Malware Detection with Machine Learning Methods (IN2207)

Thomas Stibor

Fakultät für Informatik
Technische Universität München

thomas.stibor@in.tum.de

SS2011

Organizational

- Language: English
- Lecture: Tuesday, 13:00 - 14:30, MI 01.06.011
- Exercise course: Friday, 10:00 - 10:45, MI 01.06.011
- Credits: 2+1 SWS / 4,0 ECTS
- Final exam (oral)

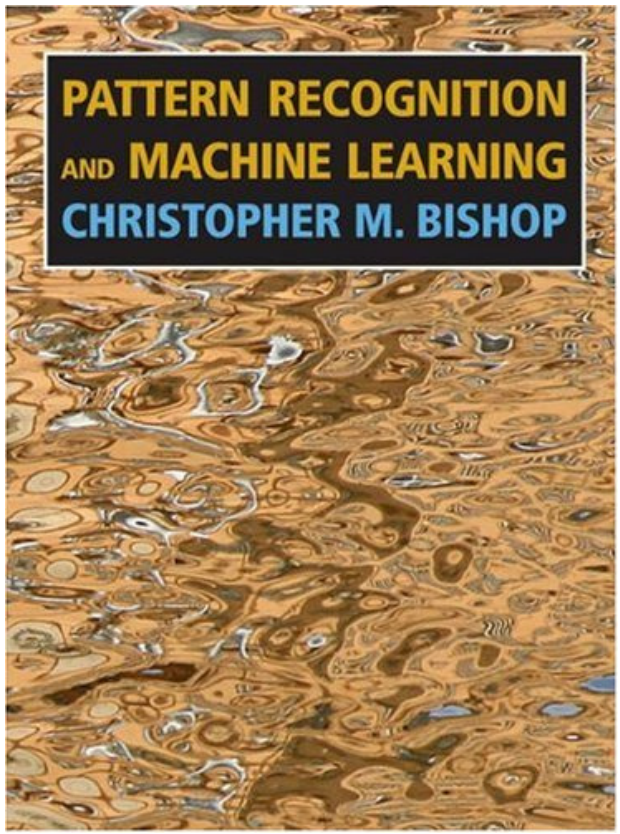
The homeworks will contain written questions and questions that require some R programming.

Exercise course on Friday 06.05.2011 will be an introduction to R (programming).

Overview

- Theoretical Limits of Virus Scanners
- Brief Warm-Up in Probability
- Naive Bayes Classifier
- Probabilistic Models (HMM, Topic Models)
- Linear Classifiers
- Neural-Networks and Support Vector Machines
- Feature Selection & Extraction
- Anomaly Detection
- Adversarial Learning

Literature



Some figures are taken from Bishop's new book (Pattern Recognition and Machine Learning).

Theoretical Limits of Virus Scanners

Can any virus be detected? What are the limits?

- Identifying a mutating virus is $\mathcal{NP}$-complete.

  Reliable Identification of Bounded-length Viruses is $\mathcal{NP}$-complete, Diomidis Spinellis, IEEE Transactions on Information Theory, Vol. 49, No. 1 (2003), pp. 280-284.

- Detecting viruses is undecidable.

  Computer Viruses: Theory and Experiments, Fred Cohen, Computers & Security, 6 (1987), pp. 22-35.

Computer Virus

Definition: A computer virus is a program that can infect other programs by modifying them to include a possibly evolved copy of it self. [Fred Cohen, Computer Viruses: Theory and Experiments, 1987]

```
program virus := 1234567;
{
  subroutine infect-executable := {
    loop:file = get-random-executable-file;
    if first-line-of-file = 1234567 then goto loop:
    prepend virus to file;
  }
  subroutine do-damage := {whatever damage is to be done}
  subroutine trigger-pulled := {return true if some condition}

  main-program := { infect-executable;
    if trigger-pulled then do-damage;
    goto next;
  }
  next: /* execute rest of the program it was prepended to. */
}
```

Identifying a Virus is $\mathcal{NP}$-complete

Idea:

Show that a virus detector D for a certain virus strain V can be used to solve the satisfiability problem, which is known to be $\mathcal{NP}$-complete.

- The satisfiability of the problem which is examined is however *not* a special case.
- The virus V is a mutating self-replicating program.
- Assume that the virus detector D can reliably determine in polynomial time whether a given candidate program C is a mutation of the virus V.
- We will use the virus detector as an oracle for determining the satisfiability of an N-term Boolean formula S.

Link between SAT and Virus Instance

Use the satisfiability formula S to create a virus archetype A and a possible instance of a virus phenotype P.
The virus is a triple

$$(f, s, c)$$

where

- f is the virus processing and replication function,
- s is a Boolean value indicating whether an instance of the virus has found a solution to S,
- c is an integer encoding the candidate values for S.

The function f maps a triple (f, s, c) into a new triple (f, s', c') and is defined as follows

$$\lambda(f, s, c).(f, s \vee S, \text{ if } c = 2^N \text{ then } c \text{ else } c + 1).$$

Link between SAT and Virus Instance (cont.)

Each term x_n in S is calculated from c through the expression

$$\left\lfloor \frac{c}{2^n} \right\rfloor \mod 2 \stackrel{?}{=} 1$$ which results in a Boolean value.

bit position c			decimal value
2	1	0	n
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	**4**
1	0	1	5
1	1	**0**	**6**
1	1	1	7

Is bit at position $c = 2$ set to 1 for decimal value $n = 4$? Result is TRUE.

Is bit at position $c = 0$ set to 1 for decimal value $n = 6$? Result is FALSE.

Will a Virus Mutation ever Satisfy S

A new generation of the virus is generated by applying f to the current generation.

1. evaluates S by extracting successive Boolean value combinations from c,
2. increments c until it reaches 2^N,
3. passes the result of the S evaluation to the next generation.

We can now ask virus detector D whether the virus archetype A

$$(f, \texttt{FALSE}, 0)$$

will ever result in a virus mutation phenotype P

$$(f, \texttt{TRUE}, 2^N)$$

that is whether one of the virus mutations will satisfy S.

Example

Consider the satisfiability of the formula

$$S = (x_0 \vee x_1) \wedge \overline{x}_0.$$

The virus replication function f will be

$$\lambda(f, s, c).(f, s \vee (x_0 \vee x_1) \wedge \overline{x}_0, c + 1)$$

the corresponding archetype A

$$(\lambda(f, s, c).(f, s \vee (x_0 \vee x_1) \wedge \overline{x}_0, c + 1), \mathtt{F}, 0),$$

where `F` denotes `FALSE` and `T` denotes `TRUE`.

Example (cont.)

This particular virus will generate a mutation P and thereby indicate that S is satisfiable in four generations through the following sequence

$$
\begin{aligned}
(\lambda(f,s,c).(f,s \vee S, c+1), \mathtt{F} \vee (\mathtt{F} \vee \mathtt{F}) \wedge \overline{\mathtt{F}}, 0+1) &\rightarrow \\
(\lambda(f,s,c).(f,s \vee S, c+1), \mathtt{F}, 1) &\rightarrow \\
(\lambda(f,s,c).(f,s \vee S, c+1), \mathtt{F} \vee (\mathtt{T} \vee \mathtt{F}) \wedge \overline{\mathtt{T}}, 1+1) &\rightarrow \\
(\lambda(f,s,c).(f,s \vee S, c+1), \mathtt{F}, 2) &\rightarrow \\
(\lambda(f,s,c).(f,s \vee S, c+1), \mathtt{F} \vee (\mathtt{F} \vee \mathtt{T}) \wedge \overline{\mathtt{F}}, 2+1) &\rightarrow \\
(\lambda(f,s,c).(f,s \vee S, c+1), \mathtt{T}, 3) &\rightarrow \\
(\lambda(f,s,c).(f,s \vee S, c+1), \mathtt{T} \vee (\mathtt{T} \vee \mathtt{T}) \wedge \overline{\mathtt{T}}, 3+1) &\rightarrow \\
(\lambda(f,s,c).(f,s \vee S, c+1), \mathtt{T}, 4) &\equiv P.
\end{aligned}
$$

Summary

It was shown that a reliable virus detector D operating in polynomial time can be used as a polynomial time satisfiable oracle and hence, reliable detection of a mutating virus is $\mathcal{NP}$-complete

IEEE TRANSACTIONS ON INFORMATION THEORY, VOL. 49, NO. 1, JANUARY 2003 (DRAFT) 280

Reliable Identification of Bounded-length Viruses is NP-complete

Diomidis Spinellis, *Member, IEEE*

***Abstract*—A virus is a program that replicates itself by copying its code into other files. A common virus protection mechanism involves scanning files to detect code patterns of known viruses. We prove that the problem of reliably identifying a bounded-length mutating virus is NP-complete by showing that a virus detector for a certain virus strain can be used to solve the satisfiability problem. The implication of this result is that virus identification methods will be facing increasing strain as virus mutation and hosting strategies mature, and that different protection methods should be developed and employed.** [1] [2]

***Index Terms*—buffer-overflow, complexity, detection, identification, mutation, NP-complete, security, virus**

I. Introduction

ONE often-used defence against computer viruses is the execution of an *anti-virus* program that detects and cleans programs that appear to be infected. Virus writers respond to this defence by trying to thwart anti-virus software through targeted attacks, mutations, or social engineering. Mutating viruses are a particularly insidious threat, because detection algorithms need to be constantly updated and to spend increasing processing time to identify new mutation types. The question of whether complexity theory is on the side of virus writers or the protection vendors could have important practical implications. In this paper we will prove that there exist realistic viruses whose reliable detection is of NP-complete complexity [1] and that therefore the general problem of reliable bounded-length virus identification is NP-complete.

II. Viral Software

Intentionally created malicious software [2]—often termed *malware*—is typically classified into Trojan horses, viruses, and worms [3]. A Trojan horse is a program that exploits the rights of its user to perform an action its user does not intend, a virus is a Trojan horse that replicates itself by copying its code into other program files [4], while a worm is an independently-running program that replicates through a network exploiting security weaknesses to invade other computers.

A number of virus prevention and detection methods have been proposed and are commonly implemented [5], [6]. Reference [7] contains an annotated bibliography of malware analysis and detection papers. Prevention methods involve limiting the flow of information between programs through the use of appropriate hardware and software protection domains, coupled with self-defence mechanisms, instrumentation, and fault-tolerance. Since the above methods will typically interfere with many legitimate operations (such as the installation of new software or the correction of an existing version) they need to be coordinated through carefully designed and executed security procedures. Unfortunately current practice in system administration often renders these methods useless. A large percentage of users typically administer their personal workstations on their own, in most cases exercising the full rights of the system administrator, without sufficient training and diligence.

Therefore, as a secondary line of defence, detection measures are often employed to locate virus instances and infections. Two often used detection measures involve either the comparison of the system's programs against known-good versions (typically condensed in the form of a checksum or a cryptographically secure signature [8]) or the comparison of files against patterns of known viruses. Since the first method depends on a known-clean system and can not be used to check software of unknown origin, the second, scanning, method is the one most commonly employed. A number of software vendors provide virus-scanning software that can search new and existing system files for patterns of all known viruses. The vendors regularly distribute updated versions of the virus patterns to keep the virus detection process up-to-date.

Virus writers however, have developed a series of countermeasures. Even early academic examples of viral code were cleverly engineered to hinder the detection of the virus [9]. Since the actual task of writing a virus is relatively simple [10], [11] modern virus code focuses on employing platform independence, stealth, effective replication, and detection countermeasures. Three pattern-matching detection countermeasures typically employed are the *encryption* of the virus body with a variable cryptographic key, the *polymorphic* generation of the decryption routine using equivalent code instructions, and, more recently, the *metamorphic* generation of the whole virus body through the addition, removal, permutation, and substitution of code sequences. Viruses that employ these techniques, such as W32/Simile [12] can be very difficult to identify. In the following section we establish that reliably detecting instances of such viruses is a problem of NP-complete complexity.

D. Spinellis is an Assistant Professor in the Department of Management Science and Technology at the Athens University of Economics and Business, Athens, Greece. E-mail: dds@aueb.gr .

[1] *IEEE Transactions on Information Theory*, 49(1):280–284, January 2003.

[2] This is a machine-readable rendering of a working paper draft that led to a publication. The publication should always be cited in preference to this draft using the reference in the previous footnote. This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder.

Virus Detection is Undecidable

Virus detection is undecidable, that is, there is no program that detects all viruses.

To proof this statement one has to show the undecidability of the language

$$V_{\mathrm{TM}} = \{\langle M, \langle P \rangle\rangle \mid M \text{ is a TM and } M \text{ detects whether } P \text{ is a virus}\}.$$

Proof sketch:

Suppose that V_{TM} is decidable and H is a decider for V_{TM}.

$$H(\langle M, \langle P \rangle\rangle) = \begin{cases} \texttt{TRUE} & \text{if } M \text{ detects } P \text{ as a virus} \\ \texttt{FALSE} & \text{if } M \text{ fails to detect } P \text{ as a virus.} \end{cases}$$

Virus Detection is Undecidable (cont.)

- On input $\langle M, \langle P \rangle \rangle$, where M is a TM and $\langle P \rangle$ a string which encodes program P, H outputs `TRUE` if M detects P as a virus. In contrast, H outputs `FALSE` if M fails to detect P as a virus.
- Construct a new Turing machine D with H as a subroutine.
- This new TM calls H to determine what M does when the input to M is its own description $\langle M \rangle$.
- Once D has determined this information, it does the opposite. That is, it outputs `FALSE` if M outputs `TRUE` and it outputs `TRUE` if M outputs `FALSE`. The following is a description of D.

Virus Detection is Undecidable (cont.)

$D =$ On input $\langle M \rangle$, where M is a TM:

1. Run H on input $\langle M, \langle M \rangle \rangle$.
2. Output the opposite of what H outputs; that is, if H outputs `TRUE`, then output `FALSE` and if H outputs `FALSE`, then output `TRUE`.

More formally

$$D(\langle M \rangle) = \begin{cases} \texttt{TRUE} & \text{if } M \text{ fails to detect } \langle M \rangle \text{ as a virus} \\ \texttt{FALSE} & \text{if } M \text{ detects } \langle M \rangle \text{ as a virus.} \end{cases}$$

What happens when we run D with its own description $\langle D \rangle$ as input?

Virus Detection is Undecidable (cont.)

In that case we get

$$D(\langle D \rangle) = \begin{cases} \texttt{TRUE} & \text{if } D \text{ fails to detect } \langle D \rangle \text{ as a virus} \\ \texttt{FALSE} & \text{if } D \text{ detects } \langle D \rangle \text{ as a virus.} \end{cases}$$

No matter what D does, it is forced to do the opposite and hence this results in a contraction.

In summary

- H outputs TRUE on input $\langle M, \langle P \rangle \rangle$ exactly when M detects P as a virus.
- D fails to detect $\langle M \rangle$ as a virus exactly when M detects $\langle M \rangle$ as a virus.
- D fails to detect $\langle D \rangle$ as a virus exactly when D detects $\langle D \rangle$ as a virus. 

Machine Learning for Information Security Problems

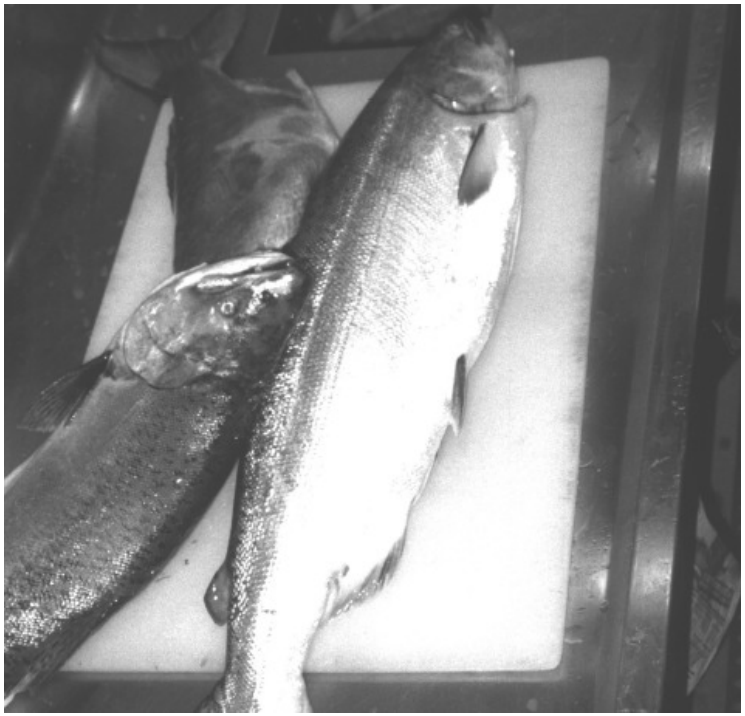
- Spam detection: Text analysis, document clustering.
- Intrusion detection: Anomaly detection, one-class learning.
- Detection of malicious behavior of processes: Sequential data analysis of system calls.
- Credit card fraud detection: Time series analysis of transactions.
- ...

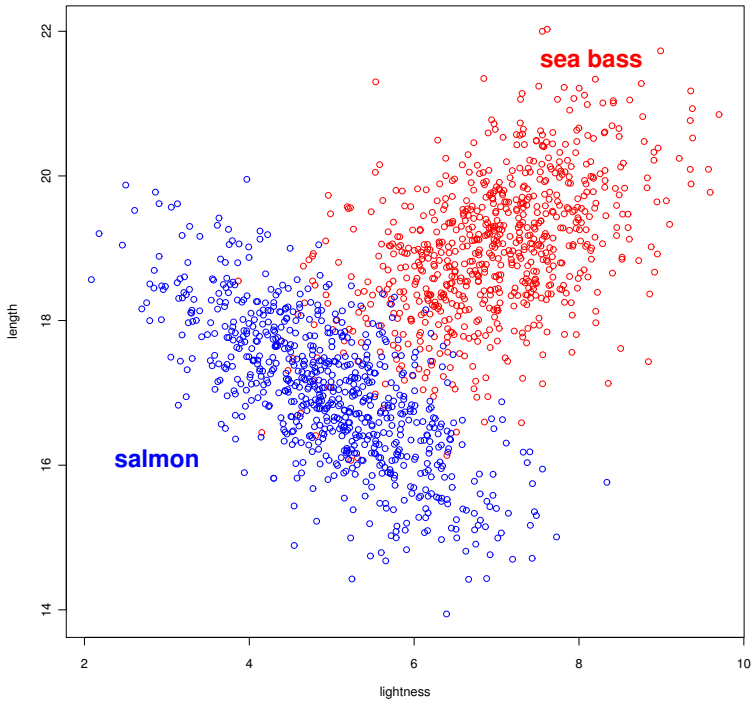
Machine learning methods can be applied for such kind of problems.

The goal of this course is to learn about the machine learning methods.

An Introductory Example

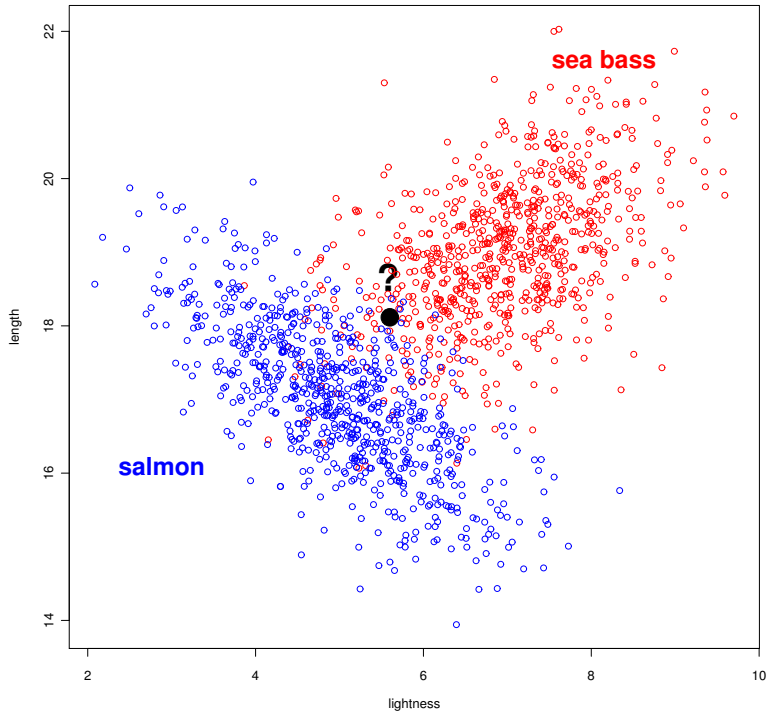
Suppose that a fishpacking factory wants to automate the process of sorting incoming fish (salmon and sea bass).





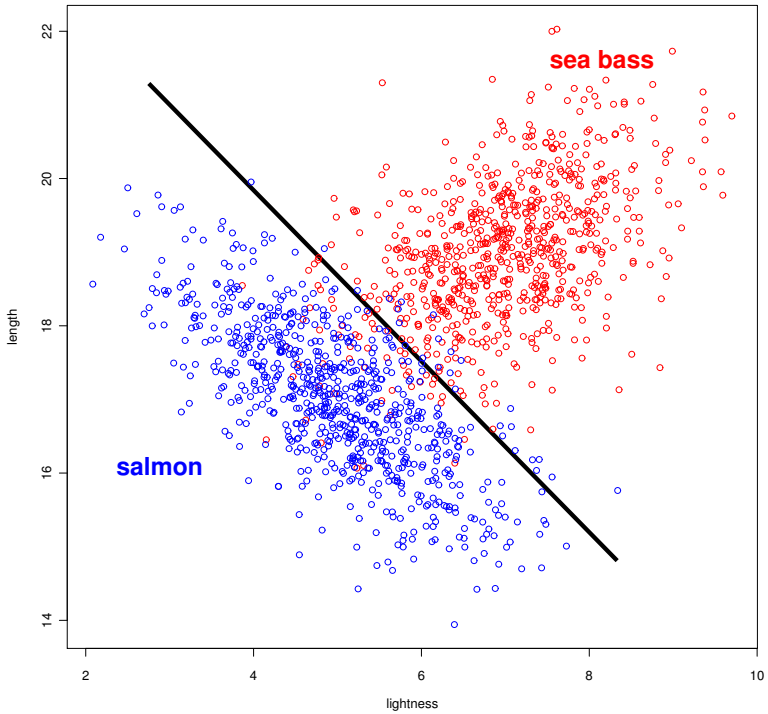
After some preprocessing, each fish is characterized by feature vector $\mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$ (pattern), where the first component is the lightness and the second component the length.

Pattern belongs to Class?



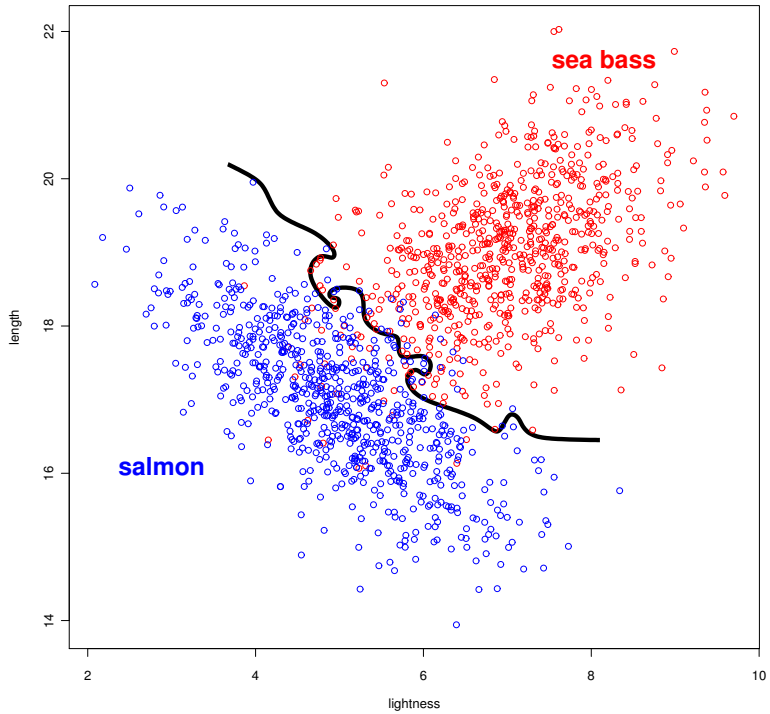
Given labeled training data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N) \in \mathbb{R}^n \times Y$ coming from some unknown probability distribution $P(\mathbf{x}, y)$. In this example, $Y \in \{\text{salmon}, \text{sea bass}\}$ and $n = 2$. Unseen (unlabeled) pattern belongs to class salmon or sea bass?

A (too underfitted) Classifier



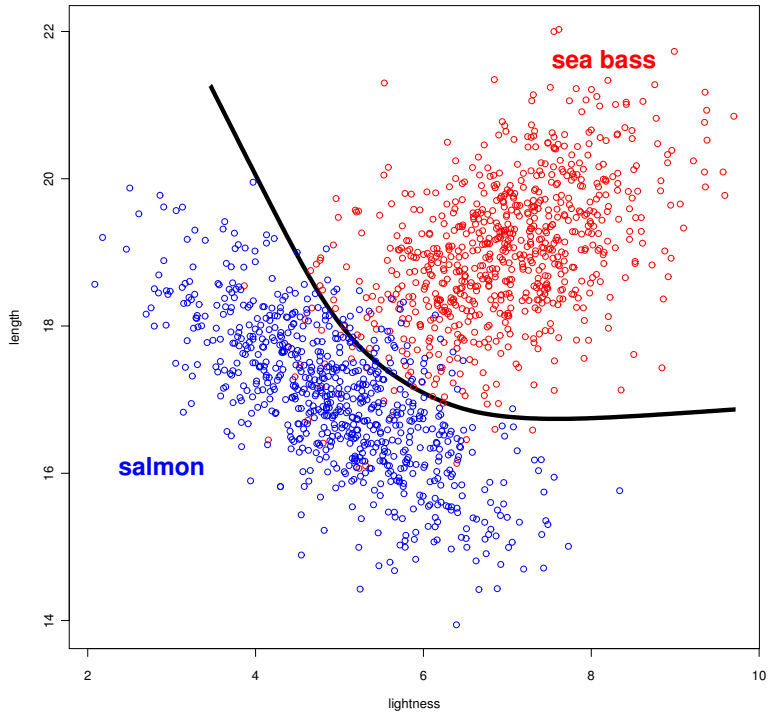
This linear separation suggests the rule: Classify the fish as salmon if its features falls below the *decision boundary*, otherwise as sea bass.

A (too overfitted) Classifier



A too complex model will lead to decision boundary that gives perfect classification accuracy on trainig set (seen patterns), but poor classification on *unseen* patterns.

A Good Classifier



Optimal tradeoff between performance on the training set and simplicity of the model. This gives high classification accuracy on unseen patterns, i.e. it gives good *generalization*.

Probability Theory

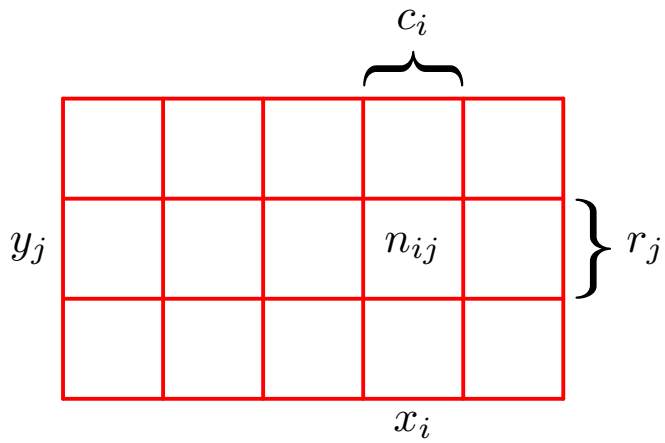
- *Uncertainty* is key concept in machine learning.
- Probability provides consistent framework for the *quantification* and *manipulation* of uncertainty.

> Probability of an event is the fraction of times that event occurs out of the total number of trials, in the limit that the total number of trials goes to infinity.

Elementary rules of probability

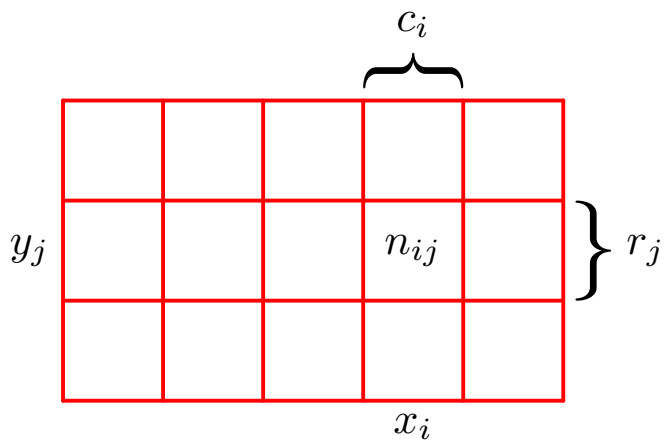
- Sum rule.
- Product rule.

Derive Sum and Product Rules



- Random variable X can take values $\{x_i\}, i = 1, 2, \ldots, M$ $(M = 5)$, Y can take values $\{y_j\}, j = 1, 2, \ldots, L$ $(L = 3)$.
- Consider a total number N of instances of these variables, and denote the number of instances where $X = x_i$ and $Y = y_i$ by n_{ij} (number of points in the corresponding cell of the array).

Derive Sum and Product Rules (cont.)



- Number of points in column i, corresponding to $X = x_i$, is denoted by c_i.
- Number of points in row j, corresponding to $Y = y_i$, is denoted by r_j.

Joint Probability

Probability that X will take value x_i *and* Y will take the value y_j is written

$$p(X = x_i, Y = y_i) = \frac{n_{ij}}{N}$$

where we are implicitly considering the limit $N \to \infty$.
Put it the other way around:

- Number of points falling in the cell i, j as a fraction of the total number of points.

Probability that X takes the value x_i irrespective of the value of Y is

$$p(X = x_i) = \frac{c_i}{N}$$

- Fraction of the total number of points that fall in column i.

Sum Rule

Number of instances in column i is just the sum of the number of instances in each cell of that column, we have $c_i = \sum_j n_{ij}$ and hence

$$\begin{aligned} p(X = x_i) &= \frac{c_i}{N} \\ &= \frac{\sum_{j=1}^{L} n_{ij}}{N} \\ &= \sum_{j=1}^{L} p(X = x_i, Y = y_j). \end{aligned}$$

Sometimes also called *marginal* probability, because it is obtained by marginalizing, or summing out, the other variables (here Y).

Product Rule

Now we are interested in finding the fraction of those points in column i that fall in cell i, j, that is

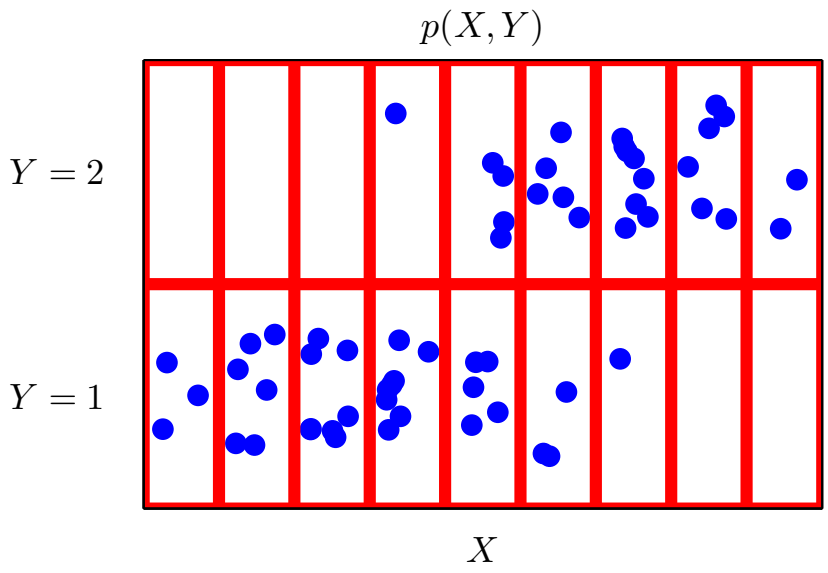
$$p(Y = y_j|X = x_i) = \frac{n_{ij}}{c_i}.$$

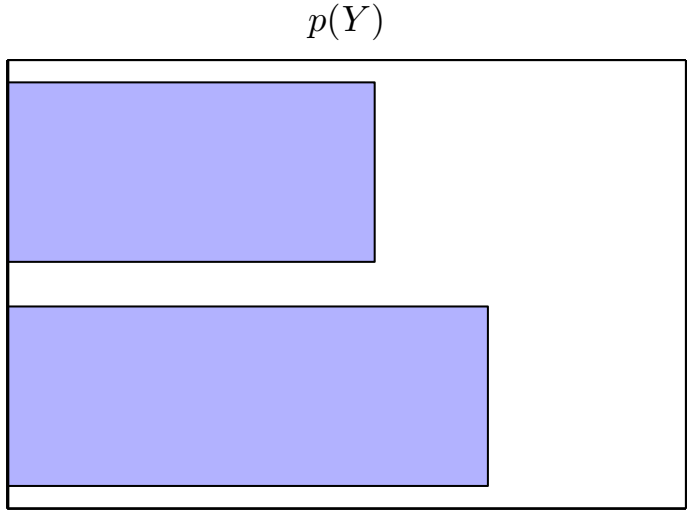
By means of this *conditional* probability, we can derive the following relationship

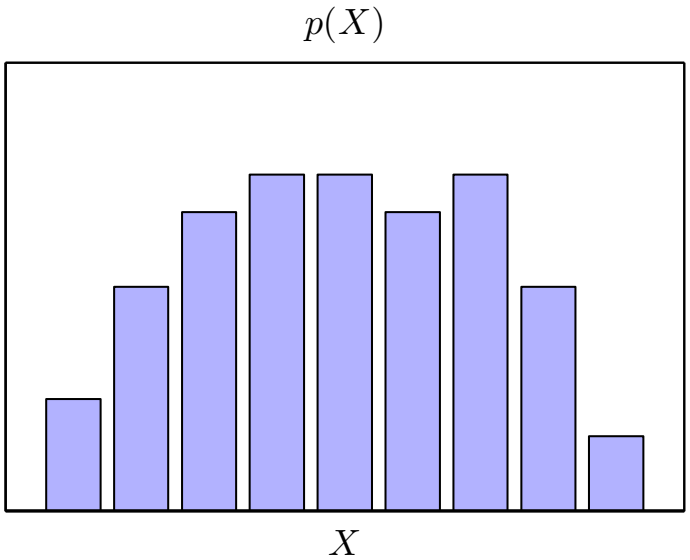
$$\begin{aligned} p(X = x_i, Y = y_j) &= \frac{n_{ij}}{N} = \frac{n_{ij}}{c_i} \cdot \frac{c_i}{N} \\ &= p(Y = y_j|X = x_i)p(X = x_i) \end{aligned}$$

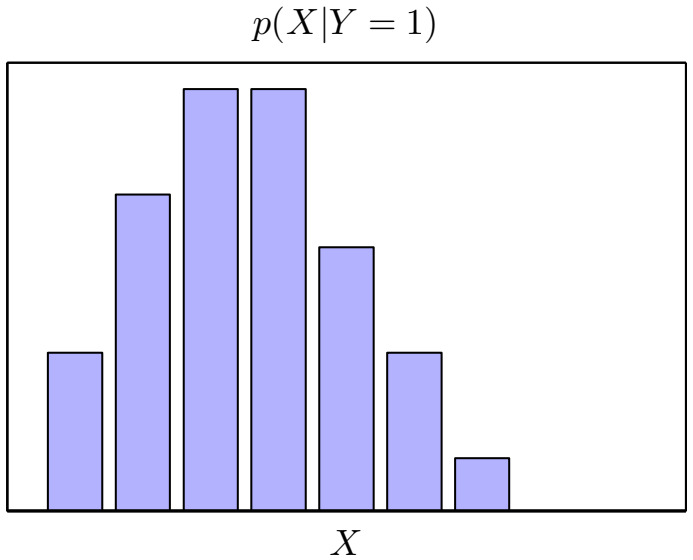
which is the product rule of probabilities.

Illustrative Example









Sum and Product Rule (Summary)

Rules of Probability

sum rule $$p(X) = \sum_{Y} p(X, Y)$$

product rule $$p(X, Y) = p(Y|X)p(X)$$

Bayes' theorem

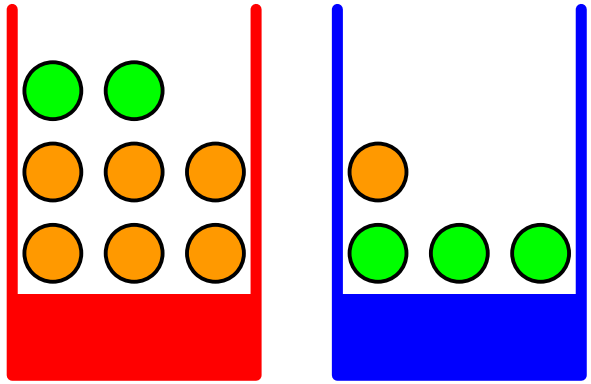
Using symmetry property $p(X, Y) = p(Y, X)$ one can derive the following relation between conditional probabilities

$$\begin{aligned} p(X, Y) &= p(Y, X) \\ p(Y|X)p(X) &= p(X|Y)p(Y) \\ p(Y|X) &= \frac{p(X|Y)p(Y)}{p(X)} \end{aligned}$$

where the denominator can be expressed by means of the sum rule as

$$\begin{aligned} p(X) &= \sum_Y p(X, Y) = \sum_Y p(Y, X) \\ &= \sum_Y p(X|Y)p(Y) \end{aligned}$$

Bishop's Bayes Example



The red box contains 6 oranges and 2 apples, the blue box contains 1 orange and 3 apples. Suppose we pick the red box 40 % of the time and the blue box 60 % of the time. Thanks to Bayes' theorem we can answer questions such as:

- What is the overall probability that we pick an apple?
- Given that we have chosen an orange, what is the probability that the box we chose was the blue one?

Bishop's Bayes Example (cont.)

For the sake of clarity let's introduce random variables B for box and F for fruit. B can take one of the two possibilities $B = r$ (for red) and $B = b$ (for blue); and $F = o$ (for orange) and $F = a$ (for apple).

The *prior probability* of selecting the red box is

$$p(B = r) = \frac{4}{10}$$

and of selecting the blue box

$$p(B = b) = \frac{6}{10}$$

Bishop's Bayes Example (cont.)

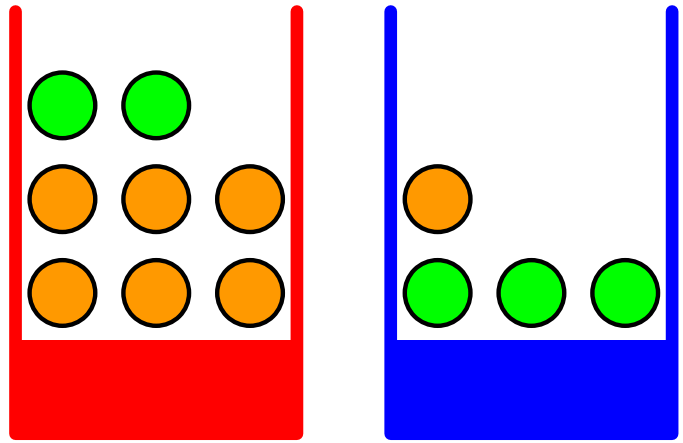
From given information we can write out all four *conditional probabilities* of given the selected box and picking the type of fruit.

$$p(F = a|B = r) = \frac{1}{4}$$

$$p(F = o|B = r) = \frac{3}{4}$$

$$p(F = a|B = b) = \frac{3}{4}$$

$$p(F = o|B = b) = \frac{1}{4}$$



Bishop's Bayes Example (cont.)

Back to our question: What is the overall probability that we pick an apple?

$$\begin{aligned} p(F=a) &= p(F=a|B=r)p(B=r) + p(F=a|B=b)p(B=b) \\ &= \frac{1}{4}\cdot\frac{4}{10} + \frac{3}{4}\cdot\frac{6}{10} = \frac{11}{20} \end{aligned}$$

from this it follows that $p(F=o) = 1 - \frac{11}{20} = \frac{9}{20}$. Although there are more oranges in total, picking an apple is more likely.

Back to our second question: Given that we have chosen an orange, what is the probability that the box we chose was the blue one, that is $p(B=b|F=o)$?

Bishop's Bayes Example (cont.)

$$p(B = b|F = o) = \frac{p(F = o|B = b)p(B = b)}{p(F = o)} = \frac{1}{4} \cdot \frac{6}{10} \cdot \frac{20}{9} = \frac{1}{3}$$

and that we chose the red box:

$$p(B = r|F = o) = \frac{p(F = o|B = r)p(B = r)}{p(F = o)} = \frac{3}{4} \cdot \frac{4}{10} \cdot \frac{20}{9} = \frac{2}{3}$$

$$p(B|F) = \frac{p(F|B)p(B)}{p(F)}, \text{ where } p(F) = \sum_B p(F|B)p(B)$$

Bayes' theorem Bishop's Example

- Probability $p(B)$ is the *prior probability* because it is the probability *before* we observe the identify of the fruit.
- After telling us the fruit is e.g. an orange, we can use Bayes' theorem to compute the probability $p(B|F)$ which is called the *posterior probability* because it is the probability *after* we have observed F.

Bayes' theorem in words

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

Independence of Random Variables

- If random variables X and Y are independent, then joint distribution factorizes into product of marginals, that is

$$p(X, Y) = p(X)p(Y).$$

- From product rule one can see

$$p(Y|X) = p(Y),$$

that is, the conditional distribution of Y given X is independent of the value of X.

Naive Bayes Classifier

Given feature variables $F_1, F_2, \ldots, F_n$ and a class variable C. The Bayes' theorem states

$$p(C \mid F_1, F_2, \ldots, F_n) = \frac{p(C)\, p(F_1, F_2, \ldots, F_n \mid C)}{p(F_1, F_2, \ldots, F_n)}.$$

Assuming that each feature F_i is conditionally independent of every other feature F_j for $i \neq j$ one obtains

$$p(C \mid F_1, F_2, \ldots, F_n) \quad = \quad \frac{p(C) \prod_{i=1}^{n} p(F_i \mid C)}{p(F_1, F_2, \ldots, F_n)}.$$

The denominator serves as a scaling factor and can be omitted in the final classification rule

$$\underset{c}{\operatorname{argmax}}\; p(C = c) \prod_{i=1}^{n} p(F_i = f_i \mid C = c).$$

Naive Bayes Classifier Example

Dataset taken from Tom Mitchell's book: Machine Learning

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Naive Bayes Classifier Example (cont.)

Should we *play tennis* when given the following information (features F_1, F_2, F_3, F_4)

$$(F_1 = \text{sunny}, F_2 = \text{cool}, F_3 = \text{high}, F_4 = \text{strong})$$

We want to compute

$$\underset{c}{\text{argmax}}\ p(C = c) \prod_{i=1}^{n} p(F_i = f_i \mid C = c).$$

Let's do it

$$p(C = \text{Yes})p(F_1 = \text{sunny}|C = \text{Yes})p(F_2 = \text{cool}|C = \text{Yes})$$
$$p(F_3 = \text{high}|C = \text{Yes})p(F_4 = \text{strong}|C = \text{Yes}) = 0.005$$

$$p(C = \text{No})p(F_1 = \text{sunny}|C = \text{No})p(F_2 = \text{cool}|C = \text{No})$$
$$p(F_3 = \text{high}|C = \text{No})p(F_4 = \text{strong}|C = \text{No}) = 0.021$$

Hence, we better play not tennis.

Probability Densities

- We considered probabilities defined over discrete sets of events.
- Consider now probabilities with respect to *continuous* variables.

If the probability of a real-valued variable x falling in the interval $(x, x + \delta x)$ is given by $p(x)\delta x$ for $\delta x \to 0$, then $p(x)$ is called the *probability density* over x.

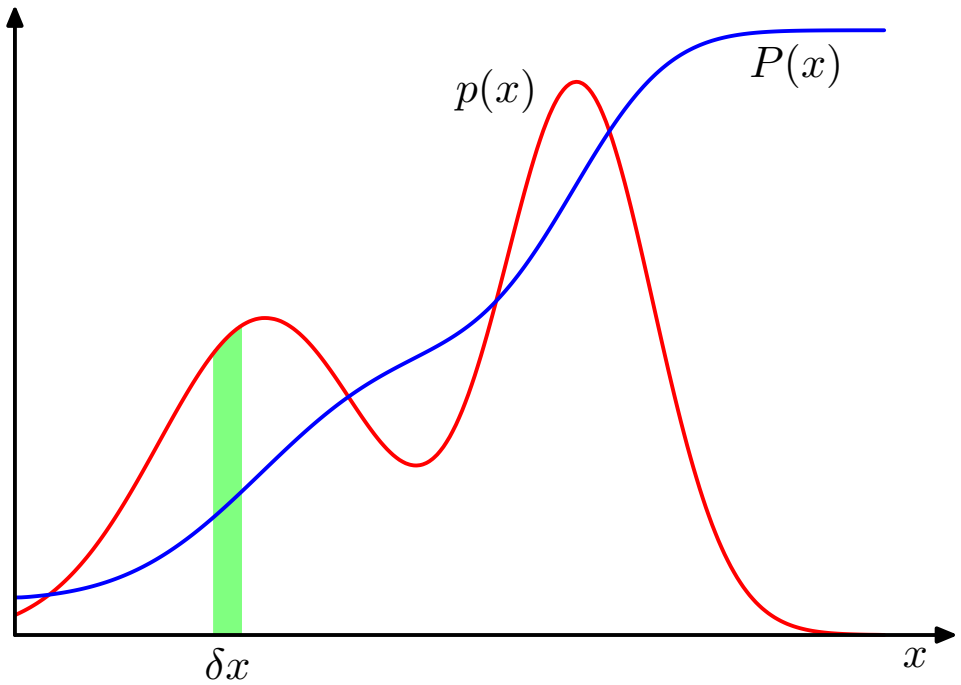
- The *probability* that x will lie in an interval (a, b) is given by

$$p(x \in (a, b)) = \int_a^b p(x)dx.$$

Probability Densities (cont.)

Probability density $p(x)$ must satisfy the two conditions

$$\begin{aligned} p(x) &\geq 0 \\ \int_{-\infty}^{\infty} p(x)dx &= 1 \end{aligned}$$



Probability density can be expressed as the derivative of a cumulative distribution function $P(x)$.

Multivariate Probability Densities

- Having a vector $\mathbf{x} = (x_1, x_2, \ldots, x_D)$ of continuous variables, we can define the joint probability density $p(\mathbf{x})$.
- Multivariate probability density must satisfy

$$\begin{aligned} p(\mathbf{x}) &\geq 0 \\ \int_{-\infty}^{\infty} p(\mathbf{x}) d\mathbf{x} &= 1 \end{aligned}$$

- Sum and product rule, as well as Bayes' theorem apply equally. If x and y are two real variables, then

$$\begin{aligned} p(x) &= \int p(x, y) dy \\ p(x, y) &= p(y|x)p(x). \end{aligned}$$

Expectations

- Average value of some function $f(x)$ under a probability distribution $p(x)$ is called the *expectation* of $f(x)$, denoted as $\mathbb{E}[f]$.

For discrete distribution it is

$$\mathbb{E}[f] = \sum_x p(x)f(x).$$

Average is weighted by the relative probabilities of the different values of x.

Continues variables are expressed in terms of an integration w.r.t. probability density

$$\mathbb{E}[f] = \int p(x)f(x)dx.$$

Expectations (cont.)

Given a finite number N of points drawn from a probability distribution or probability density, the expectation can be approximated as

$$\mathbb{E}(f) \approx \frac{1}{N} \sum_{n=1}^{N} f(x_n).$$

Subscript indicates which variable is being averaged over

$$\mathbb{E}_x[f(x,y)] = \sum_x p(x,y) f(x,y).$$

Conditional expectation w.r.t. a conditional distribution

$$\mathbb{E}_x[f \mid y] = \sum_x p(x \mid y) f(x).$$

Properties of Expectations

- If c is a constant, then $\mathbb{E}[c] = c$.
- $\mathbb{E}[ax] = a\mathbb{E}[x]$.
- $\mathbb{E}[x + y] = \mathbb{E}[x] + \mathbb{E}[y]$.
- $\mathbb{E}[x + c] = \mathbb{E}[x] + c$.

Variance and Covariance

Variance of $f(x)$ is defined as

$$\text{var}[f] = \mathbb{E}\left[(f(x) - \mathbb{E}[f(x)])^2\right]$$

and provides a measure of how much variability there is in $f(x)$ around its mean value $\mathbb{E}[f(x)]$. Variance can also be written in terms of the expectations of $f(x)$ and $f(x)^2$

$$\text{var}[f] = \mathbb{E}[f(x)^2] - \mathbb{E}[f(x)]^2.$$

The variance of variable x itself is given by

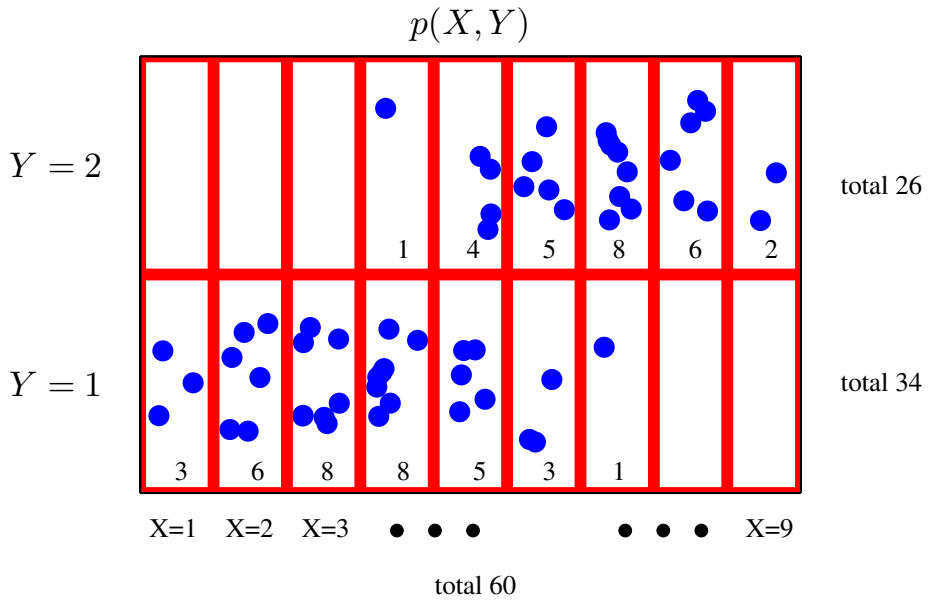
$$\text{var}[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2.$$

Variance and Covariance (cont.)

For two random variables x and y, the *covariance* is defined as

$$\begin{aligned} \mathrm{cov}[x, y] &= \mathbb{E}_{x,y}\left[\{x - \mathbb{E}[x]\}\{y - \mathbb{E}[y]\}\right] \\ &= \mathbb{E}_{x,y}[xy] - \mathbb{E}[x]\mathbb{E}[y]. \end{aligned}$$

Expectation, Variance and Covariance Example



$$\begin{aligned}\mathbb{E}[x] &= p(X=1)\,1 + p(X=2)\,2 + \ldots + p(X=9)\,9 \\ &= \frac{3}{60}\,1 + \frac{6}{60}\,2 + \ldots + \frac{2}{60}\,9 = 4.95\end{aligned}$$

$$\begin{aligned}\mathbb{E}[y] &= p(Y=1)\,1 + p(Y=2)\,2 \\ &= \frac{34}{60}\,1 + \frac{26}{60}\,2 = 1.4333\end{aligned}$$

Expectation, Variance and Covariance Example (cont.)

$$\begin{aligned}\text{var}[x] &= \mathbb{E}\left[(x - \mathbb{E}[x])^2\right] \\ &= \frac{3}{60}(1 - 4.95)^2 + \frac{6}{60}(2 - 4.95)^2 + \frac{8}{60}(3 - 4.95)^2 + \\ &\quad \ldots + (9 - 4.95)^2 \frac{2}{60} = 4.5475\end{aligned}$$

$$\begin{aligned}\text{var}[x] &= \mathbb{E}[x^2] - \mathbb{E}[x]^2 \\ &= 1^2 \frac{3}{60} + 2^2 \frac{6}{60} + 3^2 \frac{8}{60} + \\ &\quad \ldots + 8^2 \frac{6}{60} + 9^2 \frac{2}{60} - 4.95^2 = 4.5475\end{aligned}$$

Binary Variables & Bernoulli Distribution

Consider single binary random variable $x \in \{0, 1\}$, e.g. x might describe outcome of flipping a coin, with $x = 1$ representing 'heads', and $x = 0$ representing 'tails'.

