Machine Learning

Bert Kappen
SNN Radboud University, Nijmegen
Gatsby Unit, UCL London

October 9, 2023
Course setup

- 3 ec course. 7 weeks.

- Lectures are prerecorded. Available through brightspace.

- Examination based on weekly exercises only
  - You make a group of maximal 3 persons
  - Each week, you make the exercises with your group.
  - You can ask questions in tutorial class
  - Your group hands them in before the next tutorial class. Hard rule, because answers may be discussed in tutorial class.

- All course materials (slides, exercises) and schedule via http://www.snn.ru.nl/~bertk/machinellearning/
Content

1. Probabilities, Bayes rule, information theory

2. Model selection

3. Classification, perceptron, gradient descent learning rules

4. Multi layered perceptrons

5. Deep learning

6. Graphical models, latent variable models, EM

7. Variational autoencoders
Lecture 1a

Based on Mackay ch. 2: Probability, Entropy and Inference

- Probabilities

- Bayesian inference, inverse probabilities, Bayes rule
Forward probabilities

Forward probabilities is the usual way to compute the probabilities of possible outcomes given a probability model

\[ p(x|f) \rightarrow p(\text{outcome}|f) \]

Exercise 2.4

Urn with \( B \) black balls and \( W \) white balls and \( K = W + B \).

Draw \( N \) times a ball from the urn with replacement. What is the probability to draw \( N_B \) black balls?
Forward probabilities

Define $f = B/K$ the fraction of black balls in the urn. Then

$$p(N_B = 0) = (1 - f)^N$$
$$p(N_B = 1) = f(1 - f)^{N-1}N$$
$$p(N_B) = \binom{N}{N_B} f^{N_B}(1 - f)^{N-N_B}$$

Expected value and variance:

$$\mathbb{E} N_B = \sum_{N_B=0}^{N} p(N_B)N_B = \ldots = N f$$
$$\nabla N_B = \sum_{N_B=0}^{N} p(N_B)(N_B - N f)^2 = \ldots = N f(1 - f)$$

Suppose $K = 10, B = 2, f = 0.2$.
When $N = 5, N_B \approx 1 \pm 0.9$. When $N = 400, N_B \approx 80 \pm 8$. 
Inverse probabilities

Inverse probabilities do the converse: given a specific outcome, what is the probability that this has been generated by a model with parameters $f$?

$$\text{outcome} \rightarrow p(f|\text{outcome})$$

**Exercise 2.6**

11 urns $u = 0, \ldots, 10$ each with 10 balls. Urn $u$ has $u$ black balls and $10 - u$ white balls. Select one urn at random, and draw $N$ times with replacement from that urn. The outcome is that after $N = 10$ draws there are $N_B = 3$ black balls. What is the probability that urn $u$ was selected?
We treat $u$ and $N_B$ as random variables. $N$ is given and provides the context or condition in which the probabilities are calculated.

\[
p(u, N_B|N) = p(u|N)p(N_B|u, N)
\]

\[
p(u|N_B, N) = \frac{p(u|N)p(N_B|u, N)}{p(N_B|N)}
\]

\[
p(N_B|u, N) = \binom{N}{N_B} f_u^{N_B} (1 - f_u)^{N-N_B} \quad f_u = \frac{u}{10}
\]

\[
p(u|N) = p(u) = \frac{1}{11}
\]

\[
p(N_B|N) = \sum_{u=0}^{10} p(u) p(N_B|u, N) = \frac{1}{11} \binom{N}{N_B} \sum_{u=0}^{10} f_u^{N_B} (1 - f_u)^{N-N_B}
\]

\[
p(u|N_B, N) = \frac{f_u^{N_B} (1 - f_u)^{N-N_B}}{\sum_{u' = 0}^{10} f_{u'}^{N_B} (1 - f_{u'})^{N-N_B}}
\]
Left: Joint probability $p(u, N_B|N)$. Right: conditional probability $p(u|N_B, N)$. 
Bayesian inference

Note, that our 'inference' has resulted in a distribution over $u$. We do not know which urn was selected, only probabilities. This is the best we can do given the data. This is called Bayesian inference.

In general, when the models are parametrized by $\theta$, the procedure is

1. A given data set

2. Specify the prior over models $p(\theta)$.

3. Compute the likelihood of the observed data under the model with parameters $\theta$: $p(\text{data}|\theta)$.

4. Compute the posterior using Bayes’ rule

$$p(\theta|\text{data}) = \frac{p(\text{data} | \theta) p(\theta)}{p(\text{data})} \quad p(\text{data}) = \int d\theta p(\text{data} | \theta) p(\theta)$$
Exercise 2.6 continued
We draw another ball from the same urn. Given the observations so far, what is the probability that it is black?

This is computed from the posterior:

\[
p(\text{black}|N_B = 3, N = 10) = \sum_{u=0}^{10} p(\text{black}|u, N_B, N) p(u|N_B, N)
\]

\[
= \sum_{u=0}^{10} f_u p(u|N_B, N)
\]

\[
= \sum_{u=0}^{10} f_u \frac{f_u^{N_B} (1 - f_u)^{N-N_B}}{\sum_{u'=0}^{10} f_{u'}^{N_B} (1 - f_{u'})^{N-N_B}} = 0.333
\]

Compare with prediction from most likely urn \((u = 3)\) would give

\[
p(\text{black}|u = 3) = f_{u=3} = 3/10 = 0.3
\]
Exercise 2.7. The bent coin

Bent coin has probability $f$ to come up head. We do not know $f$. We toss $N$ times and get $N_H$ times head. What is $f$?
Exercise 2.7. The bent coin

Bent coin has probability $f$ to come up head. We do not know $f$. We toss $N$ times and get $N_H$ times head. What is $f$?

