathbb{R}_+$
5.     Update $\vec{w}_{t+1} = \vec{w}_t - \alpha\vec{p}_t$, and $t = t + 1$
6. Until $\|\nabla\hat{L}(\vec{w}_t)\| < tolerance$
```

Step 4 can be solved 'loosely' by taking a fixed small enough value $\alpha > 0$

Digression: Perceptron learning

In the specific case of the perceptron, the descent direction is computed by the gradient:

$$\frac{\partial}{\partial \theta_i}\hat{L}(\vec{\theta}) = \frac{1}{2}\cdot 2\cdot \hat{L}(\vec{\theta})\frac{\partial}{\partial w_i}\hat{L}(\vec{\theta}) = \hat{L}(\vec{\theta})\frac{\partial}{\partial w_i}\left(y - g\left(\sum_{i=0}^{p} w_i x_{ij}\right)\right)$$
$$= -\hat{L}(\vec{\theta})\cdot g'(in)\cdot x_i$$

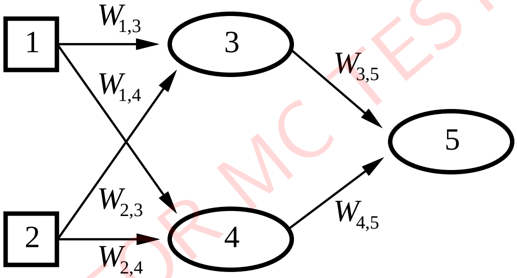
and the weight update rule (perceptron learning rule) in step 5 becomes:

$$w_{t+1,i} = w_{t,i} + \alpha\cdot \hat{L}(\vec{\theta})\cdot g'(in)x_i$$

- For threshold perceptron, $g'(in)$ is undefined:
  Original perceptron learning rule (Rosenblatt, 1957) simply omits $g'(in)$
- For sigmoid function can be calculated. See earlier.

Outline

Multilayer perceptron learning: Back-propagation



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!

Multilayer perceptron learning: Back-propagation

Output layer: same as for single-layer perceptron,

$$w_{\ell i} \leftarrow w_{\ell i} + \alpha \cdot a_\ell \cdot \Delta_i$$

where $\Delta_i = \hat{L}_i \cdot g'(\hat{L}_i)$.

Note: the general case has multiple output units hence: $\hat{\vec{L}} = (\mathbf{y} - \mathbf{h}(\vec{w}, \vec{\phi}(\vec{x})))$

Hidden layer: *back-propagate* the error from the output layer:

$$\Delta_j = g'(\text{in}_j) \sum_i w_{\ell i} \Delta_i \qquad \text{(sum over the multiple output units)}$$

Update rule for weights in hidden layer:

$$w_{k\ell} \leftarrow w_{k\ell} + \alpha \cdot a_k \cdot \Delta_\ell \ .$$

Back-propagation derivation

The squared error on a single example j is defined as

$$\hat{L} = \frac{1}{2}\sum_i (y_{ij} - a_{ij})^2$$

where the sum is over the nodes in the output layer.
Let's drop the index j for the sake of simplicity

$$\begin{aligned}
\frac{\partial \hat{L}}{\partial w_{\ell i}} &= -(y_i - a_i)\frac{\partial a_i}{\partial w_{\ell i}} = -(y_i - a_i)\frac{\partial g(\text{in}_i)}{\partial w_{\ell i}} \\
&= -(y_i - a_i)g'(\text{in}_i)\frac{\partial \text{in}_i}{\partial w_{\ell i}} = -(y_i - a_i)g'(\text{in}_i)\frac{\partial}{\partial w \ell i}\left(\sum_\ell w_{\ell i} a_\ell\right) \\
&= -(y_i - a_i)g'(\text{in}_i)a_\ell = -a_\ell \Delta_i
\end{aligned}$$

Back-propagation derivation contd.

For the hidden layer:

$$\begin{aligned}
\frac{\partial \hat{L}}{\partial w_{k\ell}} &= -\sum_i (y_i - a_i)\frac{\partial a_i}{\partial w_{k\ell}} = -\sum_i (y_i - a_i)\frac{\partial g(\mathrm{in}_i)}{\partial w_{k\ell}} \\
&= -\sum_i (y_i - a_i) g'(\mathrm{in}_i)\frac{\partial \mathrm{in}_i}{\partial w_{k\ell}} = -\sum_i \Delta_i \frac{\partial}{\partial w_{k\ell}}\left(\sum_\ell w_{\ell i} a_\ell\right) \\
&= -\sum_i \Delta_i w_{\ell i}\frac{\partial a_\ell}{\partial w_{k\ell}} = -\sum_i \Delta_i w_{\ell i}\frac{\partial g(\mathrm{in}_\ell)}{\partial w_{k\ell}} \\
&= -\sum_i \Delta_i w_{\ell i} g'(\mathrm{in}_\ell)\frac{\partial \mathrm{in}_\ell}{\partial w_{k\ell}} \\
&= -\sum_i \Delta_i w_{\ell i} g'(\mathrm{in}_\ell)\frac{\partial}{\partial w_{k\ell}}\left(\sum_k w_{k\ell} a_k\right) \\
&= -\sum_i \Delta_i w_{\ell i} g'(\mathrm{in}_\ell) a_k = -a_k \Delta_\ell
\end{aligned}$$