Probability of $x = 1$

$$p(x = 1 \,|\, \mu) = \mu \qquad \text{where } 0 \leq \mu \leq 1.$$

Probability of $x = 0$

$$p(x = 0 \,|\, \mu) = 1 - \mu.$$

Probability distribution over x is

$$\text{Bern}(x \,|\, \mu) = \mu^x \, (1 - \mu)^{1-x} \qquad (\textit{Bernoulli} \text{ distribution}).$$

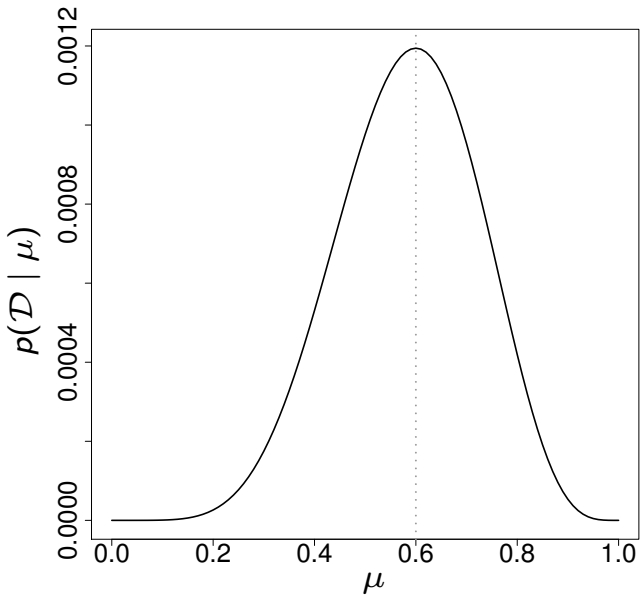
Distribution is normalized and has mean $\mathbb{E}[x] = \mu$ and variance $\text{var}[x] = \mu \, (1 - \mu)$.

Bernoulli Distribution & Maximum Likelihood

Given a dataset $\mathcal{D} = \{x_1, x_2, \ldots, x_N\}$ of observed values of x. Assume that observations are drawn independently from $p(x \,|\, \mu)$, so that

$$p(\mathcal{D} \,|\, \mu) = \prod_{n=1}^{N} p(x_n \,|\, \mu) = \prod_{n=1}^{N} \mu^{x_n} (1-\mu)^{1-x_n}.$$

Suppose we observe dataset $\mathcal{D} = \{0, 1, 1, 0, 1, 0, 1, 1, 1, 0\}$ and want to maximize $p(\mathcal{D} \,|\, \mu)$, that is, find a parameter μ such that $p(\mathcal{D} \,|\, \mu)$ is maximal.



Observe that for $\mu = 0.6 = \frac{\sum_{n=1}^{N} 1\{x_n=1\}}{|\mathcal{D}|}$ the value of $p(\mathcal{D} \,|\, \mu)$ is maximal.

Bernoulli Distribution & Maximum Likelihood (cont.)

Analytically we can set the derivative of $\ln p(\mathcal{D} \mid \mu)$ with respect to μ to zero.

$$\ln p(\mathcal{D} \mid \mu) = \sum_{n=1}^{N} \ln p(x_n \mid \mu) = \sum_{n=1}^{N} \{x_n \ln \mu + (1 - x_n) \ln(1 - \mu)\}.$$

$$\frac{\partial \ln p(\mathcal{D} \mid \mu)}{\partial \mu} = 0 \quad \Rightarrow \mu_{\mathrm{ML}} = \frac{1}{N} \sum_{n=1}^{N} x_n.$$

The maximum likelihood estimator μ_{ML} is also known as *sample mean* and is the fraction of observations of heads in the data set $\mathcal{D}$, that is,

$$\mu_{\mathrm{ML}} = \frac{\sum_{n=1}^{N} 1\{x_n = 1\}}{|\mathcal{D}|}.$$

Binomial Distribution

Suppose we toss a coin N times and are interested in the probability of obtaining m heads.

$$\mathrm{Bin}(m \,|\, N, \mu) = \left(\begin{array}{c} N \\ m \end{array} \right) \mu^m (1-\mu)^{N-m}.$$

For each observation the mean and variance are given by

$$\mathbb{E}[m] \equiv \sum_{m=0}^{N} m \,\mathrm{Bin}(m \,|\, N, \mu) = N\,\mu \qquad \text{(sum of means)}$$

$$\mathrm{var}[m] \equiv \sum_{m=0}^{N} (m - \mathbb{E}[m])^2 \,\mathrm{Bin}(m \,|\, N, \mu) = N\,\mu\,(1-\mu) \qquad \text{(sum of variances).}$$

Binomial Distribution Example

Suppose $N = 3$, $\mu = 1/2$ and $m = 2$, that is, we toss a fair coin three times and are interested in the probability of obtaining two times head.

The event space is $\Omega = \{H, T\}^3$ and the events of interests are $\mathcal{E} = \{\{H, H, T\}, \{H, T, H\}, \{T, H, H\}\}$. Observe that

$$\left(\begin{array}{c} 3 \\ 2 \end{array} \right) = |\mathcal{E}|$$

and the probability of tossing two heads in three trials

$$3\,(1/2)^2\,(1-1/2)^1 = \left(\begin{array}{c} 3 \\ 2 \end{array} \right) (1/2)^2\,(1-1/2)^1.$$

Multinomial Distribution

So far we have considered random variables that can take two possible values (e.g. head or tail). The multinomial distribution encounters discrete variables that can take on one of K possible mutually exclusive states.

Denote N the total number of observations, $\boldsymbol{\mu}$ parameter vector with $0 \leq \mu_k \leq 1$ and $\sum_{k=1}^{K} \mu_k = 1$. The *multinomial distribution* has the form

$$\text{Mult}(m_1, m_2, \ldots, m_K \,|\, \boldsymbol{\mu}, N) = \begin{pmatrix} N \\ m_1 m_2 \ldots m_K \end{pmatrix} \prod_{k=1}^{K} \mu_k^{m_k},$$

where m_k denotes the number of times outcome number k was observed over the N observations (trials) and

$$\begin{pmatrix} N \\ m_1 m_2 \ldots m_K \end{pmatrix} = \frac{N!}{m_1! m_2! \ldots m_K!}$$

(number of ways of partitioning N objects into K groups of size $m_1, m_2, \ldots, m_K$).

Multinomial Distribution (Example)

Given an urn with 10 marbles, 2 red, 3 green and 5 blue ($K = 3$). We select randomly $N = 4$ from the urn with replacement. What is the probability of selecting $m_1 = 0$ red, $m_2 = 2$ green and $m_3 = 2$ blue marbles.

$$p(red) = 2/10,\ p(green) = 3/10,\ p(blue) = 5/10.$$

$$\text{Mult}(m_1 = 0, m_2 = 2, m_3 = 2 \,|\, \mu_1 = 2/10, \mu_2 = 3/10, \mu_3 = 5/10, N = 4)$$

$$= \frac{4!}{0!\,2!\,2!}(0.2)^0\,(0.3)^2\,(0.5)^2 = 0.135,$$

where $m_1 + m_2 + m_3 = N = 4$.

Gaussian Distribution

For a single real-valued variable x, the *Gaussian* distribution is defined by

$$\mathcal{N}(x|\mu,\sigma) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$$

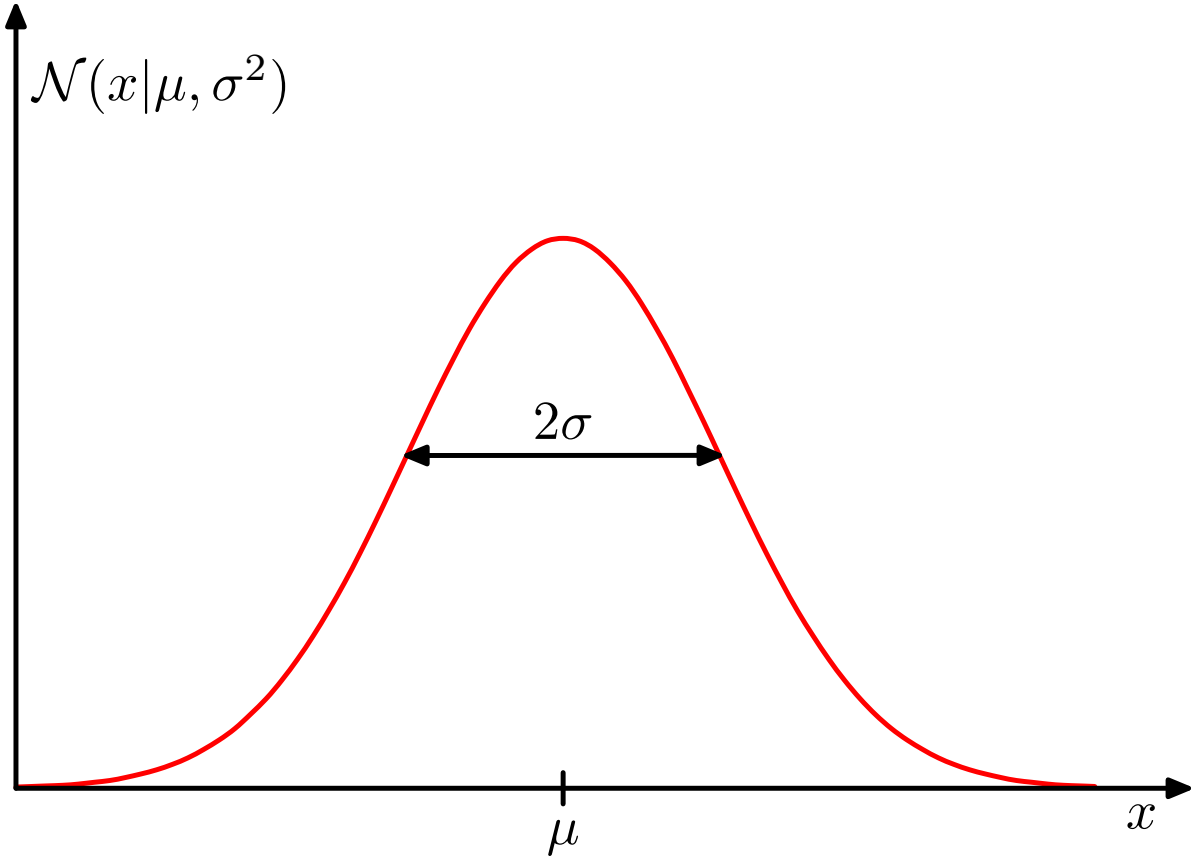
where μ is the *mean* and σ^2 the *variance* (square root of σ^2 is called *standard deviation*).
Gaussian distribution satisfies

$$\mathcal{N}(x|\mu,\sigma) \geq 0$$

$$\int_{-\infty}^{\infty} \mathcal{N}(x|\mu,\sigma)\, dx = 1$$

and therefore satisfies the two requirements for a valid *probability density*.

Gaussian Distribution (cont.)

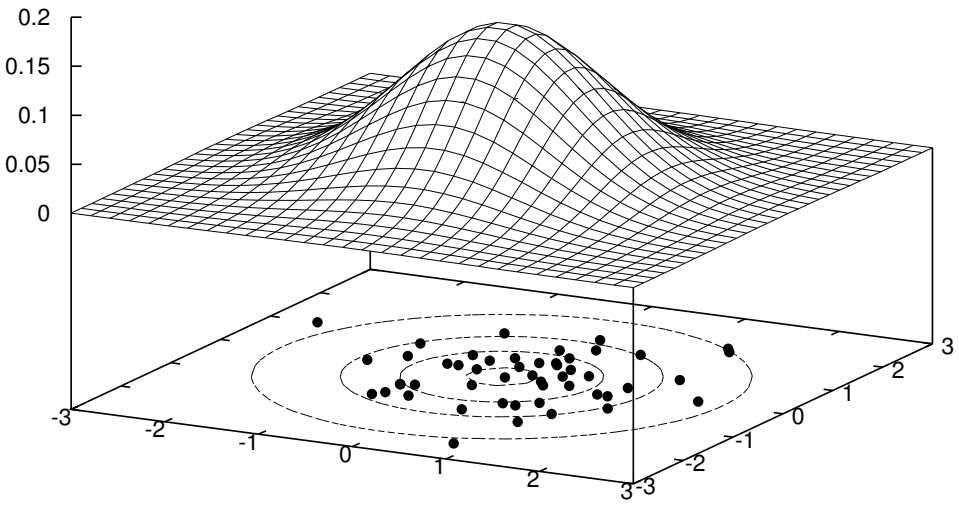


Multivariate Gaussian Distribution

Gaussian distribution defined over D-dimensional vector $\mathbf{x}$ of continuous variables is given by

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu},\boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}$$

$$\boldsymbol{\mu} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



- $\boldsymbol{\mu}$ is a D-dim. mean vector.
- $\boldsymbol{\Sigma}$ is a $D \times D$ covariance matrix, $|\boldsymbol{\Sigma}|$ denotes the determinant of $\boldsymbol{\Sigma}$.

Gaussian Distribution & Maximum Likelihood

- Given data set $\mathcal{X} = \{x_1, x_2, \ldots, x_N\}$ of observations.
- Suppose that observations are drawn independently from a Gaussian distribution whose mean μ and variance σ^2 are *unknown*.
- Goal: infer these parameters from the data set.

Data $\mathcal{X}$ is *independent and identically distributed*, hence likelihood of the data, given μ and σ^2 is

$$p(\mathcal{X} \,|\, \mu, \sigma^2) = \prod_{n=1}^{N} \mathcal{N}(x_n \,|\, \mu, \sigma^2).$$

Log likelihood of $p(\mathcal{X} \,|\, \mu, \sigma^2)$

$$\ln p(\mathcal{X} \,|\, \mu, \sigma^2) = -\frac{1}{2\sigma^2} \sum_{n=1}^{N} (x_n - \mu)^2 - \frac{N}{2} \ln \sigma^2 - \frac{N}{2} \ln(2\pi)$$

Gaussian Distribution & Maximum Likelihood (cont.)

Maximizing $\ln p(\mathcal{X} \mid \mu, \sigma^2)$, that is

$$\frac{\partial}{\partial \mu} \ln p(\mathcal{X} \mid \mu, \sigma^2) = 0$$

gives solution

$$\mu_{\mathrm{ML}} = \frac{1}{N} \sum_{n=1}^{N} x_n \qquad \text{sample mean}$$

and analog

$$\frac{\partial}{\partial \sigma} \ln p(\mathcal{X} \mid \mu, \sigma^2) = 0$$

gives sample variance

$$\sigma^2_{\mathrm{ML}} = \frac{1}{N} \sum_{n=1}^{N} (x_n - \mu_{\mathrm{ML}})^2$$

Properties of Gaussian Distribution

Expectation of $\mathbf{x} \in \mathbb{R}^D$ is

$$\begin{aligned}\mathbb{E}[\mathbf{x}] &= \frac{1}{(2\pi)^{D/2}} \frac{1}{|\mathbf{\Sigma}|^{1/2}} \int \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\} \mathbf{x}\, d\mathbf{x} \\ &= \boldsymbol{\mu}\end{aligned}$$

and covariance

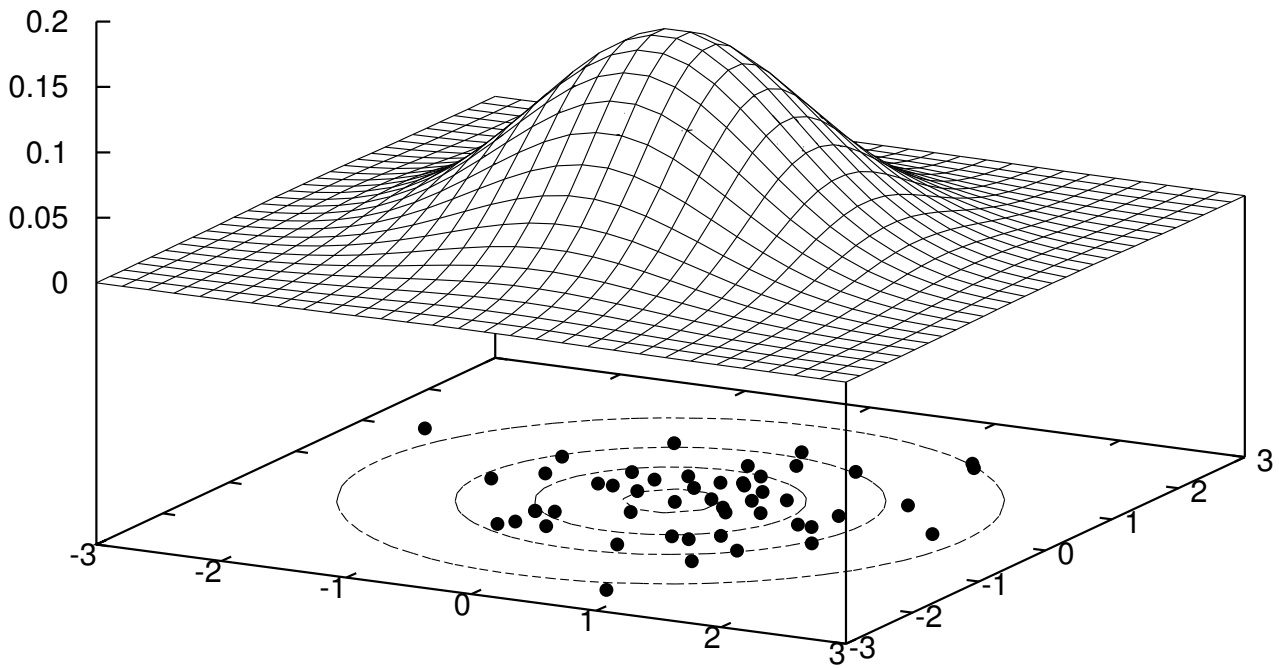
$$\begin{aligned}\text{cov}[\mathbf{x}] &= \mathbb{E}[(\mathbf{x}-\mathbb{E}[\mathbf{x}])(\mathbf{x}-\mathbb{E}[\mathbf{x}])^T] \\ \text{cov}[\mathbf{x}] &= \mathbb{E}[(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T] \\ &= \mathbf{\Sigma}.\end{aligned}$$

Covariance matrix $\mathbf{\Sigma}$ is always symmetric and positive semidefinite.
Elements σ_{ii} are variances of x_i, elements σ_{ij} are covariances of x_i and x_j.
If x_i and x_j are *statistically independent*, then $\sigma_{ij} = 0$.

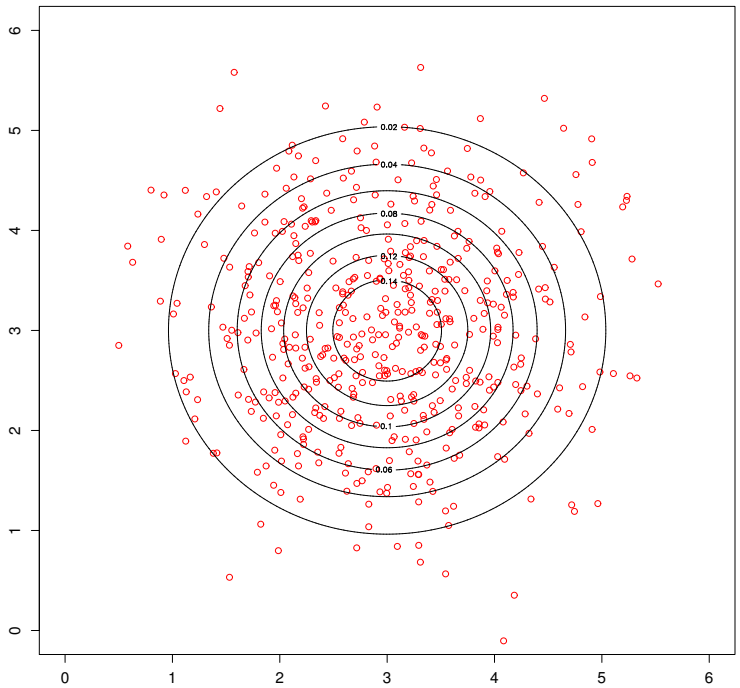
Two-dim. Gaussian Distribution

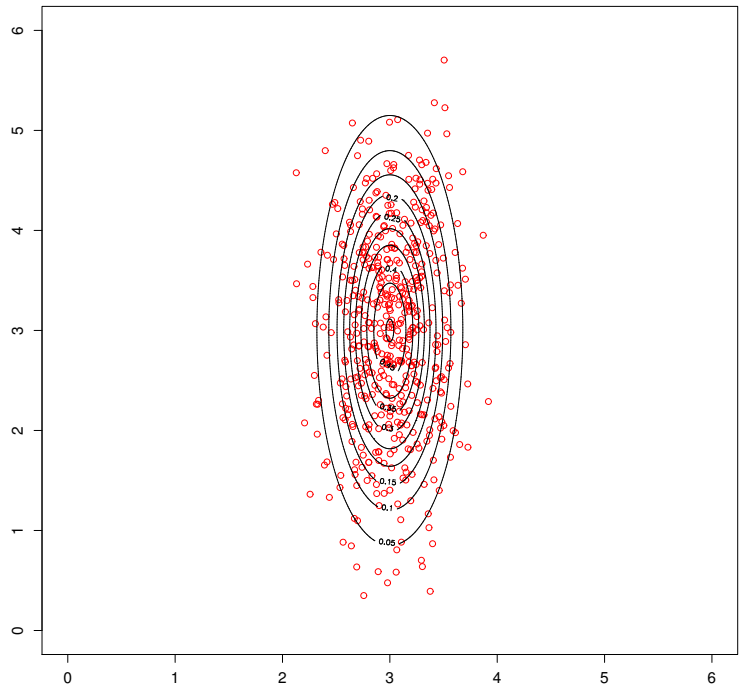
$$p(\mathbf{x}) = \frac{1}{(2\pi)^{D/2}|\mathbf{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\right\}$$

$$\boldsymbol{\mu} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad \mathbf{\Sigma} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



Two-dim. Gaussian Distribution

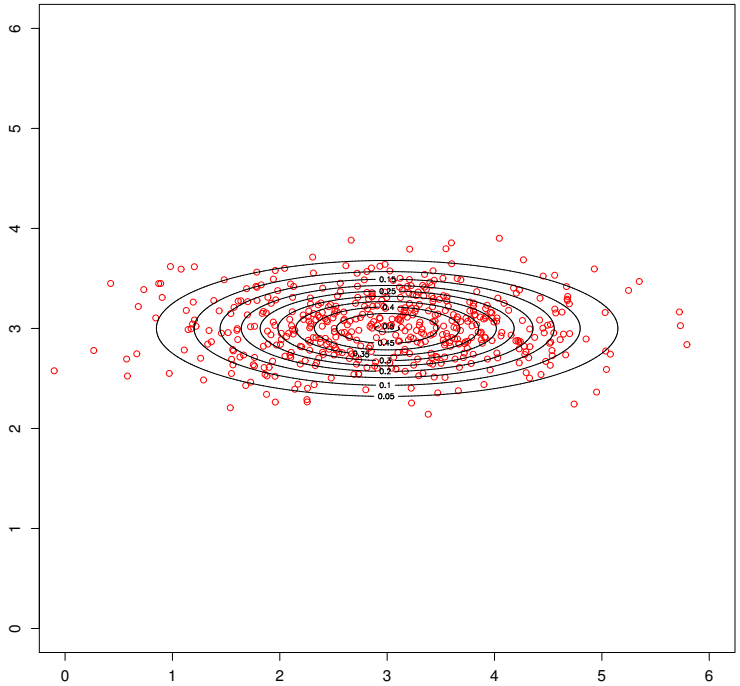




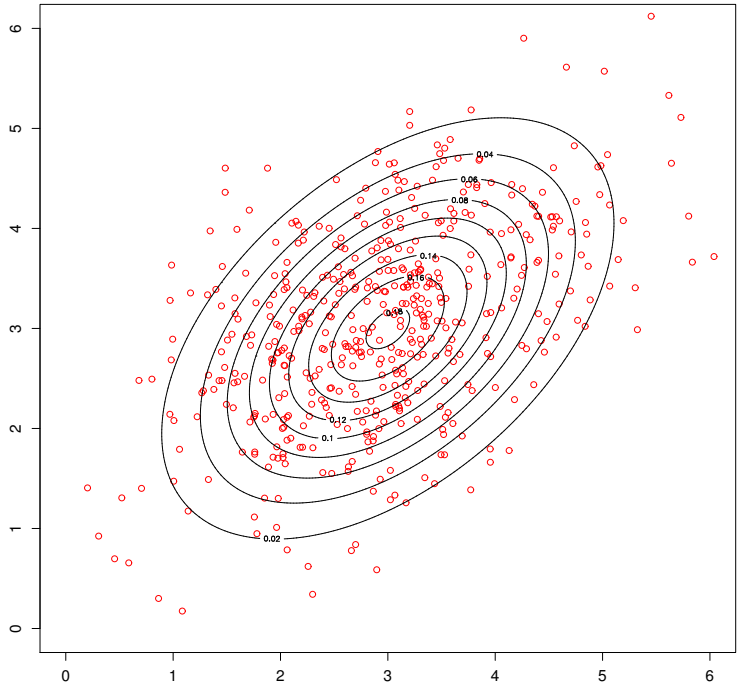
$$\boldsymbol{\mu} = \begin{bmatrix} 3 \\ 3 \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$\boldsymbol{\Sigma} = \begin{bmatrix} 0.1 & 0 \\ 0 & 1 \end{bmatrix}$$

Two-dim. Gaussian Distribution (cont.)



$$\boldsymbol{\mu} = \begin{bmatrix} 3 \\ 3 \end{bmatrix} \quad \boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0 \\ 0 & 0.1 \end{bmatrix}$$



$$\boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

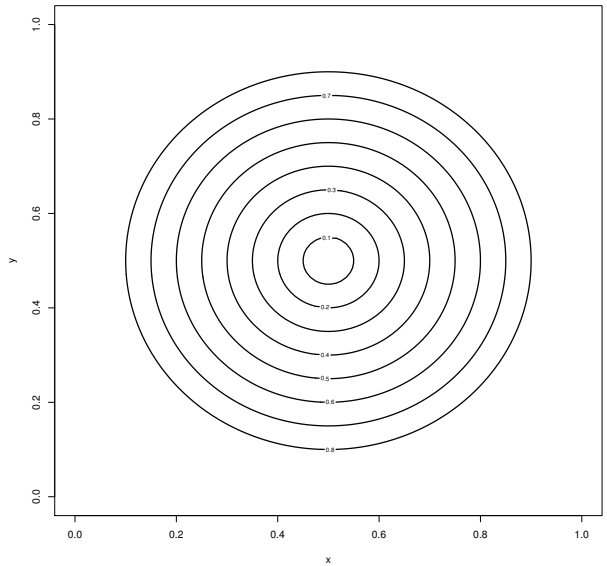
Covariance Matrix

Covariance matrix $\mathbf{\Sigma}$ is symmetric and positive definite. Example:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \mathbf{\Sigma} = \mathbf{\Sigma}^T.$$

$$x, y \in \mathbb{R}, [x \;\; y] \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = [x \;\; y] \begin{bmatrix} x \\ y \end{bmatrix} = x^2 + y^2 \geq 0, \text{ for all } x, y \in \mathbb{R}$$

```
x <- y <- seq(-1, 1, len=100);
r <- sqrt(outer(x^2, y^2, "+"));
contour(r, drawlabels = TRUE,
        levels = seq(0, 0.8, by=0.1),
        lwd = 2, ylab = "y",
        xlab = "x", cex = 2);
```



Covariance Matrix

Eigenvector and eigenvalue of covariance matrix $\mathbf{\Sigma} \in \mathbb{R}^{D \times D}$

$$\mathbf{\Sigma}\mathbf{u}_d = \lambda_d \mathbf{u}_d, \quad d = 1, 2, \ldots, D.$$

Covariance matrix can be expressed as an expansion in terms of its eigenvectors in the form

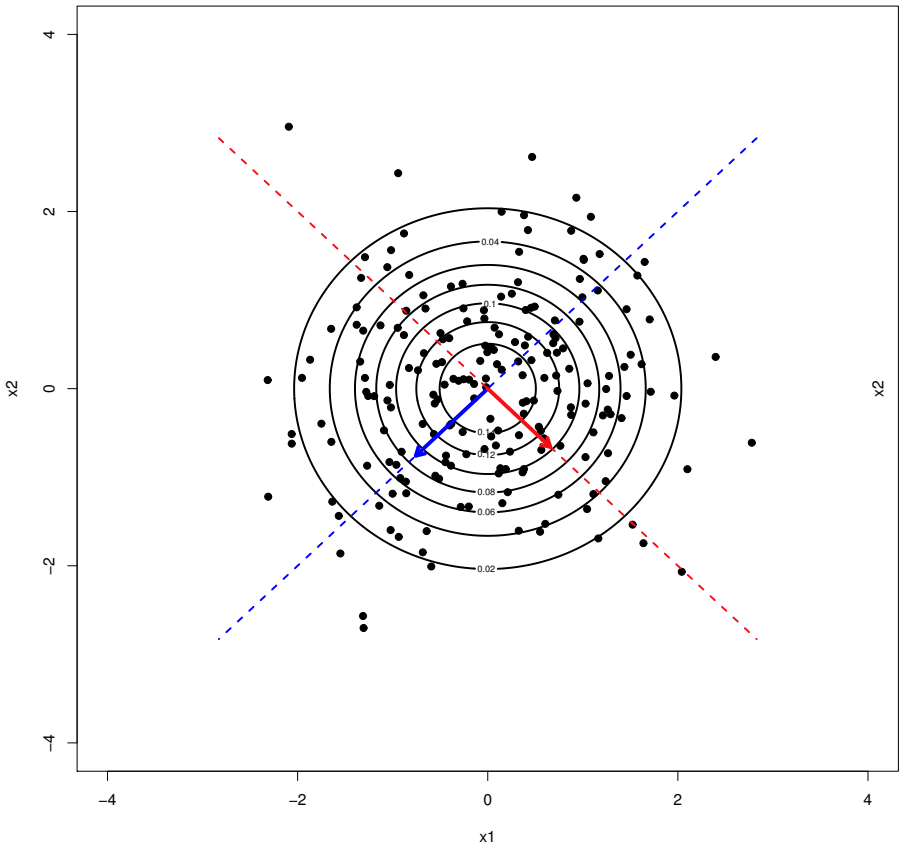
$$\mathbf{\Sigma} = \sum_{d=1}^{D} \lambda_d \mathbf{u}_d \mathbf{u}_d^T$$

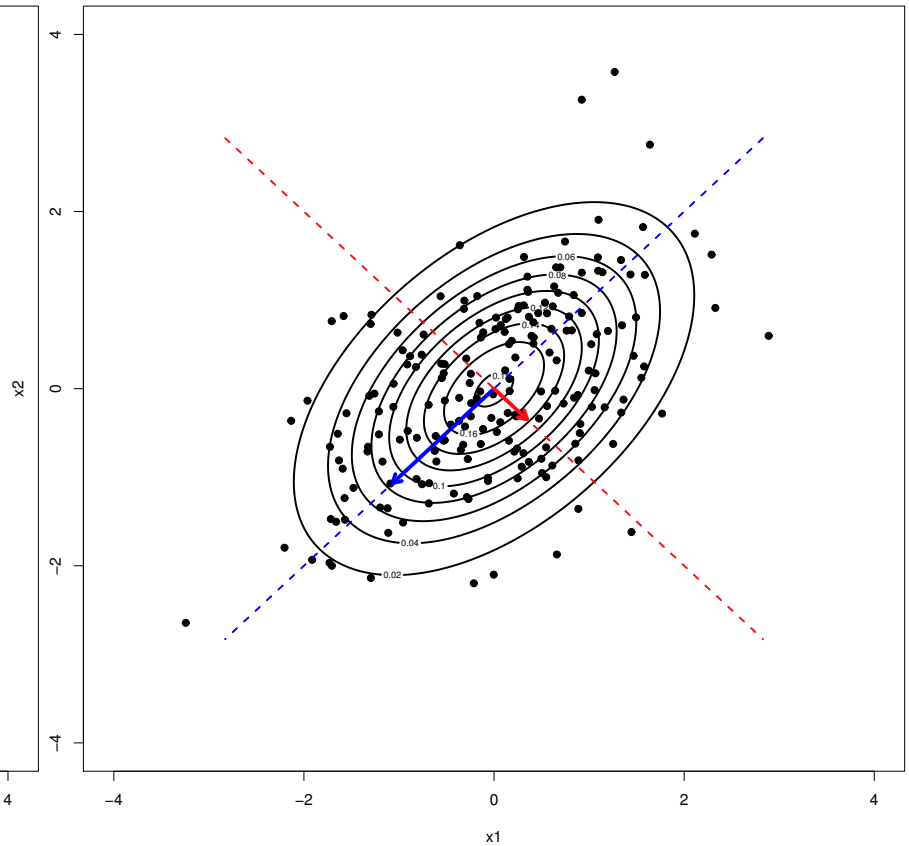
and the inverse matrix

$$\mathbf{\Sigma}^{-1} = \sum_{d=1}^{D} \frac{1}{\lambda_d} \mathbf{u}_d \mathbf{u}_d^T.$$

Covariance Matrix and Eigenvectors (Example)

$$\mathbf{\Sigma} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \qquad \mathbf{\Sigma} = \begin{bmatrix} 1 & 1/2 \\ 1/2 & 1 \end{bmatrix}$$



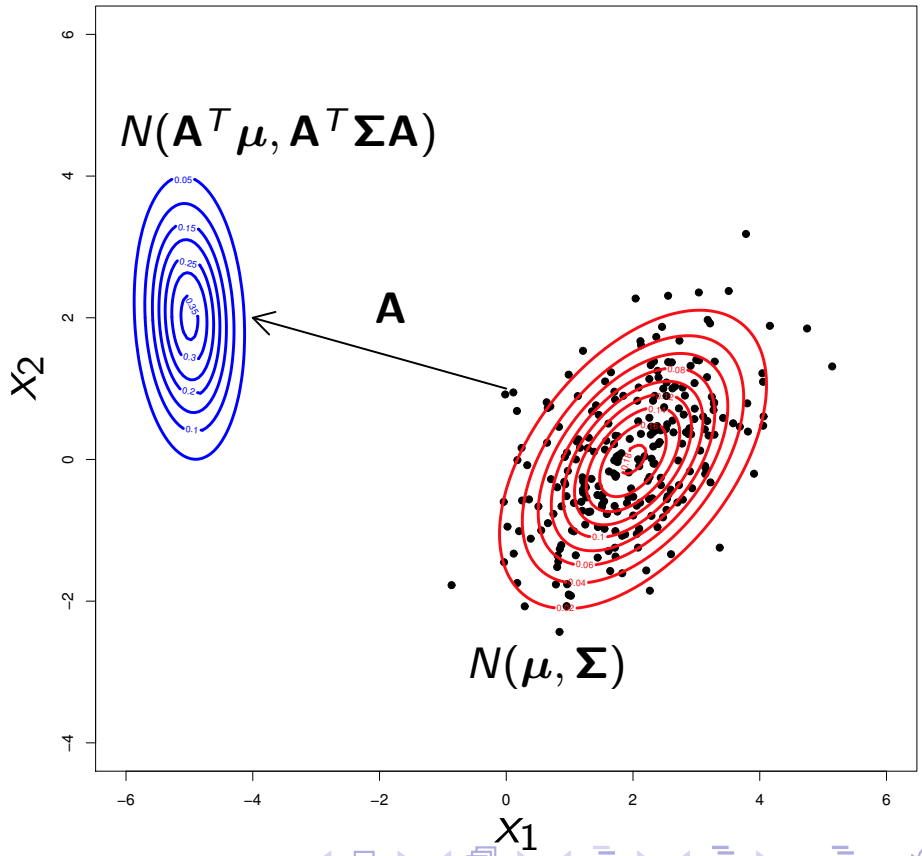


Linear Combinations of Gaussian Distributions

Suppose $p(\mathbf{x}) \sim N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, and $\mathbf{A}$ is d-by-k matrix and $\mathbf{y} = \mathbf{A}^T\mathbf{x}$ is a k-component vector, then

$$p(\mathbf{y}) \sim N(\mathbf{A}^T\boldsymbol{\mu}, \mathbf{A}^T\boldsymbol{\Sigma}\mathbf{A}).$$

When $k = 1$ and $\mathbf{A}$ is a unit-length vector $\mathbf{a}$, $y = \mathbf{a}^T\mathbf{x}$ is a scalar that represents the projection of $\mathbf{x}$ onto a line in the direction of $\mathbf{a}$, where $\mathbf{a}^T\boldsymbol{\Sigma}\mathbf{a}$ is the variance of the projection of $\mathbf{x}$ onto $\mathbf{a}$.

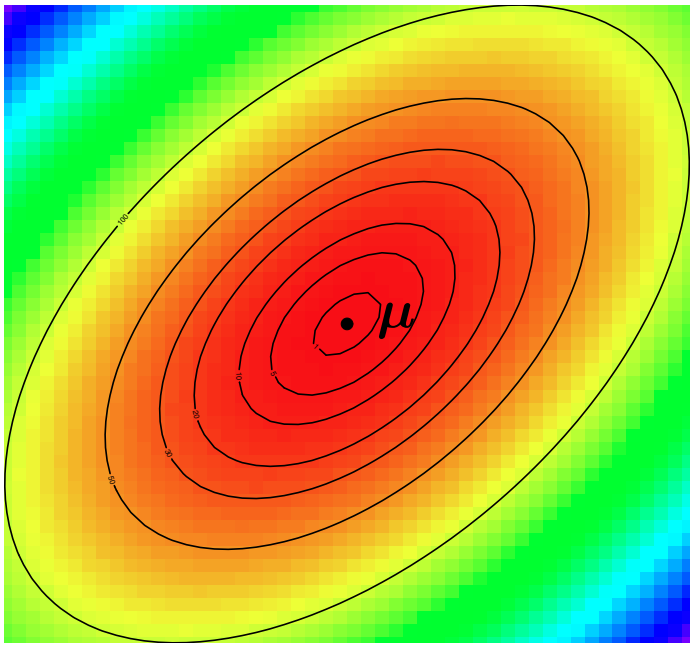


Mahalanobis Distance

Quantity

$$r^2 = (\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})$$

is called the squared *Mahalanobis Distance* from $\mathbf{x}$ to $\boldsymbol{\mu}$. If $\boldsymbol{\Sigma} = \mathbf{I}$, then r^2 is the squared Euclidean distance. Any points lying on the same contour line have equal distance to $\boldsymbol{\mu}$.



$$[x_1\, x_2] \begin{bmatrix} 1 & 1/2 \\ 1/2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} =$$

$$x_1(x_1 + 1/2\, x_2) + x_2(1/2\, x_1 + x_2) =$$

$$x_1^2 + x_1 x_2 + x_2^2.$$

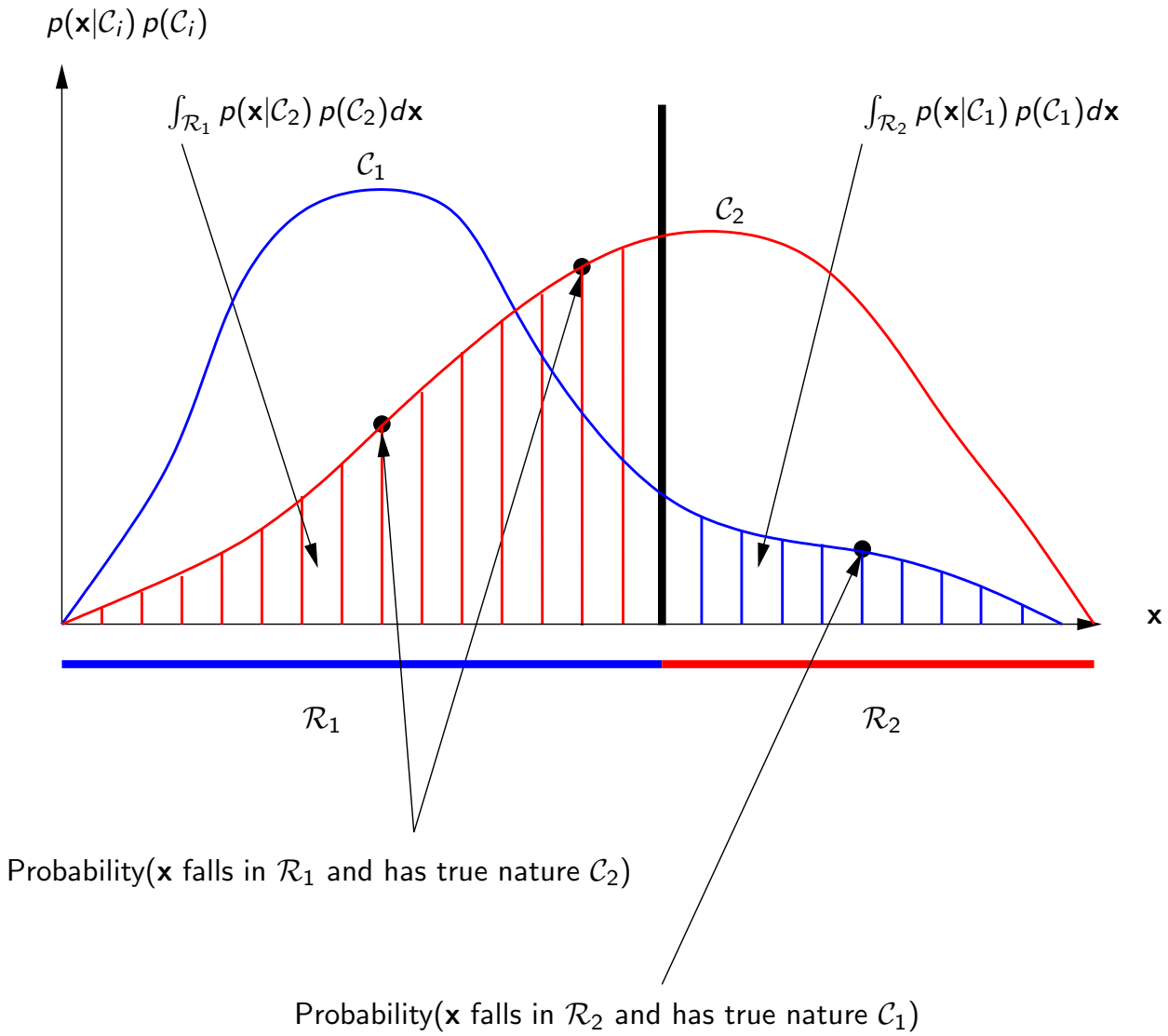
Bayesian Decision Theory

- How to make an optimal decision given the appropriate probabilities?
- Minimize the error of assigning $\mathbf{x}$ to the wrong class.
- Intuitively we would choose the class having the higher posterior probability.

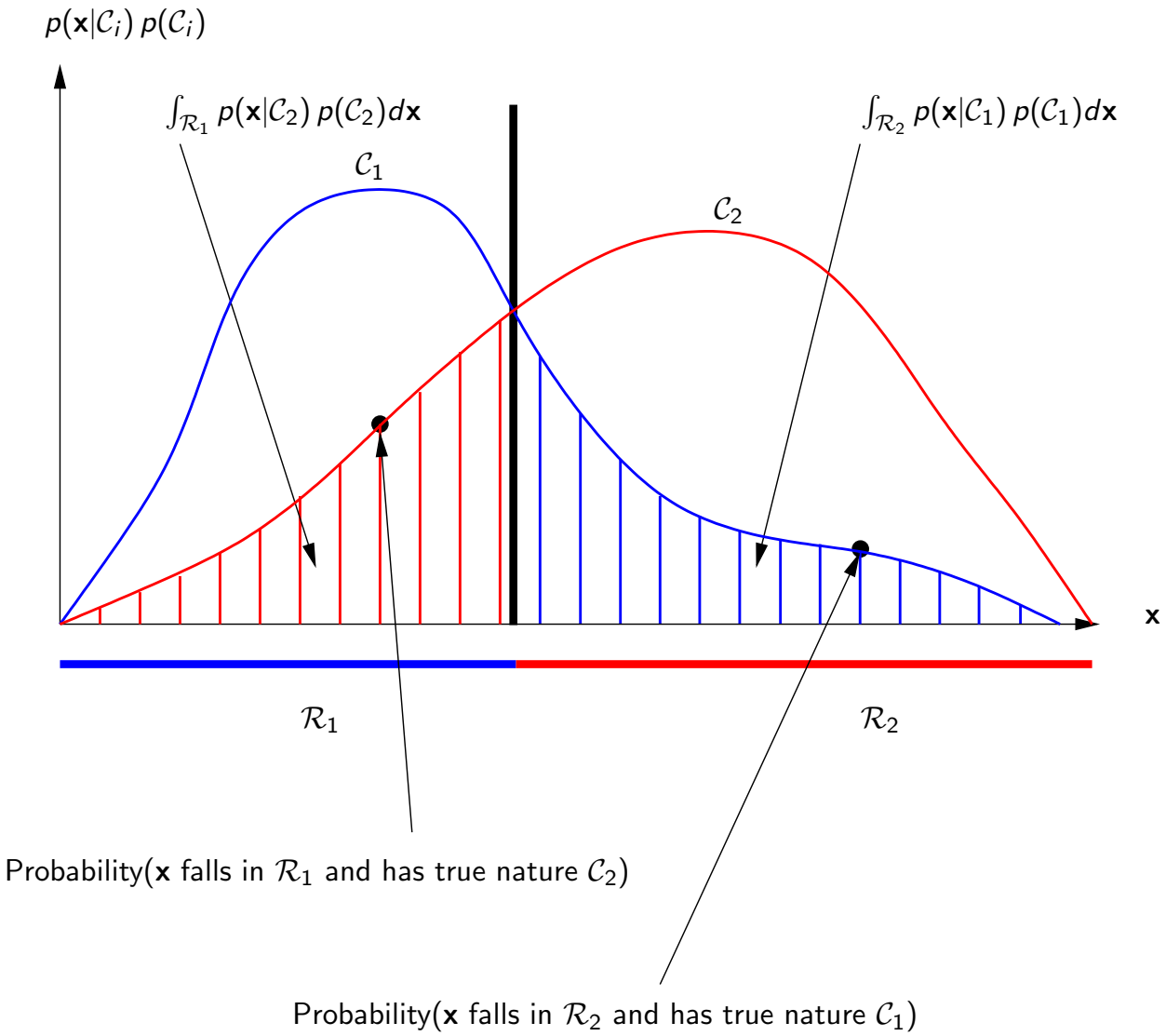
An error occurs when $\mathbf{x}$ belonging to class $\mathcal{C}_1$ is assigned to class $\mathcal{C}_2$ or vice versa. The probability of this occuring is given by:

$$\begin{aligned} p(\text{error}) &= p(\mathbf{x} \in \mathcal{R}_1, \mathcal{C}_2) + p(\mathbf{x} \in \mathcal{R}_2, \mathcal{C}_1) \\ &= \int_{\mathcal{R}_1} p(\mathbf{x}, \mathcal{C}_2) d\mathbf{x} + \int_{\mathcal{R}_2} p(\mathbf{x}, \mathcal{C}_1) d\mathbf{x} \end{aligned}$$

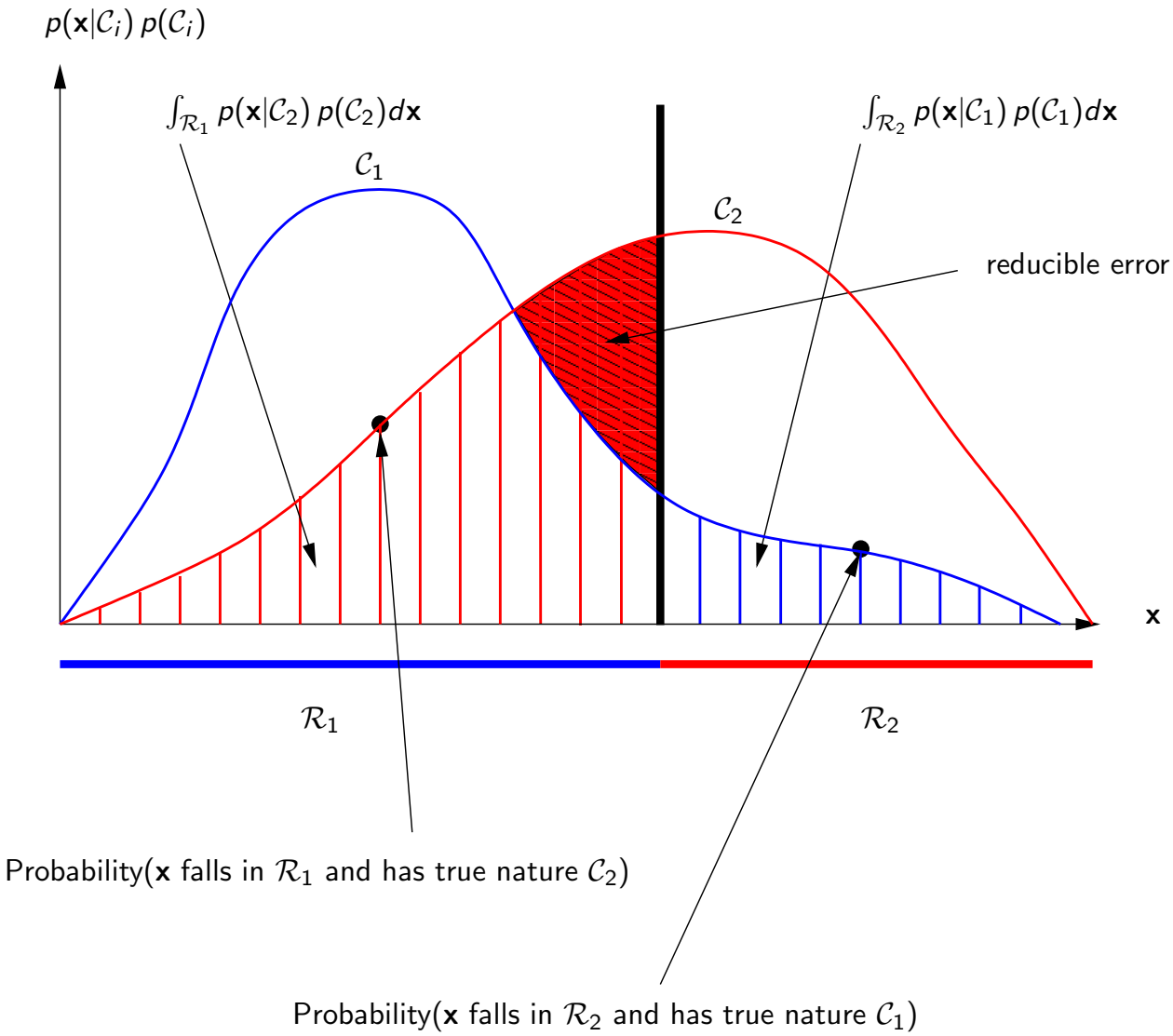
Error Probabilities in Bay. Decision Theory



Error Probabilities in Bay. Decision Theory



Error Probabilities in Bay. Decision Theory



Bayesian Estimation (Class-Conditional Densities)

In Bayesian parameter estimation, the parameter is not fixed as in ML, but is a random variable. The training data is used to convert a distribution on this variable into a posterior density.

- Let $\mathcal{D}$ denote a training sample, our goal is to compute the posterior $p(C_i \mid x, \mathcal{D})$.

Bayes formula:

$$p(C_i \mid x, \mathcal{D}) = \frac{p(x \mid C_i, \mathcal{D}) p(C_i \mid \mathcal{D})}{\sum_{i=1}^{I} p(x \mid C_i, \mathcal{D}) p(C_i \mid \mathcal{D})}.$$

Training sample $\mathcal{D}$ can be used to help us determining the class-conditional and prior density.

To simplify, we assume that prior probabilities are known or obtainable from some calculations, thus we substitute $P(C_i) = P(C_i \mid \mathcal{D})$.

Bayesian Estimation (Class-Conditional Densities) cont.

Consider supervised case: Separate training data by class into c subsets $\mathcal{D}_1, \mathcal{D}_2, \ldots, \mathcal{D}_I$ with sample in $\mathcal{D}_i$ belonging to class C_i. This assumption allows us to write the previous equation as

$$p(C_i \mid x, \mathcal{D}) = \frac{p(x \mid C_i, \mathcal{D}) p(C_i)}{\sum_{i=1}^{I} p(x \mid C_i, \mathcal{D}) p(C_i)}.$$

We have now I separate problems of the following form: Use a set of $\mathcal{D}$ of samples drawn independently according to a fixed but unknown probability distribution $p(x)$ to determine $p(x|\mathcal{D})$.

Bayesian Parameter Estimation

- Desired density $p(x)$ is unknown, we however assume that it has a known parametric form. Unknown is just the value of the parameter vector $\boldsymbol{\Theta}$. That is $p(x \mid \boldsymbol{\Theta})$ is known.
- Any information we have about $\boldsymbol{\Theta}$ prior to observing the sample is contained in prior density $p(\boldsymbol{\Theta})$.