$f$ is fixed but unknown. The key conceptual step is to treat $f$ as a random variable. Assume a prior over $f$: $p(f)$. $p(f)$ is our (subjective) prior belief in the value of $f$.

Given $f$ and $N$, we know the likelihood of the observation:

$$p(N_H|f, N) = \binom{N}{N_H} f^{N_H} (1 - f)^{N - N_H}$$

The posterior is

$$p(f|N_H, N) = \frac{p(N_H|f, N)p(f)}{p(N_H|N)}$$
Intermezzo: the Beta distribution

The Beta distribution is a probability density over a continuous random variable $x \in [0, 1]$ defined by two shape parameters $a, b > 0$.

$$
\text{Beta}(x|a, b) = \frac{\Gamma(a + b)}{\Gamma(a)\Gamma(b)} x^{a-1} (1 - x)^{b-1} \quad 0 \leq x \leq 1
$$

The prefactor ensures normalisation

$$
\int_0^1 dx x^{a-1} (1 - x)^{b-1} = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a + b)}
$$
Intermezzo: the Beta distribution

Mean and variance

\[ \mathbb{E}x = \frac{a}{a + b} \]
\[ \mathbb{V}x = \frac{ab}{(a + b)^2(a + b + 1)} \]

Note, that with \( a = fN, b = (1 - f)N \) we get

\[ \mathbb{E}x = f \quad \mathbb{V}x = \frac{f(1 - f)}{N + 1} \]

becomes more peaked for large \( N \).
Exercise 2.7. The bent coin

Assume that we have no prior knowledge of \( f \). Therefore \( p(f) = 1 \).

The posterior

\[
p(f|N_H, N) = \frac{p(N_H|f, N)p(f)}{p(N_H|N)} = \frac{\binom{N}{N_H} f^{N_H} (1 - f)^{N - N_H}}{p(N_H|N)}
\]

\[
= \frac{\Gamma(a + b)}{\Gamma(a)\Gamma(b)} f^{a-1} (1 - f)^{b-1}
\]

with \( N_H = a - 1, N - N_H = b - 1 \).

From this we can infer that

\[
p(N_H|N) = \binom{N}{N_H} \frac{\Gamma(a)\Gamma(b)}{\Gamma(a + b)} = \frac{N!}{N_H!(N - N_H)!} \frac{N_H!(N - N_H)!}{(N + 1)!} = \frac{1}{N + 1}
\]

where we used \( \Gamma(x) = (x - 1)! \) when \( x \) is integer.

Since \( p(N_H|N) = \int df p(N_H|f, N)p(f) \) we conclude that when integrating over all models, the probability of all outcomes are equally likely.
Exercise 2.7. The bent coin

So, given our experiment where we have thrown the coin $N$ times and have observed $N_H$ times 'head'. What is the probability to observe another 'head'?

\[
p(H|N_H, N) = \int df p(H|f)p(f|N_H, N)
\]

\[
= \int df f \text{Beta}(f|a = N_H + 1, b = N - N_H + 1) = \frac{a}{a + b} = \frac{N_H + 1}{N + 2}
\]

Suppose $N = N_H = 1$.

Naive answer is $f = 1$ and thus $p(H|f) = 1$.

Bayesian answer is $p(f|N_H, N) = 2f$ and $p(H|N_H, H) = \frac{2}{3}$
Lecture 1b

Based on Mackay ch. 2: Probability, Entropy and Inference

- Entropy as information
- Maximum entropy, and its relation to exponential models
- the KL divergence
  - use in variational approximation
  - relation to maximum likelihood
Entropy as information

Information is a measure of the 'degree of surprise' that a certain value gives us *given that we know the distribution*.

Unlikely events are informative, likely events less so. Thus, information decreases with the probability of the event.

Given $p(x) = \delta(x)$ (no uncertainty), observing $x$ gives us no additional information.

Let us denote $h(x)$ the information of $x$. Then if $x, y$ are independent, we want information to be additive $h(x, y) = h(x) + h(y)$. Since $p(x, y) = p(x)p(y)$ we see that

$$h(x) = -\log_2 p(x)$$

is a good candidate to quantify the information in $x$.

The expected information is

$$H[x] := \langle -\log_2 p \rangle = - \sum_x p(x) \log_2 p(x)$$

is the entropy of the distribution $p$. 
When $p$ is sharply peaked ($p(x_1) = 1, p(x_2) = \ldots = p(x_M) = 0$) then the entropy is

$$H[x] = -1 \log 1 - (M - 1)0 \log 0 = 0$$

When $p$ is flat ($p(x_i) = 1/M$) the entropy is maximal

$$H[x] = -M \frac{1}{M} \log \frac{1}{M} = \log M$$

$^1$log is $\log_2$ is this section on entropy as information.
Entropy as information

Entropy plays a role by the efficient transmission of information. Suppose we have a channel and we want to send symbols $x$ over this channel. These symbols are assumed to be randomly generated by a distribution $p(x)$.

So, there are some common symbols that are sent very often, and some other symbols that are sent rarely.

We want to encode each symbol as a binary string. Intuitively, we want to use short strings for common symbols and longer strings for rare symbols. The question is: how to do this optimally?

*Noiseless coding theorem:* Entropy is a lower bound on the average number of bits per symbol (Shannon 1948).
Entropy as information

Example 1: $x$ can have 8 values with equal probability, then $H(x) = -8 \times \frac{1}{8} \log \frac{1}{8} = 3$. We say that the distribution carries 3 bits of information.

The Thm says that we need on average at least 3 bits per symbol. In fact, we can use the normal binary representation of $x$:

$$000, 001, 010, 011, 100, 101, 110, 111$$

to use exactly 3 bits per symbol.
Entropy as information

Example 2: $x$ can have 8 values with probabilities $(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{16}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64}, \frac{1}{64})$. Then

$$H(x) = -\frac{1}{2} \log \frac{1}{2} - \frac{1}{4} \log \frac{1}{4} - \frac{1}{8} \log \frac{1}{8} - \frac{1}{16} \log \frac{1}{16} - \frac{4}{64} \log \frac{1}{64} = 2 \text{bits}$$

So the Thm says that we need on average at least two bits per symbol.