Observation of the sample $\mathcal{D}$ converts this to a posterior density $p(\boldsymbol{\Theta} \mid \mathcal{D})$, which is close (sharply peaked) about the true value of $\boldsymbol{\Theta}$.

Note that we have converted our problem if learning a probability density function to one of estimating a parameter vector.

Recall that our goal is to compute $p(x \mid \mathcal{D})$, which is as close as we can come to obtain the unknown density $p(x)$.

Bayesian Parameter Estimation (cont.)

We obtain $p(x \mid \mathcal{D})$ by integrating the joint density $p(x, \boldsymbol{\Theta} \mid \mathcal{D})$ over $\boldsymbol{\Theta}$:

$$\begin{aligned} p(x \mid \mathcal{D}) &= \int p(x, \boldsymbol{\Theta} \mid \mathcal{D})\, d\boldsymbol{\Theta}, \\ &= \int p(x \mid \boldsymbol{\Theta}, \mathcal{D}) p(\boldsymbol{\Theta} \mid \mathcal{D})\, d\boldsymbol{\Theta} \end{aligned}$$

If $p(\boldsymbol{\Theta} \mid \mathcal{D})$ peaks very sharply about some value $\hat{\boldsymbol{\Theta}}$, we obtain $p(x \mid \mathcal{D}) \approx p(x \mid \boldsymbol{\Theta})$.

Bayesian Parameter Estimation (μ)

Calculate the posterior density $p(\boldsymbol{\Theta}|\mathcal{D})$ and the desired density $p(x|\mathcal{D})$ for the case where $p(x|\mu) \sim N(\mu, \sigma^2)$, where the only unknown quantity is $\boldsymbol{\Theta} \equiv \mu$.

Whatever prior knowledge we might have about μ can be expressed by a known prior density $p(\mu)$, where me make the assumption that

$$p(\mu) \sim N(\mu_0, \sigma_0^2), \qquad \text{where both } \mu_0 \text{ and } \sigma_0 \text{ are known.}$$

Intuitively, μ_0 represents our best prior guess for μ, and σ_0^2 measures our uncertainty about this guess.

Bayesian Parameter Estimation (μ cont.)

Let $\mathcal{D} = \{x_1, x_2, \ldots, x_N\}$.

$$\begin{aligned} p(\mu|\mathcal{D}) &= \frac{p(\mathcal{D}|\mu)p(\mu)}{\int p(\mathcal{D}|\mu)p(\mu)\,d\mu} = \frac{p(\mathcal{D}|\mu)p(\mu)}{p(\mathcal{D})} \\ &= \alpha \prod_{n=1}^{N} p(x_n|\mu)p(\mu), \end{aligned}$$

where α is a normalization factor that depends on $\mathcal{D}$ but is independent of μ. Because $p(x_k|\mu) \sim N(\mu, \sigma^2)$ and $p(\mu) \sim N(\mu_0, \sigma_0^2)$ we obtain

$$p(\mu|\mathcal{D}) = \alpha \prod_{n=1}^{N} \overbrace{\frac{1}{\sqrt{2\pi}\sigma} \exp\left[-\frac{1}{2}\left(\frac{x_n - \mu}{\sigma}\right)^2\right]}^{p(x_n|\mu)} \frac{1}{\sqrt{2\pi}\sigma_0} \overbrace{\exp\left[-\frac{1}{2}\left(\frac{\mu - \mu_0}{\sigma_0}\right)^2\right]}^{p(\mu)}$$

Bayesian Parameter Estimation (μ cont.)

Additional manipulations give

$$\begin{aligned} p(\mu|\mathcal{D}) &= \alpha' \exp\left[-\frac{1}{2}\left(\sum_{n=1}^{N}\left(\frac{\mu - x_n}{\sigma}\right)^2 + \left(\frac{\mu-\mu_0}{\sigma_0}\right)^2\right)\right] \\ &= \alpha'' \exp\left[-\frac{1}{2}\left[\left(\frac{N}{\sigma^2}+\frac{1}{\sigma_0^2}\right)\mu^2 - 2\left(\frac{1}{\sigma^2}\sum_{n=1}^{N}x_n + \frac{\mu_0}{\sigma_0^2}\right)\mu\right]\right] \quad (1) \end{aligned}$$

If we write $p(\mu|\mathcal{D}) \sim N(\mu_N, \sigma_N^2)$, then μ_N and σ_N^2 can be found by equating coefficients in (1) with corresponding coefficients in the generic Gaussian of the form

$$p(\mu|\mathcal{D}) = \frac{1}{\sqrt{2\pi}\sigma_N}\exp\left[-\frac{1}{2}\left(\frac{\mu-\mu_N}{\sigma_N}\right)^2\right].$$

Bayesian Parameter Estimation (μ cont.)

Solving for μ_N and σ_N^2 gives

$$\mu_N = \left(\frac{N\sigma_0^2}{N\sigma_0^2 + \sigma^2} \right) \hat{\mu_N} + \frac{\sigma^2}{N\sigma_0^2 + \sigma^2} \mu_0, \text{ where } \hat{\mu_N} = \frac{1}{N} \sum_{n=1}^{N} x_n$$

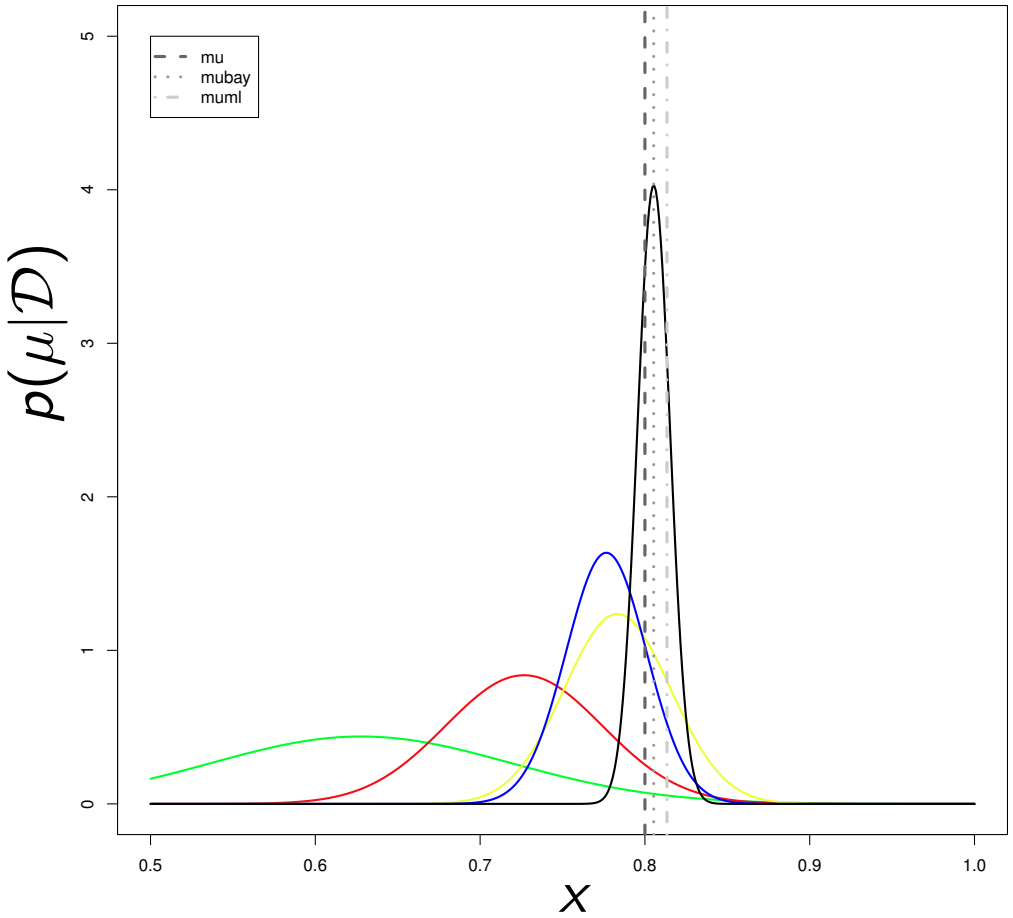
and

$$\sigma_N^2 = \frac{\sigma_0^2 \sigma^2}{N\sigma_0^2 + \sigma^2}.$$

Intuitively, μ_N represents our best guess for μ after observing a sample of size N, and σ_N^2 measures the uncertainty about this guess. Note that σ_N^2 decreases monotonically with $N \to \infty$, that is, each additional observation decreases our uncertainty about the true nature of μ. As N increases, $p(x|\mathcal{D})$ becomes more and more sharply peaked.

Bayesian Parameter Estimation (Example)

We want to apply Bayesian parameter estimation (inference) for parameter μ when given a sample $\mathcal{D} = \{x_1, x_2, \ldots, x_N\}$ where $x_n \sim N(0.8, 0.1)$ and $N = 10$. The prior is $p(\mu) \sim N(0, 1)$.



$N = 1$

$N = 2$

$N = 3$

$N = 4$

$N = 10$

History of Neural Networks

Era of Kernel Methods (SVM, Kernel-PCA, Kernel-Fisher Discriminants, etc.)
Neural Networks are however still and frequently used

1995 — Support-Vector Networks (Cortes and Vapnik)

1992 — A Training Algorithm for Optimal Margin Classifiers (Boser, Guyon and Vapnik), first paper on SVM

Era of Neural Networks

1986/1985 — Backpropagation (Rumelhart, Hinton, Williams, Le Cun (actually first proposed by Werbos, 1974))

1982 — Hopfield Network, (Hopfield), Recurrent Networks, Energy Function

Decline of neural network research

1969 — Book: Perceptrons (Minsky and Papert)

1962/1960 — Adaline (Widrow and Hoff), Perceptron (Rosenblatt)

1943 — Model of McCulloch and Pitts

Note, this historical overview is far from being complete (see books for detailed historical overview)

Method of Steepest Descent

Let $E(\mathbf{w})$ be a continuously differentiable function of some unknown (weight) vector $\mathbf{w}$.

Find an optimal solution $\mathbf{w}^\star$ that satisfies the condition

$$E(\mathbf{w}^\star) \leq E(\mathbf{w}).$$

The necessary condition for optimality is

$$\nabla E(\mathbf{w}^\star) = \mathbf{0}.$$

Let us consider the following *iterative* descent:

Start with an initial guess $\mathbf{w}^{(0)}$ and generate sequence of weight vectors $\mathbf{w}^{(1)}, \mathbf{w}^{(2)}, \ldots$ such that

$$E(\mathbf{w}^{(i+1)}) \leq E(\mathbf{w}^{(i)}).$$

Steepest Descent Algorithm

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta \nabla E(\mathbf{w}^{(i)})$$

where η is a positive constant called learning rate.

At each iteration step the algorithm applies the correction

$$\begin{aligned} \Delta \mathbf{w}^{(i)} &= \mathbf{w}^{(i+1)} - \mathbf{w}^{(i)} \\ &= -\eta \nabla E(\mathbf{w}^{(i)}) \end{aligned}$$

Steepest descent algorithm satisfies:

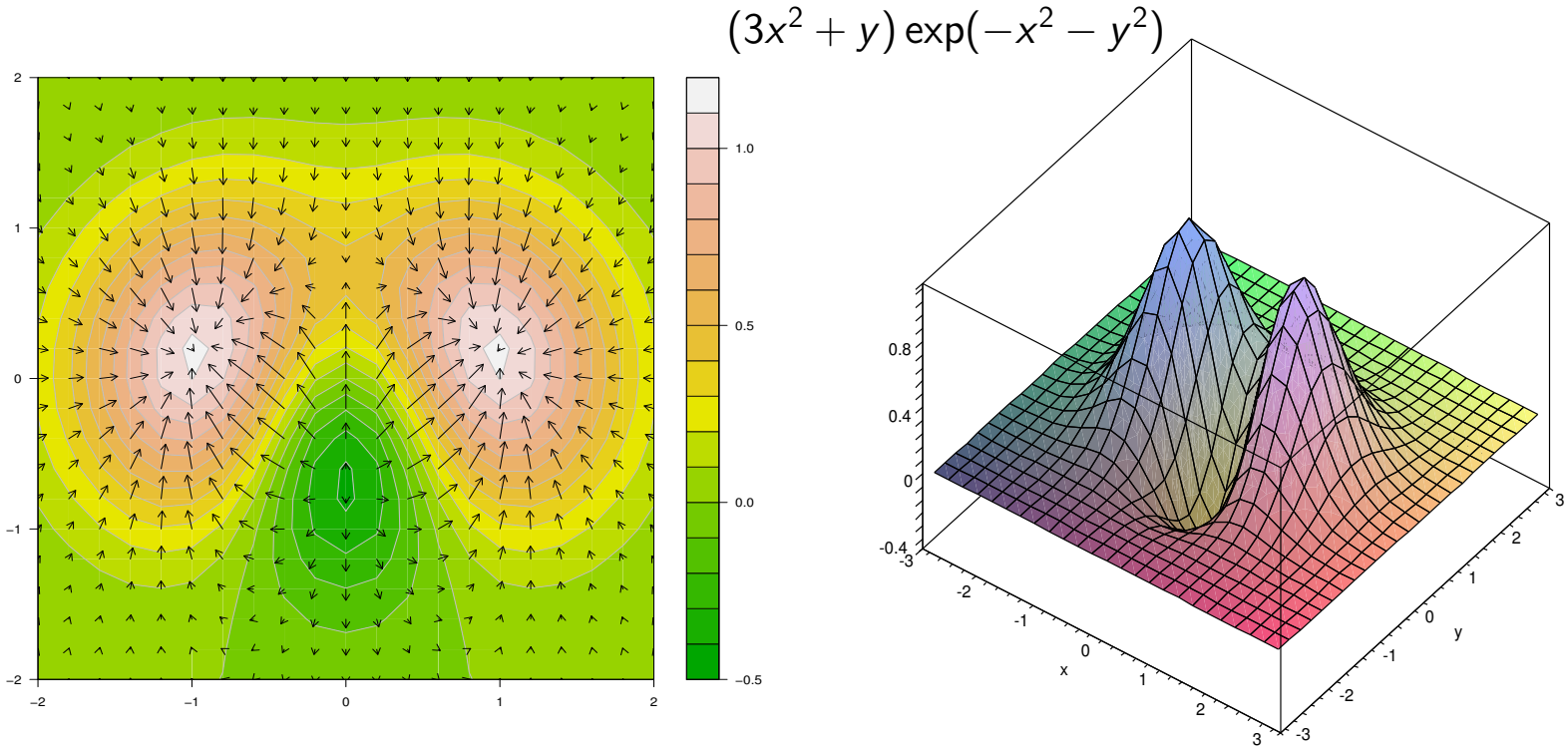
$$E(\mathbf{w}^{(i+1)}) \leq E(\mathbf{w}^{(i)}),$$

to see this, use first-order Taylor expansion around $\mathbf{w}^{(i)}$ to approximate $E(\mathbf{w}^{(i+1)})$ as $E(\mathbf{w}^{(i)}) + (\nabla E(\mathbf{w}^{(i)}))^T \Delta \mathbf{w}^{(i)}$.

Steepest Descent Algorithm (cont.)

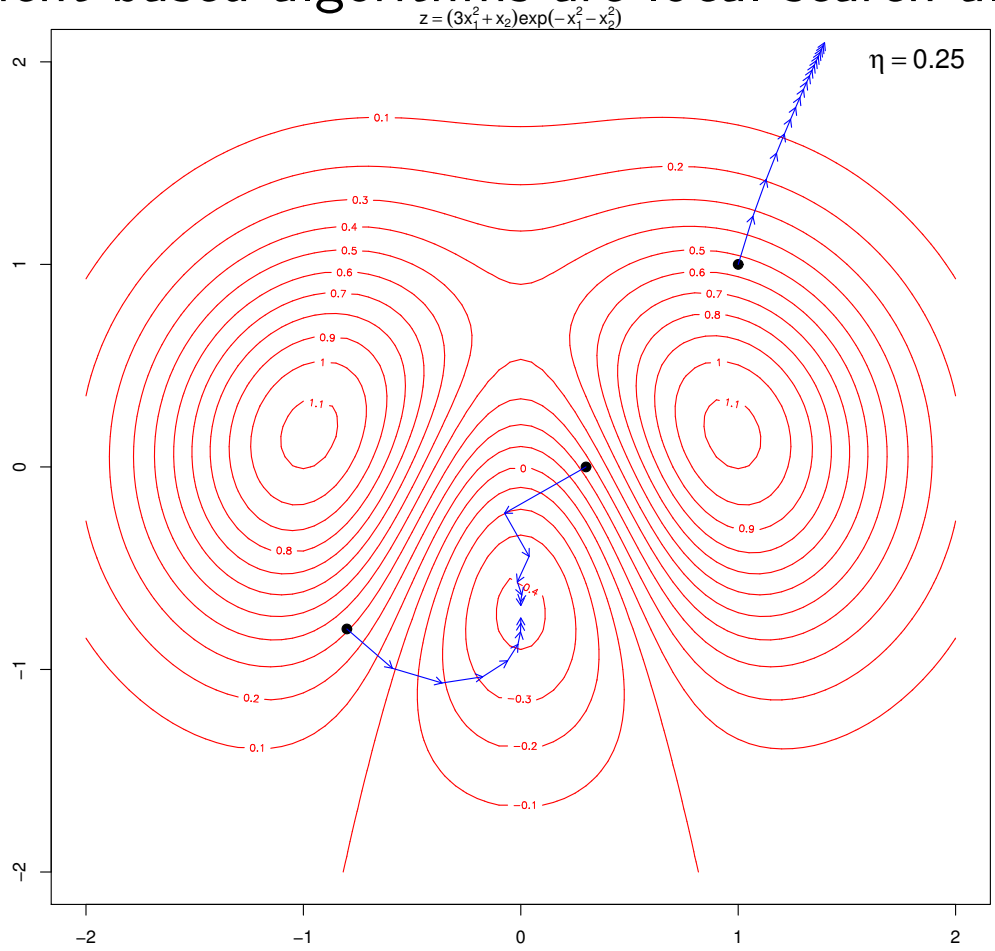
$$\begin{aligned} E(\mathbf{w}^{(i+1)}) &\approx E(\mathbf{w}^{(i)}) + (\nabla E(\mathbf{w}^{(i)}))^T \Delta \mathbf{w}^{(i)} \\ &= E(\mathbf{w}^{(i)}) - \eta \|\nabla E(\mathbf{w}^{(i)})\|^2 \end{aligned}$$

For positive learning rate η, $E(\mathbf{w}^{(i)})$ decreases in each iteration step (for small enough learning rates).



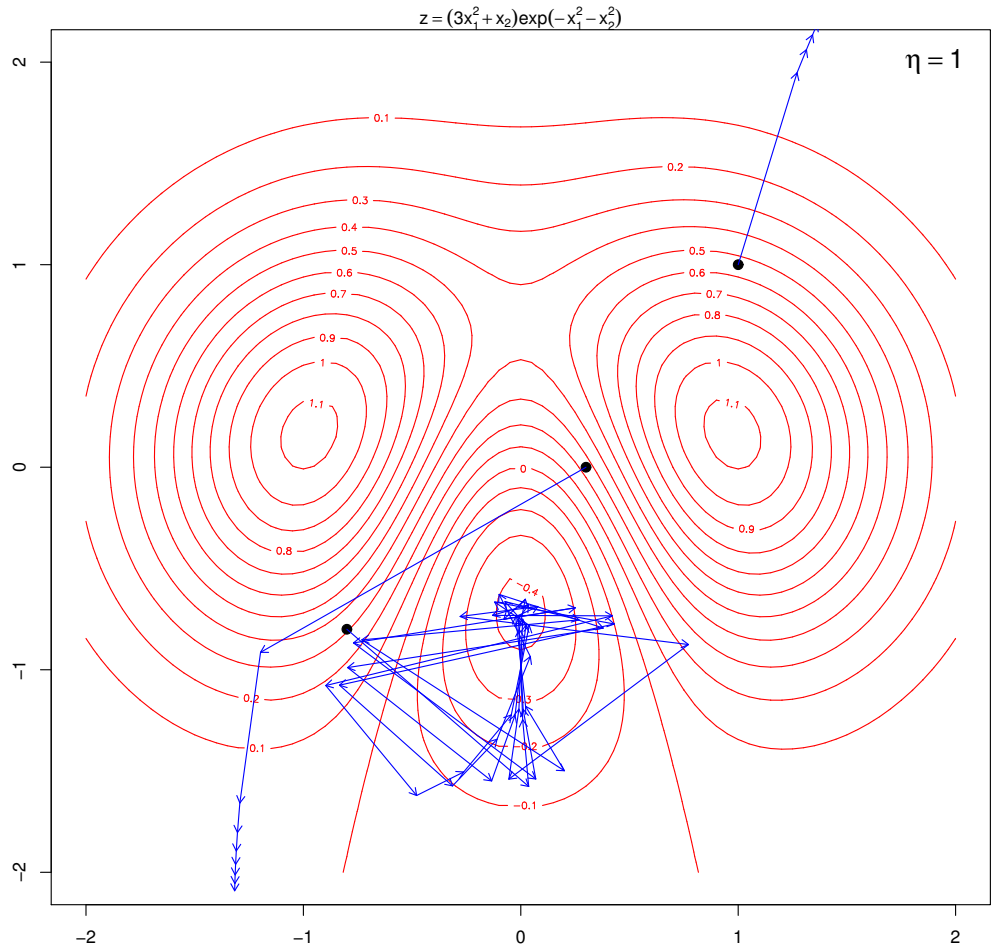
Steepest Descent Algorithm Example

Black points denote different starting values. Learning rate η is properly chosen, however for starting value $(1, 1)$, algorithm converges not to the global minimum. It follows steepest descent in the "wrong direction", in other words, gradient based algorithms are local search algorithms.



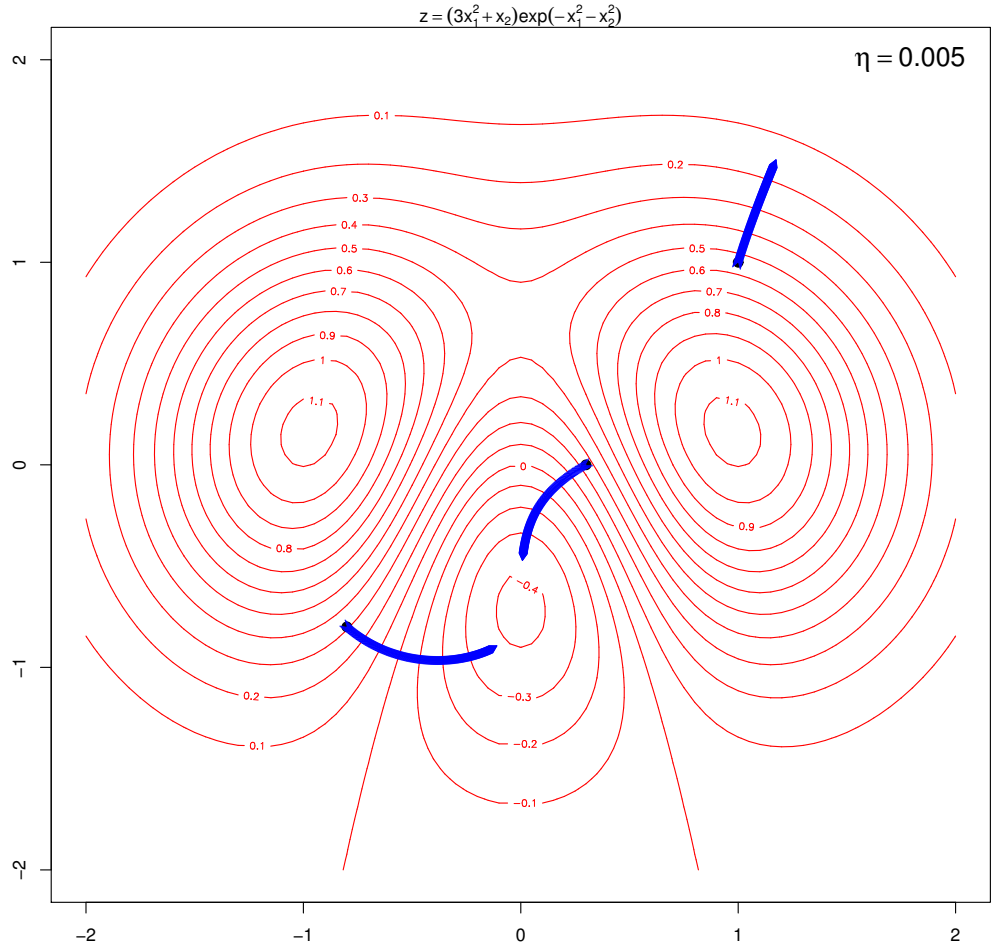
Steepest Descent Algorithm Example (cont.)

Learning rate $\eta = 1.0$ is too large, algorithm oscillates in a "*zig-zag*" manner or "*overleap*" the global minimum.

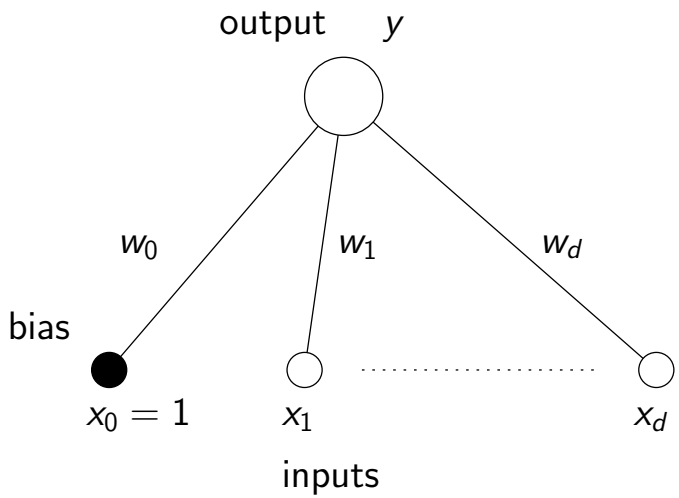


Steepest Descent Algorithm Example (cont.)

Learning rate $\eta = 0.005$ is too small, algorithm converges "very slowly".



Single-Layer Network



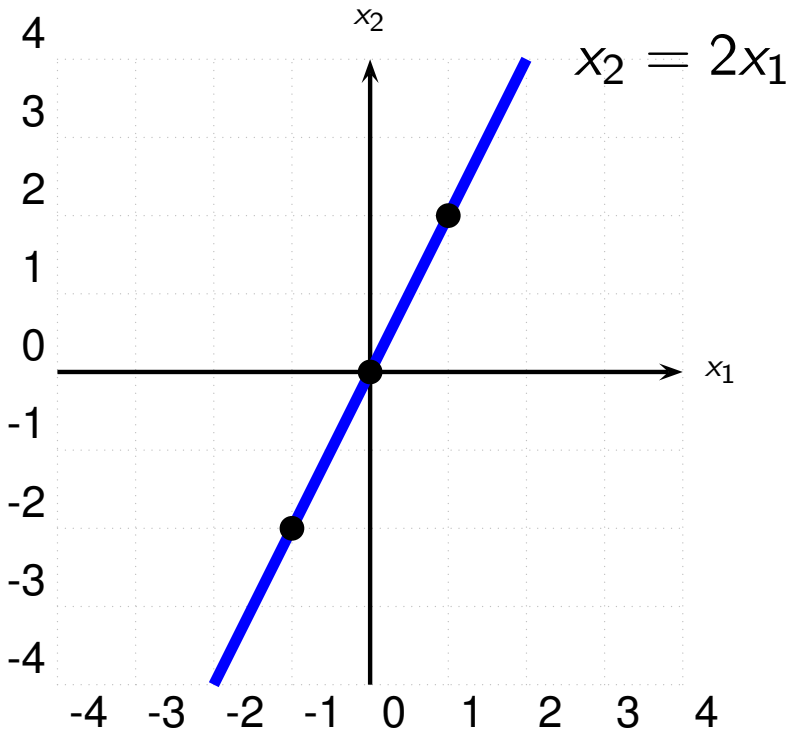
Equivalent notation:

$$y(\mathbf{x}) = \widetilde{\mathbf{w}}^T \widetilde{\mathbf{x}} = \sum_{i=0}^{d} w_i x_i$$

where $\widetilde{\mathbf{w}} = (w_0, \mathbf{w})$ and $\widetilde{\mathbf{x}} = (1, \mathbf{x})$.

Linear Classifier

- Linear classifiers are single layer neural networks.



Observe, that $x_2 = 2x_1$ can also be expressed as

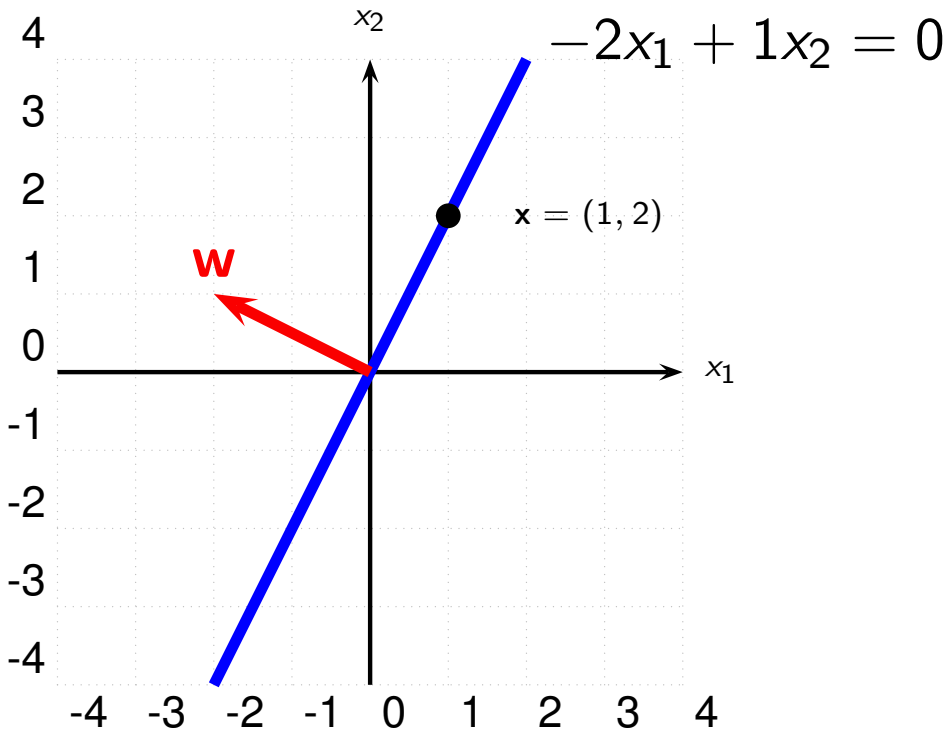
$$w_1 x_1 + w_2 x_2 = 0 \Leftrightarrow x_2 = -\frac{w_1}{w_2} x_1,$$

where for instance

$$w_1 = -2,\ w_2 = 1.$$

Furthermore, observe that all points lying on the line $x_2 = 2x_1$ satisfy $w_1 x_1 + w_2 x_2 = -2x_1 + 1x_2 = 0$.

Linear Classifier & Dot Product



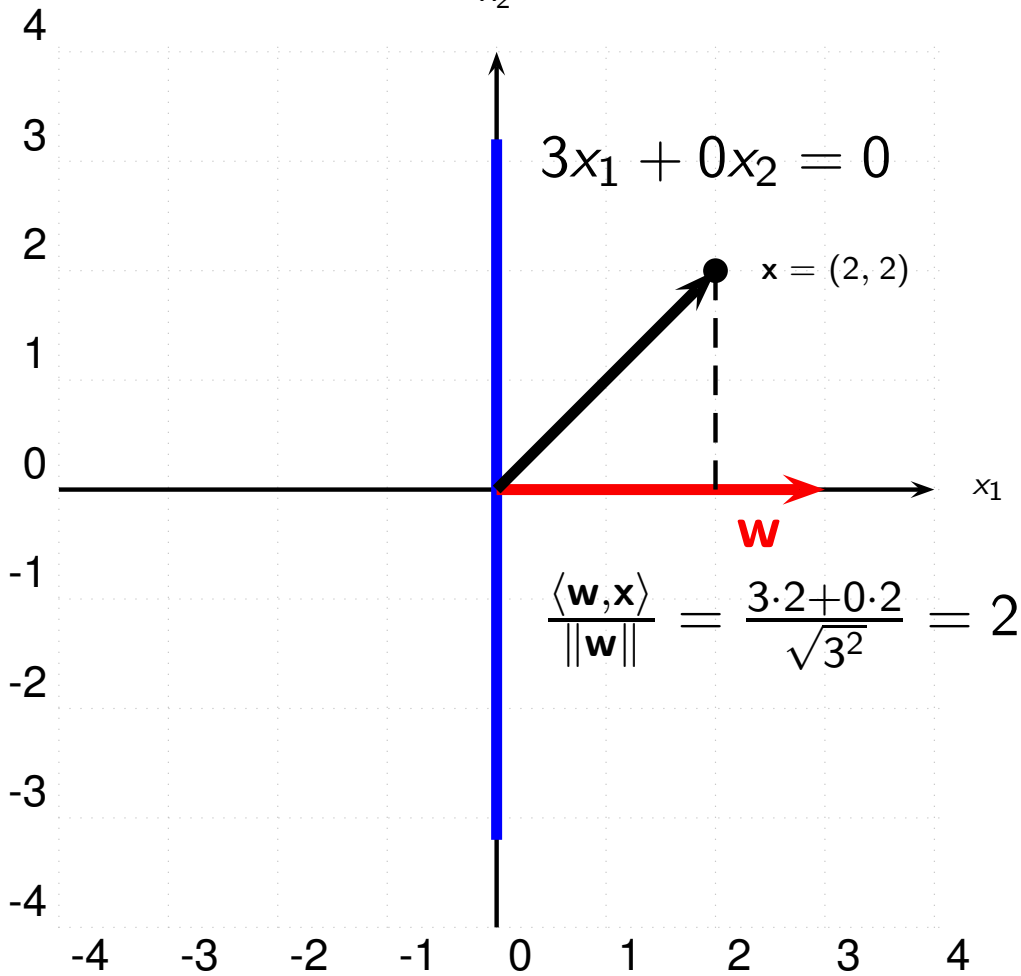
- What about the vector $\mathbf{w} = (w_1, w_2) = (-2, 1)$?
- Vector $\mathbf{w}$ is perpendicular to the line $-2x_1 + 1x_2 = 0$.
- Let us calculate the dot product of $\mathbf{w}$ and $\mathbf{x}$.

The dot product is defined as $w_1 x_1 + w_2 x_2 + \ldots + w_d x_d \stackrel{def}{=} \langle \mathbf{w}, \mathbf{x} \rangle$, for some $d \in \mathbb{N}$.

In our example $d = 2$ and we obtain $-2 \cdot 1 + 1 \cdot 2 = 0$.

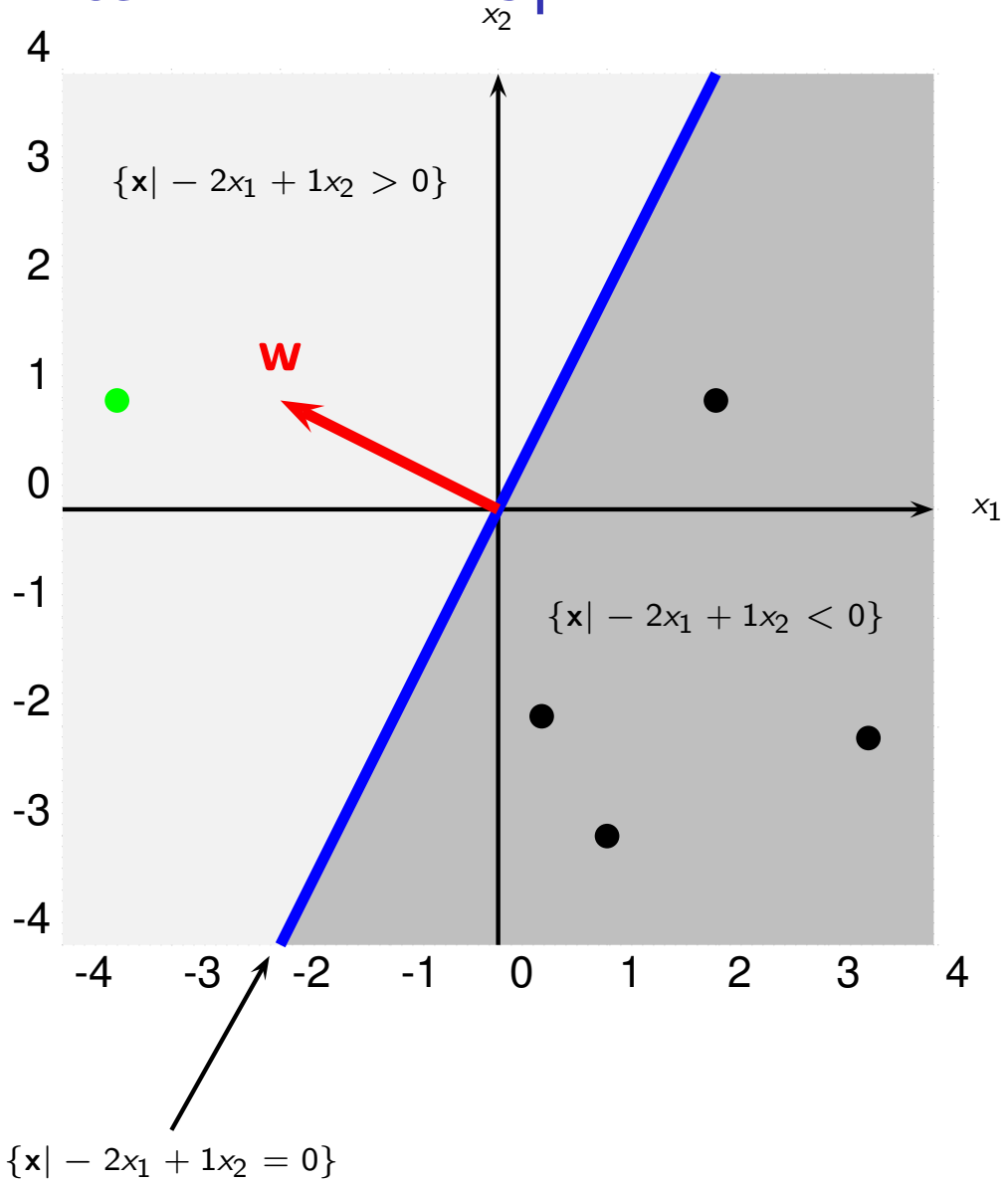
Linear Classifier & Dot Product (cont.)

Let us consider the *weight* vector $\mathbf{w} = (3, 0)$ and vector $\mathbf{x} = (2, 2)$.



Geometric interpretation of the dot product: Length of the projection of $\mathbf{x}$ onto the unit vector $\mathbf{w}/\|\mathbf{w}\|$.

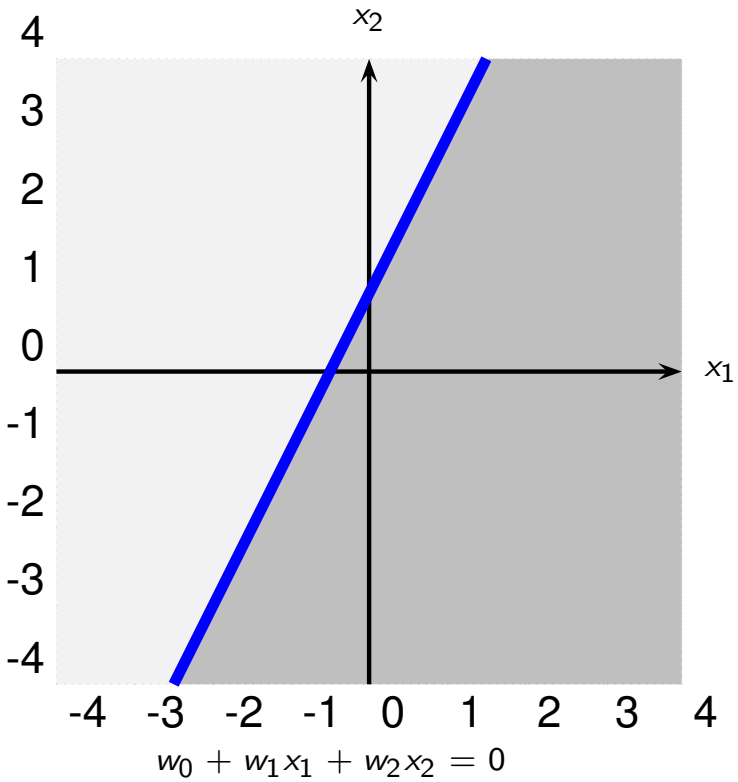
Linear Classifier & Two Half-Spaces

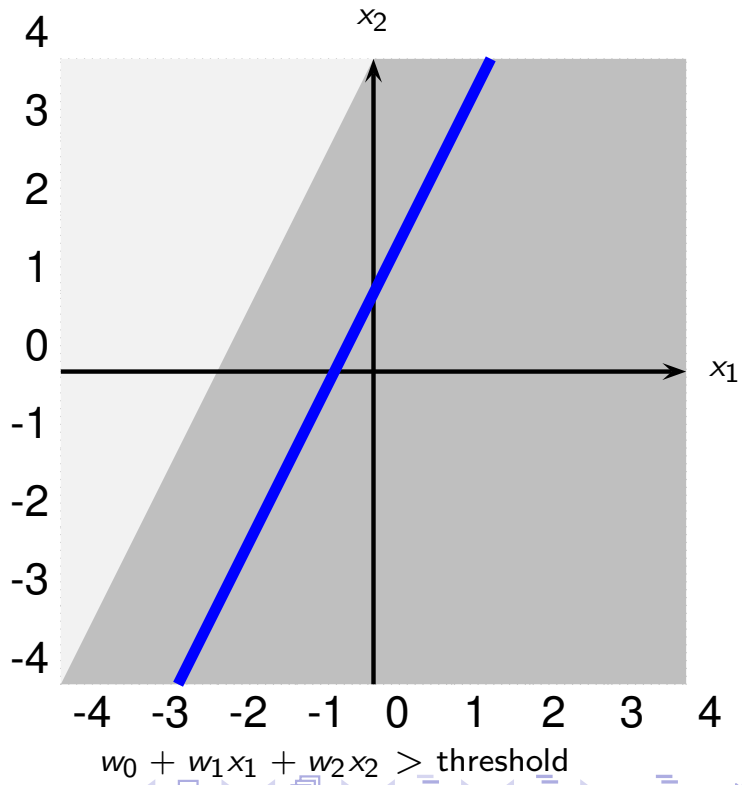


The **x**-space is separated in two half-spaces.

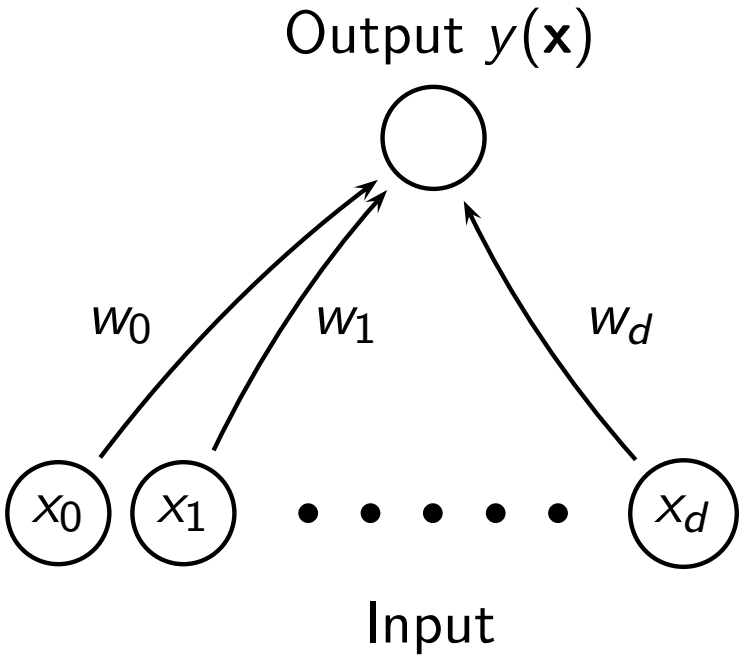
Linear Classifier & Dot Product (cont.)

- Observe, that $w_1 x_1 + w_2 x_2 = 0$ implies, that the separating line always goes through the origin.
- By adding an offset (bias), that is $w_0 + w_1 x_1 + w_2 x_2 = 0 \Leftrightarrow x_2 = -\frac{w_1}{w_2} x_1 - \frac{w_0}{w_2} \equiv y = mx + b$, one can shift the line arbitrary.

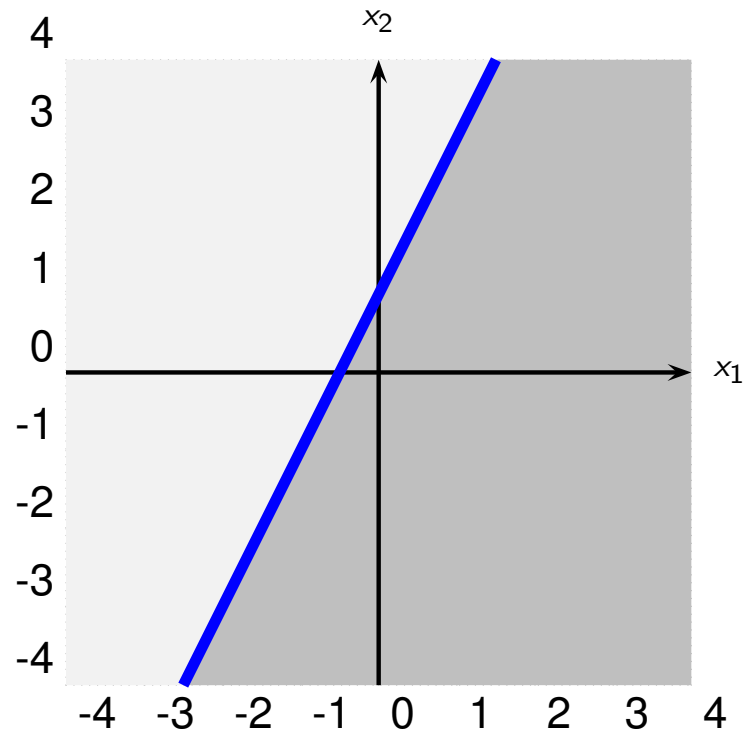




Linear Classifier & Single Layer NN



$\Leftrightarrow$



Note that $x_0 = 1$, $y(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + w_0$.

Given data which we want to separate, that is, a sample $\mathcal{X} = \{(\mathbf{x}_1, t_1), (\mathbf{x}_2, t_2), \ldots, (\mathbf{x}_N, t_N)\} \in \mathbb{R}^d \times \{-1, +1\}$.

How to determine the proper values of $\mathbf{w}$ such that the "minus" and "plus" points are separated by $y(\mathbf{x})$? Infer the values of $\mathbf{w}$ from the data by some learning algorithm.

Perceptron

Note, so far we have not seen a method for finding the weight vector $\mathbf{w}$ to obtain a linearly separation of the training set.

Let $g(a)$ be (sign) activation function

$$g(a) = \begin{cases} -1 & \text{if } a < 0 \\ +1 & \text{if } a \geq 0 \end{cases}$$

and decision function

$$g(\langle \mathbf{w}, \mathbf{x} \rangle) = g\left(\sum_{i=0}^{d} w_i x_i \right).$$

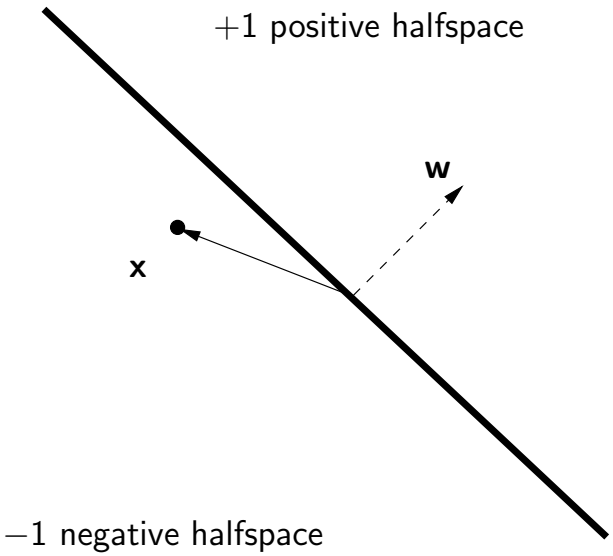
Note: x_0 is set to $+1$, that is, $\mathbf{x} = (1, x_1, \ldots, x_d)$. Training pattern consists of $(\mathbf{x}, t) \in \mathbb{R}^{d+1} \times \{-1, +1\}$

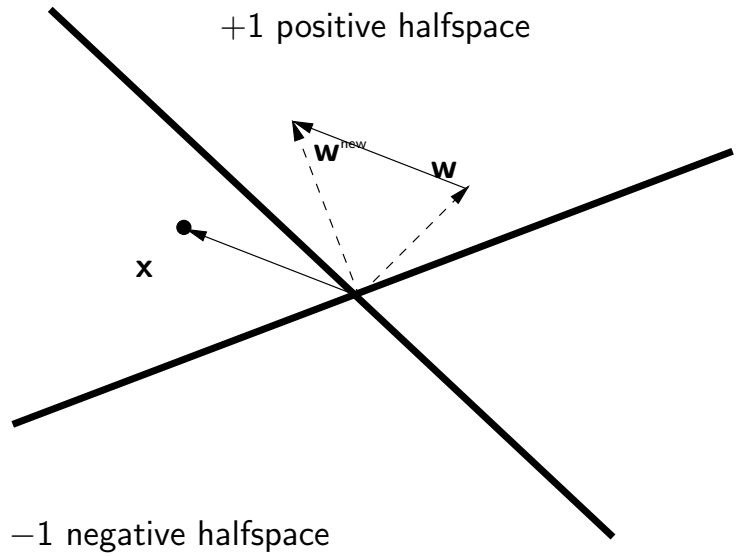
Perceptron Learning Algorithm

input : $(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N) \in \mathbb{R}^{d+1} \times \{-1, +1\}, \eta \in \mathbb{R}_+, \text{max.epoch} \in \mathbb{N}$
output: $\mathbf{w}$
begin
  Randomly initialize $\mathbf{w}$
  epoch $\leftarrow 0$
  repeat
    for $i \leftarrow 1$ **to** N **do**
      if $t_i \langle \mathbf{w}, \mathbf{x}_i \rangle \leq 0$ **then**
        $\mathbf{w} \leftarrow \mathbf{w} + \eta \mathbf{x}_i \, t_i$
    epoch $\leftarrow$ epoch $+ 1$
  until (*epoch* $=$ *max.epoch*) *or* (*no change in* $\mathbf{w}$)
  return $\mathbf{w}$
end

Training the Perceptron (cont.)

Geometrical explanation: If $\mathbf{x}$ belongs to $\{+1\}$ and $\langle \mathbf{w}, \mathbf{x} \rangle < 0 \Rightarrow$ angle between $\mathbf{x}$ and $\mathbf{w}$ is greater than 90°, rotate $\mathbf{w}$ in direction of $\mathbf{x}$ to bring missclassified $\mathbf{x}$ into the positive half space defined by $\mathbf{w}$. Same idea if $\mathbf{x}$ belongs to $\{-1\}$ and $\langle \mathbf{w}, \mathbf{x} \rangle \geq 0$.





Perceptron Error Reduction

Recall: missclassifcation results in:

$$\mathbf{w}^{\text{new}} = \mathbf{w} + \eta \mathbf{x}\, t,$$

this reduces the error since

$$\begin{aligned} -\mathbf{w}^{\text{new}}(\mathbf{x}\, t)^T &= -\mathbf{w}(\mathbf{x}\, t)^T - \underbrace{\eta}_{>0} \underbrace{(\mathbf{x}\, t)(\mathbf{x}\, t)^T}_{\|\mathbf{x}t\|^2>0} \\ &< -\mathbf{w}^T \mathbf{x} t \end{aligned}$$

How often one has to cycle through the patterns in the training set?

- A finite number of steps?

Perceptron Convergence Theorem

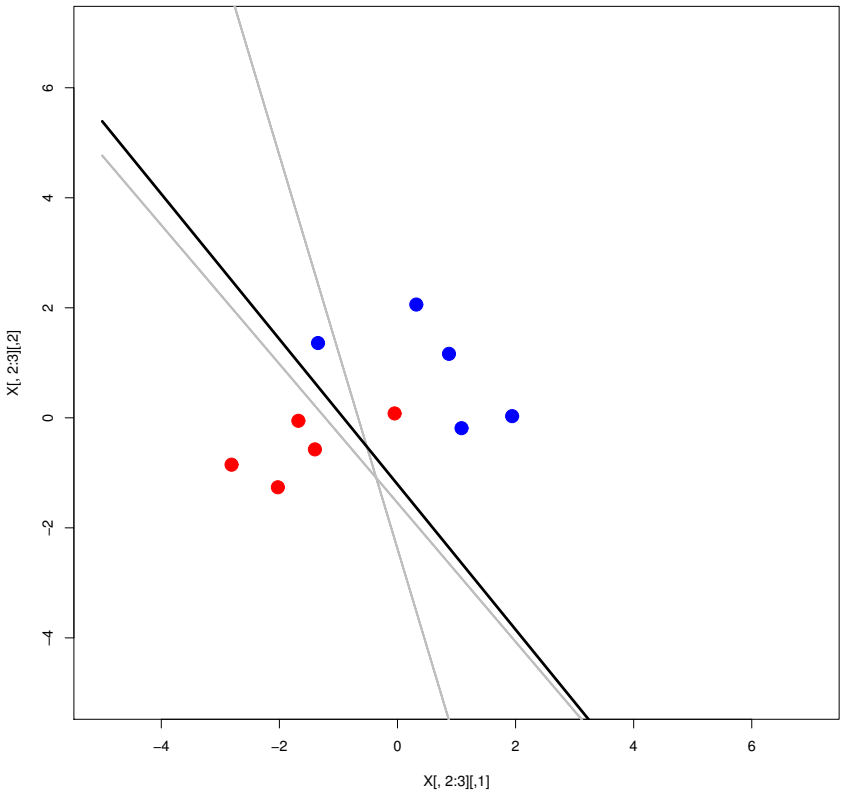
Proposition

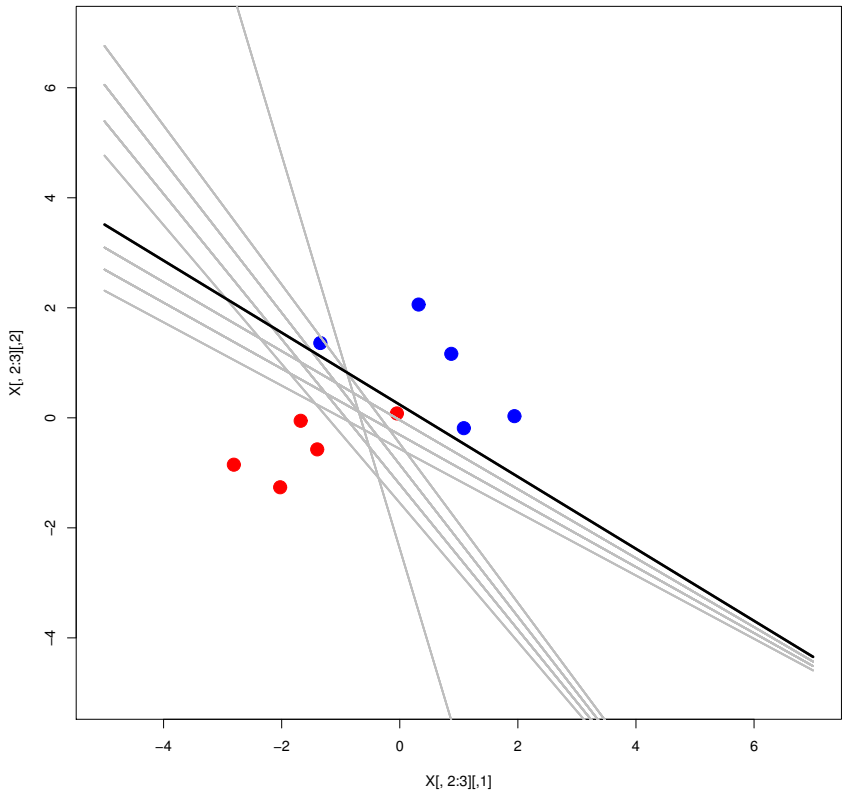
Given a finite and linearly separable training set. The perceptron converges after some finite steps [Rosenblatt, 1962].

Perceptron Algorithm (R-code)

```
##################################################
perceptron <- function(w,X,t,eta,max.epoch) {
##################################################
  N <- nrow(X)/2;
  epoch <- 0;
  repeat {
    w.old <- w;
    for (i in 1:(2*N)) {
      if ( t[i]*y(X[i,],w) <= 0 )
        w <- w + eta * t[i] * X[i,];
    }
    epoch <- epoch + 1;
    if ( identical(w.old,w) || epoch = max.epoch ) {
      break; # terminate if no change in weights or max.epoch
    }
  }
  return (w);
```

Perceptron Algorithm Visualization

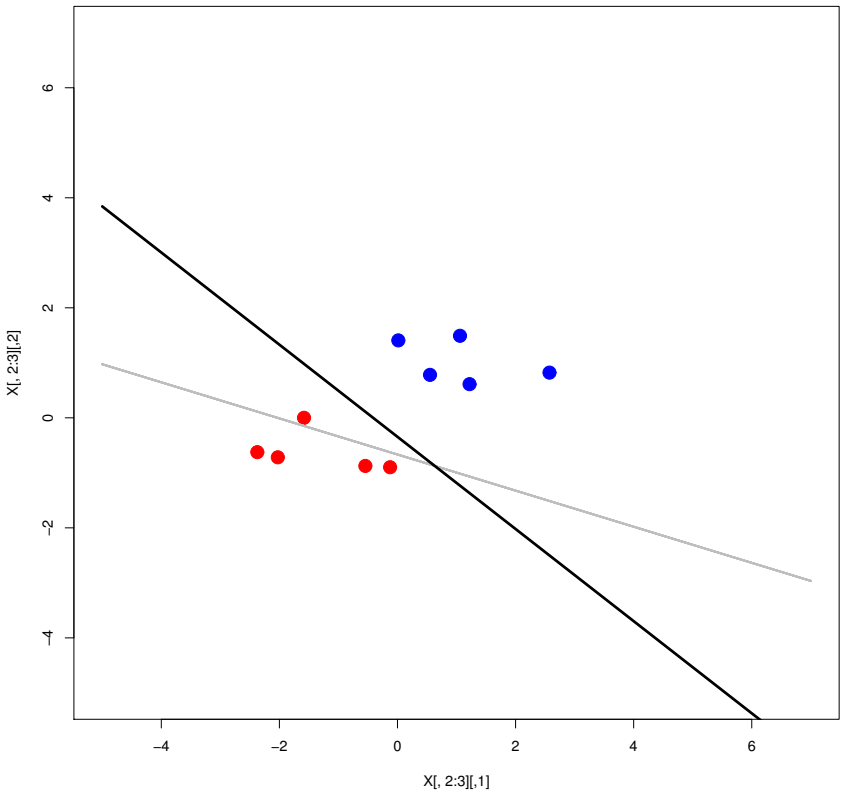


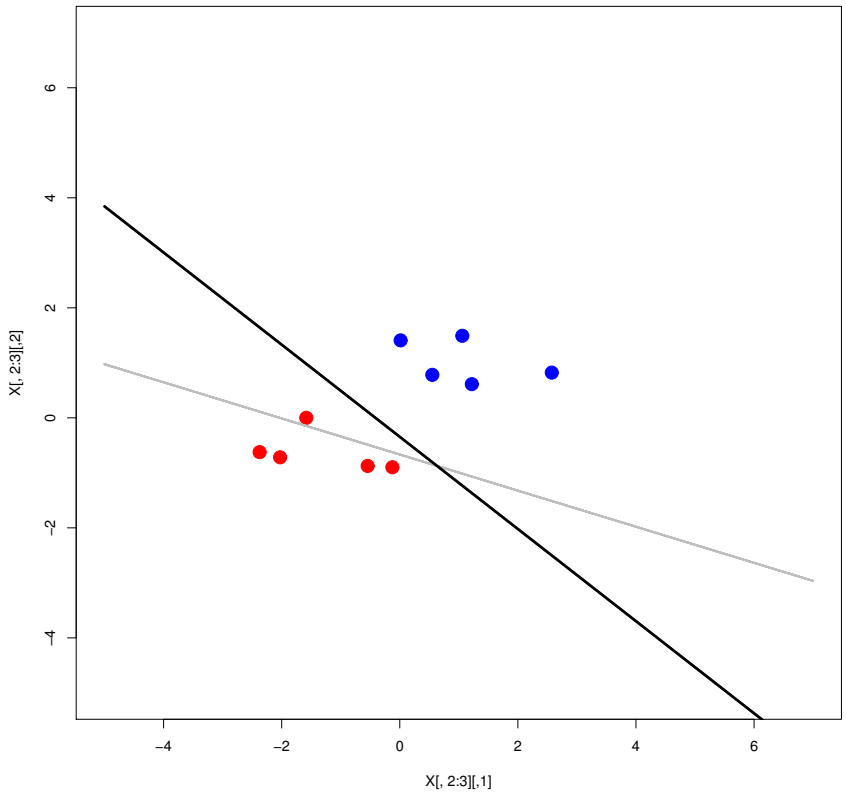


One epoch

terminate if no change in **w**

Perceptron Algorithm Visualization





One epoch

terminate if no change in **w**

Least Mean Square

Let us consider the weight correction in terms of an error function $E^{(i)} = \frac{1}{2}(\underbrace{y^{(i)}}_{g(\mathbf{w}^T\mathbf{x})} - t^{(i)})^2$, where $g(\cdot)$ is a differentiable function. Apply gradient descent rule

$$\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta \frac{\partial E^{(i)}}{\partial \mathbf{w}}, \text{ where } \quad \frac{\partial E^{(i)}}{\partial \mathbf{w}} = \underbrace{(y^{(i)} - t^{(i)})}_{\delta^{(i)}} \mathbf{x}^{(i)}$$

gives change in weights

$$\Delta \mathbf{w} = -\eta \delta^{(i)} \mathbf{x}^{(i)} = -\eta \frac{\partial E^{(i)}}{\partial \mathbf{w}}$$

Delta rule $\equiv$ {Adaline rule, Widrow-Hoff rule, Least Mean Square (LMS) }

Least Mean Square

Note, if we choose $g(a) = a$ to be the linear activation function $\mathbf{w}^T\mathbf{x}$, then there exists a closed analytical solution (pseudo-inverse solution).

Let $g(a)$ be a differentiable non-linear activation function, where $a = \mathbf{w}^T\mathbf{x}$.

$$\frac{\partial E^{(i)}}{\partial \mathbf{w}} = \delta^{(i)}\mathbf{x}^{(i)}, \text{ where } \quad \delta^{(i)} = g'(a)(y^{(i)} - t^{(i)})$$

gives change in weights

$$\Delta\mathbf{w} = -\eta\delta^{(i)}\mathbf{x}^{(i)} = -\eta\frac{\partial E^{(i)}}{\partial \mathbf{w}}$$

LMS Online/Batch Learning

Online learning:

- Update weight $\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta \frac{\partial E^{(i)}}{\partial \mathbf{w}}$ (pattern by pattern).

This type of online learning is also called *stochastic gradient descent*, it is an approximation of the true gradient.

Batch learning:

- Update weight $\mathbf{w}^{(i+1)} = \mathbf{w}^{(i)} - \eta \sum_{i=1}^{N} \frac{\partial E^{(i)}}{\partial \mathbf{w}}$ by computing derivatives for each pattern separately and then sum over all patterns.

Minimum Squared Error and Pseudoinverse

Recall that we want to minimize the squared error

$$E(\mathbf{w}) = \sum_{i=1}^{N} \frac{1}{2} \left(y^{(i)} - t^{(i)} \right)^2 \qquad \text{where } y^{(i)} = \mathbf{w}^T \mathbf{x}^{(i)}$$

Let $\mathbf{X}$ be the $N \times \widetilde{d}$ matrix where $\widetilde{d} = d + 1$ and ith row denotes training pattern $\mathbf{x}^{(i)\,T}$, $\mathbf{w}$ is weight vector, $\mathbf{t}$ class label vector.

$$\begin{pmatrix} x_{10} & x_{11} & \cdots & x_{1d} \\ x_{20} & x_{21} & \cdots & x_{2d} \\ \vdots & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ x_{N0} & x_{N1} & \cdots & x_{Nd} \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{pmatrix} = \begin{pmatrix} t_0 \\ t_1 \\ \vdots \\ \vdots \\ t_N \end{pmatrix} \qquad \mathbf{Xw} = \mathbf{t}$$

MSE and Pseudoinverse (cont.)

Problem: find weight vector $\mathbf{w}$, that is, solve $\mathbf{X}\mathbf{w} = \mathbf{t}$.

If $\mathbf{X}$ is non-singular solve $\mathbf{w} = \mathbf{X}^{-1}\mathbf{t}$, however, if $\mathbf{X}$ is rectangular (which is usually the case), then there are more equations than unknowns, that is, the equation system is overdetermined.

Let us search for $\mathbf{w}$ that minimizes the error

$$\mathbf{e} = \mathbf{X}\mathbf{w} - \mathbf{t}$$

one approach is to minimize the squared length of the error vector $\mathbf{e}$

$$J(\widetilde{\mathbf{w}}) = \|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2 = \sum_{i=1}^{N} \left(\mathbf{w}^T \mathbf{x}^{(i)} - t^{(i)}\right)^2$$

MSE and Pseudoinverse (cont.)

Forming the gradient

$$\nabla J = \sum_{i=1}^{N} 2\left(\mathbf{w}^T\mathbf{x}^{(i)} - t^{(i)}\right)\mathbf{x}^{(i)} = 2\mathbf{X}^T(\mathbf{X}\mathbf{w} - \mathbf{t})$$

and setting ∇J to zero gives $\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{t}$. Observe that $\mathbf{X}^T\mathbf{X}$ is a $\widetilde{d} \times \widetilde{d}$ matrix which often is non-singular. In the non-singular case, one can solve $\mathbf{w}$ uniquely as

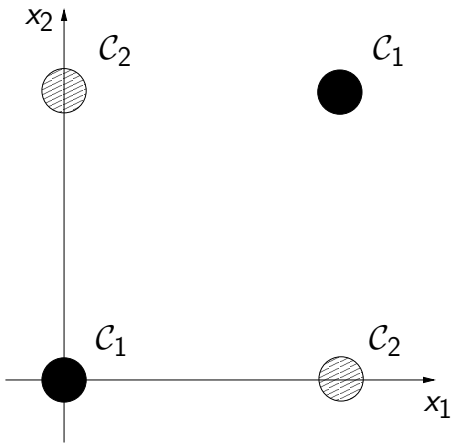
$$\begin{aligned} \mathbf{w} &= \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{t} \\ &= \mathbf{X}^{\dagger}\mathbf{t} \end{aligned}$$

The $\widetilde{d} \times N$ matrix $\mathbf{X}^{\dagger} \equiv \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T$ is called *pseudoinverse* of $\mathbf{X}$.

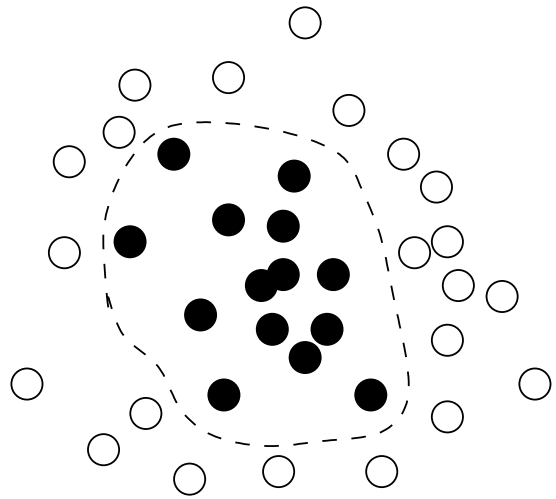
Linear Separability

Decision boundaries of single-layer networks are linear (hyperplanar in higher dimensions).

- Very restricted class of decision boundaries
- Examples:



XOR-Problem



Points are not linearly separable

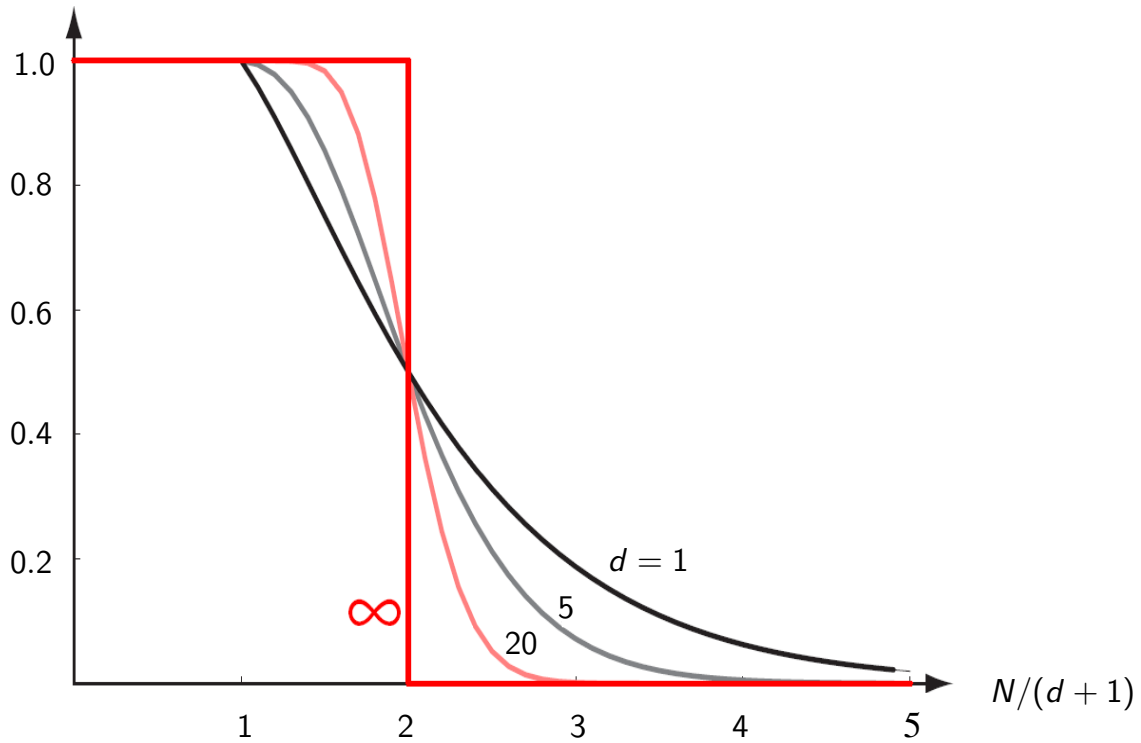
Probability for Linear Separability

- Probability that a random set of points will be linearly separable
- Suppose we have N points distributed at random in d dimensions in general position (not collinear)
- Randomly assign each of the points to one of the two classes $\mathcal{C}_1$ and $\mathcal{C}_2$ (with eq. probability)
- For N data points there are 2^N possible class assignments (dichotomies $\equiv$ binary partitions)

Question: What fraction $F(N, d)$ of these dichotomies is linearly separable?

Probability for Linear Separability (cont.)

$$F(N,d) = \begin{cases} 1 & \text{when } N \leq d+1 \\ \frac{1}{2^{N-1}} \sum_{i=0}^{d} \binom{N-1}{i} & \text{when } N \geq d+1 \end{cases}$$



If number of points is $\leq d+1$, then any labeling leads to a separable problem.

Activation functions

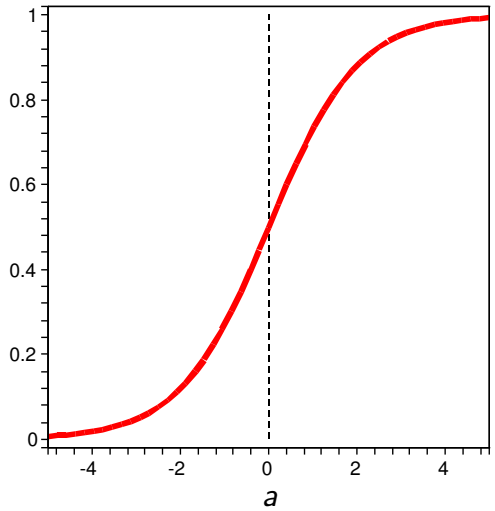
Discrimination functions of the form $y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0$ are simple linear functions of the input variables $\mathbf{x}$, where distances are measured by means of the dot product.

Let us consider the non-linear *logistic sigmoid* activation function $g(\cdot)$ for limiting the output to $(0, 1)$, that is,

$$y(\mathbf{x}) = g(\mathbf{w}^T\mathbf{x} + w_0),$$

where

$$g(a) = \frac{1}{1 + \exp(-a)}$$

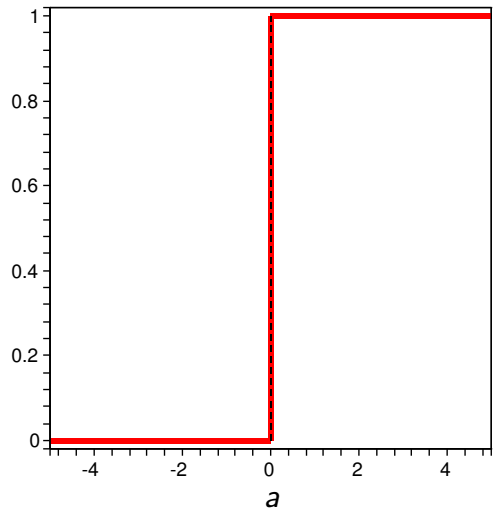


Single-layer network with a logistic sigmoid activation function can also output posterior probabilities (rather than geometric distances).

Activation functions (cont.)

Heaviside step function:

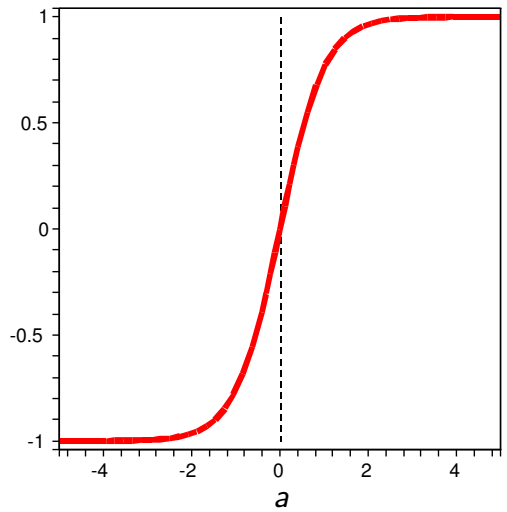
$$g(a) = \begin{cases} 0 & \text{if } a < 0 \\ 1 & \text{if } a \geq 0 \end{cases}$$



Hyperbolic tangent function:

$$g(a) = \tanh(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)}$$

Note, $\tanh(a) \in (-1, 1)$



Multi-Layer vs. Single-Layer Networks

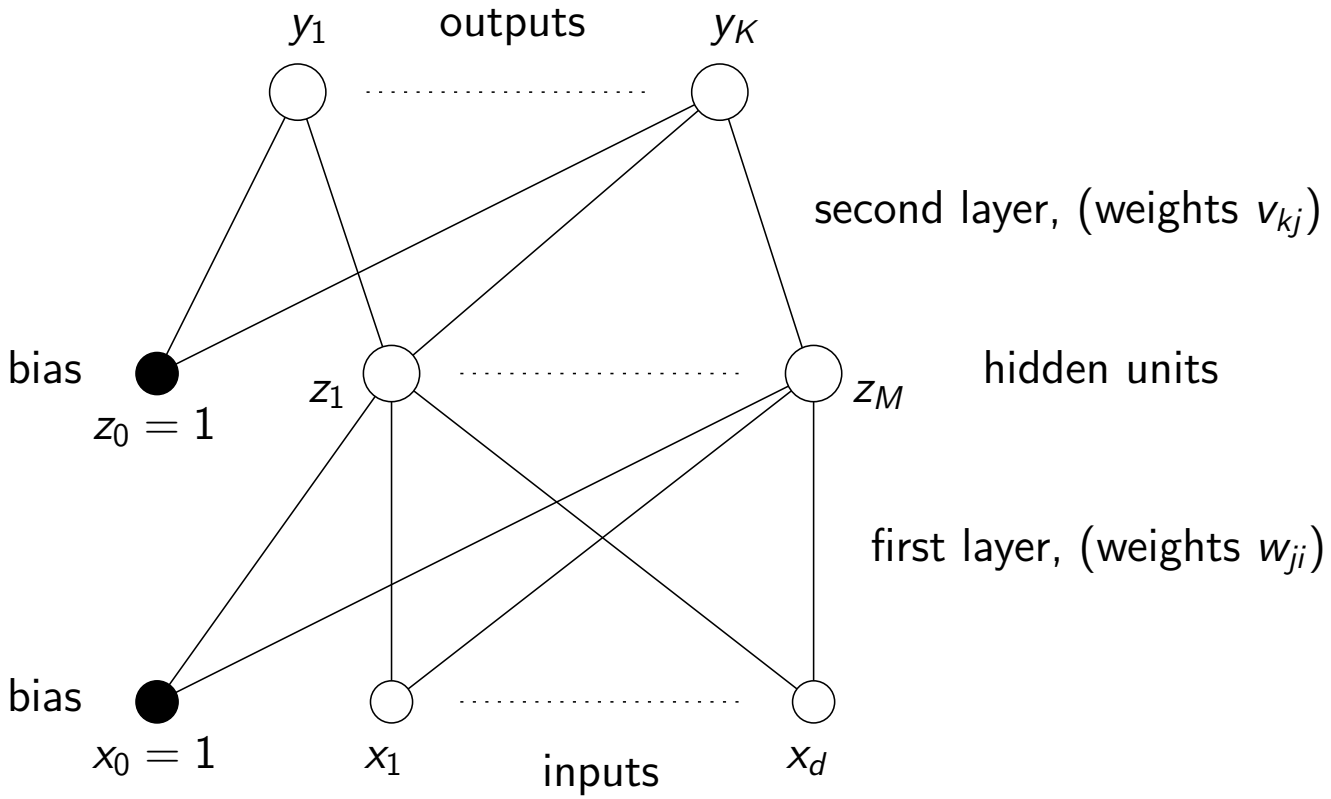
Single-layer networks

- based on a linear combination of the input variables which is transformed by linear/non-linear activation function
- are limited in terms of the range of functions they can represent

Multi-layer networks

- consist of multiple layers and are capable of approximating any continuous functional mapping
- are compared to single-layer networks not so straightforward to train

Multi-Layer Network



Connection in first layer from input unit i to hidden unit j is denoted as w_{ji}.
Connection from hidden unit j to output unit k is denoted as v_{kj}.

Multi-Layer Network (cont.)

Hidden unit j receives input

$$a_j = \sum_{i=1}^{d} w_{ji} x_i + w_{j0} = \sum_{i=0}^{d} w_{ji} x_i$$

and produces output

$$z_j = g(a_j) = g\left(\sum_{i=0}^{d} w_{ji} x_i \right).$$

Output unit k thus receives

$$a_k = \sum_{j=1}^{M} v_{kj} z_j + v_{k0} = \sum_{j=0}^{M} v_{kj} z_j$$

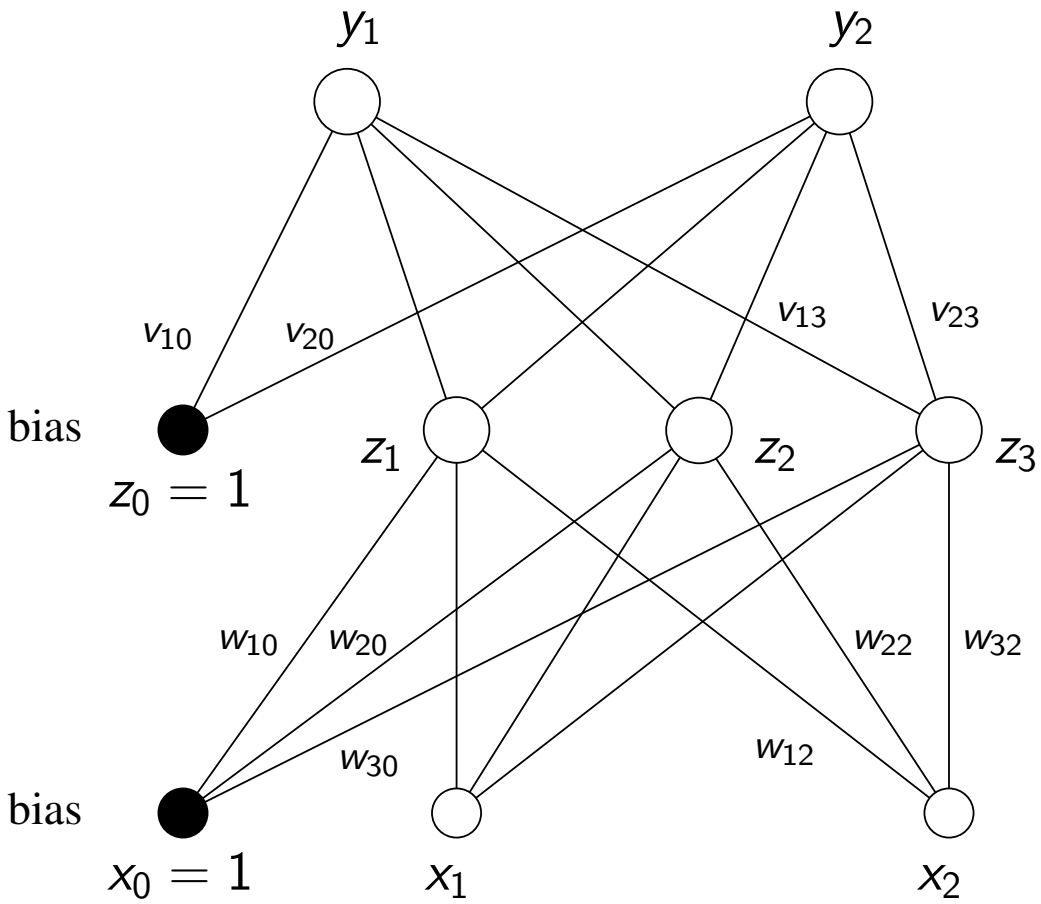
Multi-Layer Network (cont.)

and produces the final output

$$y_k = g(a_k) = g\left(\sum_{j=0}^{M} v_{kj} z_j\right) = g\left(\sum_{j=0}^{M} v_{kj}\, g\left(\sum_{i=0}^{d} w_{ji} x_i\right)\right)$$

Note that the activation function $g(\cdot)$ in the first layer can be different from those in the second layer (or other layers).

Multi-Layer Networks Example



Note: sometimes the layers of units are counted (here three layers), rather the layers of adaptive weights. In this course L-layer network is referred to a network with L layers of adaptive weights.

LMS Learning Rule for Multi-Layer Networks

- We have seen that the LMS learning rule is based on the gradient descent algorithm.
- The LMS learning rule worked because the error is proportional to the square difference between actual output y and target output t and can be evaluated for each output unit.
- In a multi-layer network we can use LMS learning rule on the hidden-to-output layer weights because target outputs are known.

Problem: we cannot compute the target outputs of the input-to-hidden weights because these values are unknown, or, to put it the other way around, how to update the weights in the first layer?

Backpropagation (Hidden-to-Output Layer)

Recall that we want to minimize the error on training patterns between actual output y_k and target output t_k:

$$E = \frac{1}{2}\sum_{k=1}^{K}(y_k - t_k)^2.$$

Backpropagation learning rule is based on gradient descent:

$$\Delta \mathbf{w} = -\eta \frac{\partial E}{\partial \mathbf{w}}, \text{ component form } \Delta w_{st} = -\eta \frac{\partial E}{\partial w_{st}}$$

Apply chain rule for differentiation:

$$\frac{\partial E}{\partial v_{kj}} = \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial v_{kj}}$$

Backprop. (Hidden-to-Output Layer) (cont.)

Gradient descent rule gives:

$$\begin{aligned} \Delta v_{kj} &= -\eta \frac{\partial E}{\partial v_{kj}} = -\eta (y_k - t_k) g'(a_k) z_j \\ &= -\eta \delta_k z_j \end{aligned}$$

where

$$\delta_k = (y_k - t_k) g'(a_k).$$

Observe that this result is identical to that obtained for LMS.

Backpropagation (Input-to-Hidden Layer)

For the input-to-hidden connection we must differentiate with respect to the w_{ji}'s which are deeply embedded in

$$E = \frac{1}{2}\sum_{k=1}^{K}\left[g\left(\sum_{j=0}^{M} v_{kj}\, g\left(\sum_{i=0}^{d} w_{ji}x_i\right)\right) - t_k\right]^2$$

Apply chain rule:

$$\begin{aligned}
\Delta w_{ji} &= -\eta\frac{\partial E}{\partial w_{ji}} = -\eta\frac{\partial E}{\partial z_j}\frac{\partial z_j}{\partial a_j}\frac{\partial a_j}{\partial w_{ji}} \\
&= -\eta\sum_{k=1}^{K}\underbrace{(y_k - t_k)g'(a_k)}_{\delta_k} v_{kj}g'(a_j)x_i \\
&= -\eta\sum_{k=1}^{K}\delta_k v_{kj}g'(a_j)x_i
\end{aligned}$$

Backprop. (Input-to-Hidden Layer) (cont.)

$$\Delta w_{ji} = -\eta \delta_j x_i$$

where

$$\delta_j = g'(a_j) \sum_{k=1}^{K} v_{kj} \delta_k$$

Observe: that we need to propagate the errors (δ's) backwards to update the weights **v** and **w**

$$\begin{aligned} \Delta v_{kj} &= -\eta \delta_k z_j \\ \delta_k &= (y_k - t_k) g'(a_k) \\ \Delta w_{ji} &= -\eta \delta_j x_i \\ \delta_j &= g'(a_j) \sum_{k=1}^{K} v_{kj} \delta_k \end{aligned}$$

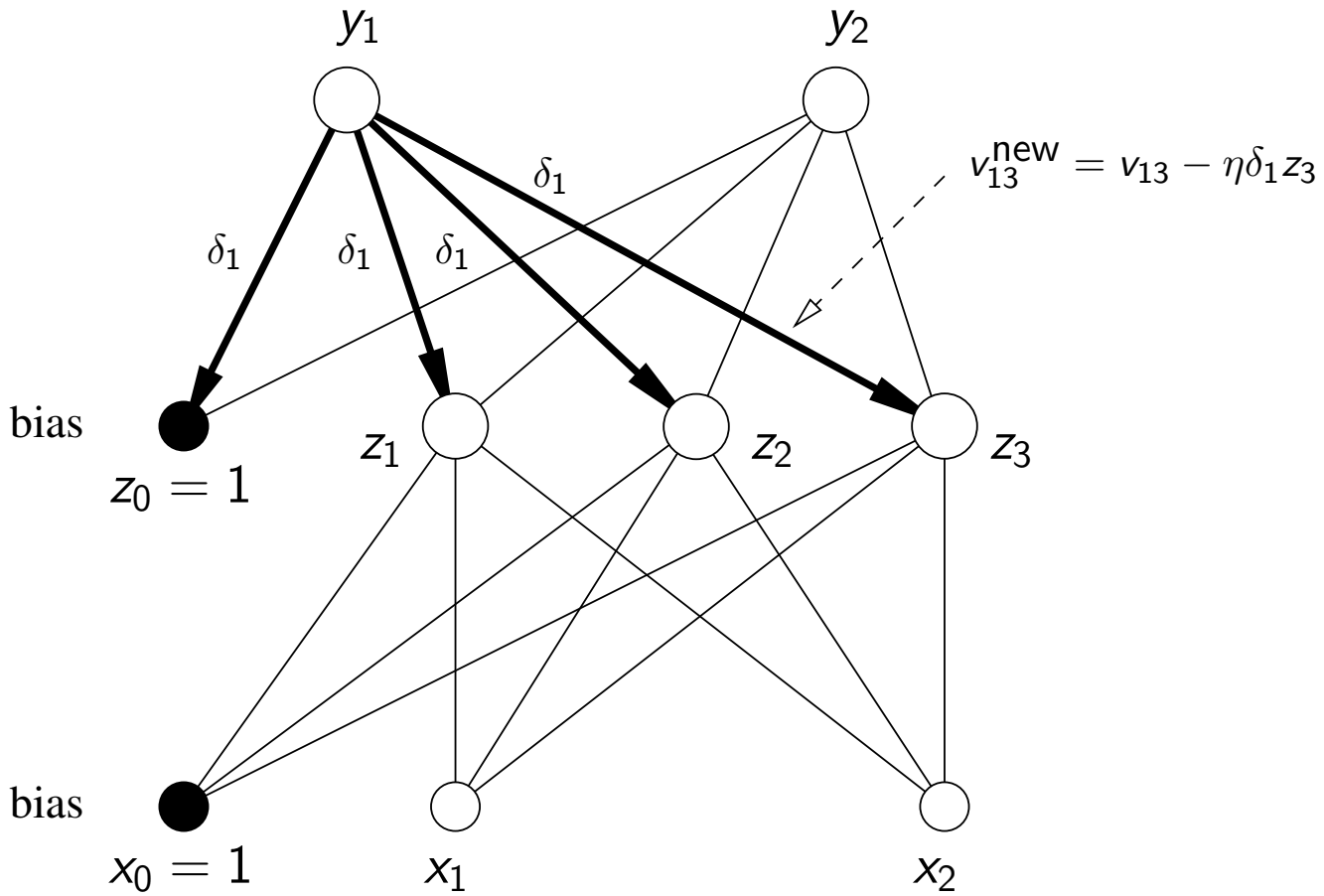
Error Backpropagation

- Apply input **x** and forward propagate through the network using $a_j = \sum_{i=0}^{d} w_{ji} x_i$ and $z_j = g(a_j)$ to find the activations of all the hidden and output units.
- Compute the deltas δ_k for all the output units using $\delta_k = (y_k - t_k) g'(a_k)$.
- Backpropagate the δ's using $\delta_j = g'(a_j) \sum_{k=1}^{K} v_{kj} \delta_k$ to obtain δ_j for each hidden unit in the network.

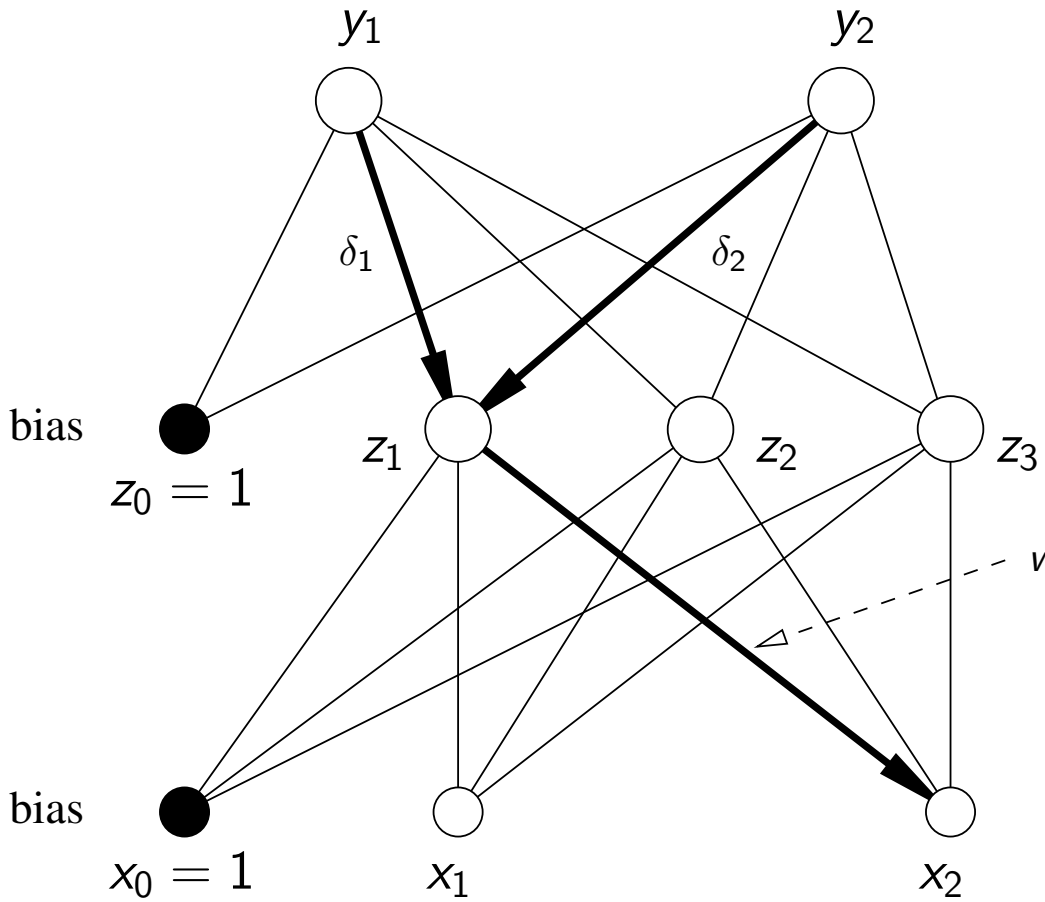
Time and space complexity:

d input units, M hidden units and K output units results in $M(d+1)$ weights in first layer and $K(M+1)$ weights in second layer. Space and time complexity is $\mathcal{O}(M\,(K+d))$. If e training epochs are performed, then time complexity is $\mathcal{O}(e\,M\,(K+d))$.

Backprop. (Output-to-Hidden Layer) Vis.



Backprop. (Hidden-to-Input Layer) Vis.



$$w_{12}^{\text{new}} = w_{12} - \eta \underbrace{[g'(a_1)(v_{11}\delta_1 + v_{21}\delta_2)]}_{\delta_j}$$

Property of Activation Functions

- In the Backpropagation algorithm the derivative of $g(a)$ is required to evaluate the δ's.
- Activation functions

$$g_1(a) = \frac{1}{1+\exp(-\beta a)} \quad \text{and} \quad g_2(a) = \tanh(\beta a)$$

obey the property

$$\begin{aligned} g_1'(a) &= \beta\, g_1(a)(1-g_1(a)) \\ g_2'(a) &= \beta(1-(g_2(a))^2) \end{aligned}$$

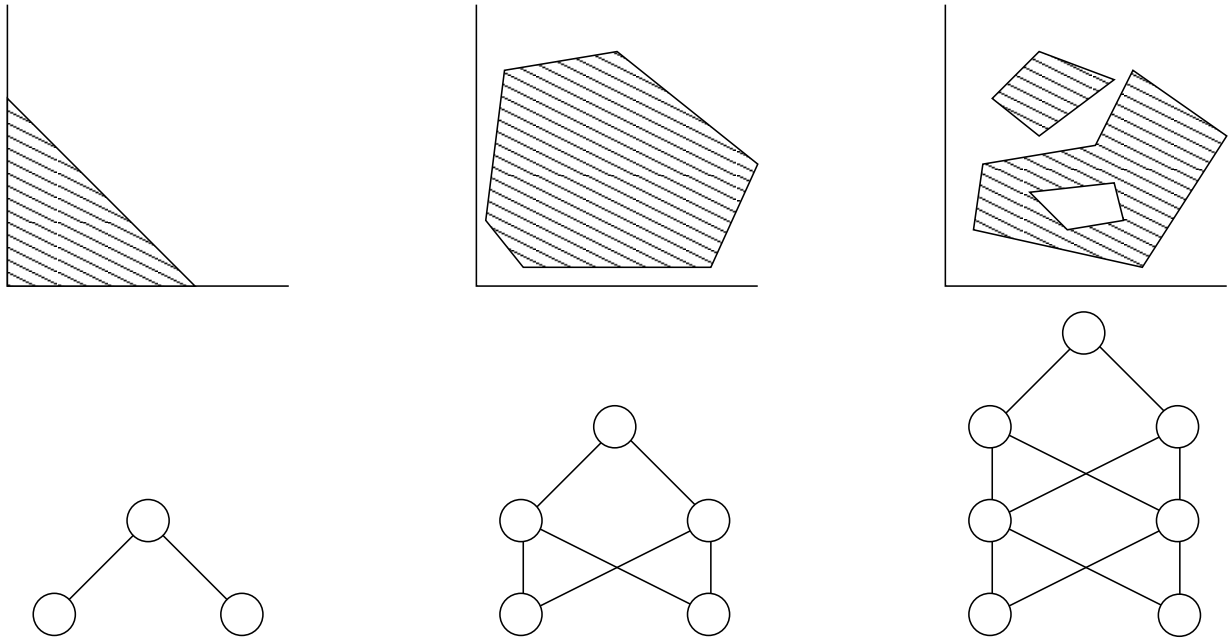
Online Backpropagation Algorithm

input : $(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N) \in \mathbb{R}^d \times \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_K\}, \eta \in \mathbb{R}_+, \text{max.epoch} \in \mathbb{N}, \epsilon \in \mathbb{R}_+$
output: $\mathbf{w}, \mathbf{v}$
begin
  Randomly initialize $\mathbf{w}, \mathbf{v}$
  epoch $\leftarrow 0$
  repeat
    for $n \leftarrow 1$ **to** N **do**
      $\mathbf{x} \leftarrow$ select pattern $\mathbf{x}_n$
      $v_{kj} \leftarrow v_{kj} - \eta \delta_k z_j$
      $w_{ji} \leftarrow w_{ji} - \eta \delta_j x_i$
    epoch $\leftarrow$ epoch $+ 1$
  until $(epoch = max.epoch)$ *or* $(\|\nabla E\| < \epsilon)$
  return $\mathbf{w}, \mathbf{v}$
end

Batch Backpropagation Algorithm

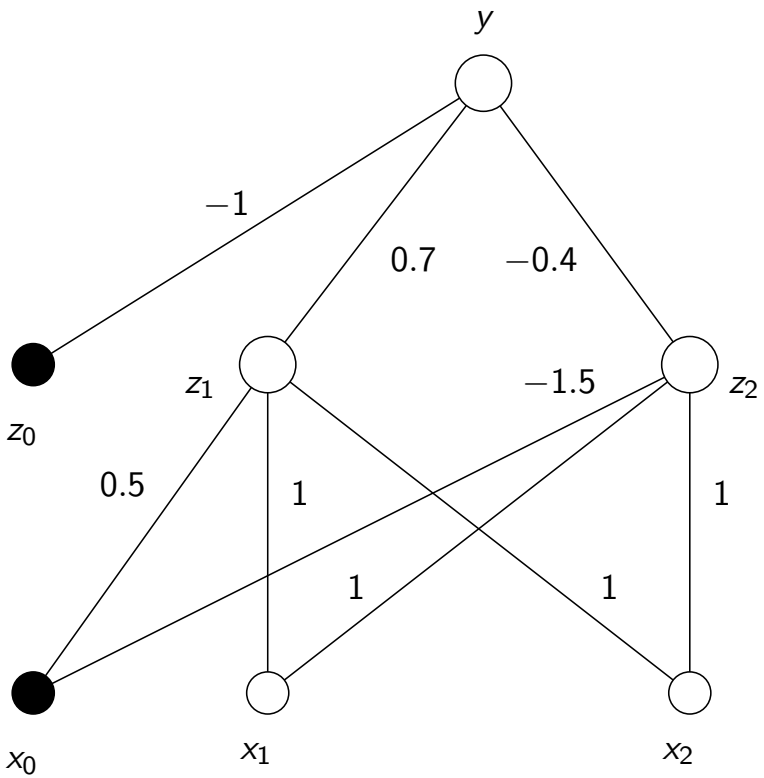
input : $(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_N, t_N) \in \mathbb{R}^d \times \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_K\}, \eta \in \mathbb{R}_+, \text{max.epoch} \in \mathbb{N}, \epsilon \in \mathbb{R}_+$
output: $\mathbf{w}, \mathbf{v}$
begin
  Randomly initialize $\mathbf{w}, \mathbf{v}$
  epoch $\leftarrow 0$, $\Delta w_{ji} \leftarrow 0$, $\Delta v_{kj} \leftarrow 0$
  repeat
    for $n \leftarrow 1$ **to** N **do**
      $\mathbf{x} \leftarrow$ select pattern $\mathbf{x}_n$
      $\Delta v_{kj} \leftarrow \Delta v_{kj} - \eta \delta_k z_j$, $\Delta w_{ji} \leftarrow \Delta w_{ji} - \eta \delta_j x_i$
    $v_{kj} \leftarrow v_{kj} + \Delta v_{kj}$
    $w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$
    epoch $\leftarrow$ epoch $+ 1$
  until $(epoch = max.epoch)$ *or* $(\|\nabla E\| < \epsilon)$
  return $\mathbf{w}, \mathbf{v}$
end

Multi-Layer Networks & Heaviside Step Func.



Possible decision boundaries which can be generated by networks having various numbers of layers and using Heaviside activation function.

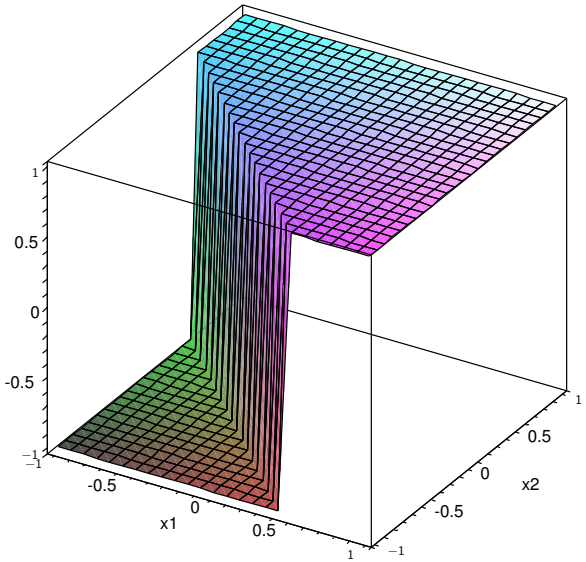
Multi-Layer NN for XOR Separability Problem

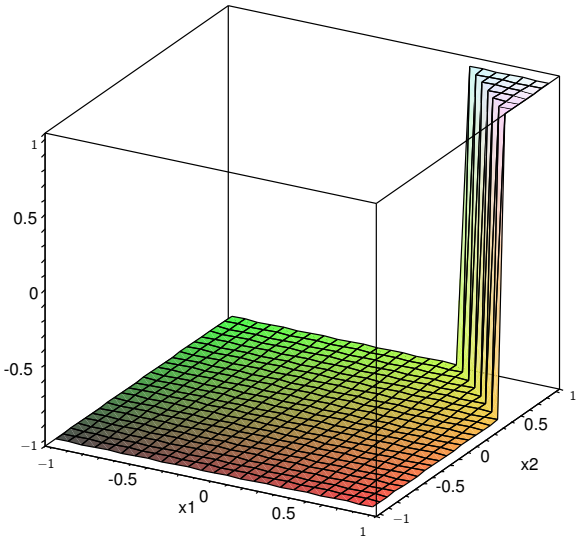


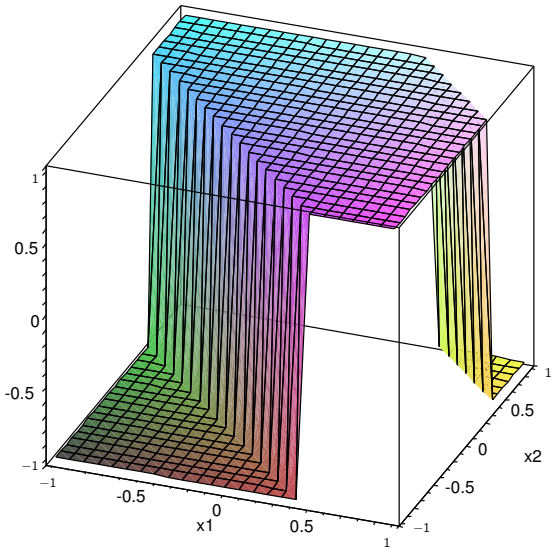
x_1	x_2	x_1 XOR x_2
-1	-1	-1
-1	$+1$	$+1$
$+1$	-1	$+1$
$+1$	$+1$	-1

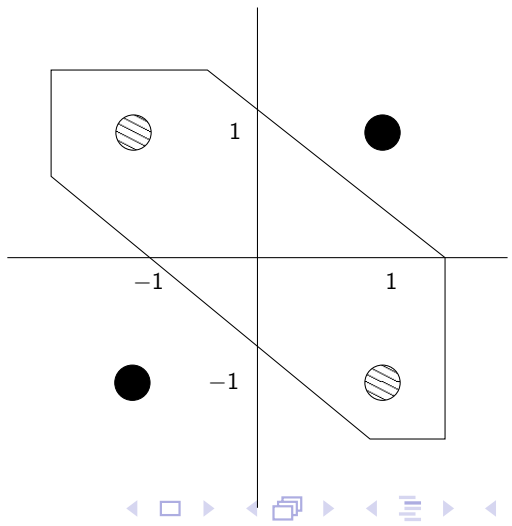
$$g(a) = \begin{cases} -1 & \text{if } a < 0 \\ +1 & \text{if } a \geq 0 \end{cases}$$

Multi-Layer NN for XOR Sep. Problem (cont.)