We can use the dumb code as before, but that requires 3 bits per symbol.

How can we construct a code that saturates the Shannon bound and only uses on average 2 bits per symbol?
Entropy as information

The idea is to encode each symbol $x$ with a string with length equal to its information $-\log p(x)$. Thus, for instance

<table>
<thead>
<tr>
<th>$x$</th>
<th>$-\log p(x)$</th>
<th>code</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>110</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1110</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>111100</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>111101</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>111110</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>111111</td>
</tr>
</tbody>
</table>

Then by construction:

$$\text{Av.code length} = \sum_x p(x) \text{length}(x) = -\sum_x p(x) \log_2 p(x) = H$$

and this is by the Thm the best you can do.
Maximum entropy and exponential family

Suppose, we are given some data, and we wish to find a distribution \( p \) to describe these data. One such approach is the maximum entropy approach, that yields the so-called exponential family models.

Denote the data as \( D = (x_1, \ldots, x_N) \). From the data, we can compute certain statistics

\[
S_a = \frac{1}{N} \sum_{i=1}^{N} \phi_a(x_i)
\]

that are averages over the data.

For instance, useful statistics might be the mean and variance of the data

\[
\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x_i \quad \sigma_x^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \bar{x})^2 = \frac{1}{N} \sum_{i} x_i^2 - \bar{x}^2
\]

\[
S_1 = \bar{x} \quad \phi_1(x) = x \quad S_2 = \sigma_x^2 + \bar{x}^2 \quad \phi_2(x) = x^2
\]
We want to find a distribution $p$ that reproduces these statistics correctly. That is to say, $p$ should be such that \[ \mathbb{E}\phi_a = \sum_x p(x)\phi_a(x) = S_a \quad a = 1, 2, \ldots \]

but in general this will not determine $p$ uniquely.

The maximum entropy solution is the broadest (least informative) distribution consistent with the statistics constraints. Thus, we wish to find

\[
\min_{p(x), \mathbb{E}\phi_a=S_a, a=0,1,2,\ldots} \int dx p(x) \log p(x)
\]

where $a = 0$ is the normalization constraint $\sum_x p(x) = 1$. This is a constrained optimization problem, that we can solve with Lagrange multipliers.

\[\text{We use } \sum_x, \text{ suggesting that } x \text{ takes discrete values. The argument can be also made for } x \text{ (multi-dimensional) continuous by simply replacing } \sum_x \text{ with } \int dx.\]

\[\phi_0(x) = 1, S_0 = 1\]
Intermezzo: Lagrange multipliers

Minimize \(f(x)\) with respect to \(x\) under the constraint: \(g(x) = 0\)

\[
\min_{x, g(x)=0} f(x)
\]

Define the Lagrangian,

\[
L(x, \lambda) = f(x) + \lambda g(x)
\]

\(\lambda\) is called a Lagrange multiplier.

Note, that

\[
\max_{\lambda} L(x, \lambda) = f(x) \quad \text{if} \quad g(x) = 0
\]

\[
\max_{\lambda} L(x, \lambda) = \infty \quad \text{otherwise}
\]

Since we restrict \(x\) to \(g(x) = 0\) we can write

\[
\min_{x, g(x)=0} f(x) = \min_{x} \max_{\lambda} L(x, \lambda)
\]
Intermezzo: Lagrange multipliers

When the functions $f, g$ are convex we are allowed to interchange the min and max:

$$\min_x \max_\lambda L(x, \lambda) = \max_\lambda \min_x L(x, \lambda)$$

Procedure:

1. Minimize $L(x, \lambda)$ w.r.t $x$. This yields a solution $x(\lambda)$ depending on $\lambda$.

2. Maximize $L(x(\lambda), \lambda)$ w.r.t. $\lambda$. The solution $\lambda^*$ is precisely such that $g(x(\lambda^*)) = 0$.

The solution of the constraint optimization problem is

$$x^* = x(\lambda^*)$$

When there are multiple constraints one introduces a Lagrange multiplier for each constraint:

$$\min_{x, g_1(x)=0, \ldots, g_M(x)=0} f(x) \rightarrow \min_x \max_{\lambda_1, \ldots, \lambda_M} f(x) + \sum_{a=1}^M \lambda_a g_a(x)$$
Minimize $f(x_1, x_2) = x_1^2 + x_2^2$ subject to $g(x_1, x_2) = x_1 + x_2 - 1 = 0$.

Lagrangian: $L(x_1, x_2, \lambda) = x_1^2 + x_2^2 + \lambda(x_1 + x_2 - 1)$

1. Maximize $L$ w.r.t. $x_{1,2}$ gives $x_1(\lambda) = x_2(\lambda) = -\frac{1}{2}\lambda$.

2. Plug into constraint: $x_1(\lambda) + x_2(\lambda) - 1$ yields $\lambda^* = -1$.
   The solution is $x_1^* = x_1(\lambda^*) = \frac{1}{2}$ and $x_1^* = x_1(\lambda^*) = \frac{1}{2}$
Maximum entropy and exponential family

The Lagrangian for the maximum entropy problem takes the form

\[ L(p, \lambda) = \sum_x p(x) \log p(x) + \sum_{a=0}^M \lambda_a \left( \sum_x p(x)\phi_a(x) - S_a \right) \]

We repeat the previous steps:

1. Minimizing \( L \) wrt \( p(x) \) yields \( p(x) = \exp \left( -1 - \lambda_0 - \sum_{a=1}^M \lambda_a \phi_a(x) \right) \).

2. \( A = \lambda_0 + 1 \) is fixed by the normalization condition \( \sum_x p(x) = 1 \). Thus

\[ p(x) = \exp \left( - \sum_{a=1}^M \lambda_a \phi_a(x) - A \right) \quad A = \log \sum_x \exp \left( - \sum_{a=1}^M \lambda_a \phi_a(x) \right) \]

This is called the exponential family model and \( A \) is the log partition sum.

The other \( \lambda_a \) are determined implicitly by the constraints \( \sum_x p(x)\phi_a(x) = S_a \).
Examples of exponential family distributions

Many distributions have an interpretation in terms of maximum entropy:

1. The Bernoilli distribution. \( x = 0, 1 \) is a binary variable and \( p(x) = f^x(1 - f)^{1-x} \).
   We can write
   \[
   p(x) = \exp\left( x \log \frac{f}{1 - f} + \log(1 - f) \right)
   \]
   Thus \( \phi_1(x) = x, \lambda_1 = -\log \frac{f}{1 - f}, A = -\log(1 - f) \). It is the max ent distribution on a binary variable for a given mean value. But in this case the mean determines the distribution completely!

2. The Gaussian distribution
   \[
   p(x) \propto \exp\left( -\frac{(x - \mu)^2}{2\sigma^2} \right) \propto \exp\left( \frac{\mu}{\sigma^2}x - \frac{1}{2\sigma^2}x^2 \right) = \exp\left( -\lambda_1\phi_1(x) - \lambda_2\phi_2(x) \right)
   \]
   \[\phi_1(x) = x, \phi_2(x) = x^2, \lambda_1 = -\frac{\mu}{\sigma^2}, \lambda_2 = \frac{1}{2\sigma^2}.\]
   It is the max ent distribution on a continuous variable with given mean and variance.
KL-divergence

Relative entropy or Kullback-Leibler (KL) divergence:

$$KL(p||q) = \sum_i p_i \log \left( \frac{p_i}{q_i} \right)$$

- We need that when $q_i = 0$ also $p_i = 0$. Otherwise $p_i \log \left( \frac{p_i}{q_i} \right)$ is not defined. (The reverse is no problem: when $p_i = 0$ and $q_i \neq 0$ we have $p_i \log \left( \frac{p_i}{q_i} \right) = 0$)

- $KL(p||q) \neq KL(q||p)$

- $KL(p||q) \geq 0$, $KL(p||q) = 0 \iff p = q$ (use convex functions)

- with continuous variables: $KL(p||q) = - \int p(x) \ln \left\{ \frac{q(x)}{p(x)} \right\} dx$
**Convex functions**

Convex function: every chord lies on or above the function.

\[ f \text{ is convex } \iff f(\lambda a + (1 - \lambda)b) \leq \lambda f(a) + (1 - \lambda)f(b) \quad \forall \lambda \in [0, 1], \forall a, b \]

- Examples: \( f(x) = ax + b \), \( f(x) = x^2 \), \( f(x) = -\ln(x) \) and \( f(x) = x \ln(x) \)

- Convex: \( \cup \) shaped. Concave: \( \cap \) shaped.

- Convex \( \iff \) second derivative non-negative.
Convex functions/Jensen’s inequality

Convex functions satisfy *Jensen’s inequality* (Proof see Ex. Mackay 2.14).

\[
f \left( \sum_{i=1}^{M} \lambda_i x_i \right) \leq \sum_{i=1}^{M} \lambda_i f(x_i)
\]

where \( \lambda_i \geq 0, \sum \lambda_i = 1 \), for any set points \( x_i \).

In other words:

\[
f(\langle x \rangle) \leq \langle f(x) \rangle
\]

We can use Jensen’s inequality to show that \( \text{KL}(p||q) \geq 0 \) (Ex. Mackay 2.26). Use \( \lambda_i = p_i \), making use of the fact that \( -\ln(x) \) is convex:

\[
\text{KL}(p||q) = -\sum_i p_i \ln \left( \frac{q_i}{p_i} \right) \geq -\ln \left( \sum_i p_i \frac{q_i}{p_i} \right) = -\ln \left( \sum_i q_i \right) = 0
\]
The variational approximation: Approximate one distribution with another

Often in machine learning we encounter distributions that are too complex to use. One can then approximate such a complex distribution by a simpler distribution.

For a given complex distribution \( p(x) \), one way to define the approximation is to consider a parametrized set of distributions \( q(x|\theta) \) and to find

\[
\theta = \arg\min_{\theta} KL(q|p) \quad KL(q|p) = \sum_x q(x) \log q(x) - \sum_x q(x) \log p(x)
\]

Note, that minimizing \( KL(q|p) \) is equivalent to maximizing the entropy of \( q \) subject to a constraint on the 'energy' \( \log p(x) \). This is called the variational approximation.

The variational approximation is used extensively in machine learning, as we will discuss when we treat the EM algorithm and variational EM algorithms later.
The variational approximation: Approximate one distribution with another

As an example consider $p$ a elongated Gaussian distribution (green). Approximate with the simpler factorized distribution $q(z_1, z_2) = q(z_1)q(z_2)$.