Expressive Power of Multi-Layer Networks

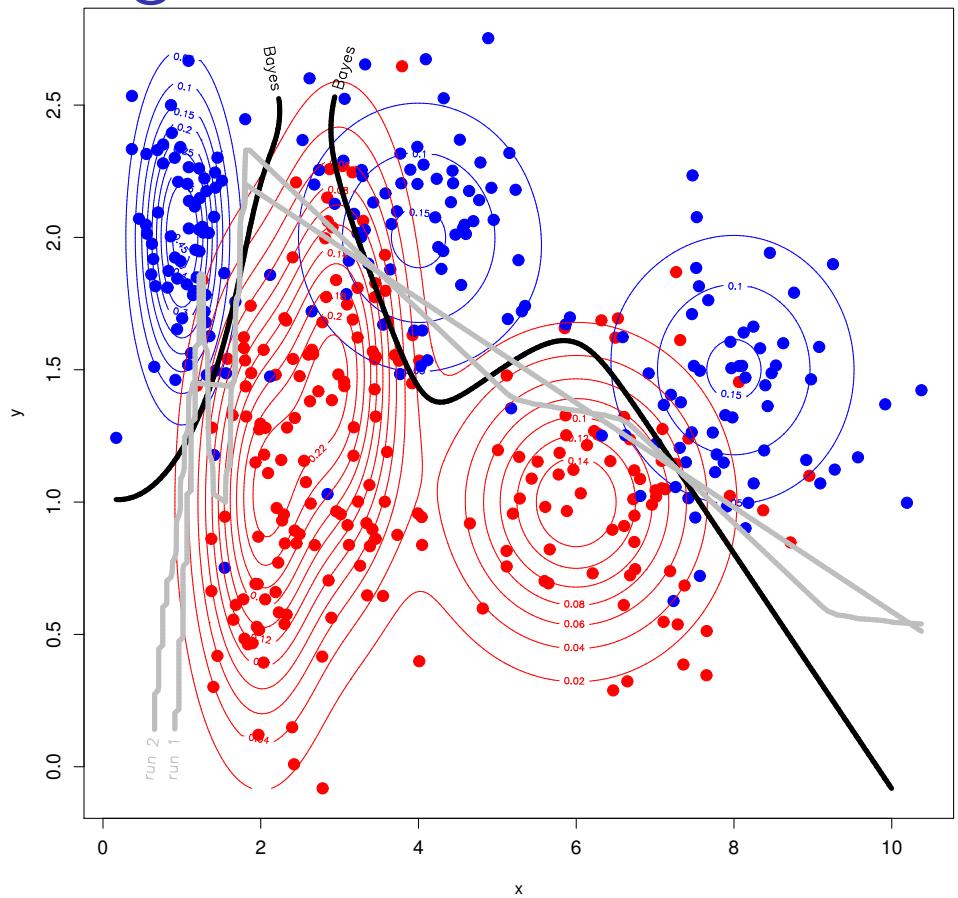
With a two-layer network and a sufficient number of hidden units, *any* type of function can be represented when given proper nonlinearities and weights.

The famous mathematician Andrey Kolmogorov proved that any continuous function $y(\mathbf{x})$ defined on the unit hypercube $[0,1]^n, n \geq 2$ can be represented in the form

$$y(\mathbf{x}) = \sum_{j=1}^{2n+1} \Xi_j \left(\sum_{i=1}^{d} \Psi_{ij}(x_i) \right)$$

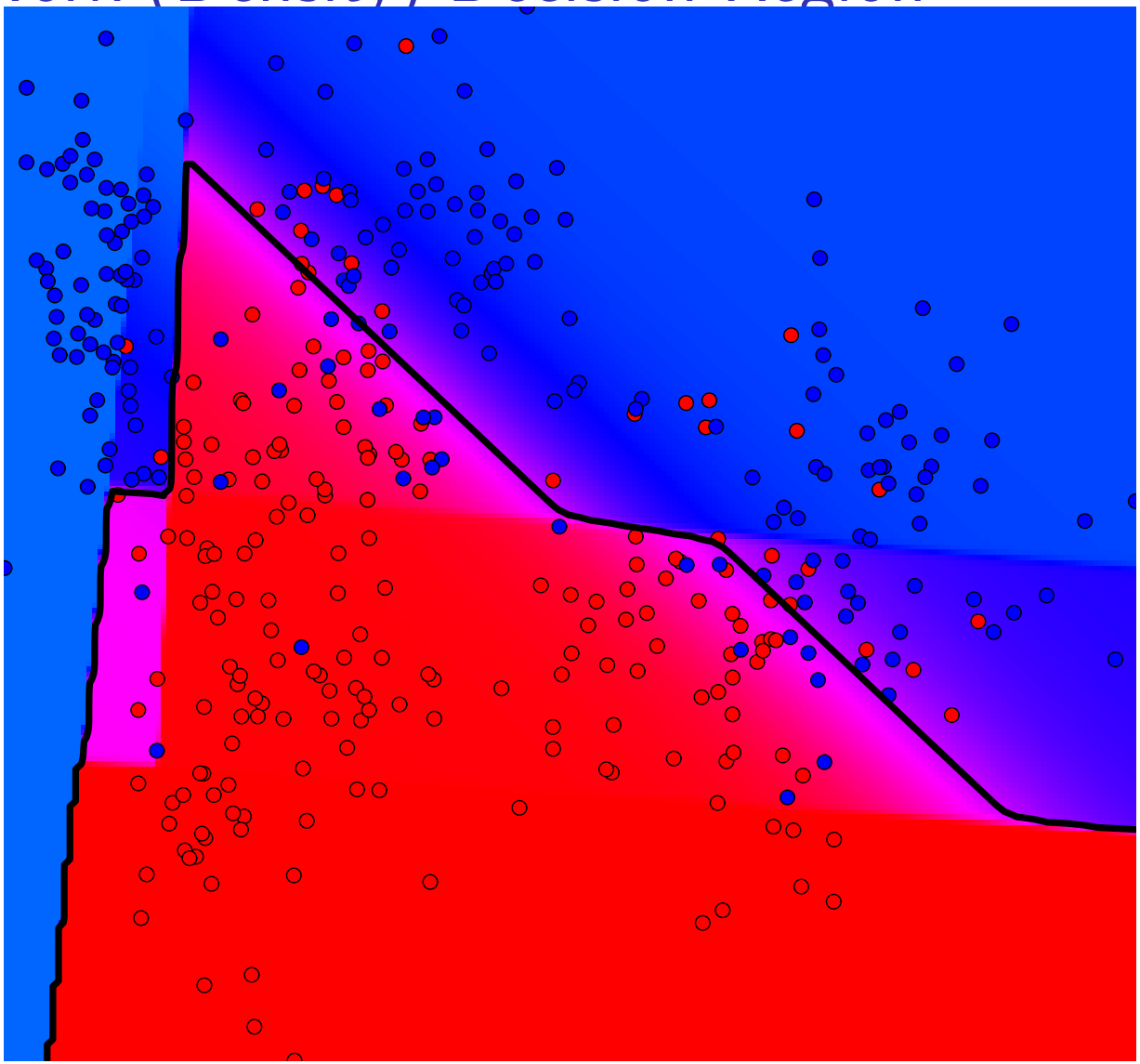
for properly chosen Ξ_j and Ψ_{ij}.

Bayes Decision Region vs. Neural Network



Points from blue and red class are generated by a mixture of Gaussians. Black curve shows optimal separation in a Bayes sense. Gray curve shows neural network separation of two independent backpropagation learning runs.

Neural Network (Density) Decision Region



Overfitting/Underfitting & Generalization

Consider the problem of polynomial curve fitting where we shall fit the data using a polynomial function of the form:

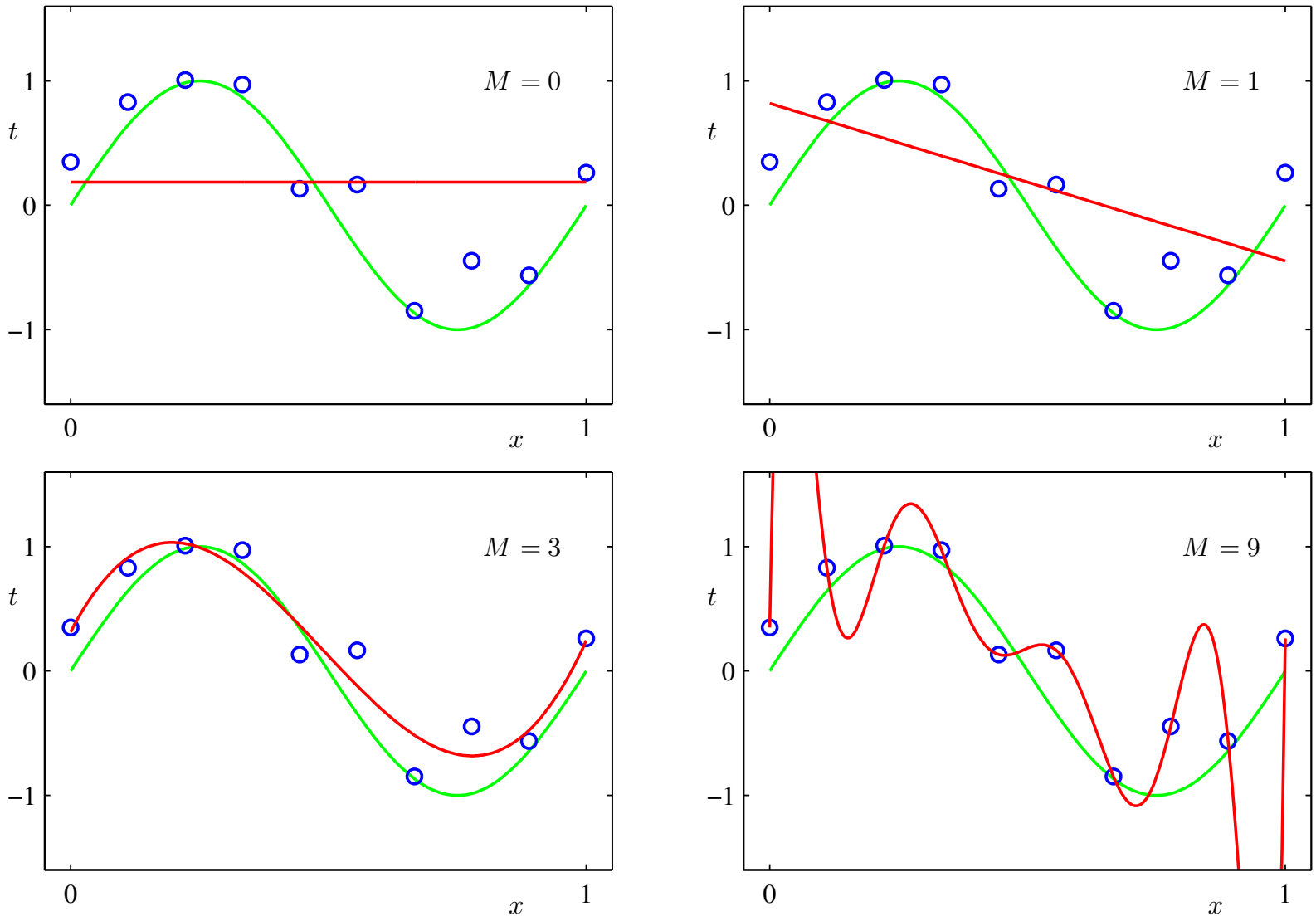
$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j.$$

We measure the misfit of our predictive function $y(x, \mathbf{w})$ by means of error function which we like to minimize:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^{N} (y(x_i, \mathbf{w}) - t_i)^2$$

where t_i is the corresponding target value in the given training data set.

Polynomial Curve Fitting



Polynomial Curve Fitting (cont.)

	$M=0$	$M=1$	$M=3$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

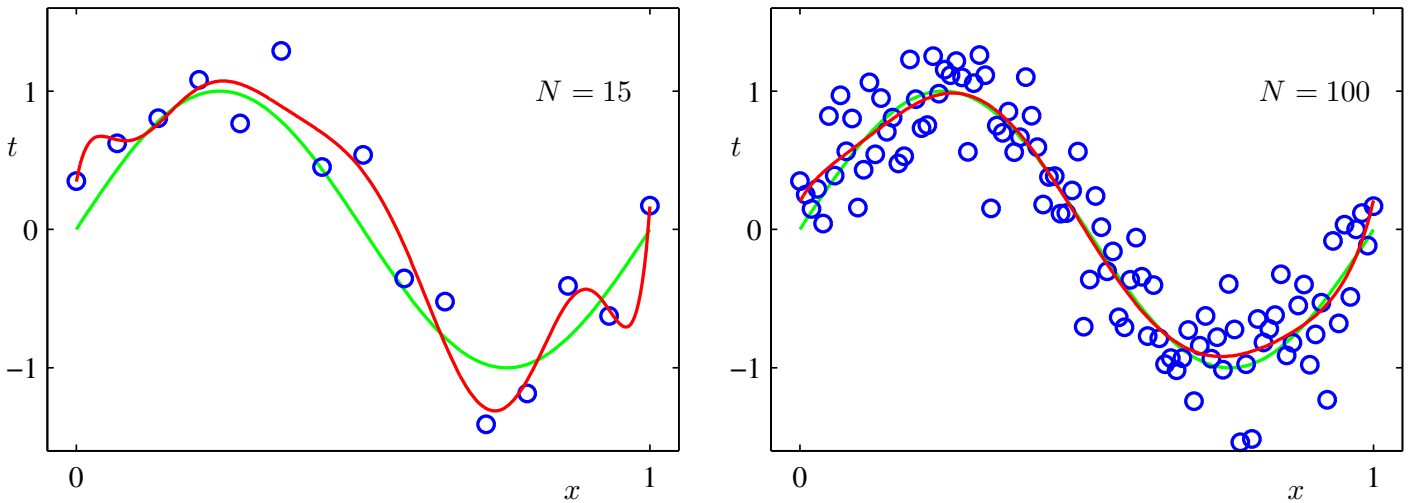
Table: Coefficients $\mathbf{w}^\star$ obtained from polynomials of various order. Observe the dramatically increase as the order of the polynomial increases (this table is taken from Bishop's book).

Polynomial Curve Fitting (cont.)

Observe:

- if M is too small then the model underfits the data
- if M is too large then the model overfits the data

If M is too large then the model is more flexible and is becoming increasingly tuned to random noise on the target values. It is interesting to note that the overfitting problem become less severe as the size of the data set increases.



Polynomial Curve Fitting (cont.)

One technique that can be used to control the overfitting phenomenon is the *regularization*.

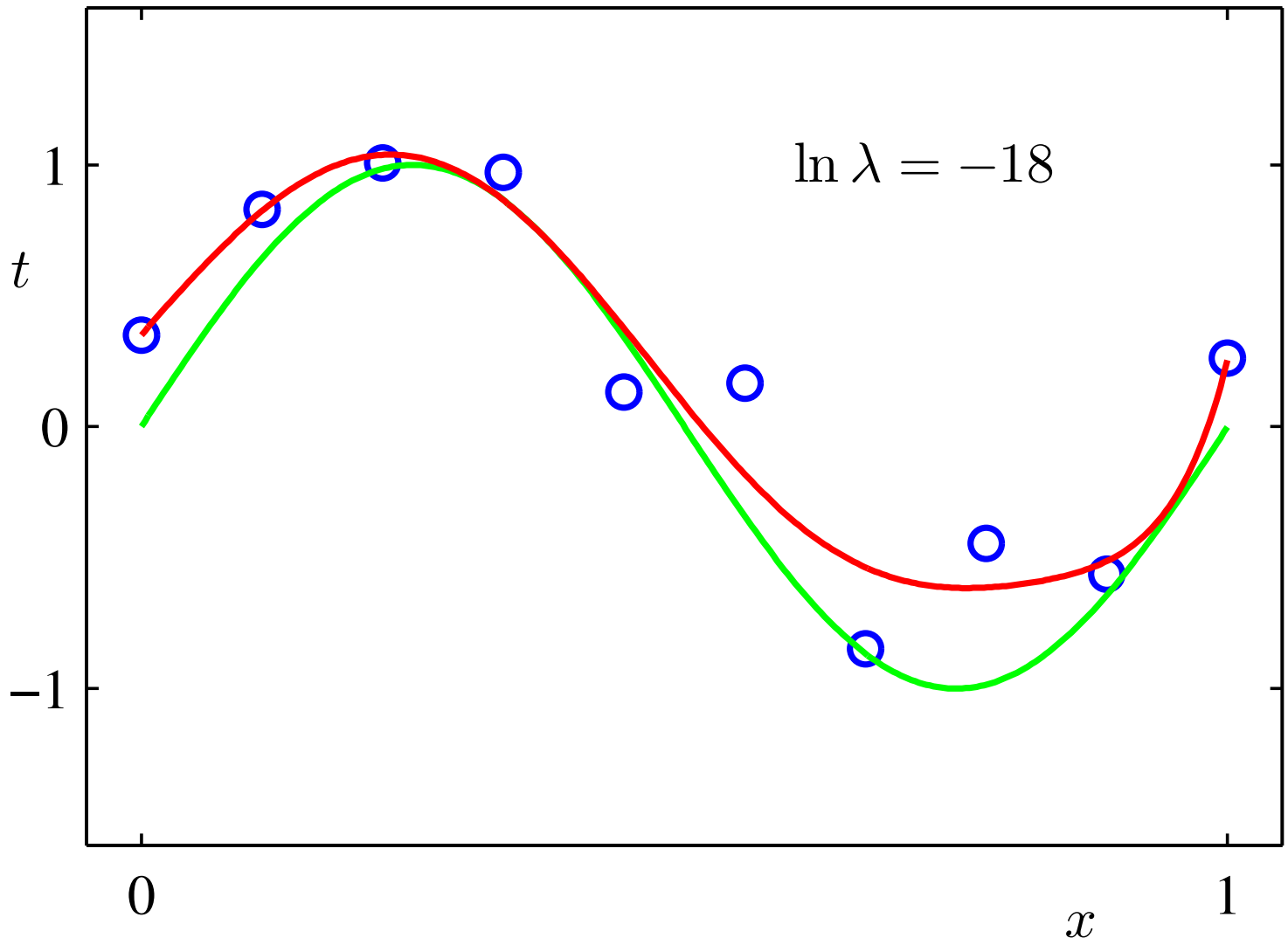
- Regularization involves adding a penalty term to the error function in order to discourage the coefficients from reaching large values.

The modified error function has the form:

$$\widehat{E}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(y(x_i,\mathbf{w}) - t_i)^2 + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}.$$

By means of the penalty term one reduces the value of the coefficients (shrinkage method).

Regularized Polynomial Curve Fitting $M = 9$



Regularization in Neural Networks

- Number of input/output units is generally determined by the dimensionality of the data set.
- Number of hidden units M is free parameter that can be adjusted to obtain best predictive performance.
- Generalization error is not a simple function of M due to the presence of local minima in the error function.
- One straightforward way to deal with this problem is to increase stepwise the value of M and to choose the specific solution having the smallest test error.

Regularization in Neural Networks (cont.)

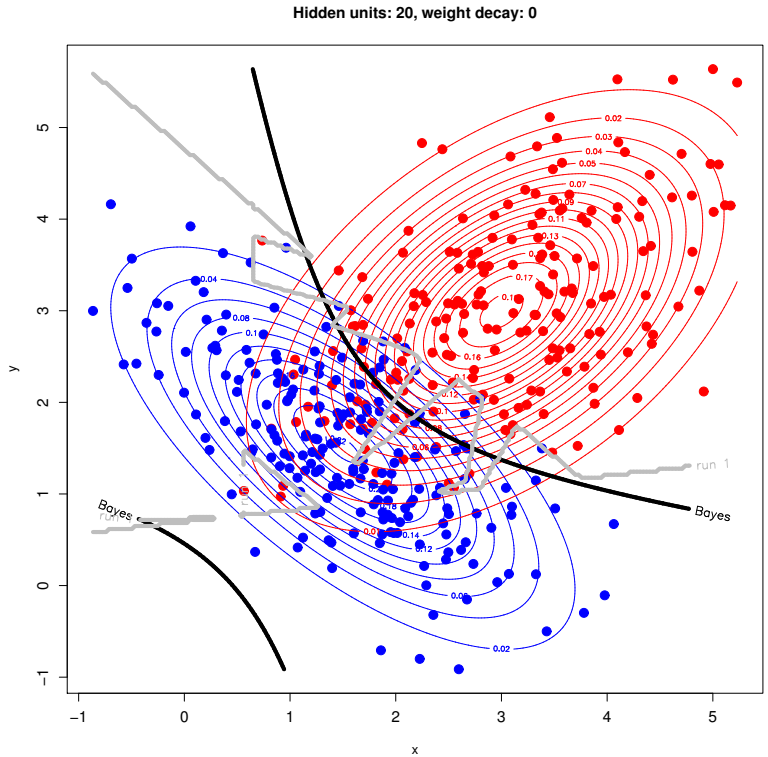
Equivalent to the regularized curve fitting approach, we can choose a relatively large value for M and control the complexity by the addition of a regularized term to the error function.

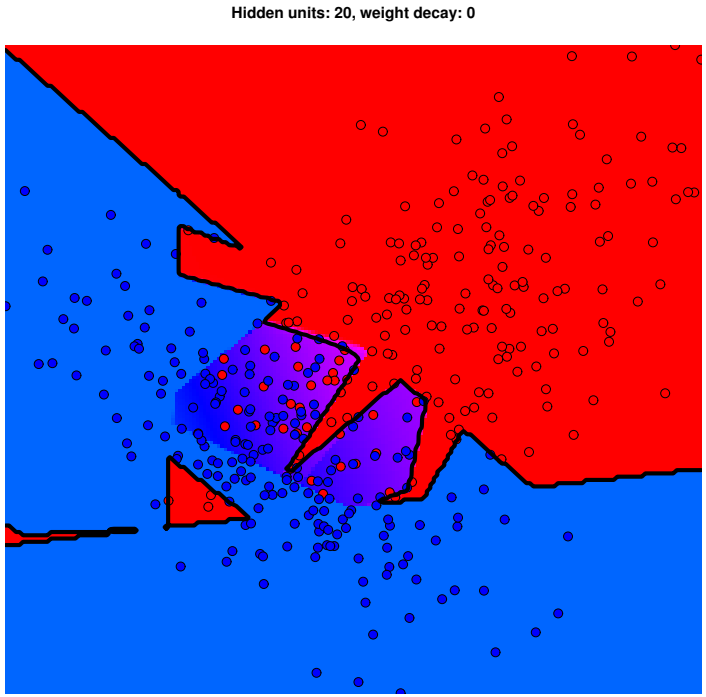
$$\widehat{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

This form of regularization in neural networks is known as *weight decay*.

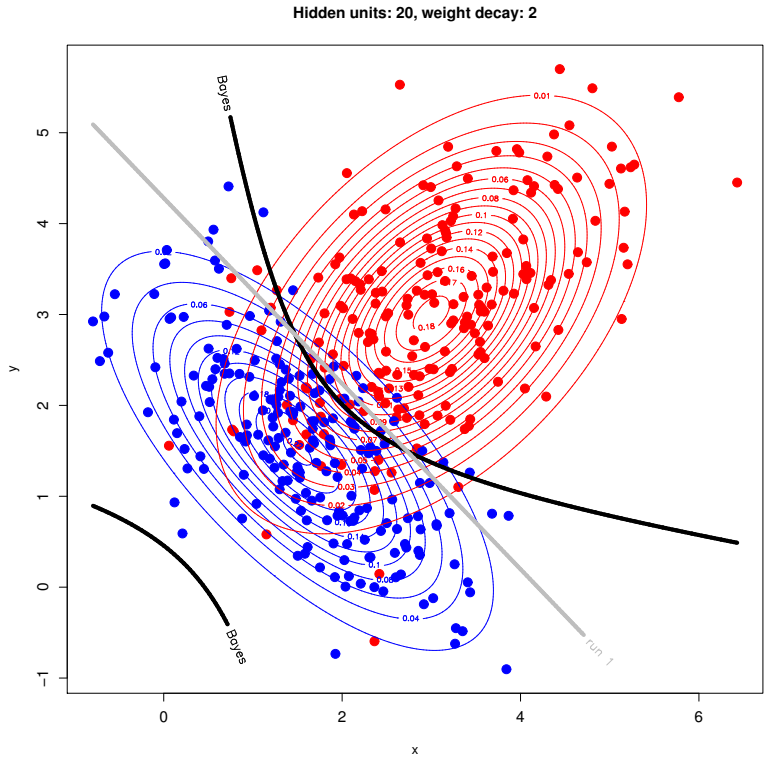
- Weight decay encourages weight values to decay towards zero, unless supported by the data.
- It can be considered as an example of a parameter shrinkage method because parameter values are shrunk towards zero.
- It can be also interpreted as the removal of non-useful connections during training.

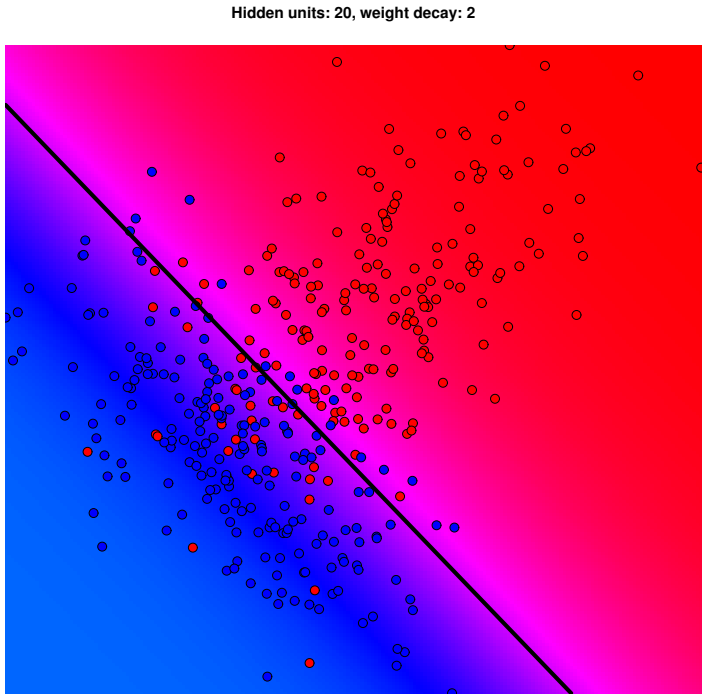
A too Overfitted Neural Network Model



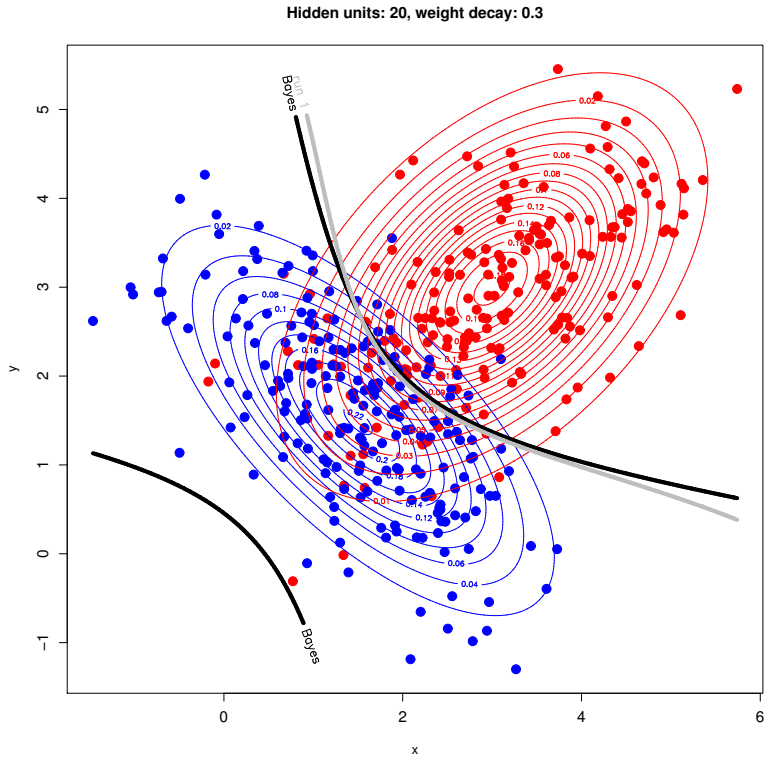


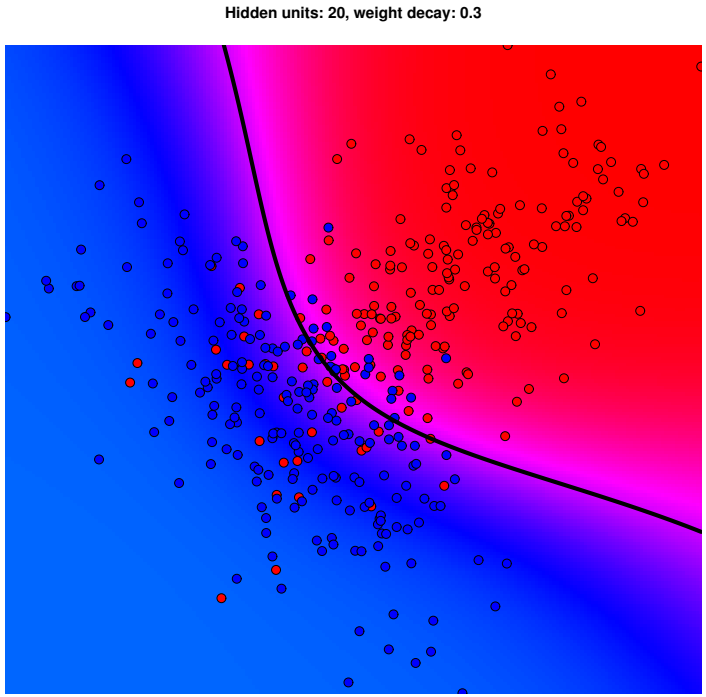
A too Underfitted Neural Network Model





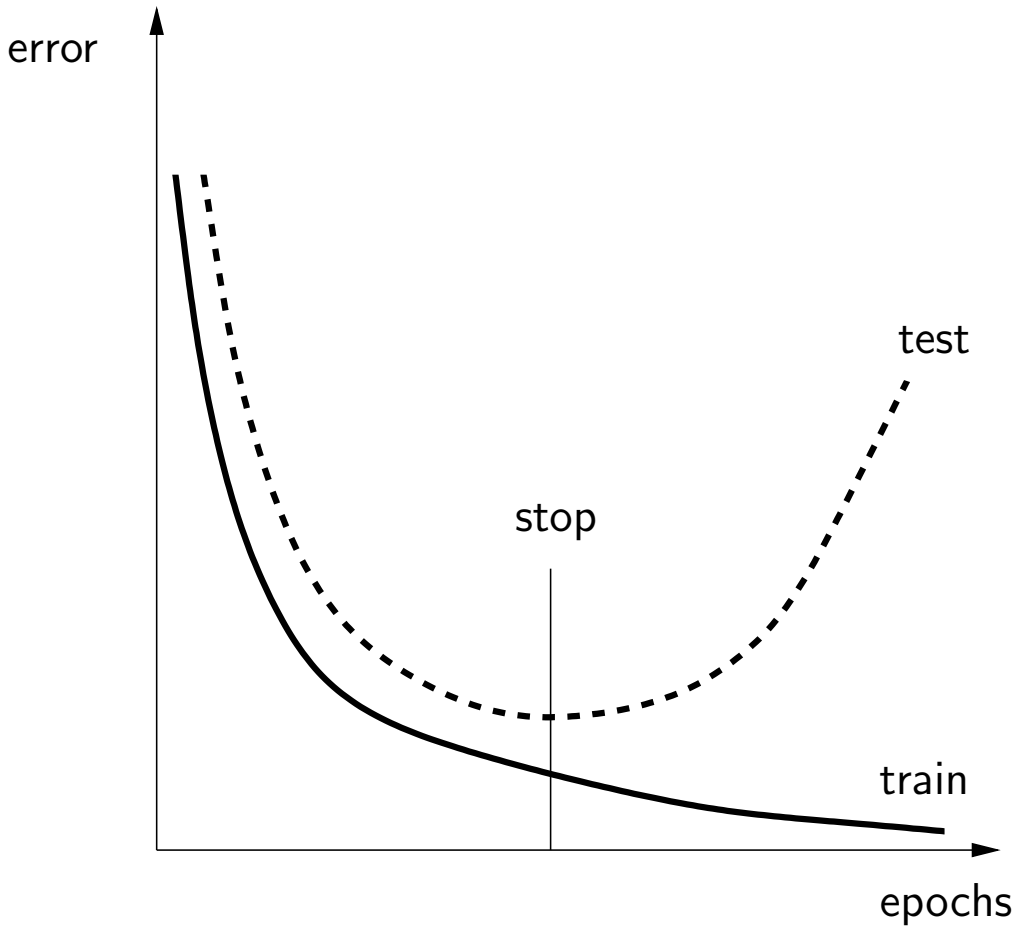
Model Complexity is Properly Penalized



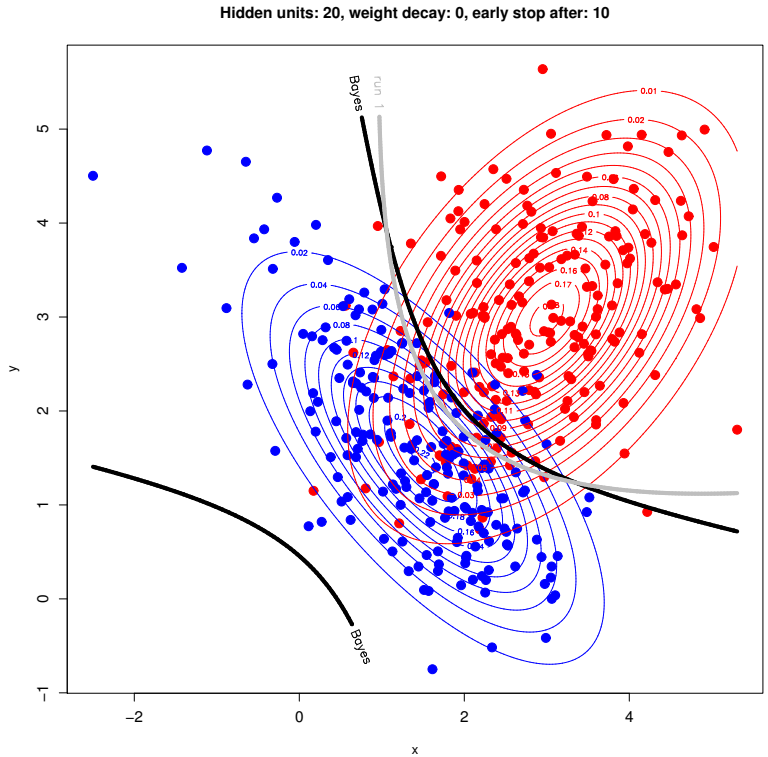


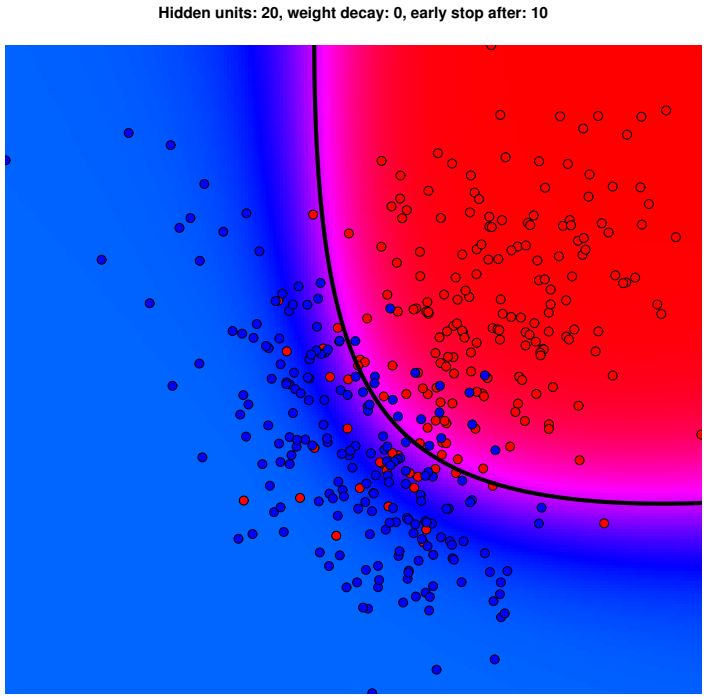
Regularization by Early Stopping

- Another alternative of regularization as a way of controlling the effective complexity of a network is the procedure of *early stopping*.

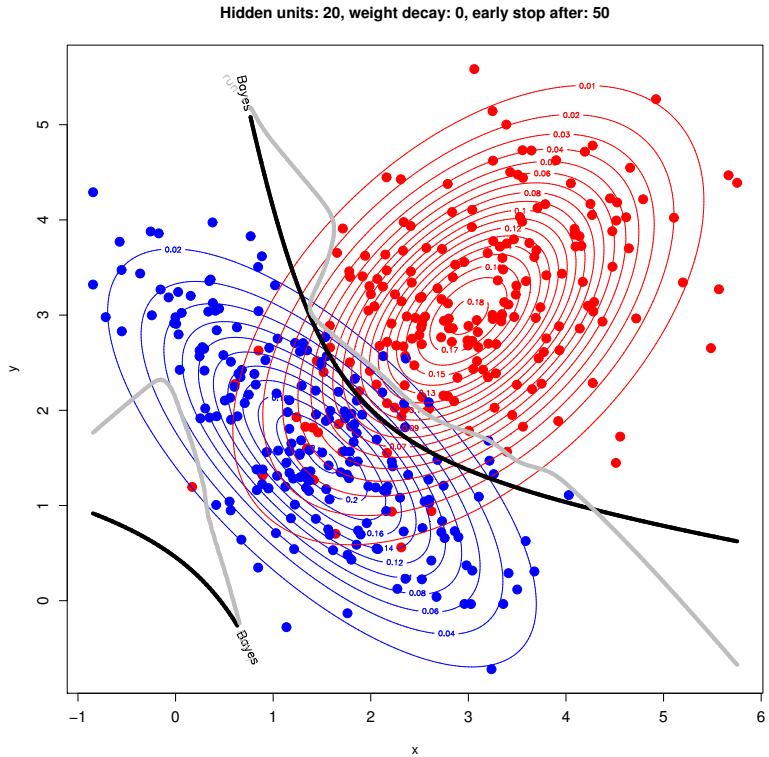


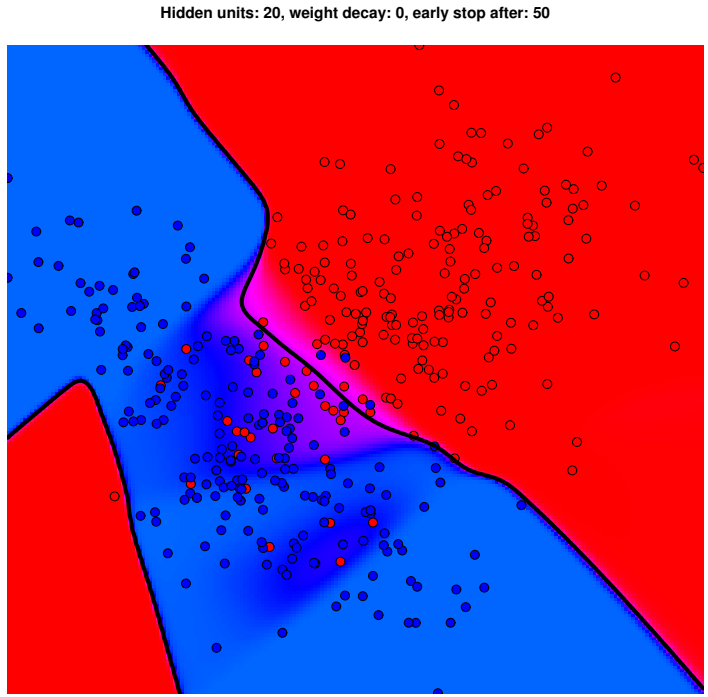
Example Early Stopping after 10 Epochs



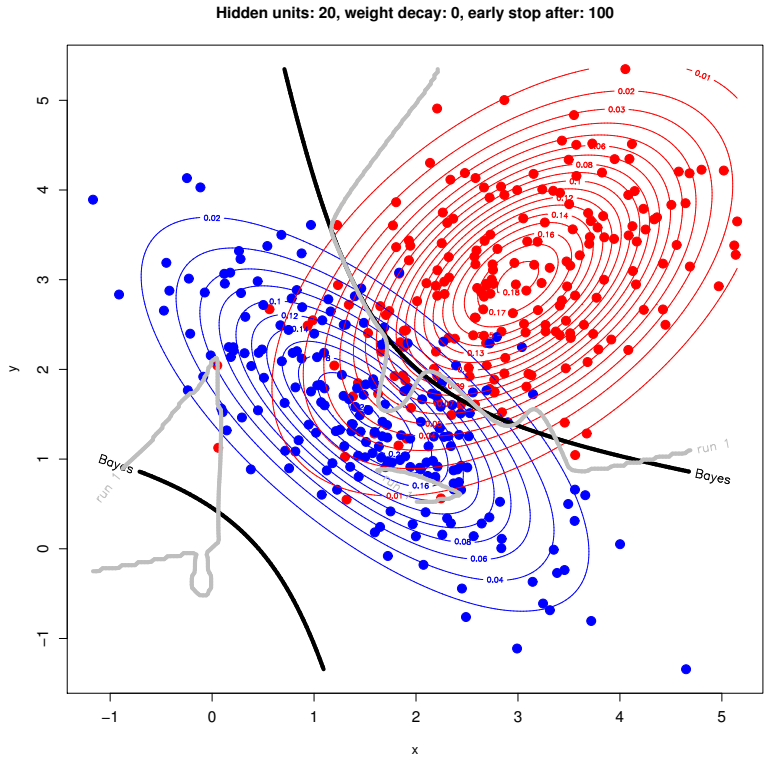


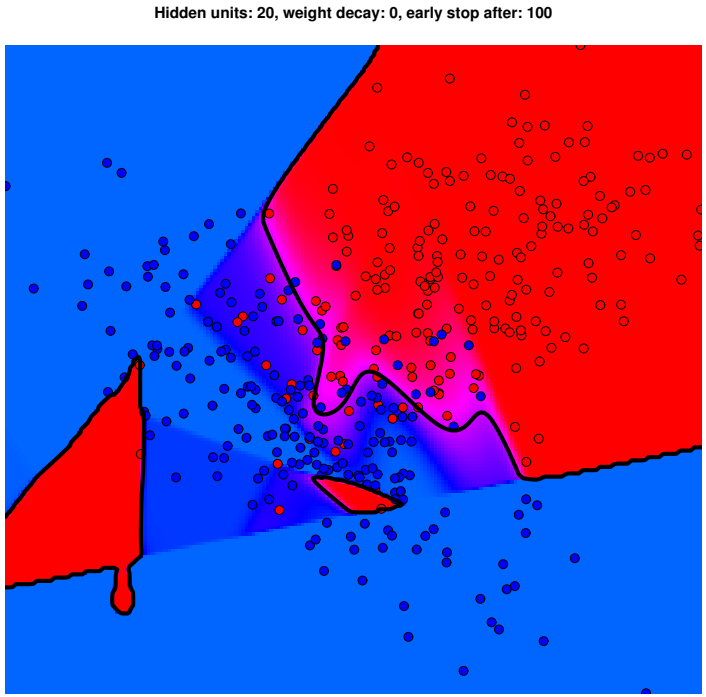
Example Early Stopping after 50 Epochs





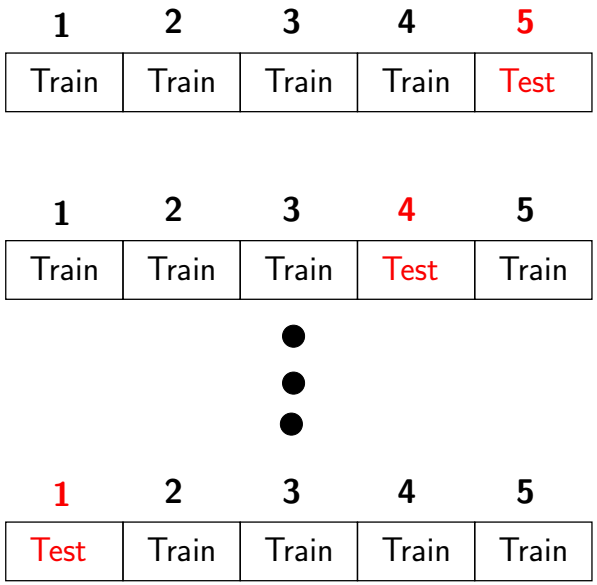
Example Early Stopping after 100 Epochs





Cross-Validation & Prediction Error

Most commonly used method for estimating the prediction error (generalization error) on new data is *cross-validation*.



Split training data into K roughly equal-sized parts (e.g. $K = 5$). Learn neural network on $K - 1$ training parts and predict the not seen testing part. Perform this for $k = 1, 2, \ldots, K$ and combine the K estimates of prediction error (test error).

Cross-Validation & Prediction Error (cont.)

Typical choices of K are 5 or 10. If training data set consists of N data points and $K = N$, then one obtains the *leave-one-out* cross-validation method.

```
# number of folds
folds <- 10
samps <- sample(rep(1:folds, length=n), n, replace=FALSE)

for (fold.iter in 1:folds) {

  train <- dataset[samps!=fold.iter,] # fit the model
  test  <- dataset[samps==fold.iter,] # predict

  # perform here neural network training on set train
  # perform here neural network prediction on set test
}
```

Bias-Variance Dilemma

- Idea: decompose generalization error into the two terms (bias,variance).
- Let us consider our curve fitting problem again. Let $h(\mathbf{x})$ be the true (but unknown) function with continuous valued output with noise.
- We want to estimate $h(\cdot)$ based on training data sets $\mathcal{D}$ each of size N.
- The natural way to measure the effectiveness of the estimator is to use the mean-square deviation from the desired optimal.
- Averaging over all training data sets $\mathcal{D}$ one gets the decomposition:

$$\mathbb{E}_{\mathcal{D}}\left[\{y(\mathbf{x},\mathcal{D}) - h(\mathbf{x})\}^2\right]$$
$$= \underbrace{\{\mathbb{E}_{\mathcal{D}}[y(\mathbf{x},\mathcal{D})] - h(\mathbf{x})\}^2}_{\text{bias}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\left[\{y(\mathbf{x},\mathcal{D}) - \mathbb{E}_{\mathcal{D}}[y(\mathbf{x},\mathcal{D})]\}^2\right]}_{\text{variance}}$$

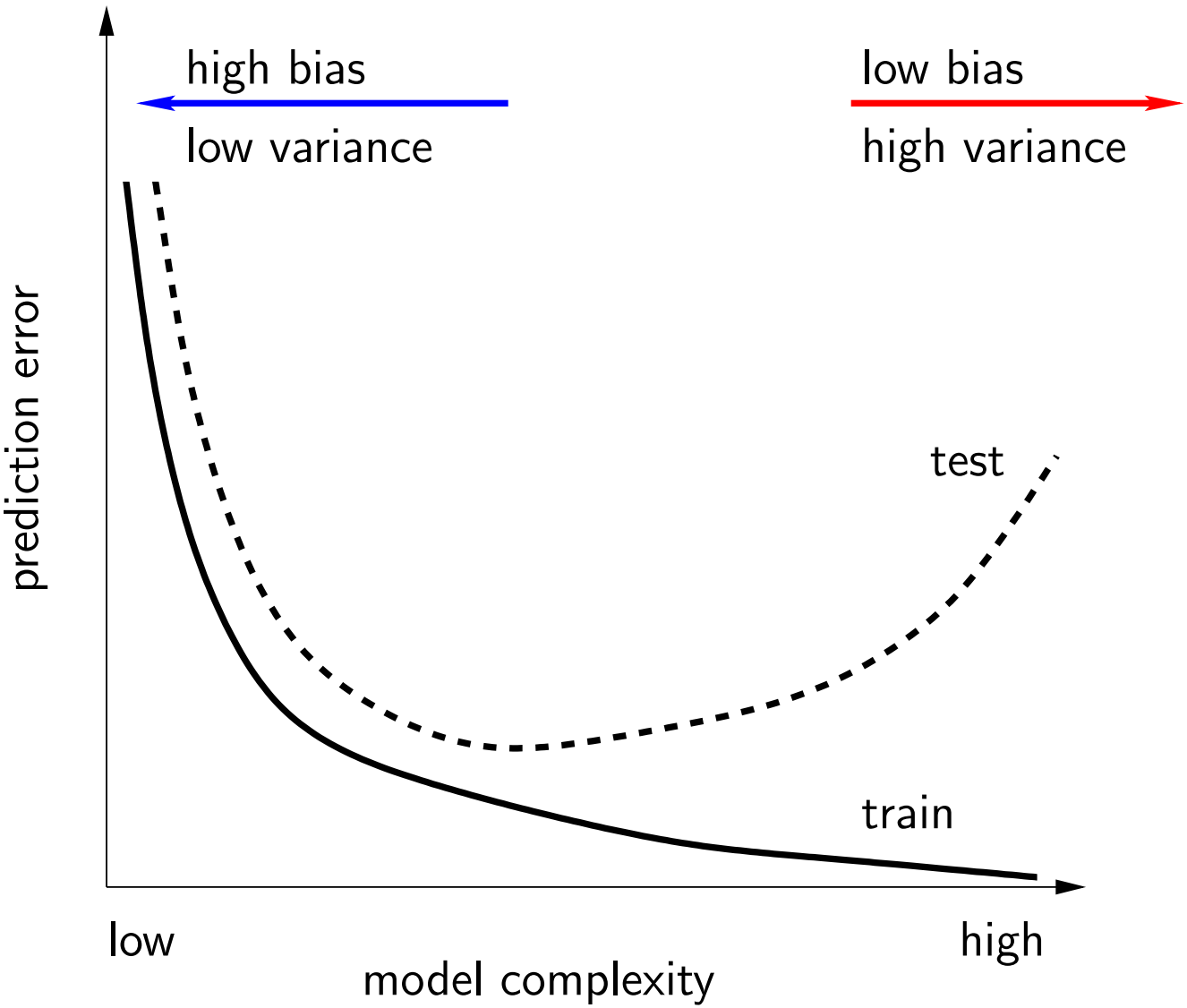
Bias-Variance Dilemma (cont.)

The bias-variance dilemma is general phenomenon and also occurs as the under/overfitting problem in neural networks. In the context of neural networks:

- Bias is a measure of how much the network output, averaged over all possible data sets differs from the desired function.
- Variance is a measure of how much the network output varies between data sets.

In early stage of training, the bias is large because the network is far from the desired function. If the network is trained too long, then the network will also have learned the noise specific in the data set.

Bias-Variance Dilemma (cont.)



Momentum

- Gradient descent can be very slow if η is too small, and can oscillate widely if η is too large.
- Idea: use fraction of the previous weight change and actual gradient term to control non-radical revisions in the updates.

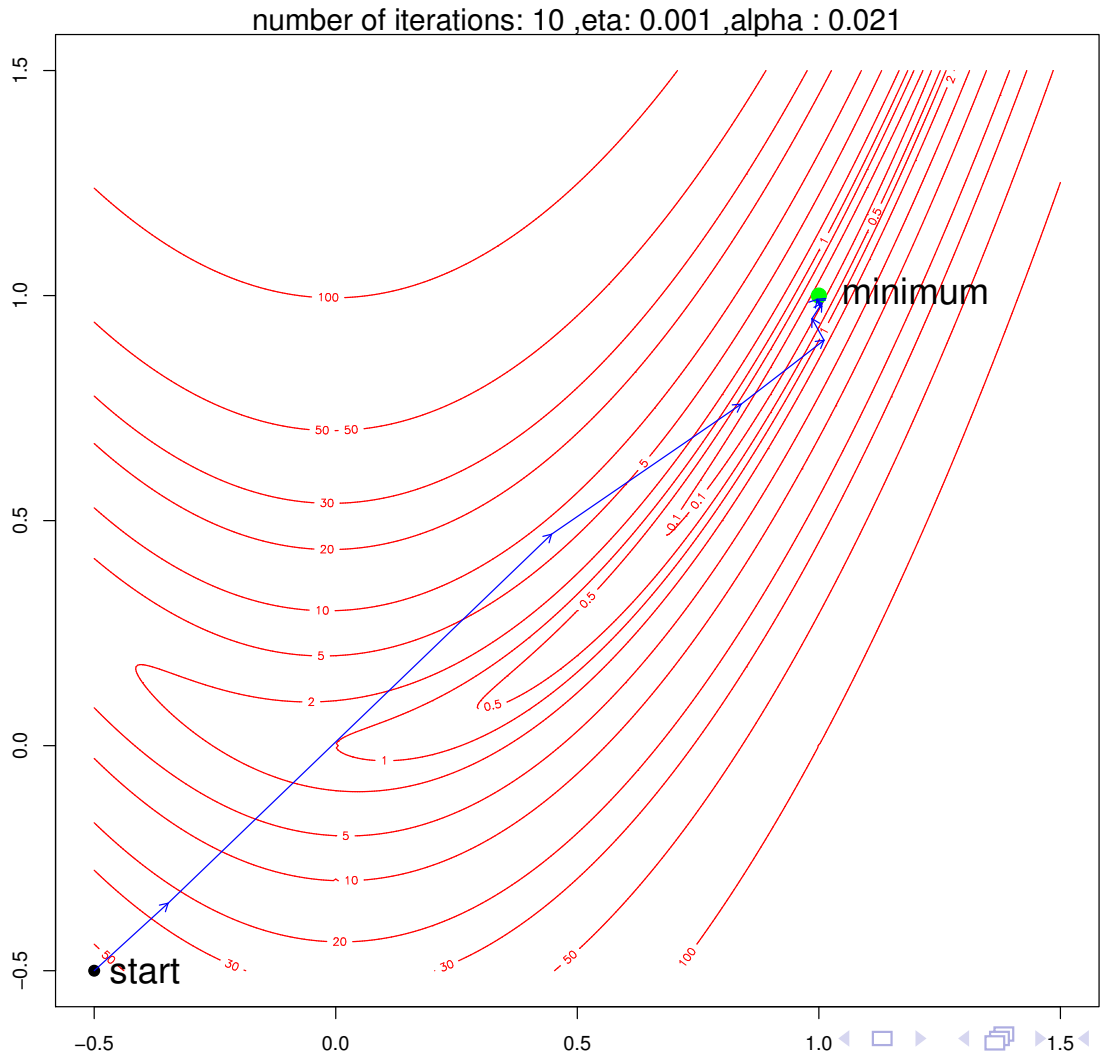
$$\Delta \mathbf{w}^{\text{new}} = -\eta \frac{\partial E}{\partial \mathbf{w}} + \alpha \Delta \mathbf{w}, \qquad 0 \leq \alpha \leq 1.$$

Momentum:

- can cancel side-to-side oscillations across the error valley,
- can cause a faster convergence when weight updates are all in the same direction because the learning rate is amplified.