The variational approximation tends to under estimate the variance (Exercise extra_opgaven 2.4). The reverse $KL(p||q)$ estimates the variance correctly (but often cannot be computed).
The reverse $KL(p|q)$ estimates the variance correctly

It is quite easy to understand how the reverse KL estimates the variational approximation.

$$KL(p|q) = \int dx_1 \ldots dx_n p(x_1, \ldots, x_n) \log \frac{p(x_1, \ldots, x_n)}{q_1(x_1) \ldots q_n(x_n)}$$

$$= \int dx_1 \ldots dx_n p(x_1, \ldots, x_n) \left( \log p(x_1, \ldots, x_n) - \sum_{i=1}^n \log q_i(x_i) \right)$$

$$= \text{const.} - \sum_{i=1}^n \int dx_1 \ldots dx_n p(x_1, \ldots, x_n) \log q_i(x_i)$$

$$= \text{const.} - \sum_{i=1}^n \int dx_i p_i(x_i) \log q_i(x_i)$$

$$= \text{const.} - \sum_{i=1}^n KL(p_i|q_i)$$

So the inverse KL solution is $q_i = p_i$, with $p_i(x_i) = \int dx_{\setminus i} p(x_i, x_{\setminus i})$ the marginal distribution of $p$ on $x_i$.  

\[4\] $x_{\setminus i}$ is the vector of components excluding variable variable $i$. So if $x = (x_1, x_2, x_3)$ then $x_{\setminus 2} = (x_1, x_3)$. 
The maximum likelihood method

Given a data set \( D = (x_1, \ldots, x_N) \) and model \( p(x|\theta) \), a common approach to estimate \( \theta \) is the so-called maximum likelihood method.

It is assumed that the data set is generated from this model, each data point independently. The probability to observe the entire data set \( D \) is

\[
p(D|\theta) = \prod_{i=1}^{N} p(x_i|\theta)
\]

The best value of \( \theta \) is given by maximizing \( p(D|\theta) \) wrt \( \theta \). Equivalently, one maximizes the log likelihood

\[
L = \frac{1}{N} \log p(D|\theta) = \frac{1}{N} \sum_{i=1}^{N} \log p(x_i|\theta)
\]

As a simple exercise, define the Gaussian model \( p(x|\theta) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(x-\theta)^2\right) \). Then you can compute the maximum likelihood solution by differentiating \( L \) wrt \( \theta \). You will find that \( \theta = \frac{1}{N} \sum_{i=1}^{N} x_i \), ie. the mean of the data.
Relation of KL divergence to maximum likelihood

We can understand the maximum likelihood method in terms of the KL divergence.

Define the *empirical distribution*\(^6\)

\[
q(x) = \frac{1}{N} \sum_{i=1}^{N} \delta_{x,x_i}
\]

Then

\[
KL(q|p) = \sum_x q(x) \log \frac{q(x)}{p(x|\theta)} = -\sum_x q(x) \log p(x|\theta) + \text{const.} = -\frac{1}{N} \sum_{i=1}^{N} \log p(x_i|\theta)
\]

Thus, minimizing \(KL(q|p)\) is equivalent to maximizing the log likelihood.

---

\(^6\)We consider that \(x\) is a discrete variable. If \(x\) is continuous, the expression is \(q(x) = \frac{1}{N} \sum_{i=1}^{N} \delta(x-x_i)\).
Lecture 2

Based on Mackay chapter 3, 28 and 27.

• Bayesian estimating of discrete and continuous parameters distribution from data

• Model comparison
  – Evidence framework and Occam’s razor
  – Laplace approximation of the evidence
  – Example of polynomial curve fitting
Bayesian inference over discrete variable. Example of two dice

Exercise 3.1. [2, p. 50] A die is selected at random from two twenty-faced dice on which the symbols 1–10 are written with nonuniform frequency as follows.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of faces of die A</td>
<td>6</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Number of faces of die B</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The randomly chosen die is rolled 7 times, with the following outcomes:
5, 3, 9, 3, 8, 4, 7.

What is the probability that the die is die A?
Bayesian inference over discrete variable. Example of two dice

Denote $x = A$ or $B$ the two dice. Denote $y = 1, \ldots, 10$ the possible outcomes of a single throw of a die. Denote $D = \{5, 3, 9, 3, 8, 4, 7\} = \{y_1, \ldots, y_7\}$ the observed data.

We wish to compute the probability of $x = A, B$ given the observed data $D$. This is $p(x|D)$. Bayes’ rule says:

$$ p(x|D) = \frac{p(D|x)p(x)}{p(D)} $$
Bayesian inference over discrete variable. Example of two dice

\( p(D|x) \) is the probability to observe the data under the two different models (dice):

\[
p(D|x = A) = \prod_{i=1}^{7} p(y_i|x = A) = \frac{1 \times 3 \times 1 \times 3 \times 1 \times 2 \times 1}{20^7} = \frac{18}{20^7}
\]

\[
p(D|x = B) = \prod_{i=1}^{7} p(y_i|x = B) = \frac{2 \times 2 \times 1 \times 2 \times 2 \times 2 \times 2}{20^7} = \frac{64}{20^7}
\]

\( p(x) = 1/2 \) is the prior probability to choose a die.

Thus

\[
\frac{p(D|x = A)}{p(D|x = B)} = \frac{p(x = A|D)}{p(x = B|D)} = \frac{9}{32}
\]

The result is

\[
p(x = A|D) = \frac{9}{41} \quad p(x = B|D) = \frac{32}{41}.
\]
Bayesian inference over continuous variable. Example of decaying particle

\( \lambda \) is the unknown characteristic decay length.

\( x_i, i = 1, \ldots, N \) are the observed actual decay lengths.

Given \( \lambda \) the probability to observe \( x_i \) is

\[
p_0(x_i|\lambda) = \frac{1}{\lambda} e^{-x_i/\lambda}
\]

When we can observe only \( 1 \leq x \leq 20 \), then

\[
p(x|\lambda) = \begin{cases} \frac{1}{Z} p_0(x|\lambda) & 1 \leq x \leq 20 \\ 0 & \text{else} \end{cases}
\]

\[
Z(\lambda) = \int_1^{20} dx p_0(x|\lambda) = e^{-1/\lambda} - e^{-20/\lambda}
\]

Check that \( \int_0^{\infty} dx p_0(x|\lambda) = 1 \).
Since the particles decay independently, the probability of the data set \( D = \{ x_1, \ldots, x_N \} \) is the product of the probability of the data points

\[
p(D|\lambda) = \prod_{i=1}^{N} p(x_i|\lambda)
\]

Bayes’ rule

\[
p(\lambda|D) = \frac{p(D|\lambda)p(\lambda)}{p(D)} = \frac{1}{p(D)} \frac{1}{(\lambda Z(\lambda))^N} \exp \left( - \sum_{i=1}^{N} \frac{x_i}{\lambda} \right) p(\lambda)
\]

\[
p(D) = \int_{0}^{\infty} \frac{1}{(\lambda Z(\lambda))^N} \exp \left( - \sum_{i=1}^{N} \frac{x_i}{\lambda} \right) p(\lambda)
\]

\( p(\lambda|D) \) for \( N = 1 \) and \( x_1 = 3, 5, 12 \) and \( p(\lambda) \propto 1 \).

NB: When \( x_1 = 3 \) we find \( \lambda \approx 3 \) (effect of box is absent). When \( x_1 = 12 \) we find \( \lambda \gtrsim 10 \).
A bent coin is tossed $N$ times with $N_H$ times outcome 'head' and $N - N_H$ times outcome 'tail'. We consider two hypotheses:

- $H_0$: the coin is fair with probability 'head' is $1/2$
- $H_1$: the coin is not fair. $\lambda$ is the probability of outcome head. Our prior assumption about $\lambda$ is $p(\lambda) = 1$.

Assuming equal prior probabilities on the hypotheses, $p(H_0) = p(H_1) = 1/2$, what is the probability of each of the hypotheses after seeing the data $D$?

Bayes’ rule and equal prior probabilities:

$$p(H_i|D) = \frac{p(D|H_i)p(H_i)}{p(D)}$$
$$p(H_1|D) = \frac{p(D|H_1)}{p(D|H_0)}$$
$$H_0:\]

$$p(D|H_0) = \left(\frac{1}{2}\right)^{N_H} \left(\frac{1}{2}\right)^{N-N_H} = \left(\frac{1}{2}\right)^N$$

$$H_1:\]

$$p(D|\lambda, H_1) = \lambda^{N_H} (1 - \lambda)^{N-N_H}$$

$$p(\lambda|H_1) = 1 \quad \left(\text{NB} \int_0^1 d\lambda p(\lambda|H_1) = 1\right)$$

$$p(D|H_1) = \int_0^1 d\lambda p(\lambda|H_1)p(D|\lambda, H_1) = \int_0^1 d\lambda \lambda^{N_H} (1 - \lambda)^{N-N_H} = \frac{N_H!(N-N_H)!}{(N+1)!}$$

Thus

$$\frac{p(H_1|D)}{p(H_0|D)} = 2^N \frac{N_H!(N-N_H)!}{(N+1)!}$$
Consider a different hypothesis $H_0$: The coin is unfair with probability 'head' $p_0 = 1/6$.

Then

$$\frac{p(H_1|D)}{p(H_0|D)} = \frac{N_H!(N-N_H)!}{(N+1)!} \frac{p_0^{N_H}(1 - p_0)^{N-N_H}}{p_0^{N_H}(1 - p_0)^{N-N_H}}$$

| $F$ | Data $(F_a, F_b)$ | $P(H_1|s,F)$ | $P(H_0|s,F)$ |
|-----|------------------|--------------|--------------|
| 6   | (5, 1)           | 222.2        |              |
| 6   | (3, 3)           | 2.67         |              |
| 6   | (2, 4)           | 0.71         | = 1/1.4      |
| 6   | (1, 5)           | 0.356        | = 1/2.8      |
| 6   | (0, 6)           | 0.427        | = 1/2.3      |
| 20  | (10, 10)         | 96.5         |              |
| 20  | (3, 17)          | 0.2          | = 1/5        |
| 20  | (0, 20)          | 1.83         |              |

Table 3.5. Outcome of model comparison between models $H_1$ and $H_0$ for the 'bent coin'. Model $H_0$ states that $p_a = 1/6$, $p_b = 5/6$.

The simple model $H_0$ 'likes' $(1, 5)$ or $(3, 17)$. The complex model $H_1$ 'likes' all outcomes.
Exercise 3.15.\[2,\ p.63\] A statistical statement appeared in *The Guardian* on Friday January 4, 2002:

When spun on edge 250 times, a Belgian one-euro coin came up heads 140 times and tails 110. ‘It looks very suspicious to me’, said Barry Blight, a statistics lecturer at the London School of Economics. ‘If the coin were unbiased the chance of getting a result as extreme as that would be less than 7%’.

But do these data give evidence that the coin is biased rather than fair? [Hint: see equation (3.22).]

Denote $H_0$ the assumption that the coin is fair ($f = \frac{1}{2}$). Then

$$p(N_H = 140|N = 250, H_0) = \binom{N}{N_H} f^{N_H} (1 - f)^{N - N_H}$$

approaches a Gaussian with mean $fN = 125$ and variance $Nf(1-f) = \frac{250}{4}$. p-value is the probability $p(N_H \geq 140|N = 250, H_0) = 0.07$. 
Bayesian approach compares the likelihood of the assumption $H_0$ with the assumption $H_1$ that the coin is biased with flat prior.

\[
p(N_H|N, H_0) = \binom{N}{N_H}f^{N_H}(1 - f)^{N - N_H} \quad f = \frac{1}{2}
\]

\[
p(N_H|N, H_1) = \int df p(N_H|N, f, H_1)p(f|H_1) = \frac{1}{N + 1} \text{ see slide 15}
\]

\[
\frac{p(N_H|N, H_1)}{p(N_H|N, H_0)} = 0.48
\]
Figure 28.2. How many boxes are behind the tree?
Model comparison: The evidence framework

Consider models $H_i$ with parameters $w_i$, priors $p(w_i|H_i)$ and likelihoods $p(D|w_i, H_i)$. For each $H_i$, we can compute the posterior distribution over $w_i$

$$p(w_i|D, H_i) = \frac{p(D|w_i, H_i)p(w_i|H_i)}{p(D|H_i)}$$

$$p(D|H_i) = \int dw_i p(D|w_i, H_i)p(w_i|H_i)$$

The normalization constants $p(D|H_i)$ are called the model evidence and appear as the likelihood in model comparison

$$p(H_i|D) = \frac{p(D|H_i)p(H_i)}{p(D)}$$
Occam’s razor

Simple models explain only few data sets (but really well!) Complex models explain many data sets by using the freedom of $w$.

$$p(D|H) = \int dw p(D|w, H)p(w|H)$$

Since $p(D|H_2)$ is broader, $p(D|H_1)$ is higher because of normalization: $\int dD p(D|H) = 1$.