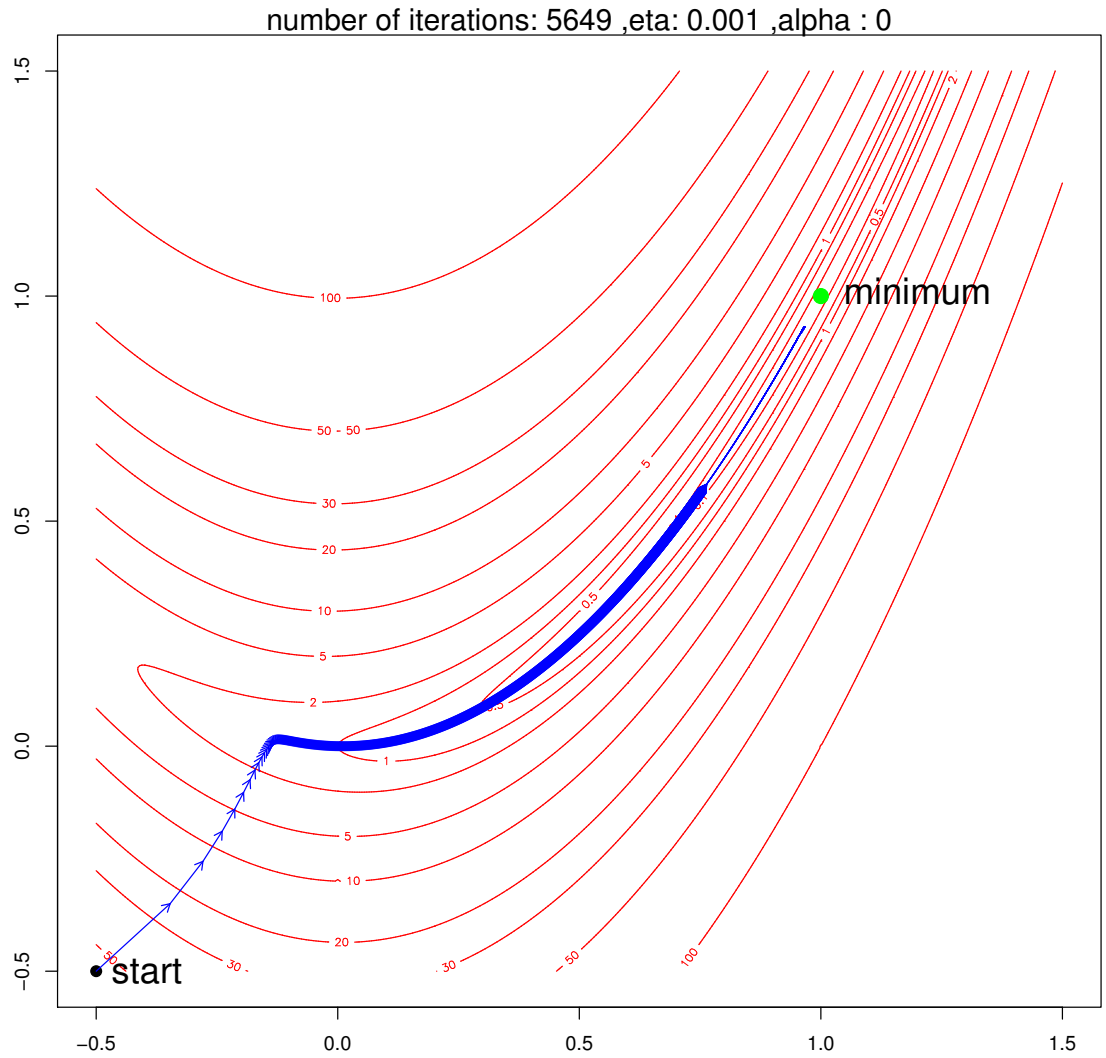
Momentum Example Rosenbrock Function

Rosenbrock function $f(x, y) = (1 - x)^2 + 100(y - x^2)^2$ has global minimum $f(x, y) = (0, 0)$ at $(1, 1)$. Momentum param. $\alpha = 0.021$, learning rate $\eta = 0.001$.



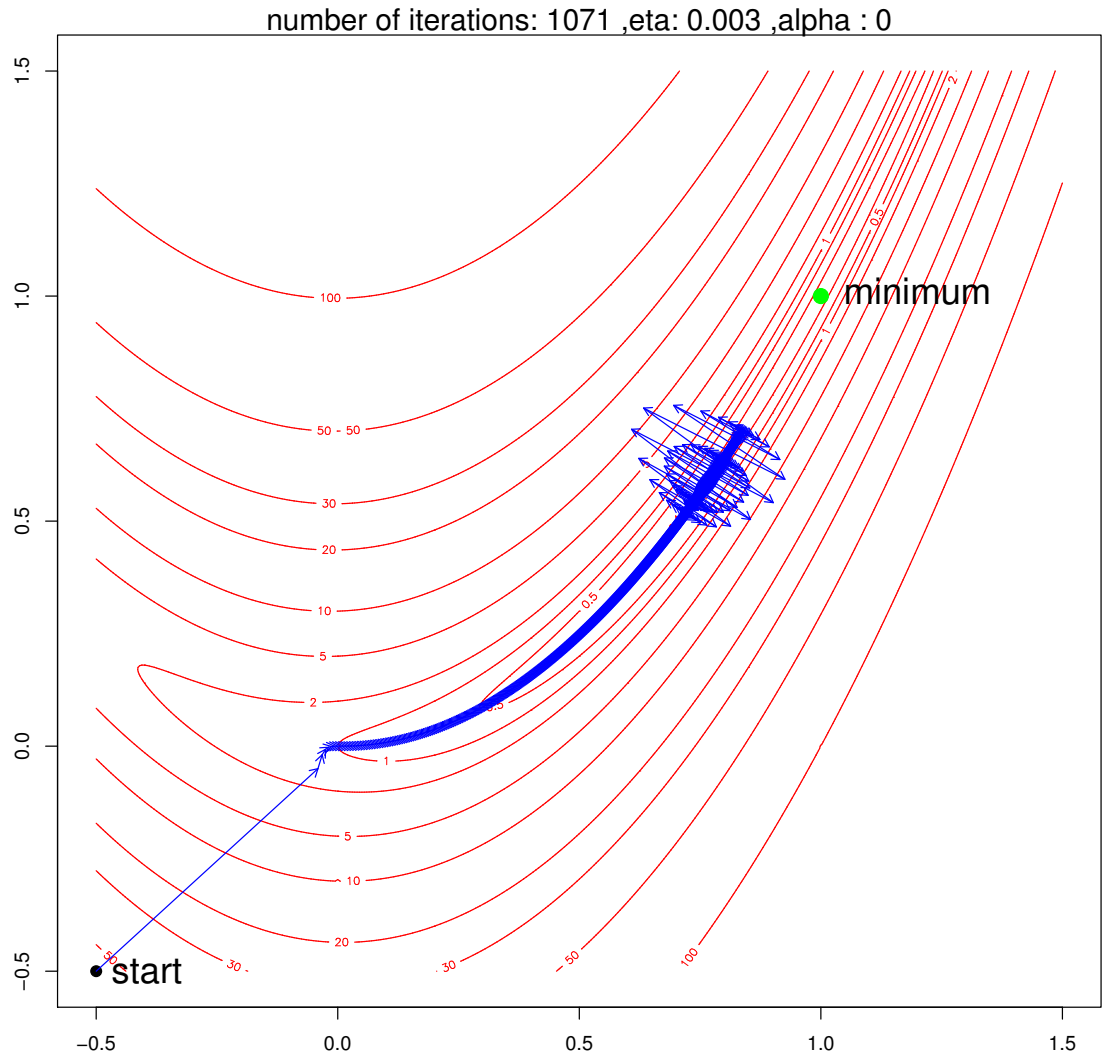
Momentum Example Rosenbrock Func. (cont.)

Setting $\alpha = 0$ (no momentum)



Momentum Example Rosenbrock Func. (cont.)

Setting $\alpha = 0$ (no momentum) and a larger learning rate η



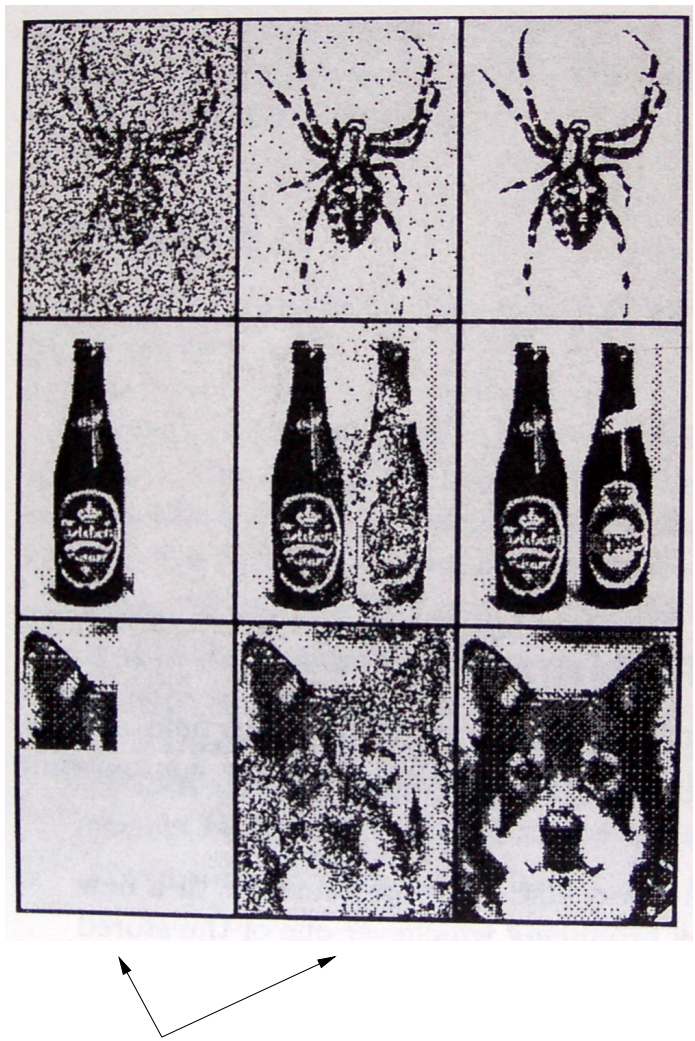
Adaptive Parameter

- Adjust the parameters automatically as learning progress.
- Check if particular weight update decreased the error function.
  - If not we overleaped and η should be reduced.
  - If several steps in a row have decreased the error, we are too conservative and could try increasing η

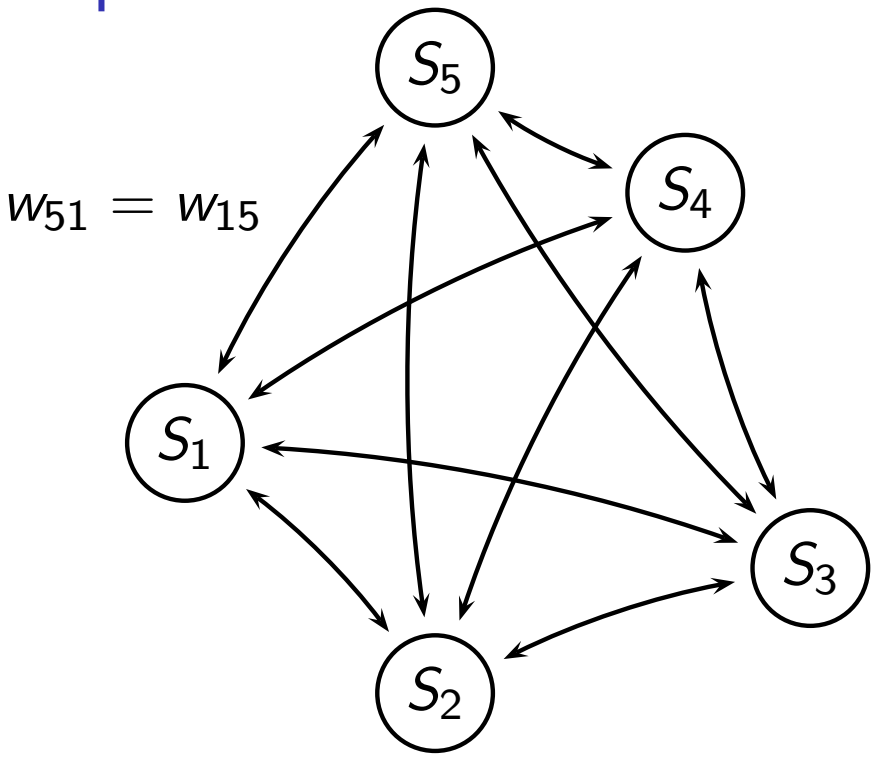
$$\Delta\eta = \begin{cases} +a & \text{if } \Delta E < 0 \\ -b\eta & \text{if } \Delta E > 0 \\ 0 & \text{otherwise} \end{cases} \qquad \text{consistently (e.g. last } K \text{ steps)}$$

Hopfield Network Introductory Example

- Suppose we want to store N binary images in some memory.
- The memory should be *content-addressable* and insensitive to small errors.
- We present corrupted images to the memory (e.g. our brain) and recall the corresponding images.



Hopfield Network



- w_{ij} denotes weight connection from unit j to unit i
- no unit has connection with itself $w_{ii} = 0, \forall i$
- connections are symmetric $w_{ij} = w_{ji}, \forall i, j$

State of unit i can take values ± 1 and is denoted as S_i. State dynamics are governed by activity rule:

$$S_i = \text{sgn}\left(\sum_j w_{ij} S_j\right), \text{ where } \text{sgn}(a) = \begin{cases} +1 & \text{if } a \geq 0, \\ -1 & \text{if } a < 0 \end{cases}$$

Learning Rule in a Hopfield Network

Learning in Hopfield networks:

- Store a set of desired memories $\{\mathbf{x}^{(n)}\}$ in the network, where each memory is a binary pattern with $x_i \in \{-1, +1\}$.
- The weights are set using the sum of outer products

$$w_{ij} = \frac{1}{N} \sum_n x_i^{(n)} x_j^{(n)},$$

where N denotes the number of units (N can also be some positive constant, e.g. number of patterns). Given a $m \times 1$ column vector $\mathbf{a}$ and $1 \times n$ row vector $\mathbf{b}$. The outer product $\mathbf{a} \otimes \mathbf{b}$ (short $\mathbf{a}\,\mathbf{b}$) is defined as the $m \times n$ matrix

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \otimes [b_1\ b_2\ b_3] = \begin{bmatrix} a_1 b_1 & a_1 b_2 & a_1 b_3 \\ a_2 b_1 & a_2 b_2 & a_2 b_3 \\ a_3 b_1 & a_3 b_2 & a_3 b_3 \end{bmatrix}, \quad m = n = 3$$

Learning in Hopfield Network (Example)

Suppose we want to store patterns $\mathbf{x}^{(1)} = [-1, +1, -1]$ and $\mathbf{x}^{(2)} = [+1, -1, +1]$.

$$\begin{bmatrix} -1 \\ +1 \\ -1 \end{bmatrix} \otimes [-1, +1, -1] = \begin{bmatrix} +1 & -1 & +1 \\ -1 & +1 & -1 \\ +1 & -1 & +1 \end{bmatrix}$$

$$+$$

$$\begin{bmatrix} +1 \\ -1 \\ +1 \end{bmatrix} \otimes [+1, -1, +1] = \begin{bmatrix} +1 & -1 & +1 \\ -1 & +1 & -1 \\ +1 & -1 & +1 \end{bmatrix}$$

Learning in Hopfield Network (Example) (cont.)

$$\mathbf{W} = \frac{1}{3}\begin{bmatrix} 0 & -2 & +2 \\ -2 & 0 & -2 \\ +2 & -2 & 0 \end{bmatrix}$$

Recall: no unit has connection with itself.

The storage of patterns in the network can also be interpreted as constructing stable states. The condition for patterns to be stable is:

$$\text{sgn}\left(\sum_j w_{ij} x_i\right) = x_i \,, \forall i.$$

Suppose we present pattern $\mathbf{x}^{(1)}$ to the network and want to restore the corresponding pattern.

Learning in Hopfield Network (Example) (cont.)

Let us assume that the network states are set as follows: $S_i = x_i,\ \forall i$. We can restore pattern $\mathbf{x}^{(1)} = [-1, +1, -1]$ as follows:

$$S_1 = \text{sgn}\left(\sum_{j=1}^{3} w_{1j} S_j\right) = -1 \qquad S_2 = \text{sgn}\left(\sum_{j=1}^{3} w_{2j} S_j\right) = +1$$

$$S_3 = \text{sgn}\left(\sum_{j=1}^{3} w_{3j} S_j\right) = -1$$

Can we also restore the original patterns by presenting "similar" patterns which are corrupted by noise?

Updating States in a Hopfield Network

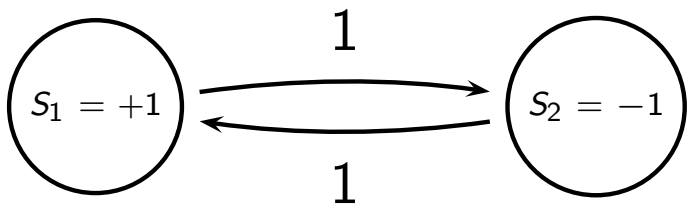
Synchronous updates:

- all units update their states $S_i = \text{sgn}\left(\sum_j w_{ij} S_j\right)$ simultaneously.

Asynchronous updates:

- one unit at a time updates its state. The sequence of selected units may be a fixed sequence or a random sequence.

Synchronously updating states can lead to oscillation (no convergence to a stable state).



Aim of a Hopfield Network

Our aim is that by presenting a corrupted pattern, and by applying iteratively the state update rule the Hopfield network will settle down in a stable state which corresponds to the desired pattern.

Hopfield network is a method for

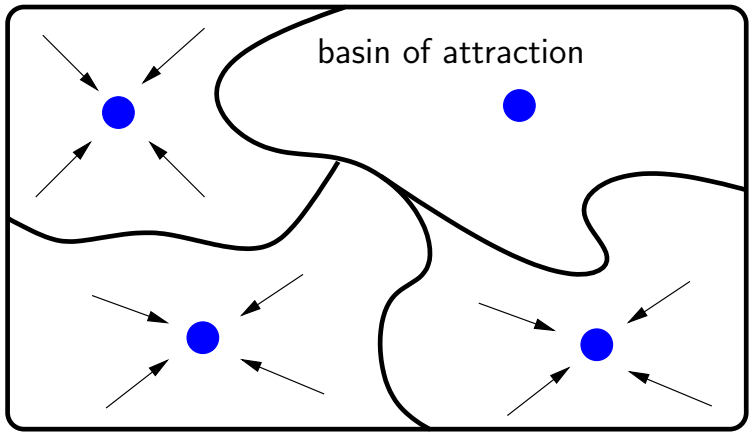
- pattern completion
- error correction.

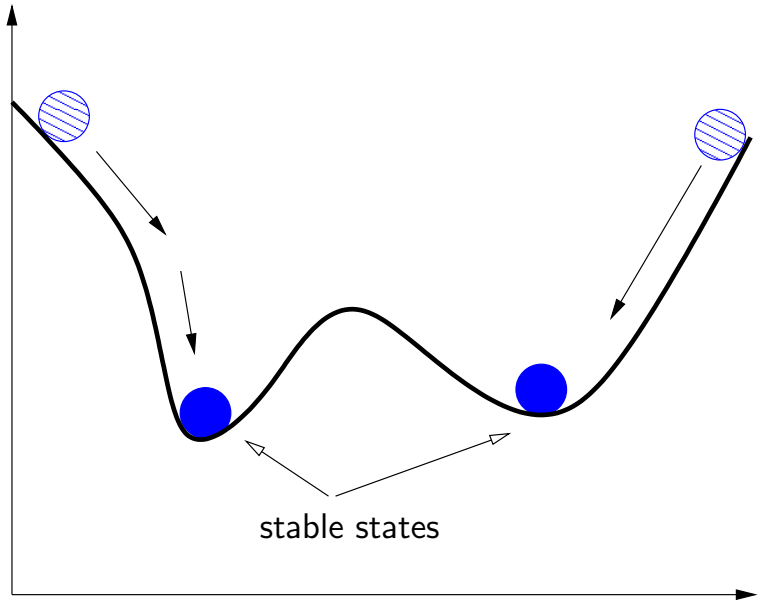
The state of a Hopfield network can be expressed in terms of the energy function

$$E = -\frac{1}{2}\sum_{i,j} w_{ij} S_i S_j$$

Hopfield observed that if a state is a local minimum in the energy function, it is also a stable state for the network.

Basin of Attraction and Stable States

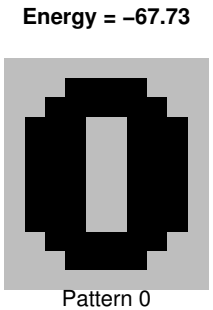


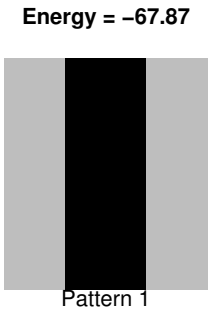


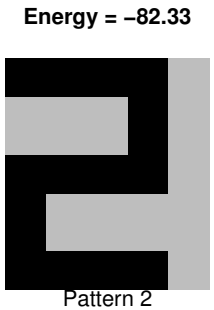
Within the space the stored patterns $\mathbf{x}^{(n)}$ are acting like attractors.

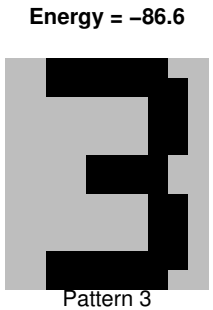
Haykin's Digit Example

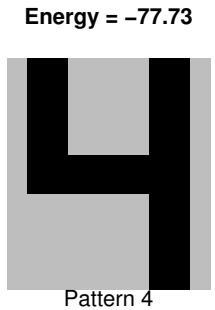
Suppose we stored the following digits in the Hopfield network:

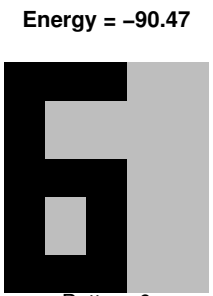


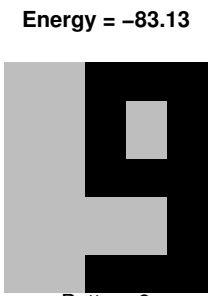


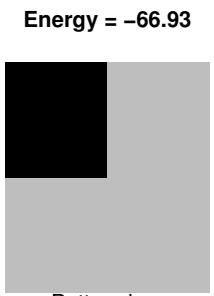




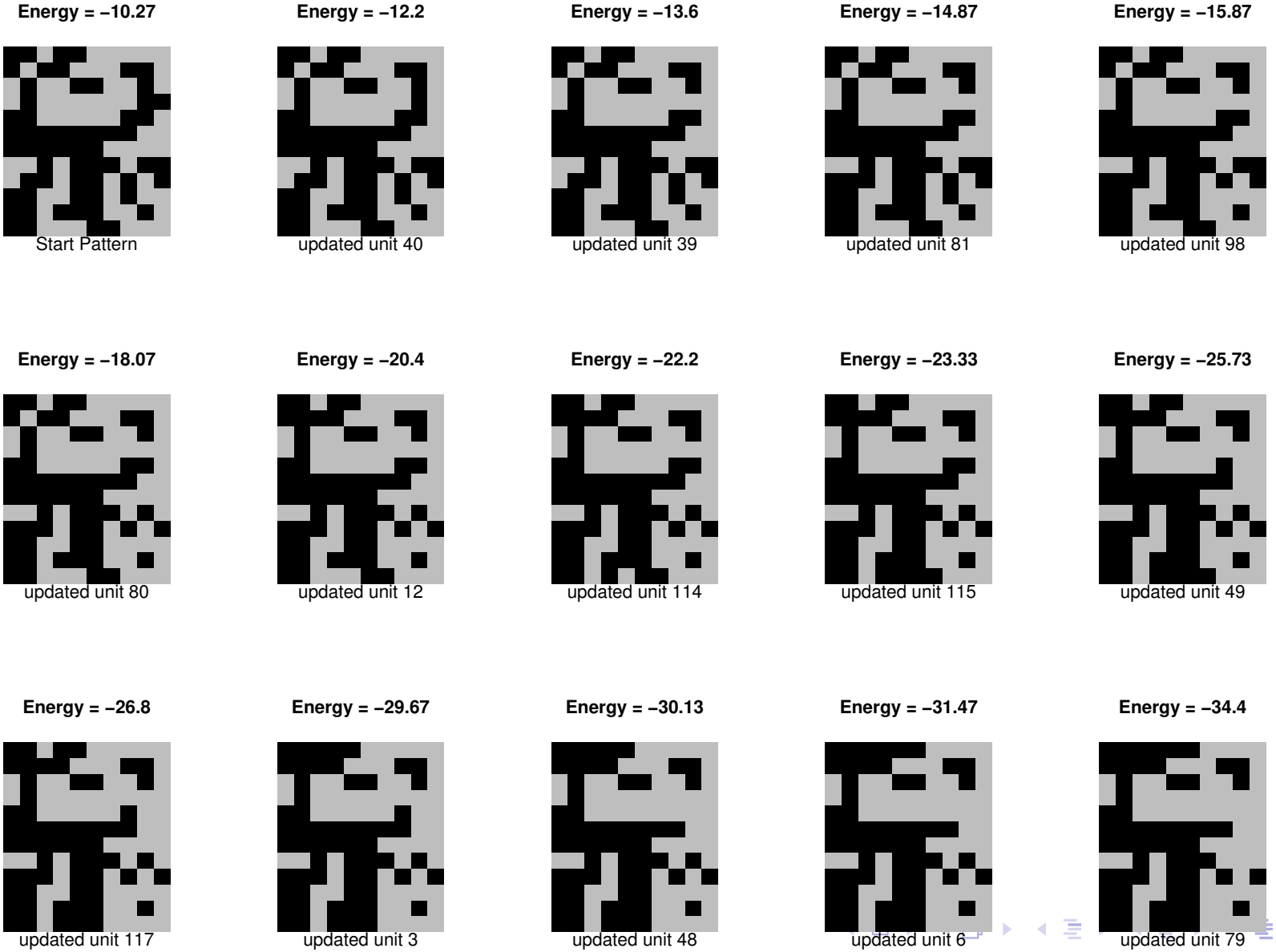






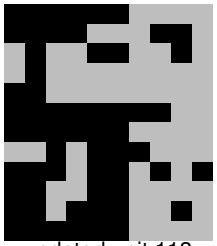


Updated States of Corrupted Digit 6



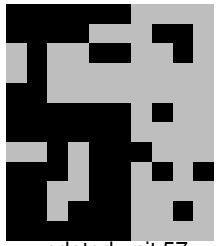
Updated States of Corrupted Digit 6 (cont.)

Energy = –36.73



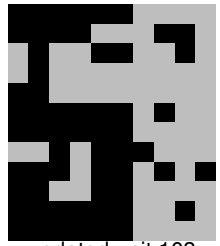
updated unit 113

Energy = –38.4



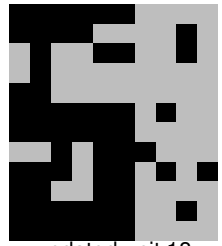
updated unit 57

Energy = –41.07



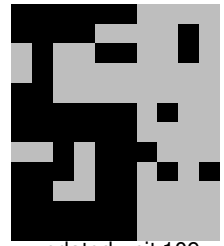
updated unit 103

Energy = –42.4



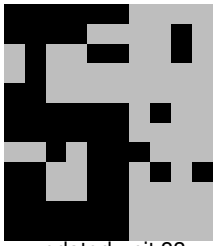
updated unit 18

Energy = –45.27



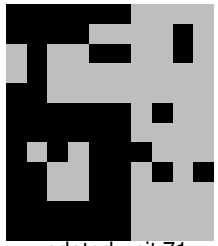
updated unit 109

Energy = –47.6



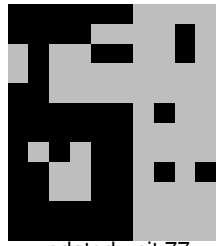
updated unit 83

Energy = –50.4



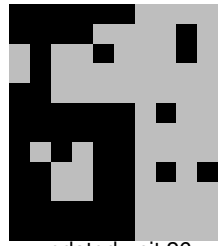
updated unit 71

Energy = –52.67



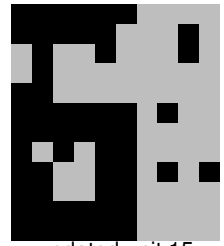
updated unit 77

Energy = –56.47



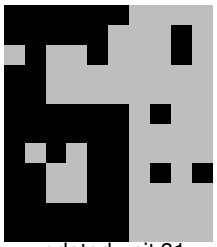
updated unit 26

Energy = –58.4



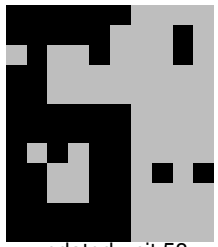
updated unit 15

Energy = –60.67



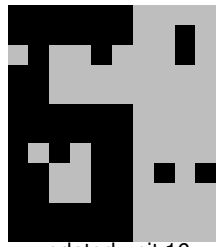
updated unit 31

Energy = –63.33



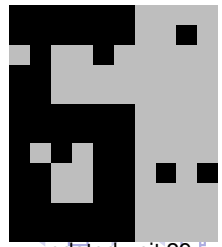
updated unit 58

Energy = –64.47



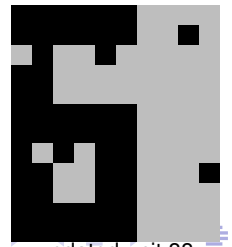
updated unit 16

Energy = –68



updated unit 29

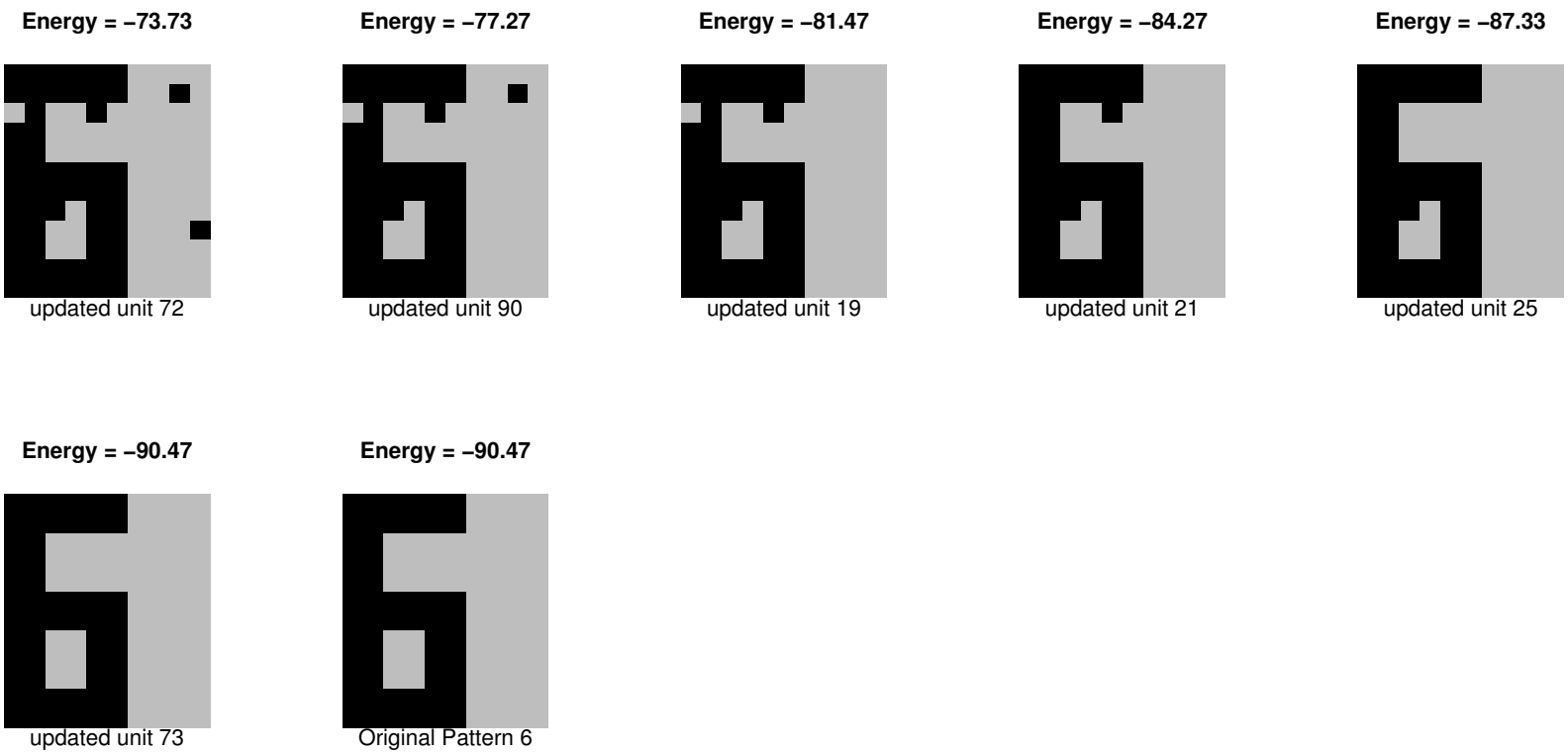
Energy = –71.27



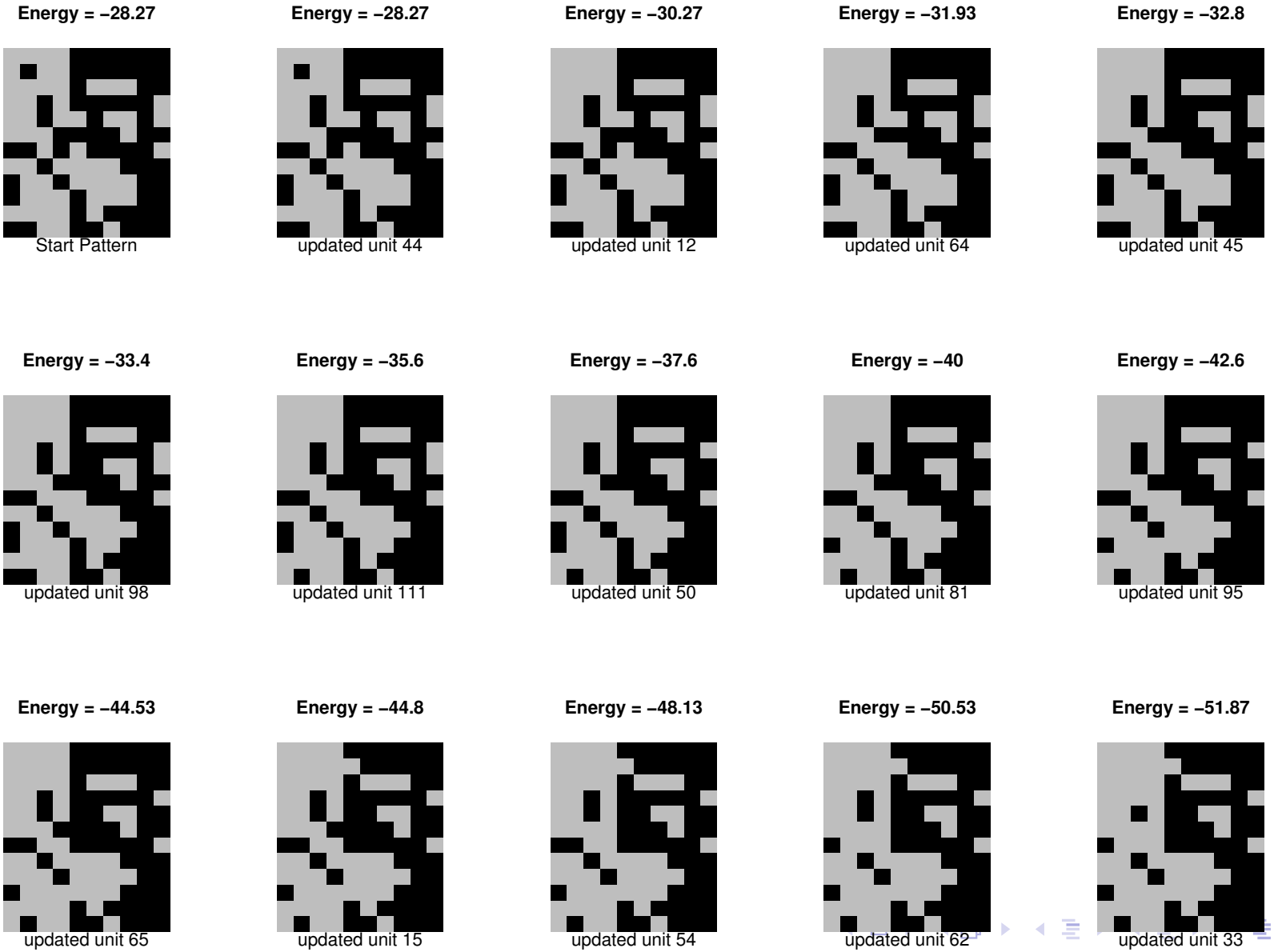
updated unit 88

Updated States of Corrupted Digit 6 (cont.)

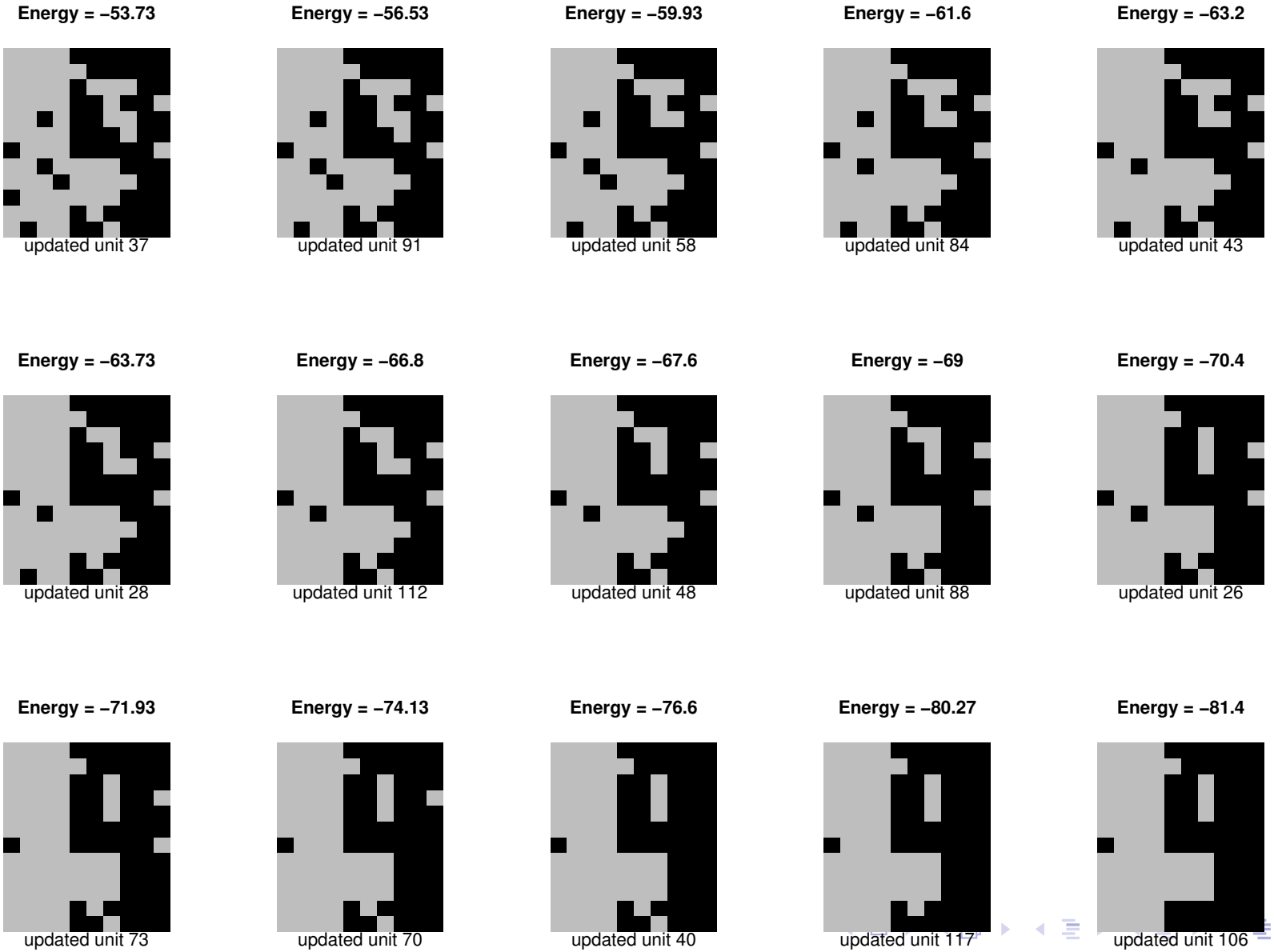
The resulting pattern (stable state with energy -90.47) matches the desired pattern.



Recall a Spurious Pattern

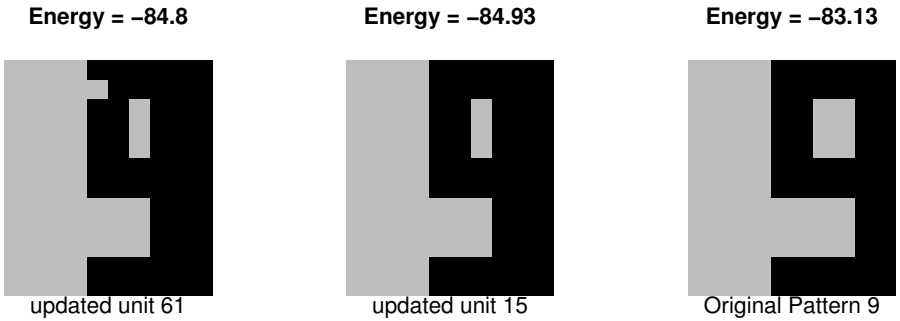


Recall a Spurious Pattern (cont.)

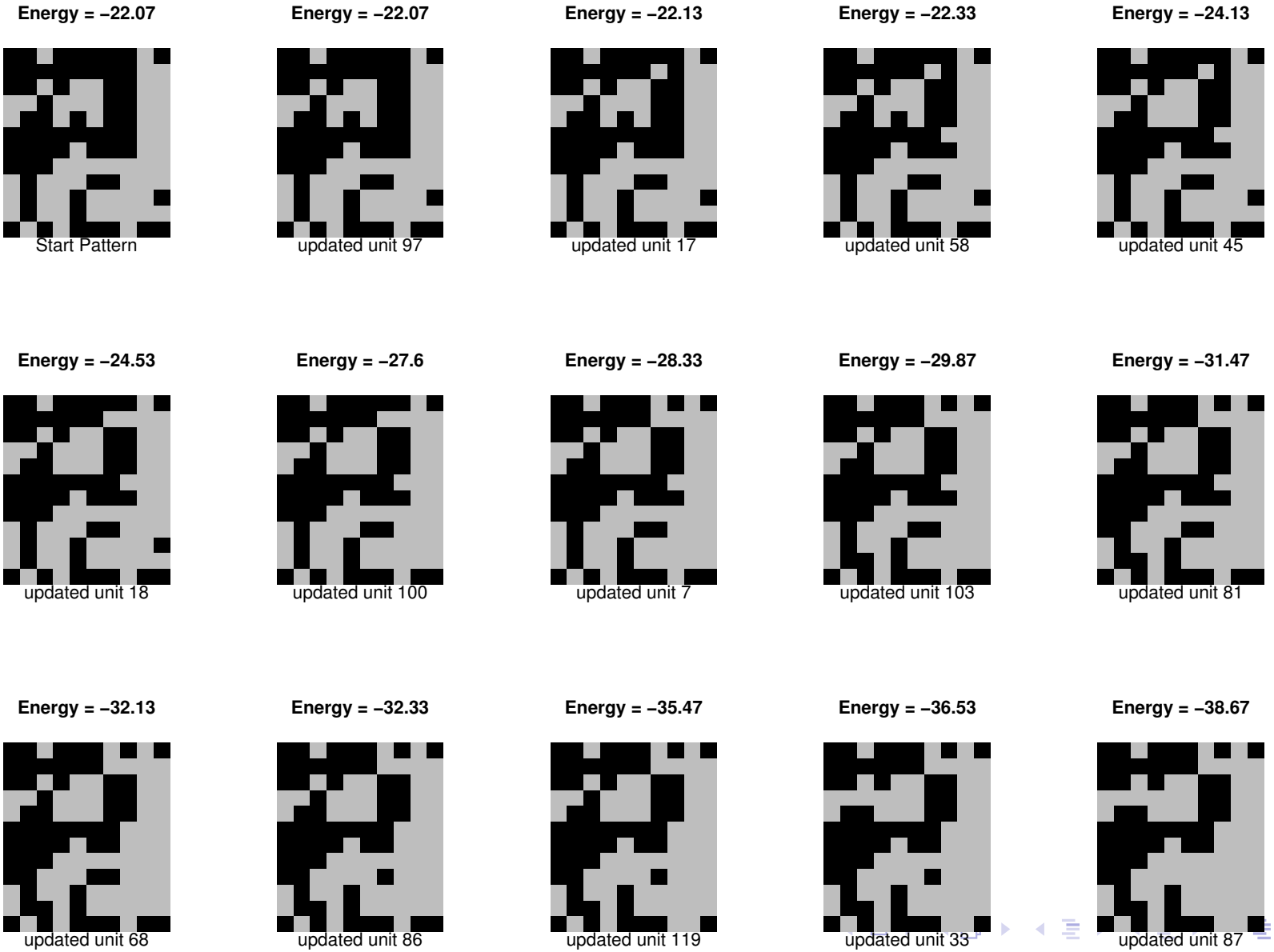


Recall a Spurious Pattern (cont.)

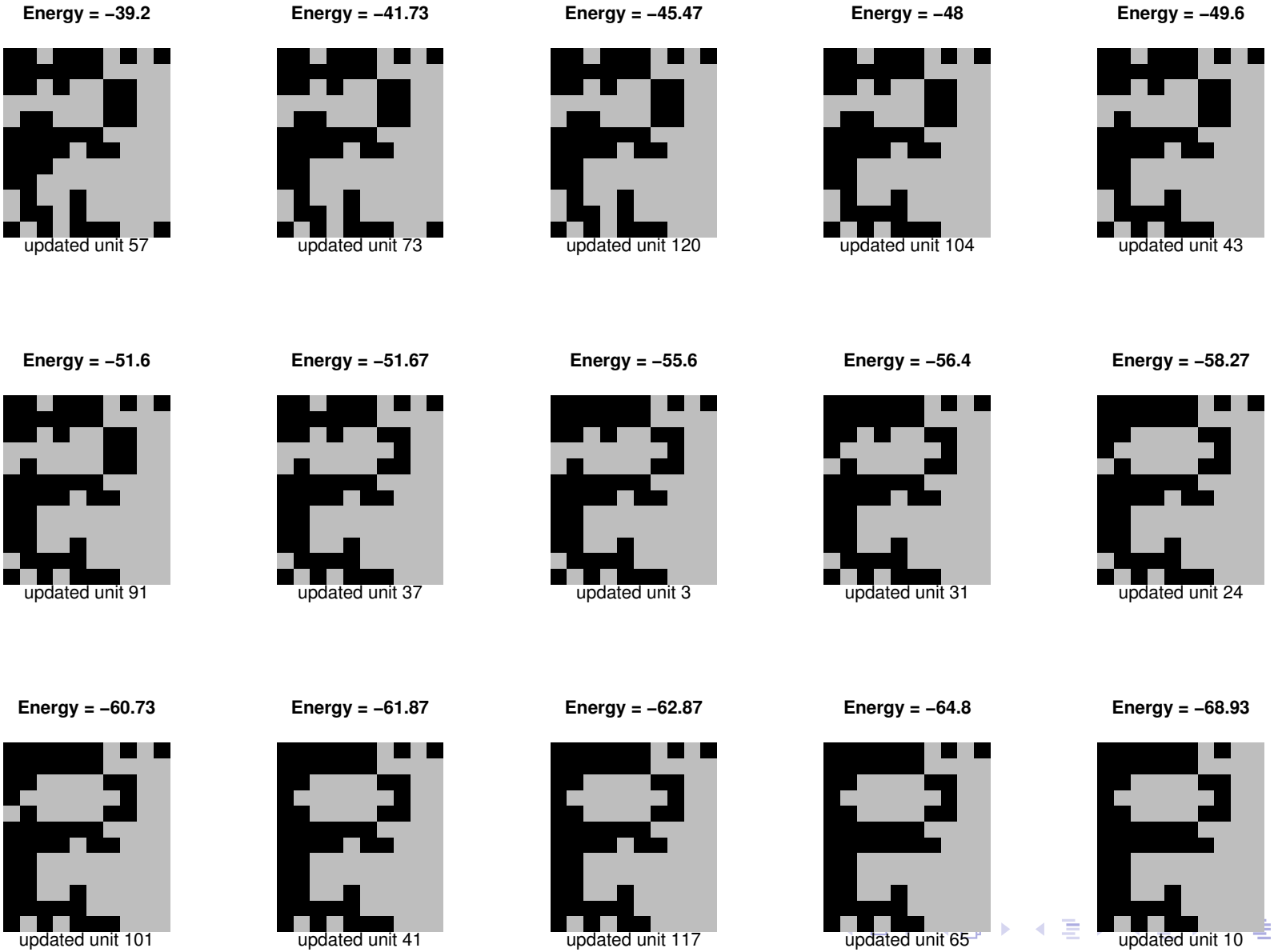
The Hopfield network settled down in local minima with energy -84.93. This pattern however is *not* the desired pattern. It is a pattern which was not stored in the network.