Occam’s razor: both $H_{1,2}$ explain $D \in C_1$, but the simpler model $H_1$ is more likely.

$^8$For instance for the bent coin, compare the very restrictive prior $p(w|H) = \delta(w - \frac{1}{2})$ with the 'liberal' prior $p(w|H) = 1$. 
Three models with equal priors $p(\mathcal{H}_i)$ but different priors $p(w|\mathcal{H}_i)$.

$w, D$ are one dimensional and $p(D|w, \mathcal{H}_i) = \mathcal{N}(D|w, \epsilon)$ for all models.

Joint distribution $p(D, w|\mathcal{H}_i) = p(D|w, \mathcal{H}_i)p(w|\mathcal{H}_i)$ shown as as point clouds.

Given particular $D$, $p(D|\mathcal{H}_i)$ is highest for $i = 2$. It is the simplest model that explains the data.
Occam’s razor: A 1-dimensional heuristic argument

The posterior over parameters is always more narrow than the prior

\[ p(w|D, \mathcal{H}_i) \propto p(D|w, \mathcal{H}_i)p(w|\mathcal{H}_i) \]

Denote their width by \( \sigma_{w|D} \) and \( \sigma_w \).

Assume \( p(w|\mathcal{H}_i) = \frac{1}{\sigma_w} \) in a box of size \( \sigma_w \) and \( p(D|w, \mathcal{H}_i) = p(D|w_{\text{MP}}, \mathcal{H}_i) \) in box of size \( \sigma_{w|D} \), then

\[
 p(D|\mathcal{H}_i) = \int dw p(D|w, \mathcal{H}_i)p(w|\mathcal{H}_i) \approx \underbrace{p(D|w_{\text{MP}}, \mathcal{H}_i)}_{\text{best fit}} \frac{\sigma_{w|D}}{\sigma_w} \]

Complex models fit better but have larger \( \sigma_w \).
Laplace method

The idea behind the Laplace approximation is simple. We assume that an unnormalized probability density $P^*(x)$, whose normalizing constant

$$Z_P = \int P^*(x) \, dx$$

(27.1)

is of interest, has a peak at a point $x_0$. We Taylor-expand the logarithm of $P^*(x)$ around this peak:

$$\ln P^*(x) \simeq \ln P^*(x_0) - \frac{c}{2} (x - x_0)^2 + \cdots,$$

(27.2)

where

$$c = -\left. \frac{\partial^2}{\partial x^2} \ln P^*(x) \right|_{x=x_0}.$$

(27.3)

We then approximate $P^*(x)$ by an unnormalized Gaussian,

$$Q^*(x) \equiv P^*(x_0) \exp \left[ -\frac{c}{2} (x - x_0)^2 \right],$$

(27.4)

and we approximate the normalizing constant $Z_P$ by the normalizing constant of this Gaussian,

$$Z_Q = P^*(x_0) \sqrt{\frac{2\pi}{c}}.$$

(27.5)
Laplace method

We can generalize this integral to approximate \( Z_P \) for a density \( P^*(x) \) over a \( K \)-dimensional space \( x \). If the matrix of second derivatives of \(- \ln P^*(x)\) at the maximum \( x_0 \) is \( A \), defined by:

\[
A_{ij} = -\frac{\partial^2}{\partial x_i \partial x_j} \ln P^*(x) \bigg|_{x=x_0},
\]

so that the expansion (27.2) is generalized to

\[
\ln P^*(x) \simeq \ln P^*(x_0) - \frac{1}{2} (x - x_0)^\top A (x - x_0) + \cdots,
\]

then the normalizing constant can be approximated by:

\[
Z_P \simeq Z_Q = P^*(x_0) \frac{1}{\sqrt{\det \frac{1}{2\pi} A}} = P^*(x_0) \sqrt{\frac{(2\pi)^K}{\det A}}.
\]

Predictions can be made using the approximation \( Q \). Physicists also call this widely-used approximation the saddle-point approximation.

See book Mackay ch. 27 for details on the Gaussian integral.
The Laplace approximation to the evidence

When there is much data, the posterior becomes more peaked. We can use the Laplace method to approximate the evidence.

\[
p(D|H_i) = \int dw p(D|w, H_i)p(w|H_i) \approx p(D|w_{\text{MP}}, H_i)p(w_{\text{MP}}|H_i) \sqrt{\frac{(2\pi)^M}{\det A}}
\]

\[
\log p(D|H_i) \propto \log p(D|w_{\text{MP}}, H_i) - \frac{1}{2} \log \det \left( \frac{A}{2\pi} \right)
\]

with \( A = -\nabla \nabla \log p(w_{\text{MP}}|D, H_i) \).

Since \( \det A = \prod_{i=1}^{M} \lambda_i \) we get \( \log \det A \propto M \).
1.1. Polynomial curve fitting

Given training set \((x_n, t_n), n = 1, \ldots, N\), find the best curve.