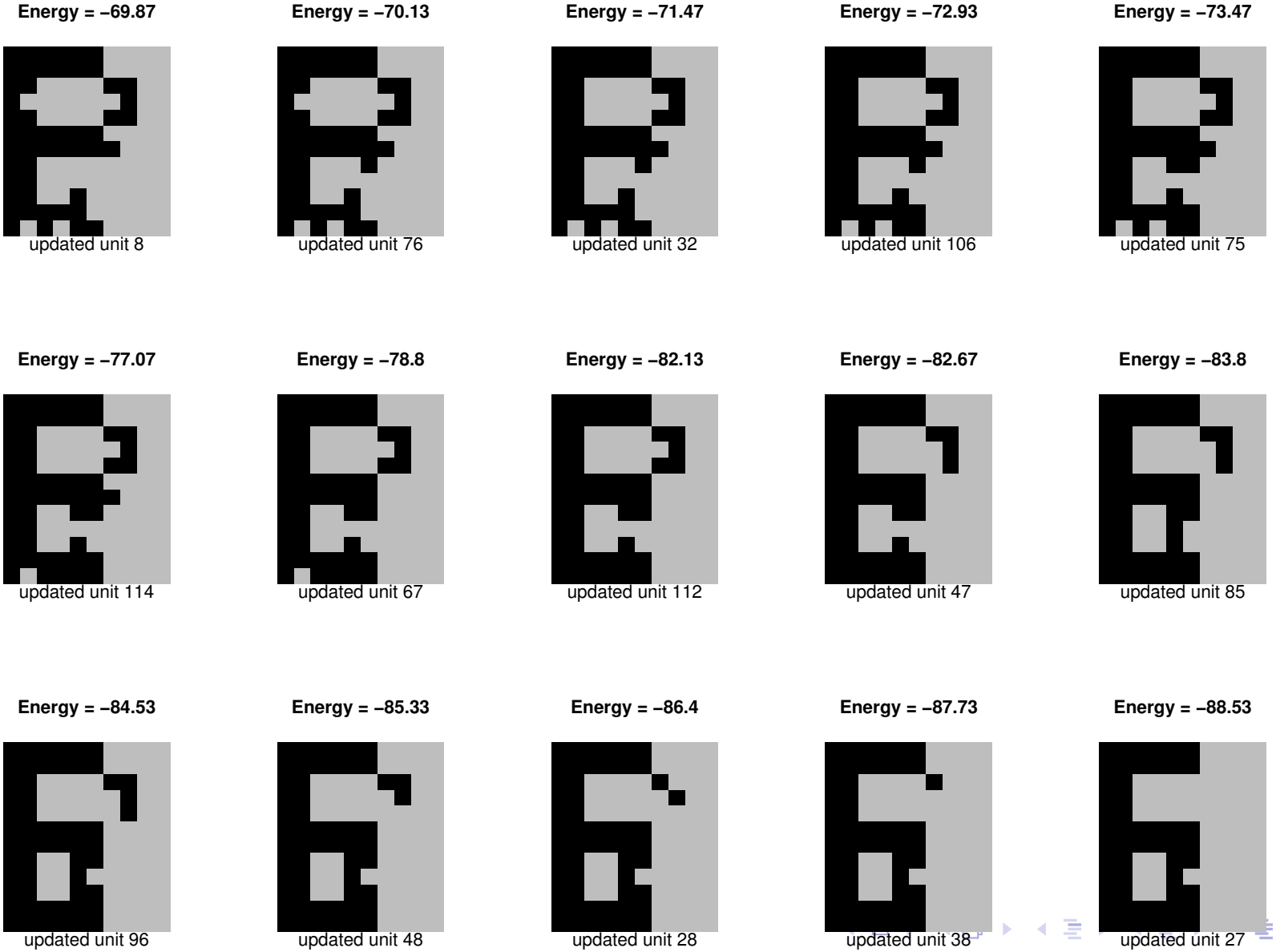
Incorrect Recall of Corrupted Pattern 2



Incorrect Recall of Corrupted Pattern 2 (cont.)

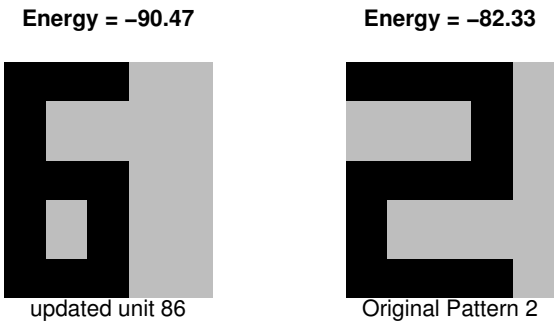


Incorrect Recall of Corrupted Pattern 2 (cont.)



Incorrect Recall of Corrupted Pattern 2 (cont.)

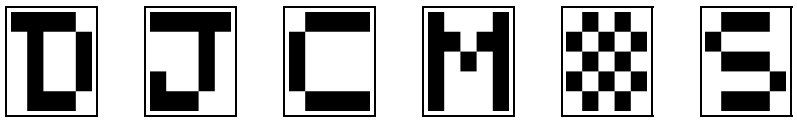
Although we presented the corrupted pattern 2, the Hopfield network settled down in the stable state that corresponds to pattern 6.

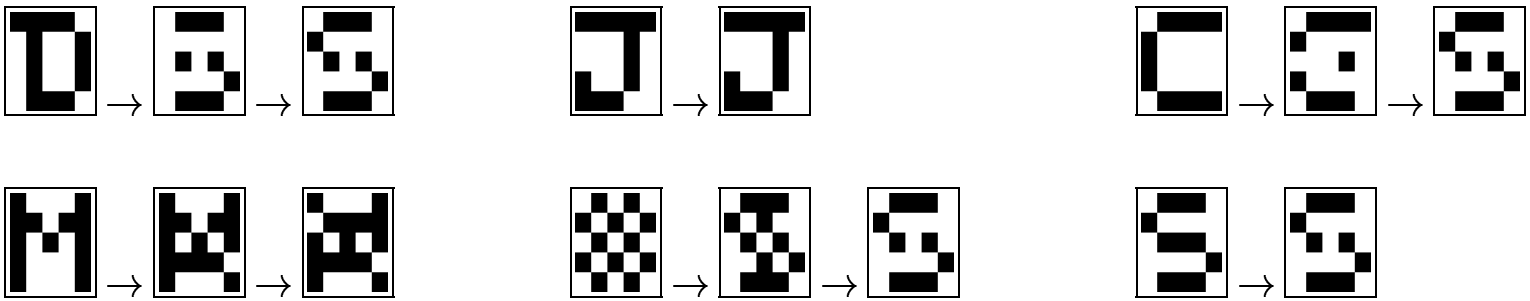


MacKay's Example of an Overloaded Network

Six patterns are stored in the Hopfield network, however most of them are not stable states.

Desired memories:





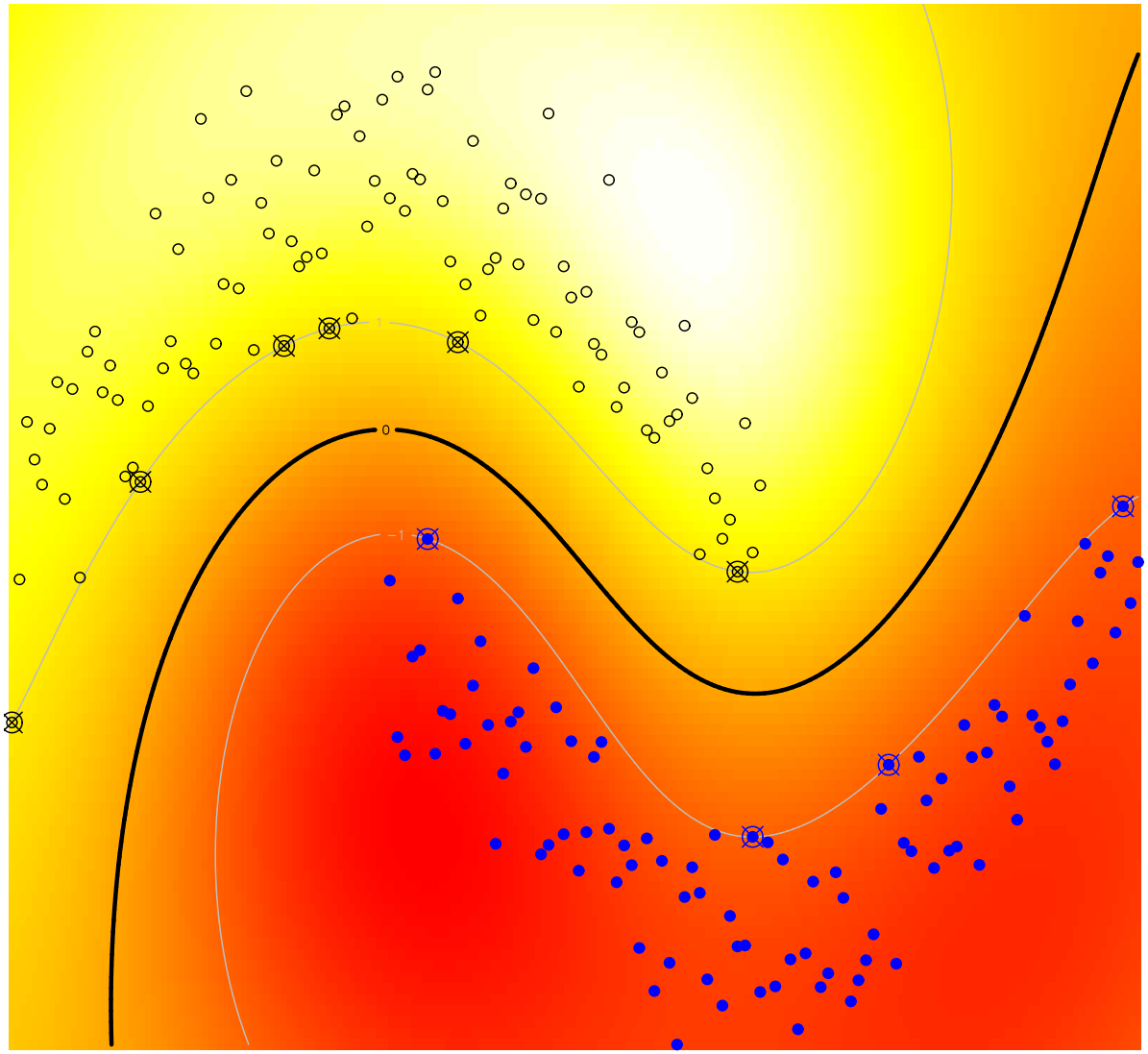
Spurious states represent stable states that are different from the stored desired patterns.

Spurious States and Capacity

- Reversed states $((-1) \cdot \mathbf{x}^{(n)})$ have same energy as the original patterns $\mathbf{x}^{(n)}$.
- Stable mixture states are not equal to any single pattern. They corresponds to a linear combination of an odd number of patterns.
- Spin glass states are local minima that are not correlated with any finite number of the original patterns.

Capacity:
What is the relation between the number d of units and the maximum number N_{max} of patterns one can store by allowing some small error. If $N_{\text{max}} = \frac{d}{4 \log d}$ then most of stored patterns can be recalled perfectly.

Support Vector Machine (SVM)



First papers on SVM

A Training Algorithm for Optimal Margin Classifiers, Bernhard E. Boser and Isabelle M. Guyon and Vladimir Vapnik, Proceedings of the fifth annual workshop on Computational learning theory (COLT), 1992
http://www.clopinet.com/isabelle/Papers/colt92.ps.Z

Support-Vector Networks, Corinna Cortes and Vladimir Vapnik, Machine Learning, 20(3):273-297, 1995
http://homepage.mac.com/corinnacortes/papers/SVM.ps

Extracting Support Data for a Given Task, Bernhard Schölkopf and Christopher J.C. Burges and Vladimir Vapnik, First International Conference on Knowledge Discovery & Data Mining, 1995
http://research.microsoft.com/~cburges/papers/kdd95.ps.gz

Literature (Papers, Bookchapter, T.Report)

Statistical Learning and Kernel Method, Bernhard Schölkopf, Microsoft Research Technical Report, 2000, ftp://ftp.research.microsoft.com/pub/tr/tr-2000-23.pdf

An Introduction to Kernel-based Learning Algorithms, K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf, IEEE Neural Networks, 12(2):181-201, 2001, http://ieeexplore.ieee.org/iel5/72/19749/00914517.pdf

Support Vector Machines, S. Mika, C. Schäfer, P. Laskov, D. Tax and K.-R. Müller, Bookchapter : Handbook of Computational Statistics (Concepts and Methods), http://www.xplore-stat.de/ebooks/scripts/csa/html/

Support Vector Machines for Classification and Regression, S.R. Gunn, Technical Report, http://www.ecs.soton.ac.uk/∼srg/publications/pdf/SVM.pdf

Literature (Paper, Books)

A Tutorial on Support Vector Machines for Pattern Recognition, Christopher J.C. Burges, Kluwer Academic Publishers, Boston http://research.microsoft.com/~cburges/papers/SVMTutorial.pdf

Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond, Bernhard Schölkopf and Alexander J. Smola, MIT Press, 2001

An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, Nello Cristianini and John Shawe-Taylor, Cambridge University Press, 2000

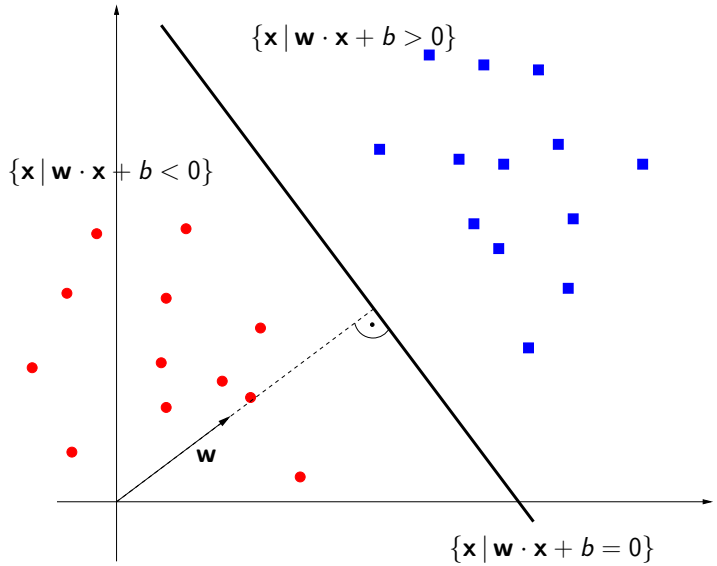
Kernel Methods for Pattern Analysis, John Shawe-Taylor and Nello Cristianini, Cambridge University Press, 2004

The Nature of Statistical Learning Theory, Vladimir N. Vapnik, Springer-Verlag, sec. edition, 1999

Literature (WWW and Source Code)

- http://www.kernel-machines.org/
- http://www.support-vector.net/
- http://www.learning-with-kernels.org/
- http://www.pascal-network.org/
- http://www.r-project.org/ (packages e1071,kernlab)

Scaling Freedom (Problem)



Multiplying $\mathbf{w}$ and b by the same constant κ gives the same hyperplane, represented in terms of different parameters

$$\kappa[(\mathbf{w} \cdot \mathbf{x}) + b] = 0 \Leftrightarrow (\kappa\mathbf{w} \cdot \mathbf{x} + \kappa b) = 0 \Leftrightarrow (\mathbf{w}' \cdot \mathbf{x}) + b' = 0.$$

Example: $\mathbf{w} := (5, -4)^T, b := 2;\ 5x - 4y + 2 = 0 \Leftrightarrow y = \frac{5}{4}x + \frac{1}{2}$,
$\kappa := 3;\ 15x - 12y + 6 = 0 \Leftrightarrow y = \frac{5}{4}x + \frac{1}{2}$

Canonical Hyperplane

A *canonical* hyperplane with respect to given m examples $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m) \in \mathcal{X} \times \{\pm 1\}$ is defined as a function

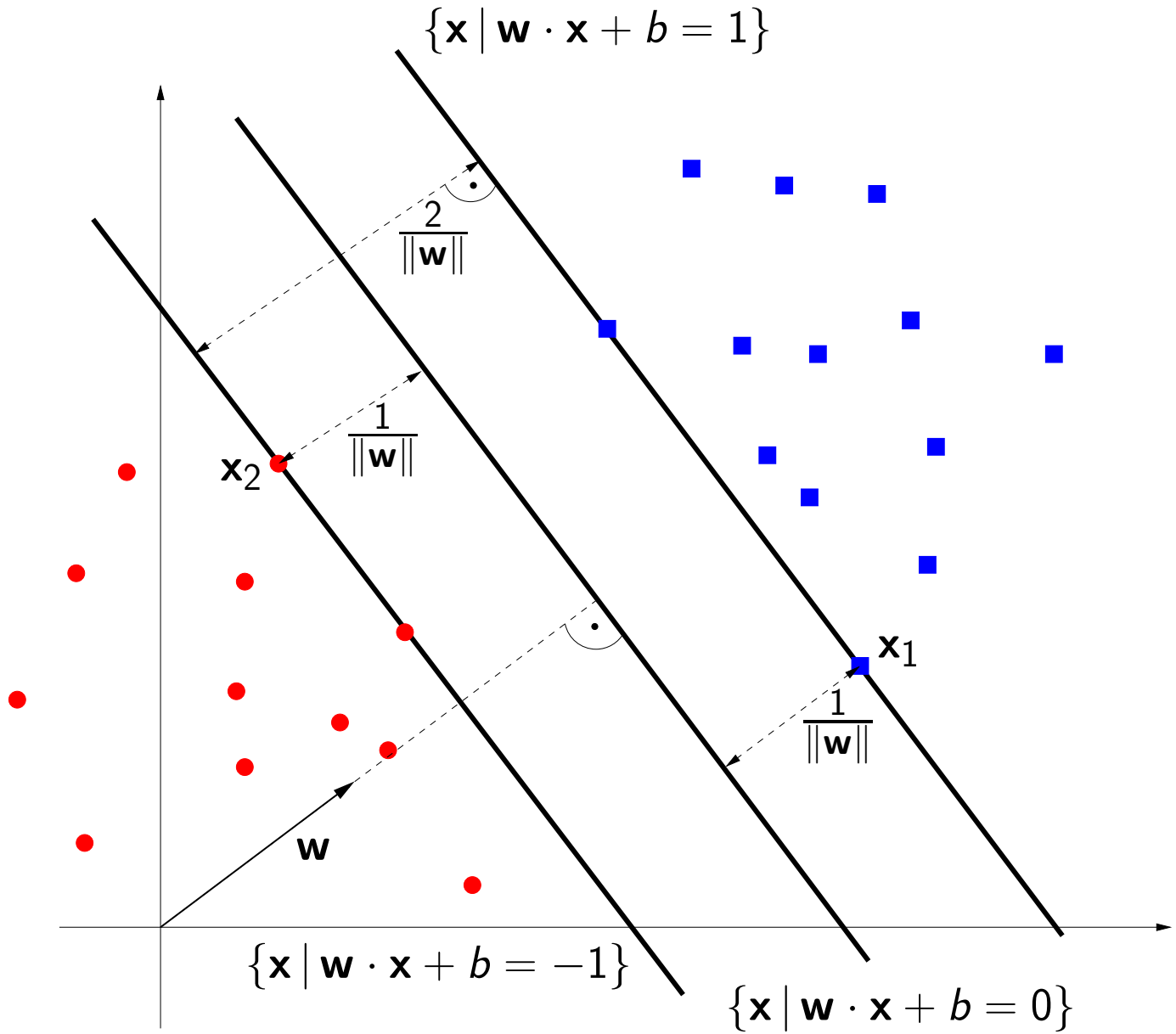
$$f(\mathbf{x}) = (\mathbf{w} \cdot \mathbf{x}) + b$$

where $\mathbf{w}$ is normalized such that

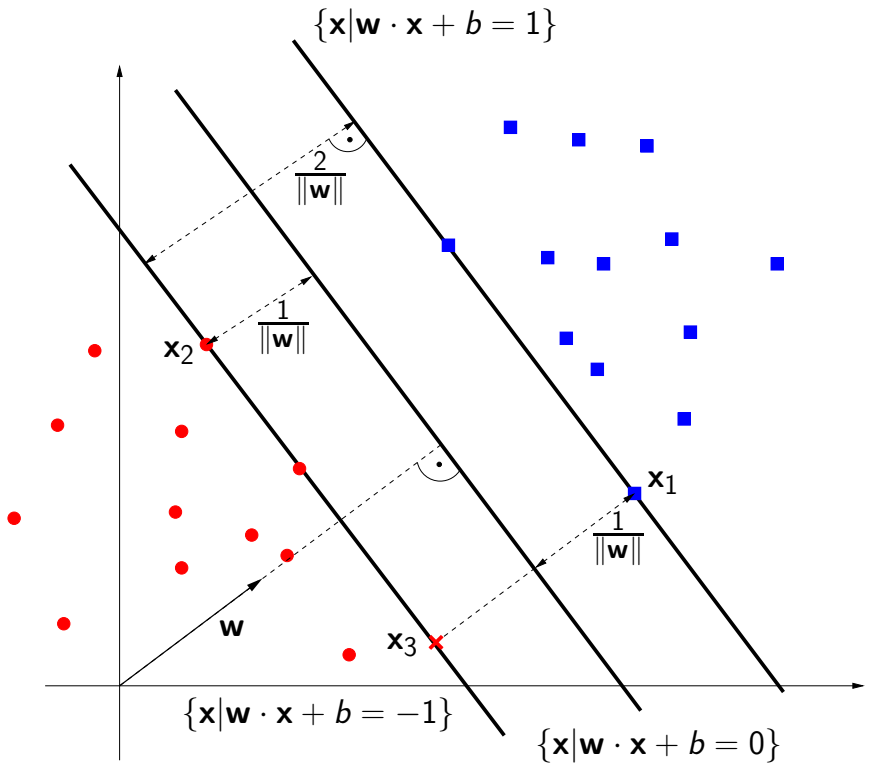
$$\min_{i=1,\ldots,m} |f(\mathbf{x}_i)| = 1$$

i.e. scaling $\mathbf{w}$ and b such that the points closest to the hyperplane satisfy $|(\mathbf{w} \cdot \mathbf{x}_i) + b| = 1$

Plus/Minus and Zero Hyperplane



Optimal Separating Hyperplane



Let $\mathbf{x}_3$ be any point on the "minus" hyperplane and let $\mathbf{x}_1$ be the closest point to $\mathbf{x}_3$.

$$\mathbf{w}\cdot\mathbf{x}_1+b=+1$$
$$\mathbf{w}\cdot\mathbf{x}_3+b=-1$$
$$\mathbf{x}_1=\mathbf{x}_3+\lambda\mathbf{w}$$
$$\mathbf{w}\cdot\underbrace{(\mathbf{x}_3+\lambda\mathbf{w})}_{\mathbf{x}_1}+b=1$$

$$\underbrace{\mathbf{w}\cdot\mathbf{x}_3+b}_{-1}+\lambda\|\mathbf{w}\|^2=1\Rightarrow\lambda=\frac{2}{\|\mathbf{w}\|^2}.$$

We are interested in margin, i.e.

$$\|\mathbf{x}_1-\mathbf{x}_3\|=\|\lambda\mathbf{w}\|=\lambda\|\mathbf{w}\|=\frac{2}{\|\mathbf{w}\|^2}\|\mathbf{w}\|=\frac{2}{\|\mathbf{w}\|}$$

Formulation as an Optimization Problem

Hyperplane with maximum margin (the smaller the norm of the weight vector $\mathbf{w}$, the larger the margin):

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to} \quad & y_i\left((\mathbf{w}\cdot\mathbf{x}_i)+b\right) \geq 1, \quad i = 1,\ldots,m \end{aligned}$$

Introduce Lagrange multipliers $\alpha_i \geq 0$ and a Lagrangian

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i (y_i((\mathbf{x}_i \cdot \mathbf{w}) + b) - 1)$$

At the extrema, we have

$$\frac{\partial}{\partial b} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0, \quad \frac{\partial}{\partial \mathbf{w}} L(\mathbf{w}, b, \boldsymbol{\alpha}) = 0$$

Optimization and Kuhn-Tucker Theorem

leads to solution

$$\sum_{i=1}^{m} \alpha_i y_i = 0, \qquad \mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i.$$

The extreme point solution obtained has an important property that results from optimization known as the Kuhn-Tucker theorem. The theorem says:

> Lagrange multiplier can be non-zero only if the corresponding inequality constraint is an equality at the solution.

This implies that *only* the training examples $\mathbf{x}_i$ that lie on the plus and minus hyperplane have their corresponding α_i non-zero.

Relevant/Irrelevant Support Vector

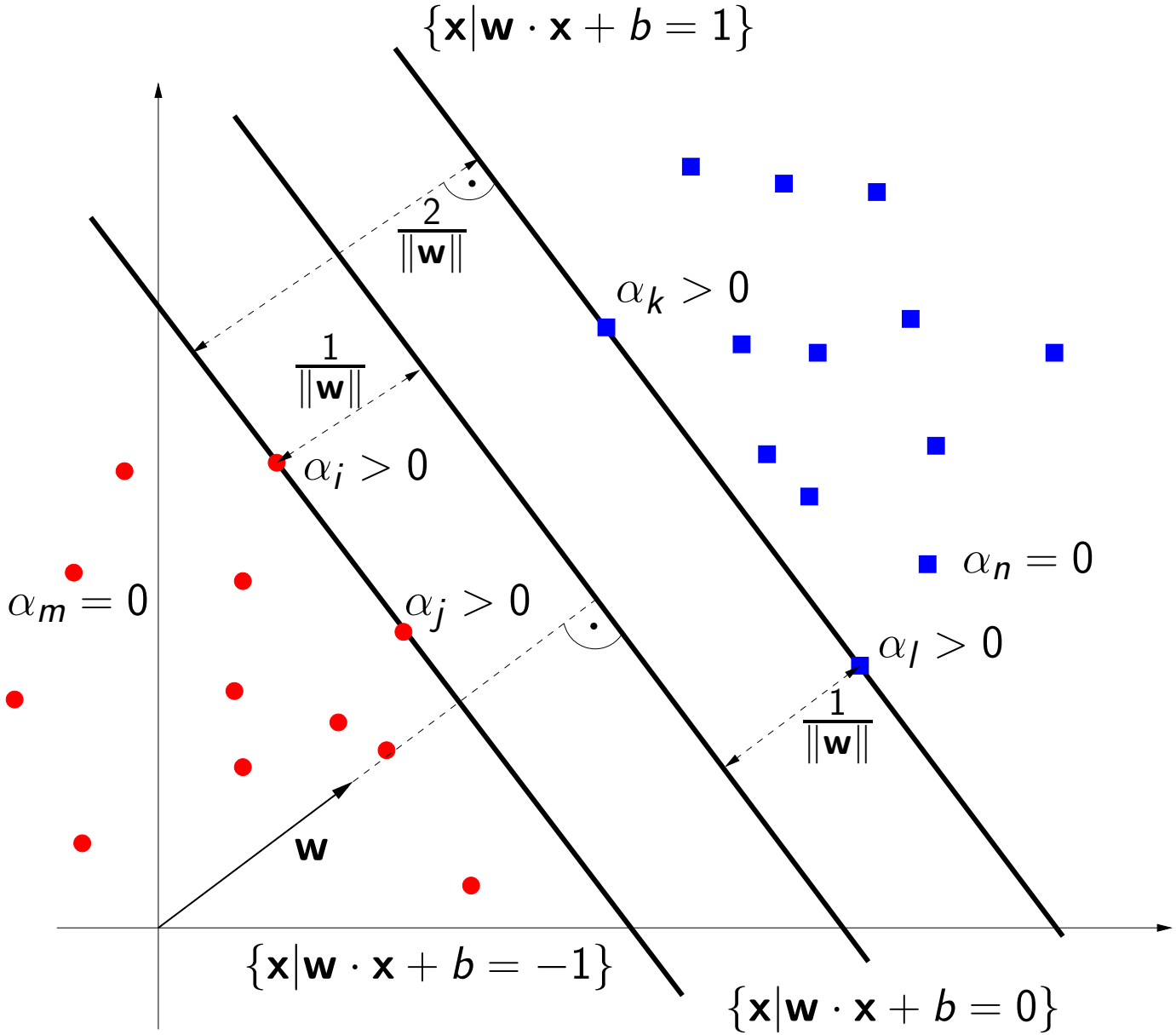
More formally, the Karush-Kuhn-Tucker complementarity conditions say:

$$\alpha_i \left[y_i((\mathbf{x}_i \cdot \mathbf{w}) + b) - 1 \right] = 0, \quad i = 1, \ldots, m$$

the Support Vectors lie on the margin. That means for all training points

$$\begin{array}{lll} [y_i((\mathbf{x}_i \cdot \mathbf{w}) + b)] > 1 & \Rightarrow \alpha_i = 0 & \rightarrow \mathbf{x}_i \text{ irrelevant} \\ [y_i((\mathbf{x}_i \cdot \mathbf{w}) + b)] = 1 & \text{(on margin)} & \rightarrow \mathbf{x}_i \text{ Support Vector} \end{array}$$

Relevant/Irrelevant Support Vector



Dual Form

The dual form has many advantages

- Formulate optimization problem without **w** (mapping **w** in high-dimensional spaces).
- Formulate optimization problem by means of α, y_i and dot product $\mathbf{x}_i \cdot \mathbf{x}_j$.
- Quadratic Programming Solver.

$$\text{maximize} \qquad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$\text{subject to} \qquad \alpha_i \geq 0, \quad i = 1, \ldots, m \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0$$

Hyperplane Decision Function

The solution is determined by the examples on the margin. Thus

$$\begin{aligned} f(\mathbf{x}) &= \operatorname{sgn}\left((\mathbf{x}\cdot\mathbf{w})+b\right) \\ &= \operatorname{sgn}\left(\sum_{i=1}^{m} y_i\alpha_i\left(\mathbf{x}\cdot\mathbf{x}_i\right)+b\right) \end{aligned}$$

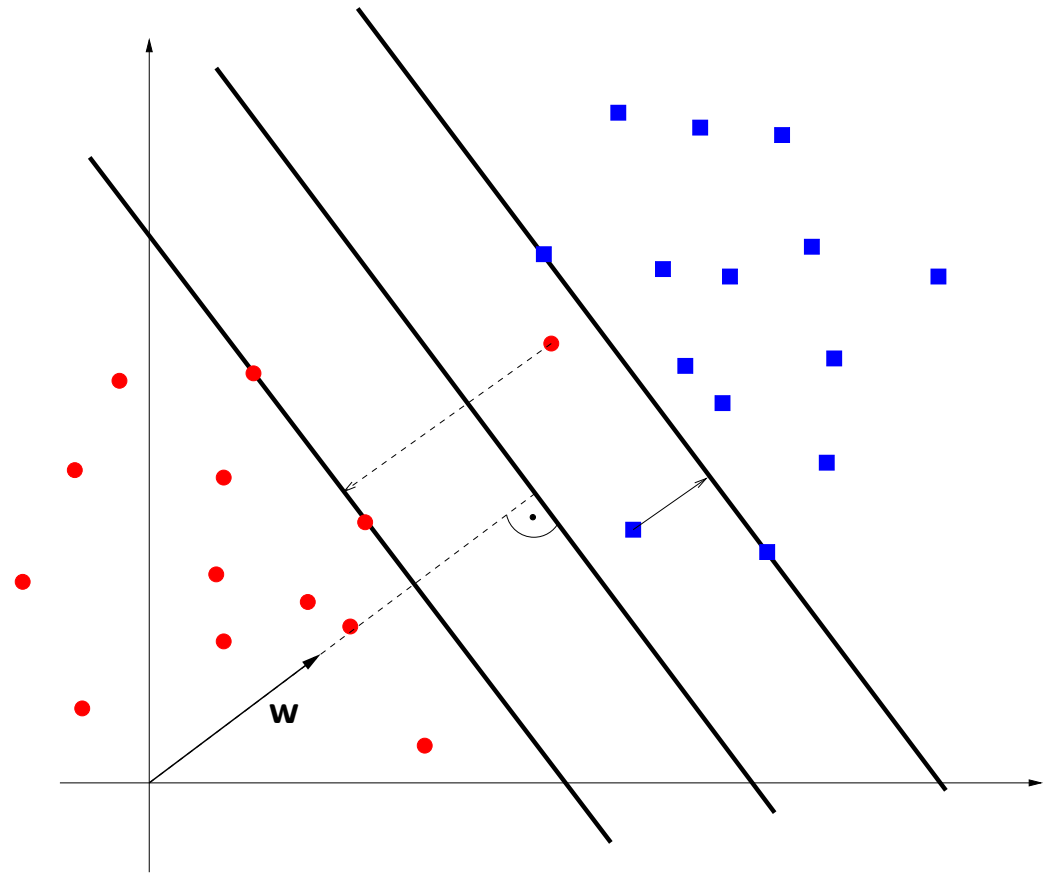
where

$$\alpha_i\left[y_i((\mathbf{x}_i\cdot\mathbf{w})+b)-1\right]=0, \quad i=1,\ldots,m$$

and

$$\mathbf{w}=\sum_{i=1}^{m}\alpha_i y_i \mathbf{x}_i$$

Non-separable Case



Case where the constraints $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1$ cannot be satisfied, i.e. $\alpha_i \to \infty$

Relax Constraints (Soft Margin)

Modify the constraints to

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i, \text{ with } \xi_i \geq 0$$

and add

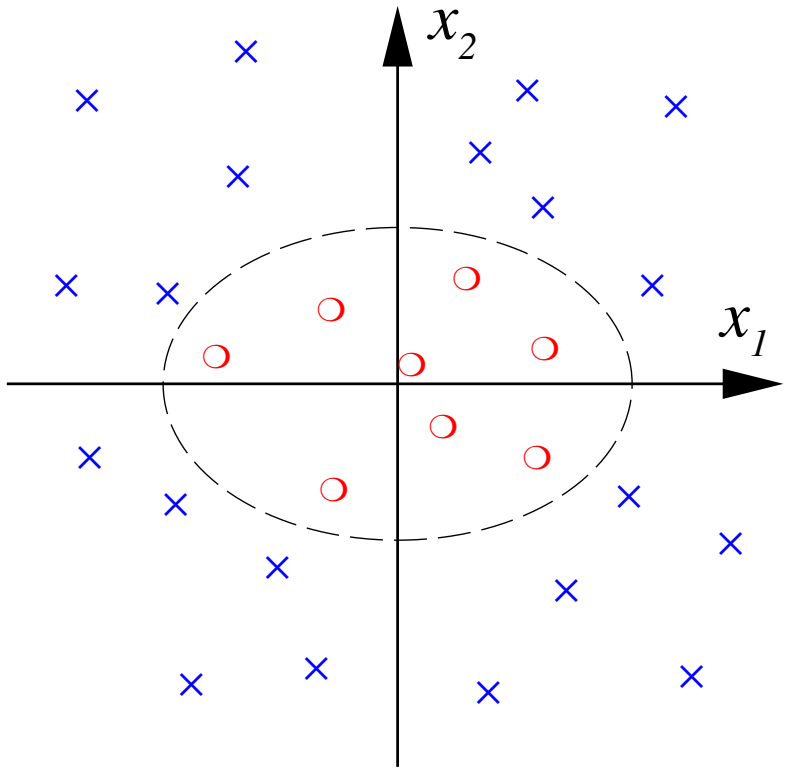
$$C \cdot \sum_{i=1}^{m} \xi_i$$

i.e. distance of error points to their correct place in the objective function

$$\text{minimize} \qquad \frac{1}{2}\|\mathbf{w}\|^2 + C \cdot \sum_{i=1}^{m} \xi_i$$

Same dual, with additional constraints $0 \leq \alpha_i \leq C$

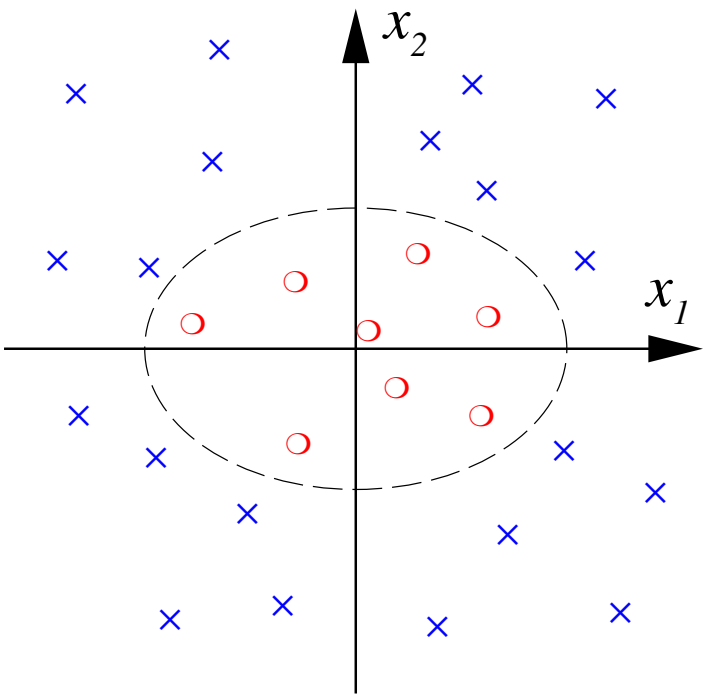
Non-separable Case (Part 2)

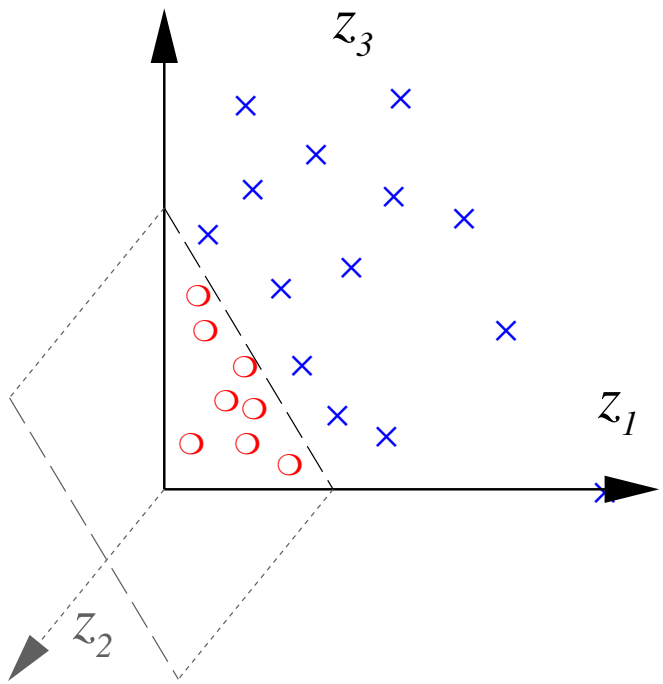


This data set is not properly separable with lines (also when using many slack variables)

Separate in Higher-Dim. Space

Map data in higher-dimensional space and separate it there with a hyperplane





$$\begin{aligned} \Phi : \mathbb{R}^2 &\rightarrow \mathbb{R}^3 \\ (x_1, x_2) &\mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2}x_1x_2, x_2^2) \end{aligned}$$

Feature Space

Apply the mapping

$$\begin{aligned} \Phi : \mathbb{R}^N &\rightarrow \mathcal{F} \\ \mathbf{x} &\mapsto \Phi(\mathbf{x}) \end{aligned}$$

to the data $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m \in \mathcal{X}$ and construct separating hyperplane in $\mathcal{F}$ instead of $\mathcal{X}$. The samples are preprocessed as $(\Phi(\mathbf{x}_1), y_1), \ldots, (\Phi(\mathbf{x}_m), y_m) \in \mathcal{F} \times \{\pm 1\}$.

Obtained decision function:

$$\begin{aligned} f(\mathbf{x}) &= \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i \left(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}_i)\right) + b\right) \\ &= \operatorname{sgn}\left(\sum_{i=1}^{m} y_i \alpha_i \, k(\mathbf{x}, \mathbf{x}_i) + b\right) \end{aligned}$$

How about patters $\mathbf{x} \in \mathbb{R}^N$ and product features of order d? $\mathrm{Dim}(\mathcal{F})$ grows like N^d. Example $N = 16 \times 16$, and $d = 5 \longrightarrow$ dimension 10^{10}.

Kernels

A *kernel* is a function k, such that for all $\mathbf{x}, \mathbf{y} \in \mathcal{X}$

$$k(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{y})),$$

where Φ is a mapping from $\mathcal{X}$ to an dot product feature space $\mathcal{F}$.

The $m \times m$ matrix K with elements $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ is called kernel matrix or Gram matrix. The kernel matrix is symmetric and positive semi-definite, i.e. for all $a_i \in \mathbb{R}$, $i = 1, \ldots, m$, we have

$$\sum_{i,j=1}^{m} a_i a_j K_{ij} \geq 0$$

Positive semi-definite kernels are exactly those giving rise to a positive semi-definite kernel matrix K for all m and all sets $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\} \subseteq \mathcal{X}$.

The Kernel Trick Example

Example : compute 2nd order products of two "pixels", i.e.

$$\mathbf{x} = (x_1, x_2) \text{ and } \Phi(\mathbf{x}) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$$

$$\begin{aligned}
(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{z})) &= (x_1^2, \sqrt{2}x_1x_2, x_2^2)(z_1^2, \sqrt{2}z_1z_2, z_2^2)^T \\
&= ((x_1, x_2)(z_1, z_2)^T)^2 \\
&= (\mathbf{x} \cdot \mathbf{z}^T)^2 \\
&= : k(\mathbf{x}, \mathbf{z})
\end{aligned}$$

Kernel without knowing Φ

Recall: mapping $\Phi : \mathbb{R}^N \to \mathcal{F}$. SVM depends on the data through dot products in $\mathcal{F}$, i.e. functions of the form

$$\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

- With k such that $k(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$, it is not necessary to even know what $\Phi(\mathbf{x})$ is.

Example: $k(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{\gamma}\right)$, in this example $\mathcal{F}$ is infinite dimensional.

Feature Space (Optimization Problem)

Quadratic optimization problem (soft margin) with kernel:

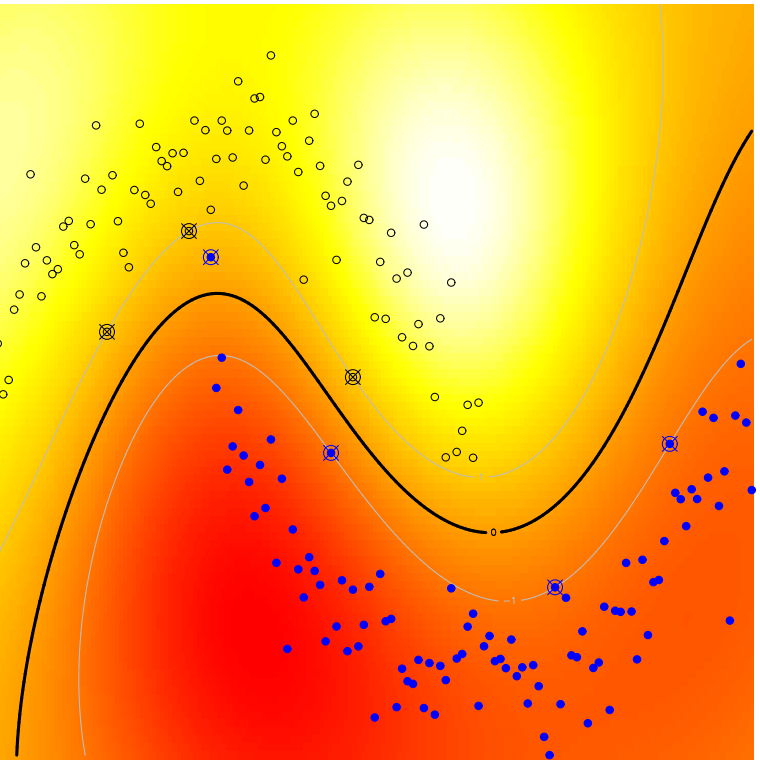
$$\text{maximize} \qquad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{subject to} \qquad 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, m \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0$$

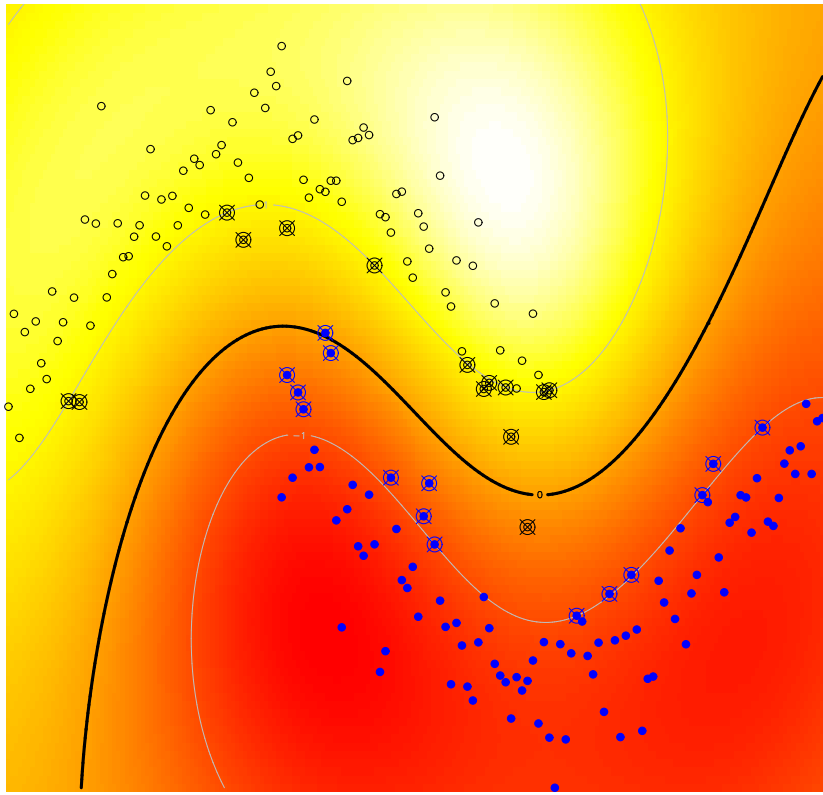
(Standard) Kernels

$$
\begin{aligned}
\text{Linear} \quad & k_0(\mathbf{u},\mathbf{v}) &&= (\mathbf{u}\cdot\mathbf{v}) \\
\text{Polynomial} \quad & k_1(\mathbf{u},\mathbf{v}) &&= ((\mathbf{u}\cdot\mathbf{v})+\Theta)^d \\
\text{Gaussian} \quad & k_2(\mathbf{u},\mathbf{v}) &&= \exp\left(-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{\gamma}\right) \\
\text{Sigmoidal} \quad & k_3(\mathbf{u},\mathbf{v}) &&= \tanh(\kappa(\mathbf{u}\cdot\mathbf{v})+\Theta)
\end{aligned}
$$

SVM Results for Gaussian Kernel

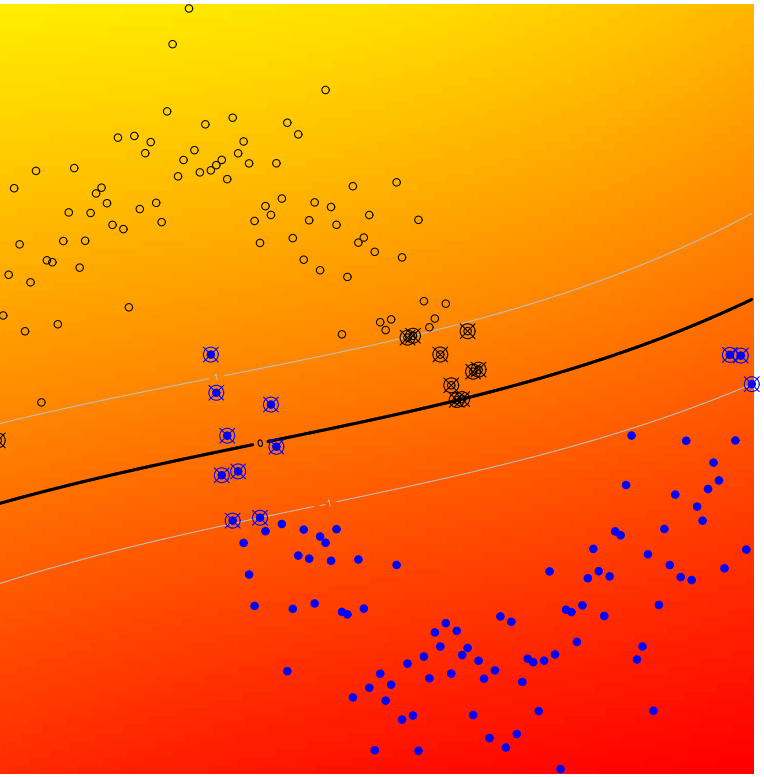


$\gamma = 0.5,\ C = 50$

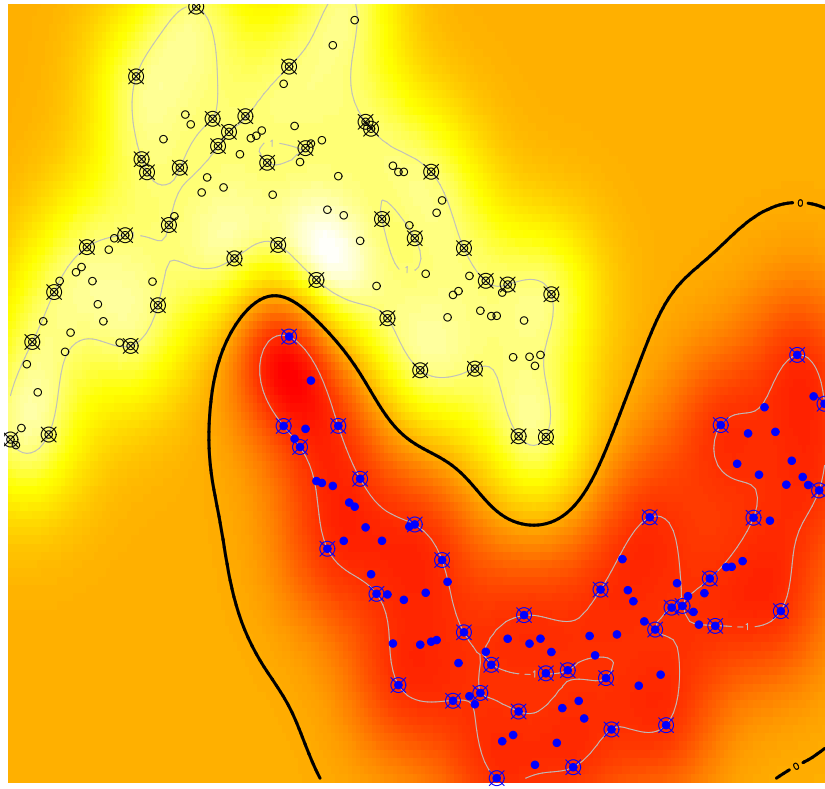


$\gamma = 0.5,\ C = 1$

SVM Results for Gaussian Kernel (cont.)



$\gamma = 0.02,\ C = 50$



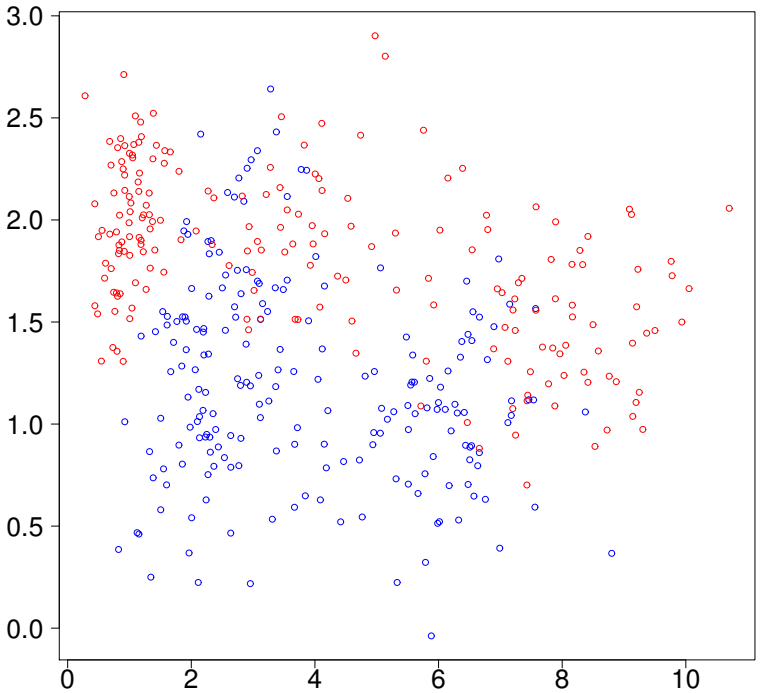
$\gamma = 10,\ C = 50$

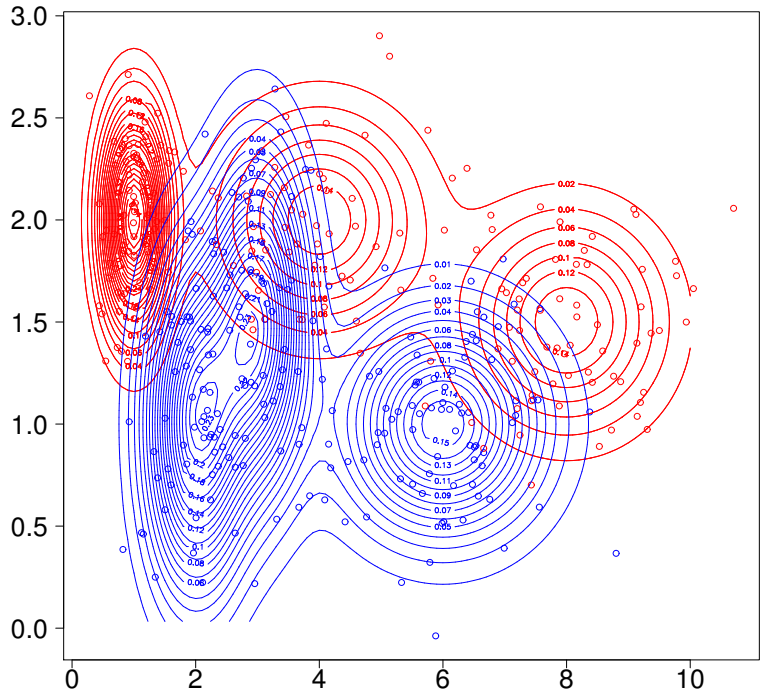
Link to Statistical Learning Theory

Pattern Classification:

$$\text{Learn } f : \mathcal{X} \to \{\pm 1\} \text{ from input-output training data}$$

We assume that data is generated from some unknown (but fixed) probability distribution $P(\mathbf{x}, y)$





(Empirical) Risk

Learn f from training set $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m) \in \mathcal{X} \times \{\pm 1\}$, such that the expected missclassification error on a test set, also drawn from $P(\mathbf{x}, y)$,

$$R[f] = \int \frac{1}{2} |f(\mathbf{x}) - y| dP(\mathbf{x}, y) \quad \text{Expected Risk}$$

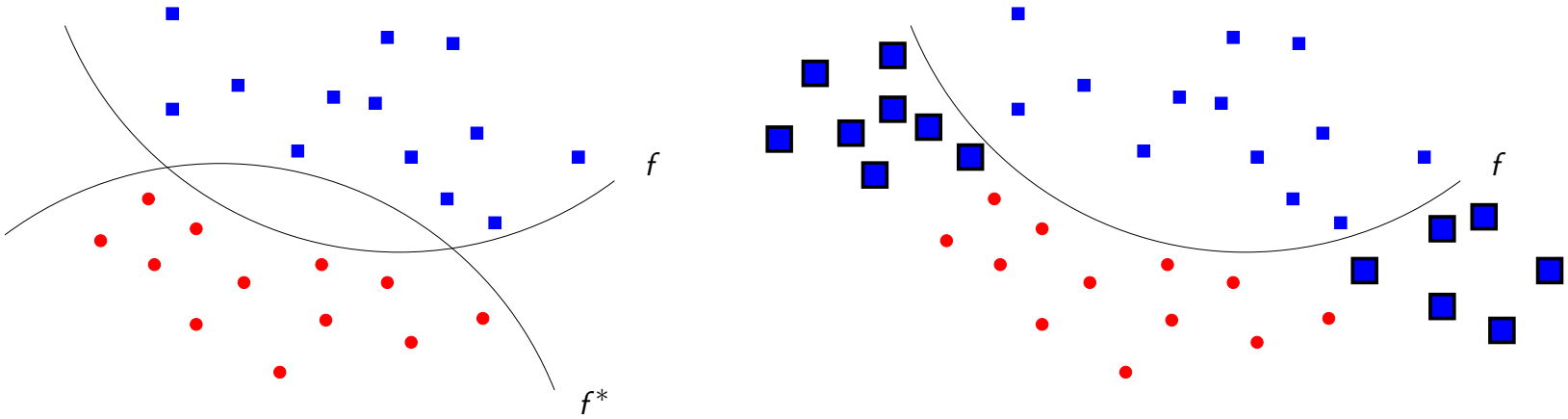
is minimal.

Problem: we cannot minimize the expected risk, because P is unknown. Minimize instead the average risk over training set, i.e.

$$R_{\text{emp}}[f] = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} |f(\mathbf{x}_i) - y_i| \quad \text{Empirical Risk}$$

Problems with Empirical Risk

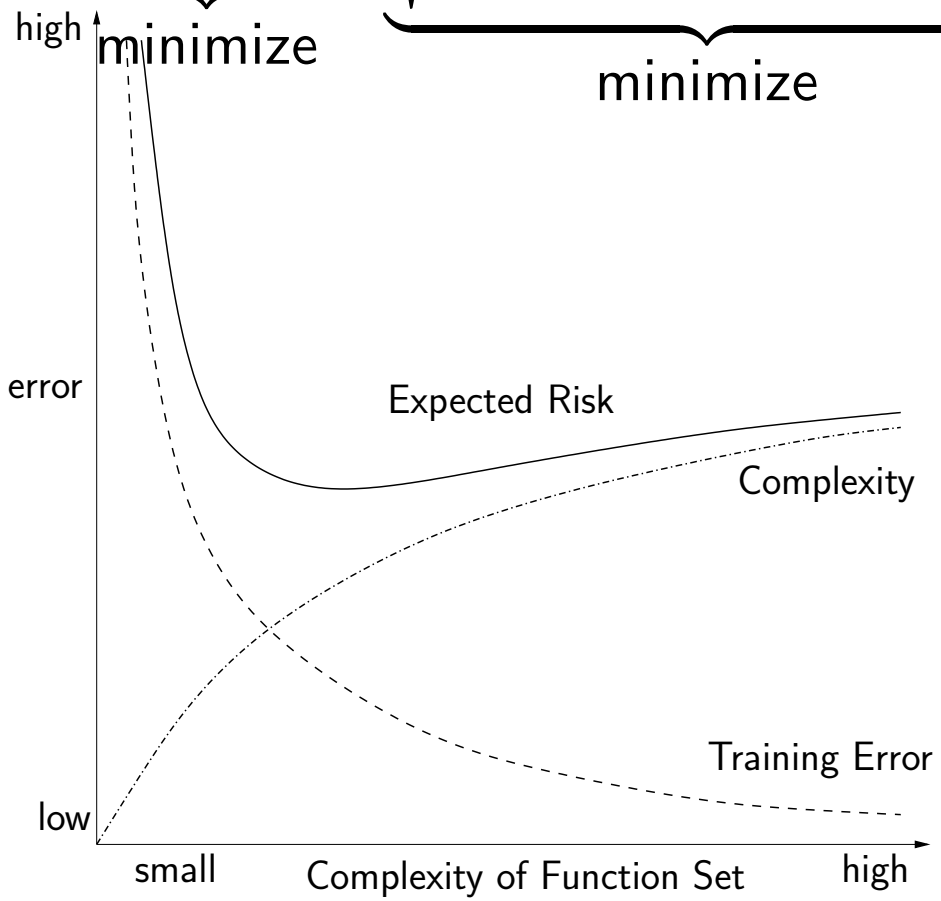
Minimizing the empirical risk (training error), does not imply a small test error. To see this, note that for each function f and any test set $(\overline{\mathbf{x}}_1, \overline{y}_1), \ldots, (\overline{\mathbf{x}}_m, \overline{y}_m) \in \mathcal{X} \times \{\pm 1\}$, satisfying $\{\overline{\mathbf{x}}_1, \ldots, \overline{\mathbf{x}}_{\overline{m}}\} \cap \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} = \{\}$, there exists another function f^* such that $f^*(\mathbf{x}_i) = f(\mathbf{x}_i)$ for all $i = 1, \ldots, m$, *but* $f^*(\overline{\mathbf{x}}_i) \neq f(\overline{\mathbf{x}}_i)$ (on all test samples) for all $i = 1, \ldots, \overline{m}$



A Bound for Pattern Classification

For any $f \in F$ and $m > h$, with a probability of at least $1 - \eta$,

$$R[f] \leq \underbrace{R_{\text{emp}}[f]}_{\text{minimize}} + \underbrace{\sqrt{\frac{h(\ln \frac{2m}{h} + 1) - \ln(\eta/4)}{m}}}_{\text{minimize}}$$

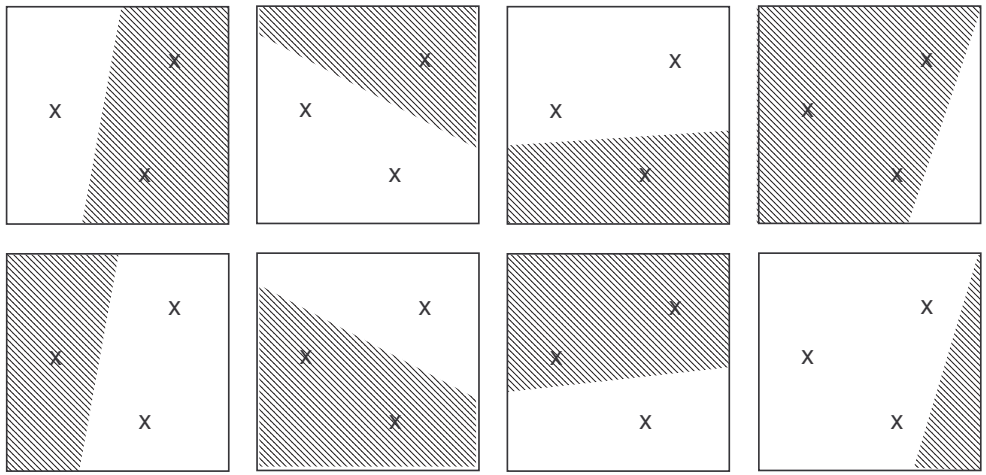


Measure Complexity of Function Set

How to measure the complexity of a given function set ?

The Vapnik-Chervonenkis (short VC) dimension is defined as the maximum number of points that can be labeled in all possible ways.

- In $\mathbb{R}^2$ we can shatter three non-collinear points.
- But we can never shatter four points in $\mathbb{R}^2$.
- Hence the VC dimension is $h = 3$.



VC Dimension

- Separating hyperplanes in $\mathbb{R}^N$ have VC dimension $N + 1$.
- Hence: separating hyperplanes in high-dimensional feature spaces have extremely large VC dimension, and may not generalize well.
- But, "margin" hyperplanes can still have a small VC dimension.

VC Dimension of Margin Hyperplanes

Consider hyperplanes $(\mathbf{w} \cdot \mathbf{x}) = 0$ where w is normalized such they are in a canonical form w.r.t. a set of points $X^* = \{\mathbf{x}_1, \ldots, \mathbf{x}_r\}$, i.e.,

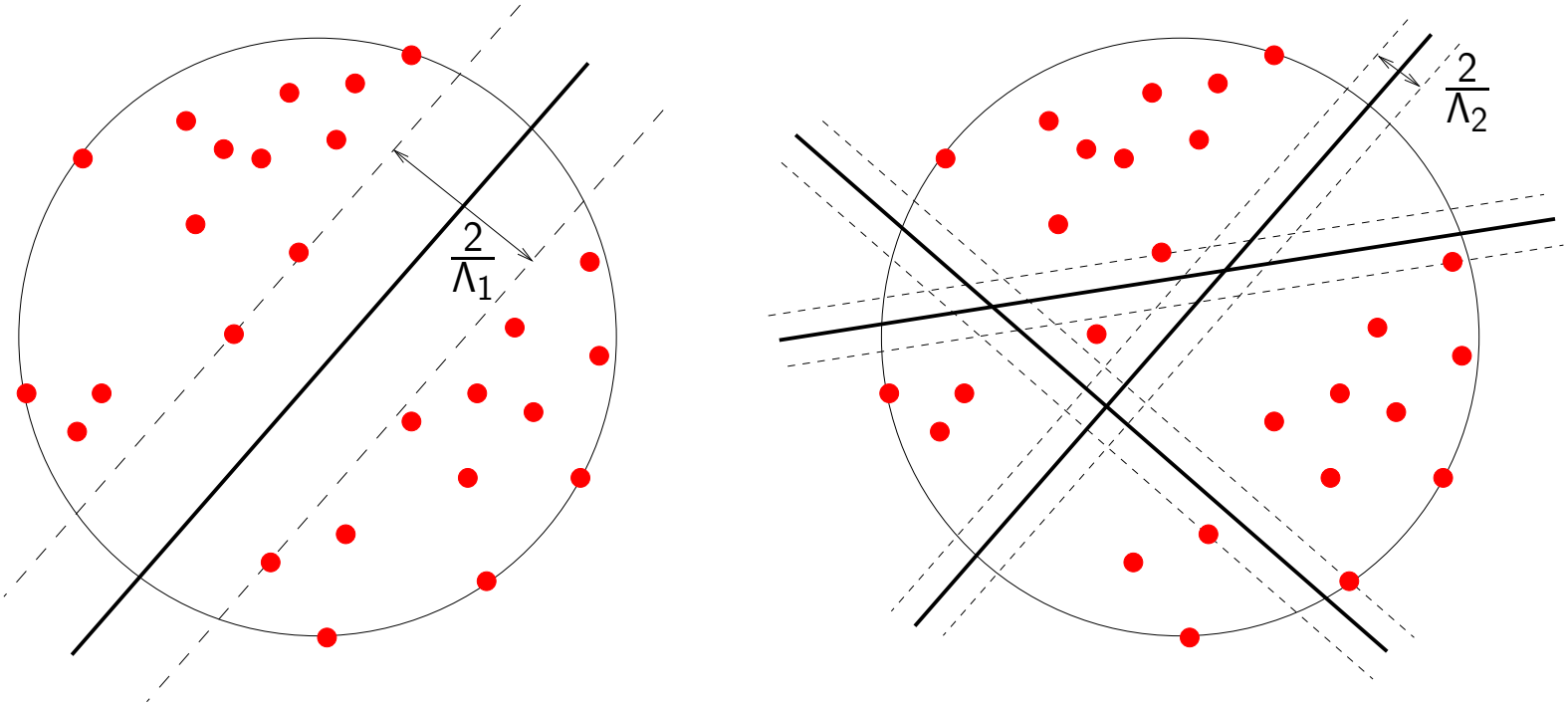
$$\min_{i=1,\ldots,r} |(\mathbf{w} \cdot \mathbf{x}_i)| = 1.$$

The set of decision functions $f_{\mathbf{w}}(\mathbf{x}) = \mathrm{sgn}(\mathbf{x} \cdot \mathbf{w})$ defined on X^* and satisfying the constraint $\|\mathbf{w}\| \leq \Lambda$ has VC dimension satisfying

$$h \leq \min(R^2\Lambda^2 + 1, N + 1)$$

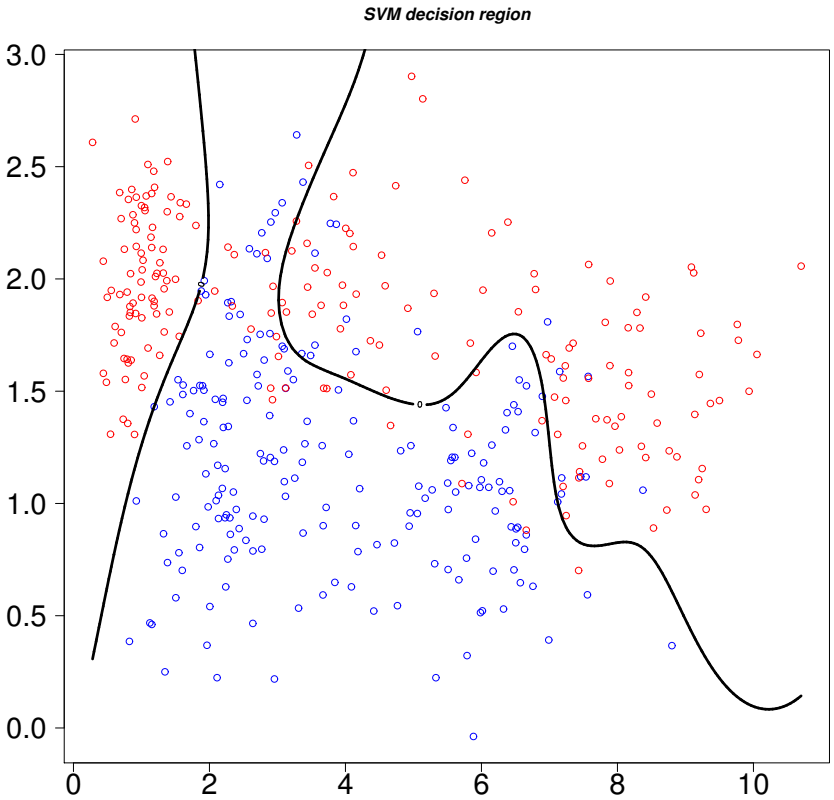
Here, R is the smallest sphere around the origin containing X^*.

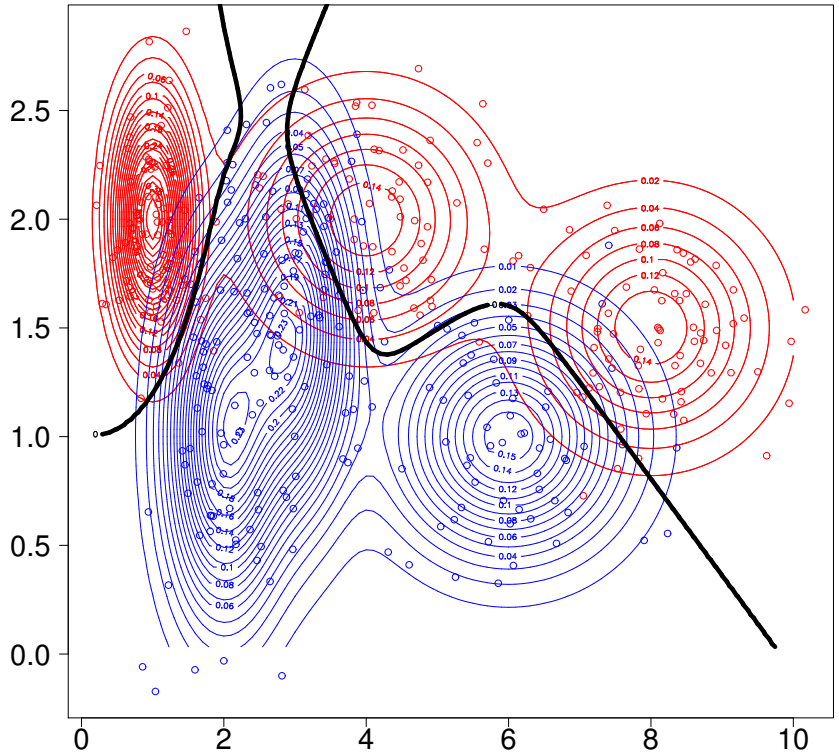
Smallest Sphere and VC Dimension



Hyperplanes with a large margin ($\frac{2}{\Lambda_1}$) induce only a small number of possibilities to separate the data, i.e. the VC dimension is small (left figure). In contrast, smaller margins ($\frac{2}{\Lambda_2}$) induce more separating hyperplanes, i.e. the VC dimension is large (right figure).

SVM (RBF Kernel) and Bayes Decision





Implementation (Matrix notation)

Recall:

$$\text{maximize} \qquad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$\text{subject to} \qquad \alpha_i \geq 0, \quad i = 1, \ldots, m \text{ and } \sum_{i=1}^{m} \alpha_i y_i = 0$$

This can be expressed in a matrix notation as

$$\min_{\alpha} \frac{1}{2} \alpha^T H \alpha + c^T \alpha$$

Implementation (Matrix notation) cont.

where

$$H = ZZ^T, \quad c^T = (-1, \ldots, -1)$$

with constraints

$$\alpha^T Y = 0, \alpha_i \geq 0, i = 1, \ldots, m$$

where

$$Z = \begin{bmatrix} y_1 \mathbf{x}_1 \\ y_2 \mathbf{x}_2 \\ \vdots \\ y_m \mathbf{x}_m \end{bmatrix} \quad Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}.$$

Implementation (Maple)

```
CalcAlphaVector := proc(LPM::Matrix, Kernel, C)

local H,i,j,k,rows,a,b,c,bl,bu,Aeq,
      beq,Y,QPSolution;

rows := RowDimension(LPM);
H := Matrix(1..rows,1..rows);
a := Vector(2);
b := Vector(2);
beq := Vector([0]);
Y := Vector(rows);

for k from 1 to rows do
  Y[k] := LPM[k,3];
end do;
```

Implementation (Maple) cont.

```
for i from 1 to rows do
  for j from 1 to rows do

    a[1] := LPM[i,1];
    a[2] := LPM[i,2];

    b[1] := LPM[j,1];
    b[2] := LPM[j,2];

    H[i,j] := Y[i] * Y[j] * Kernel(a,b);

  end do;
end do;
```

Implementation (Maple) cont.

```
c   := Vector(1..rows, fill = -1);
bl  := Vector(1..rows, fill = 0);
bu  := Vector(1..rows, fill = C);
Aeq := convert(Transpose(Y),Matrix);

QPSolution := QPSolve([c,H],
              [NoUserValue,NoUserValue,Aeq,beq],
              [bl,bu]);

return QPSolution[2];

end proc:
```

Using SVM in R (e1071)

Example, how to use SVM classification in R:

```
library(e1071);

# load famous Iris Fisher data set
data(iris);
attach(iris);

# number of rows
n <- nrow(iris);
# number of folds
folds <- 10
samps <- sample(rep(1:folds, length=n),
                n, replace=FALSE)
```

Using SVM in R (e1071) cont.

```
# Using the first fold:
train <- iris[samps!=1,] # fit the model
test <- iris[samps==1,] # predict

# features 1 -> 4 from training set
x_train <- train[,1:4];
# class labels
y_train <- train[,5];

# determine the hyperplane
model <- svm(x_train,y_train,type="C-classification",
             cost=10,kernel="radial",
             probability = TRUE, scale = FALSE);
```

Using SVM in R (e1071) cont.

```
# features 1 -> 4 from test set (no class labels)
x_test <- test[,1:4];

# predict class labels for x_test
pred <- predict(model,x_test,decision.values = TRUE);

# get true class labels
y_true <- test[,5];

# check accuracy:
table(pred, y_true);
# determine test error
sum(pred != y_true)/length(y_true);
# do that for all folds, i.e. train <- iris[samps!=2,]
# test <- iris[samps==2,] , ........
```

SVM Multi-Class Classification

- A SVM is a binary classifier, that is, the class labels can only take two values: ± 1.
- Many real-world problems, however, have more than two classes (e.g. optical character recognition).

One Versus the Rest: To get M-class classifiers, construct set of binary classifiers $f^1, f^2, \ldots, f^M$, each trained to separate one class from rest.

Combine them to get a multi-class classification according to the maximal output *before* applying the sgn function.

$$\underset{j=1\ldots M}{\operatorname{argmax}}\, g^j(x), \text{ where } g^j(x) = \sum_{i=1}^{m} y_i \alpha_i^j k(x, x_i) + b^j.$$

and

$$f^j(x) = \operatorname{sgn}(g^j(x))$$

SVM Multi-Class Classification (cont.)

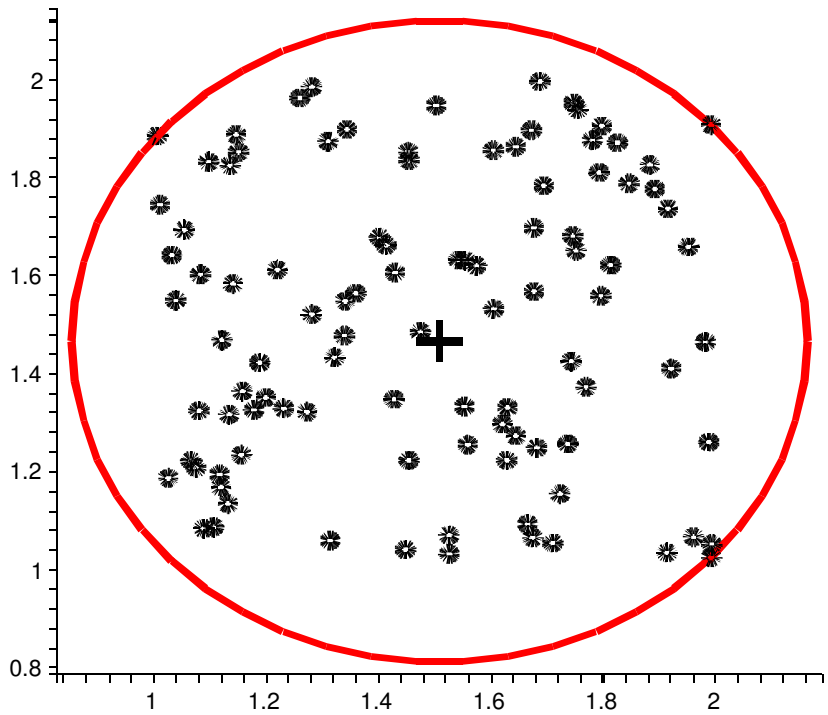
- Recall: $g^j(x)$ returns a signed real-valued value which can be interpreted as the distance from the separation (hyper)plane to the point x.
- Value can also be interpreted as a confidence value. The larger the value the more *confident* one is that the point x belong to the positive class.
- Hence, assign point x to the class whose confidence value is largest for this point.

SVM Pairwise Classification

- Train a classifier for each possible pair of classes.
- For M classes, this results in $\binom{M}{2} = \frac{(M-1)M}{2}$ binary classifiers.
- Classify an unknown point x by applying each of the $\binom{M}{2}$ binary classifiers and count how many times point x was assigned to that class label.
- Class label with highest count is then the considered label for the unknown point x.

One-Class SVM for Novelty Detection

Idea: enclose data with a hypersphere and classify new data as *normal* if it falls within the hypersphere and otherwise as anomalous data.



Minimum Enclosing Hypersphere

Given normal data $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m\} \in \mathbb{R}^d$ and let r be the radius of the hypersphere and $\mathbf{c} \in \mathbb{R}^d$ the center. To find the minimum enclosing hypersphere we have to solve the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & r^2 \\ \text{subject to} \quad & \|\Phi(\mathbf{x}_i) - \mathbf{c}\|^2 \leq r^2, \quad i = 1, \ldots, m. \end{aligned}$$

Lagrangian multiplier $\alpha_i \geq 0$ for each constraint

$$L(\mathbf{c}, r, \boldsymbol{\alpha}) = r^2 + \sum_{i=1}^{m} \alpha_i \left\{ \|\Phi(\mathbf{x}_i) - \mathbf{c}\|^2 - r^2 \right\}$$

Minimum Enclosing Hypersphere (cont.)

Setting the derivatives with respect to $\mathbf{c}$ and r to zero

$$\frac{\partial L(\mathbf{c}, r, \boldsymbol{\alpha})}{\partial \mathbf{c}} = 2\sum_{i=1}^{m} \alpha_i(\Phi(\mathbf{x}_i) - \mathbf{c}) = \mathbf{0}$$

$$\frac{\partial L(\mathbf{c}, r, \boldsymbol{\alpha})}{\partial r} = 2r\left(1 - \sum_{i=1}^{m} \alpha_i\right) = 0$$

one obtains the following equations

$$\sum_{i=1}^{m} \alpha_i = 1 \text{ and } \mathbf{c} = \sum_{i=1}^{m} \alpha_i \Phi(\mathbf{x}_i). \tag{2}$$

Minimum Enclosing Hypersphere (cont.)

Inserting relation (2) into

$$\begin{aligned} L(\mathbf{c}, r, \boldsymbol{\alpha}) &= r^2 + \sum_{i=1}^{m} \alpha_i \left\{ \|\Phi(\mathbf{x}_i) - \mathbf{c}\|^2 - r^2 \right\} \\ &= \sum_{i=1}^{m} \alpha_i \|\Phi(\mathbf{x}_i) - \mathbf{c}\|^2 \\ &= \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_i, \mathbf{x}_i) - \sum_{i,j=1}^{m} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) \end{aligned}$$

gives the dual form.

Minimum Enclosing Hypersphere (cont.)

To find $\boldsymbol{\alpha}$ in dual form, solve optimization problem:

$$\text{maximize} \qquad W(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_i, \mathbf{x}_i) - \sum_{i,j=1}^{m} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{subject to} \qquad \sum_{i=1}^{m} \alpha_i = 1, \text{ and } \alpha_i \geq 0, \quad i = 1, \ldots, m.$$

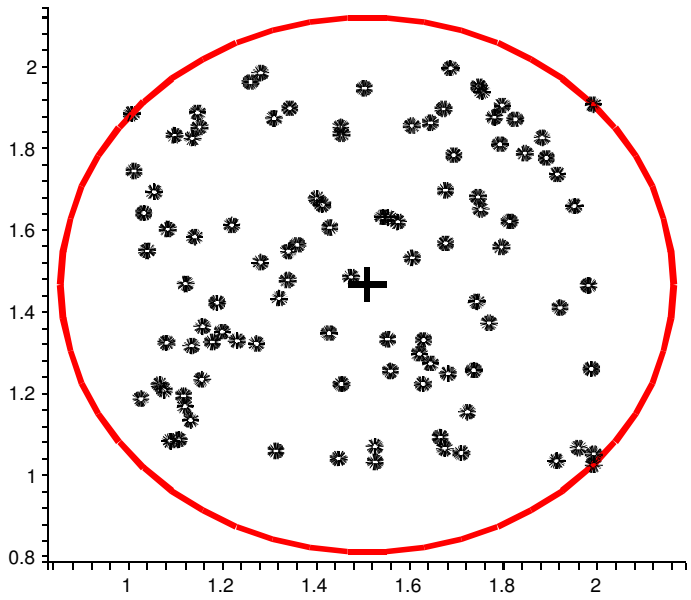
Recall: Lagrange multiplier can be non-zero only if the corresponding inequality constraint is an equality at the solution.

Minimum Enclosing Hypersphere (cont.)

The KKT complementarity conditions are satisfied by the optimal solutions $\boldsymbol{\alpha}, (\mathbf{c}, r)$

$$\alpha_i \left\{ \|\Phi(\mathbf{x}_i) - \mathbf{c}\|^2 - r^2 \right\}, \quad i = 1, \ldots, m.$$

This implies that only training examples $\mathbf{x}_i$ that lie on the surface of the optimal hypersphere have their corresponding $\alpha_i > 0$.



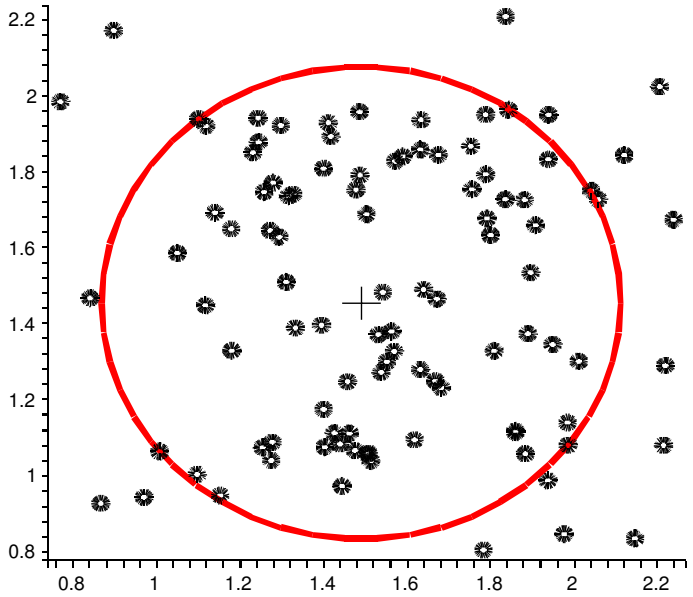
Decision Function

$$
\begin{aligned}
f(\mathbf{x}) &= \operatorname{sgn}(r^2 - \|\Phi(\mathbf{x}) - \mathbf{c}\|^2) \\
&= \operatorname{sgn}\left(r^2 - \left\{(\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x})) - 2\sum_{i=1}^{m} \alpha_i (\Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}_i))\right.\right. \\
&\quad \left.\left. + \sum_{i,j=1}^{m} \alpha_i \alpha_j (\Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j))\right\}\right) \\
&= \operatorname{sgn}\left(r^2 - \left\{k(\mathbf{x}, \mathbf{x}) - 2\sum_{i=1}^{m} \alpha_i k(\mathbf{x}, \mathbf{x}_i)\right.\right. \\
&\quad \left.\left. + \sum_{i,j=1}^{m} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)\right\}\right)
\end{aligned}
$$

Soft Enclosing Hypersphere

If we have some noise in our training set the "hard" enclosing hypersphere approach may force a larger radius than should really be needed. In other words, the solution would not be *robust*.

Aim: Find minimum enclosing hypersphere that contains (allmost) all training examples, but not some small portion of extreme training examples.



Soft Enclosing Hypersphere (cont.)

Introduce slack variables $\boldsymbol{\xi}, \xi_i \geq 0, i = 1, \ldots, m$

$$\begin{aligned} \text{minimize} \quad & r^2 + C \sum_{i=1}^{m} \xi_i \\ \text{subject to} \quad & \|\Phi(\mathbf{x}_i) - \mathbf{c}\|^2 \leq r^2 + \xi_i, \quad \xi_i \geq 0,\ i = 1, \ldots, m. \end{aligned}$$

Lagrangian multiplier $\alpha_i, \beta_i \geq 0$ for each constraint

$$\begin{aligned} L(\mathbf{c}, r, \boldsymbol{\alpha}, \boldsymbol{\beta}) \quad = \quad & r^2 + C \sum_{i=1}^{m} \xi_i \\ & + \sum_{i=1}^{m} \alpha_i \left\{ \|\Phi(\mathbf{x}_i) - \mathbf{c}\|^2 - r^2 - \xi_i \right\} - \sum_{i=1}^{m} \beta_i \xi_i \end{aligned}$$

Soft Enclosing Hypersphere (cont.)

Setting partial derivatives to $\mathbf{0}$ gives

$$\sum_{i=1}^{m} \alpha_i = 1, \quad \mathbf{c} = \sum_{i=1}^{m} \alpha_i \Phi(\mathbf{x}_i)$$

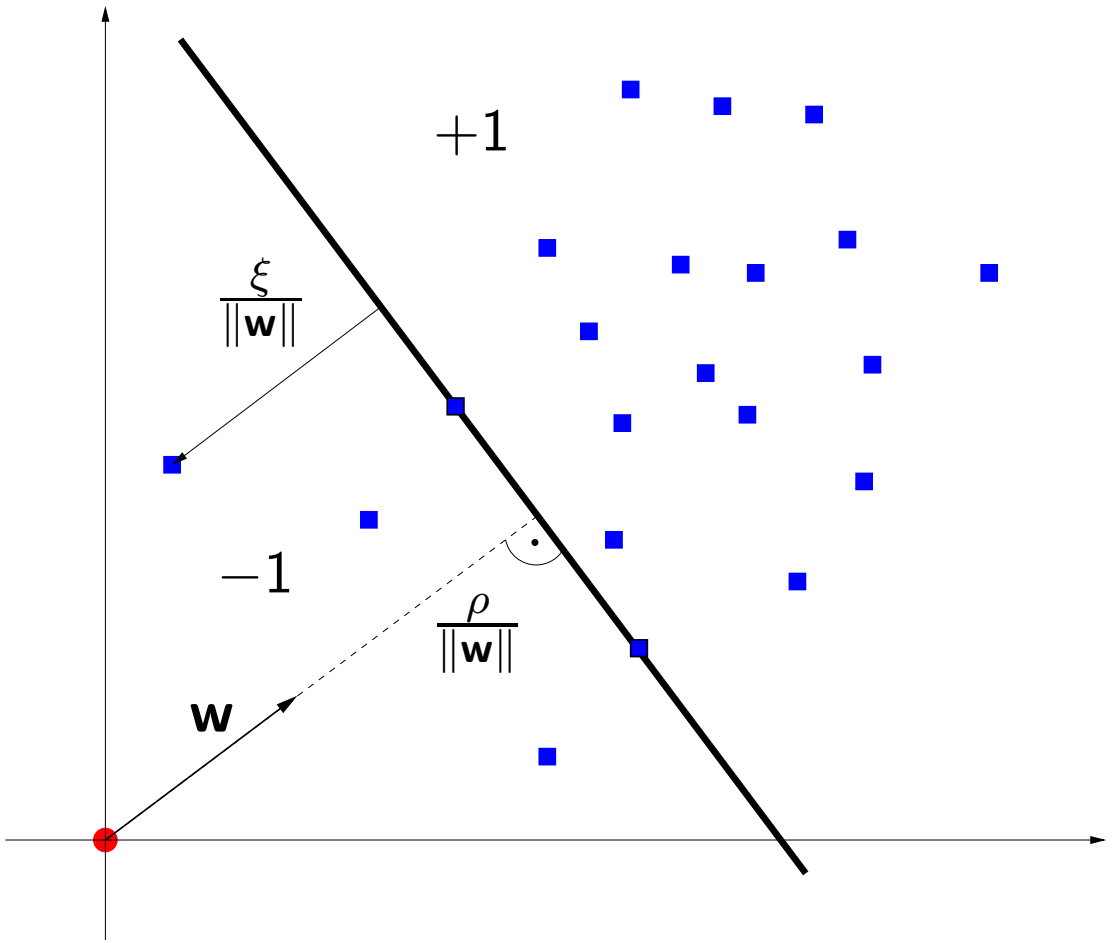
This leads to the dual form

$$\text{minimize} \quad \sum_{i,j=1}^{m} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^{m} \alpha_i k(\mathbf{x}_i, \mathbf{x}_i)$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{m} \alpha_i = 1$$

Hyperplane One-Class SVM

Idea: Separate in high-dimensional feature space $\mathcal{F}$, the points from the origin (circled point) with a maximum distance, and allow $\nu \cdot m$ many "outliers" which lie between the origin and the hyperplane, i.e. the -1 side.



Hyperplane One-Class SVM (cont.)

Normal vector of the hyperplane is determined by solving the primal quadratic optimization problem

$$\text{minimize} \qquad \tfrac{1}{2}\|\mathbf{w}\|^2 + \tfrac{1}{\nu m}\textstyle\sum_i \xi_i - \rho \tag{3}$$

$$\text{subject to} \quad (\mathbf{w}\cdot\Phi(\mathbf{x}_i)) \geq \rho - \xi_i, \xi_i > 0,\ i = 1,\ldots,m. \tag{4}$$

Lagrangian multiplier $\alpha_i, \beta_i \geq 0$ for each constraint

$$\begin{aligned} L(\mathbf{w},\boldsymbol{\xi},\rho,\boldsymbol{\alpha},\boldsymbol{\beta}) &= \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{\nu m}\sum_i \xi_i - \rho \\ &\quad - \sum_{i=1}^{m}\alpha_i((\mathbf{w}\cdot\Phi(\mathbf{x}_i)) - \rho + \xi_i) - \sum_{i=1}^{m}\beta_i\xi_i \end{aligned}$$

Reformulating (3) and (4) to a dual optimization problem in terms of a kernel function $k(\cdot,\cdot)$, one obtains

Hyperplane One-Class SVM (cont.)

$$\text{maximize} \qquad \frac{1}{2}\sum_{i,j=1}^{m}\alpha_i\alpha_j k(\mathbf{x}_i,\mathbf{x}_j) \tag{5}$$

$$\text{subject to } 0 \leq \alpha_i \leq \frac{1}{\nu m}, i = 1,\ldots,m \text{ and } \sum_{i=1}^{m}\alpha_i = 1. \tag{6}$$

Differentiating the primal with respect to $\mathbf{w}$, one gets $\mathbf{w} = \sum_{i=1}^{m}\alpha_i\Phi(\mathbf{x}_i)$.

Recall KKT theorem: For $\alpha_i > 0$ the corresponding pattern $\mathbf{x}_i$ satisfies

$$\rho = (\mathbf{w}\cdot\Phi(\mathbf{x}_i)) = \sum_{j=1}^{m}\alpha_j k(x_j, x_i)$$

Hyperplane One-Class SVM (cont.)

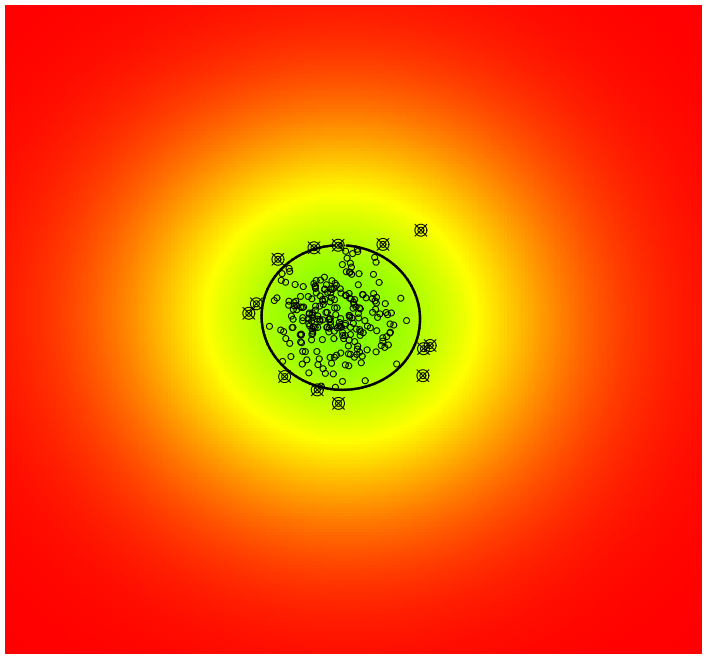
The decision function (left/right side of the hyperplane):

$$\begin{aligned} f(\mathbf{x}) &= \text{sgn}((\mathbf{w} \cdot \Phi(\mathbf{x}_i)) - \rho) \\ &= \text{sgn}\left(\sum_{i=1}^{m} \alpha_i k(\mathbf{x}_i, \mathbf{x}) - \rho\right) \end{aligned}$$

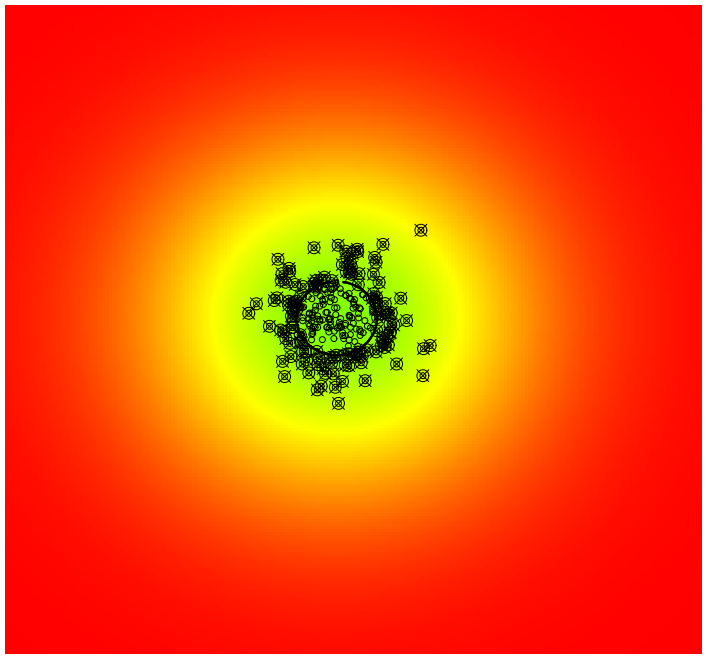
ν-Property:

- ν is an upper bound on the fraction of outliers.
- ν is a lower bound on the fraction of Support Vectors.

Hyperplane One-Class SVM Example



$\nu = 0.05$



$\nu = 0.5$

Support Vector Regression

Basic idea: map the data $\mathbf{x}$ into a high-dimensional feature space $\mathcal{F}$ via a nonlinear mapping Φ, and do *linear* regression in this space.

$$f(\mathbf{x}) = (\mathbf{w} \cdot \Phi(\mathbf{x})) + b \text{ with } \Phi : \mathbb{R}^d \rightarrow \mathcal{F}, \mathbf{w} \in \mathcal{F}.$$

Linear regression in a high dimensional feature space corresponds to *nonlinear* regression in the low dimensional space $\mathbb{R}^d$.

Vapnik's ϵ-insensitive loss function:

$$|y - f(\mathbf{x})|_\epsilon := \max\{0, |y - f(\mathbf{x})| - \epsilon\}$$

Find function $f(\mathbf{x})$ that has at most ϵ deviation from all the targets y_i

Support Vector Regression (cont.)

Estimate linear regression $f(\mathbf{x}) = (\mathbf{w} \cdot \Phi(\mathbf{x})) + b$ leads to the problem of minimizing the term

$$\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} |y_i - f(\mathbf{x}_i)|_\epsilon$$

In the soft margin case one needs two types of slack variables $(\boldsymbol{\xi}, \boldsymbol{\xi}^*)$ for the two cases $f(\mathbf{x}_i) - y_i > \epsilon$ and $y_i - f(\mathbf{x}_i) > \epsilon$.

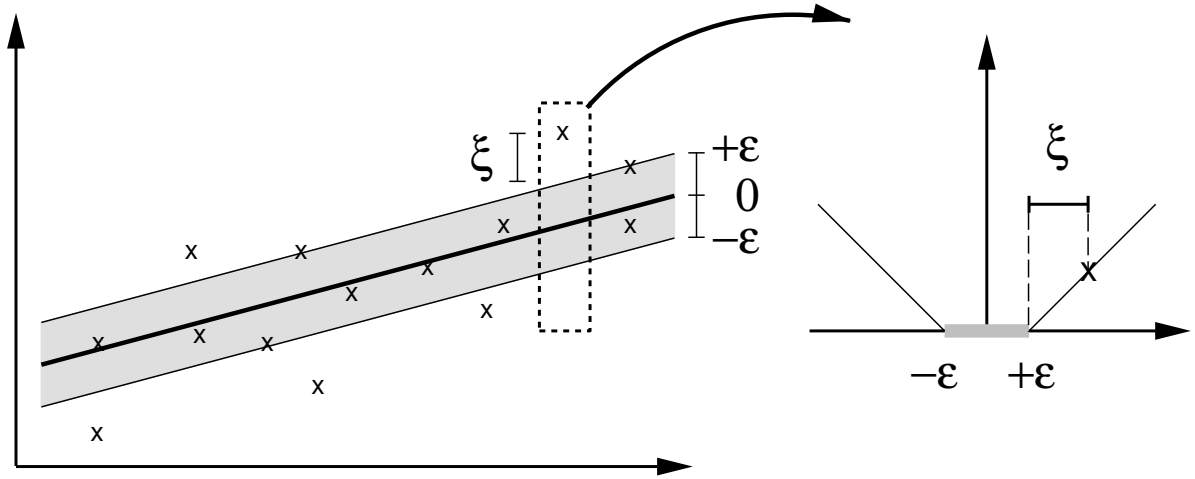


Figure is taken from Schölkopf's and Smola's book (Learning with Kernels)

Support Vector Regression (cont.)

Optimization problem is given by:

$$
\begin{aligned}
\text{minimize} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C \cdot \sum_{i=1}^{n}(\xi_i + \xi_i^*) \\
\text{subject to} \quad & f(\mathbf{x}_i) - y_i \leq \epsilon + \xi_i \\
& y_i - f(\mathbf{x}_i) \leq \epsilon + \xi_i^* \\
& \xi_i, \xi_i^* \geq 0 \qquad \text{for all } i = 1, \ldots, n
\end{aligned}
$$

Support Vector Regression (cont.)

Introducing Lagrange multipliers $\boldsymbol{\alpha}, \boldsymbol{\alpha}^*$ (dual form):

$$\begin{aligned}
\text{maximize} \quad & -\epsilon \sum_{i=1}^{n}(\alpha_i^* + \alpha_i) + \sum_{i=1}^{n}(\alpha_i^* - \alpha_i)y_i \\
& -\frac{1}{2}\sum_{i,j}^{n}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)k(\mathbf{x}_i, \mathbf{x}_j) \\
\text{subject to} \quad & 0 \leq \alpha_i, \alpha_i^* \leq C \text{ for all } i = 1, \ldots, n \text{ and} \\
& \sum_{i=1}^{n}(\alpha_i^* - \alpha_i) = 0
\end{aligned}$$

Regression estimate takes the form

$$f(\mathbf{x}) = \sum_{i=1}^{n}(\alpha_i^* - \alpha_i)k(\mathbf{x}_i, \mathbf{x}) + b$$

Support Vector Regression (cont.)

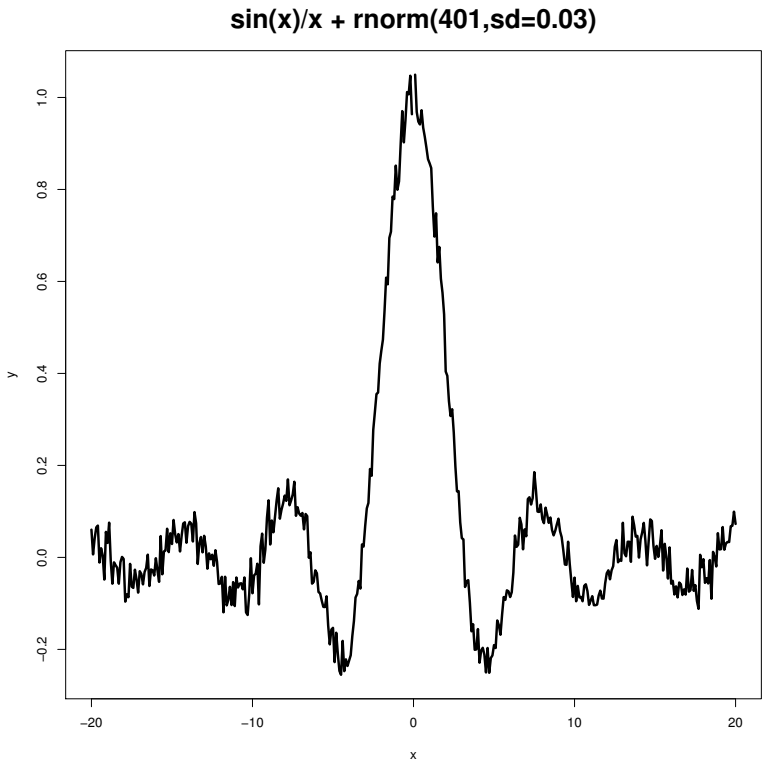
Offset b can be computed by exploiting Karush-Kuhn-Tucker conditions: $f(\mathbf{x}_i) - y_i \leq \epsilon + \xi_i$ becomes an equality with $\xi_i = 0$ if $0 < \alpha_i < C$ and $y_i - f(\mathbf{x}_i) \leq \epsilon + \xi_i^*$ becomes an equality with $\xi_i^* = 0$ if $0 < \alpha_i^* < C$ that is:

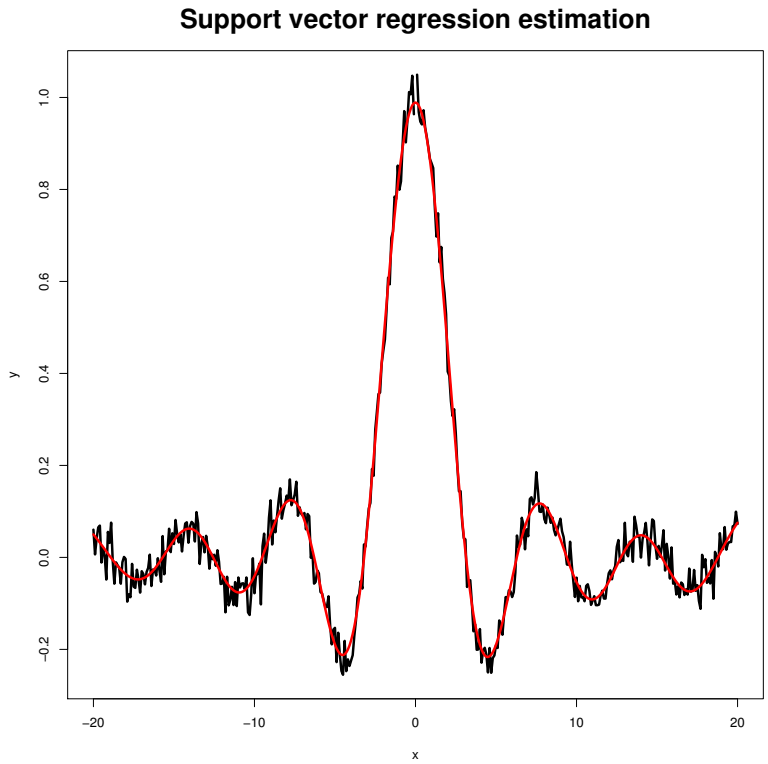
$$\begin{aligned} \alpha_i(\epsilon + \xi_i - y_i + (\mathbf{w} \cdot \Phi(\mathbf{x}_i)) + b) &= 0 \\ \alpha_i^*(\epsilon + \xi_i^* + y_i - (\mathbf{w} \cdot \Phi(\mathbf{x}_i)) - b) &= 0 \end{aligned}$$

and leads to solution

$$\begin{aligned} b &= y_i - (\mathbf{w} \cdot \Phi(\mathbf{x}_i)) - \epsilon \quad \text{for } \alpha_i \in (0, C) \\ b &= y_i - (\mathbf{w} \cdot \Phi(\mathbf{x}_i)) + \epsilon \quad \text{for } \alpha_i^* \in (0, C) \end{aligned}$$

Support Vector Regression Example






```
library(kernlab);
x <- seq(-20,20,0.1);
y <- sin(x)/x + rnorm(401,sd=0.03);
# train SVM
reg.svm <- ksvm(x,y,epsilon=0.01,kpar=list(sigma=16),cross=3);
plot(x,y,type="l",lwd=3);
lines(x,predict(reg.svm,x),col="red",lwd=3);
```