We will fit the data using \(M\)-th order polynomial

\[
y(x, w) = \sum_{j=0}^{M} w_j \Phi_j(x)
\]

Minimize the error

\[
E(w) = \frac{1}{2} \sum_{n=1}^{N} (y(x_n, w) - t_n)^2
\]
Model comparison, model selection

For instance $\Phi_j(x) = x^j$. $y(x, w) = \sum_{j=0}^{M} w_j x^j$ is a $M$th order polynomial.

How to choose $M$?
Split the data in a training set and a test set. And define

\[
E_{\text{train}}(w) = \frac{1}{2} \sum_{n \in \text{train}} (y(x_n, w) - t_n)^2 \quad E_{\text{test}}(w) = \frac{1}{2} \sum_{n \in \text{test}} (y(x_n, w) - t_n)^2
\]

Too simple (small \( M \)) → poor fit
Too complex (large \( M \)) → overfitting (fits the noise)
<table>
<thead>
<tr>
<th></th>
<th>M=0</th>
<th>M=1</th>
<th>M=3</th>
<th>M=9</th>
</tr>
</thead>
<tbody>
<tr>
<td>$w_0^*$</td>
<td>0.19</td>
<td>0.82</td>
<td>0.31</td>
<td>0.35</td>
</tr>
<tr>
<td>$w_1^*$</td>
<td>-1.27</td>
<td>8</td>
<td></td>
<td>232</td>
</tr>
<tr>
<td>$w_2^*$</td>
<td>-25</td>
<td></td>
<td>5321</td>
<td></td>
</tr>
<tr>
<td>$w_3^*$</td>
<td>-17</td>
<td></td>
<td>48568</td>
<td></td>
</tr>
<tr>
<td>$w_4^*$</td>
<td></td>
<td></td>
<td>-231639</td>
<td></td>
</tr>
<tr>
<td>$w_5^*$</td>
<td></td>
<td></td>
<td>640042</td>
<td></td>
</tr>
<tr>
<td>$w_6^*$</td>
<td></td>
<td></td>
<td>-10618000</td>
<td></td>
</tr>
<tr>
<td>$w_7^*$</td>
<td></td>
<td></td>
<td>10424000</td>
<td></td>
</tr>
<tr>
<td>$w_8^*$</td>
<td></td>
<td></td>
<td>-557683</td>
<td></td>
</tr>
<tr>
<td>$w_9^*$</td>
<td></td>
<td></td>
<td>-125201</td>
<td></td>
</tr>
</tbody>
</table>
Bayesian treatment

Label the different models $H_i = M$.

$$p(t|x, w, M) = \mathcal{N}(t|y(x, w), \beta) = \sqrt{\frac{\beta}{2\pi}} \exp\left(-\frac{\beta}{2} (t - y(x, w))^2\right)$$

$\beta = 1/\sigma^2$ is the precision (given) and $y(x, w) = \sum_{i=0}^M w_i \Phi_i(x)$.
Bayesian treatment

Likelihood and prior

\[
p(D|w, M, \beta) = \prod_{n=1}^{N} p(t_n|x_n, w, M) = \left(\frac{\beta}{2\pi}\right)^{N/2} \exp\left(-\frac{\beta}{2} \sum_{n=1}^{N} (t_n - y(x_n, w))^2\right)
\]

\[
p(w|M, \alpha) = \left(\frac{\alpha}{2\pi}\right)^{M/2} \exp\left(-\frac{\alpha}{2} \sum_{i=0}^{M} w_i^2\right)
\]

The evidence is (we assume \(\alpha, \beta\) given):

\[
p(D|M, \alpha, \beta) = \int dw p(D|w, M, \beta)p(w|M, \alpha)
\]

\[
= \left(\frac{\beta}{2\pi}\right)^{N/2} \left(\frac{\alpha}{2\pi}\right)^{M/2} \int dw \exp\left(-\frac{\beta}{2} \sum_{n=1}^{N} (t_n - y(x_n, w))^2 - \frac{\alpha}{2} \sum_{i=0}^{M} w_i^2\right)
\]

Since \(y\) is linear in \(w_i\), the Gaussian integral can be computed exactly.
Bayesian treatment

Denote \( f(w) = \frac{\beta}{2} \sum_{n=1}^{N} (t_n - y(x_n, w))^2 + \frac{\alpha}{2} \sum_{i=0}^{M} w_i^2 \). Then

\[
p(D|M, \alpha, \beta) = \left( \frac{\beta}{2\pi} \right)^{N/2} \left( \frac{\alpha}{2\pi} \right)^{M/2} \int dw \exp (-f(w))
\]

We can use some optimization method to minimize \( f \), yielding the value \( w_{MP} \) (for most probable), and

\[
f(w) = f(w_{MP}) + \frac{1}{2} (w - w_{MP})^T A (w - w_{MP}) \quad A_{ij} = \alpha \delta_{ij} + \beta \sum_{n=1}^{N} \Phi_i(x_n) \Phi_j(x_n)
\]

\[
p(D|M, \alpha, \beta) = \left( \frac{\beta}{2\pi} \right)^{N/2} \left( \frac{\alpha}{2\pi} \right)^{M/2} \exp (-f(w_{MP})) \int dw \exp \left( - \frac{1}{2} (w - w_{MP})^T A (w - w_{MP}) \right)
\]

\[
= p(D|w_{MP}, M, \beta)p(w_{MP}|M, \alpha) \sqrt{\frac{(2\pi)^M}{\det A}}
\]
Evidence framework

\[ \log p(D|M, \alpha, \beta) = \log p(D|w_{MP}, M, \beta) + \log p(w_{MP}|M, \alpha) - \frac{1}{2} \log \det \left( \frac{A}{2\pi} \right) \]

versus \( M \) for fixed \( \alpha, \beta \). \( M = 1 \) improves over \( M = 0 \). \( M = 2 \) does not improve over \( M = 1 \). \( M = 3 \) improves over \( M = 2 \). Models \( M = 4 - 8 \) have different likelihood but increasing complexity.
Classification problems

- Some background on deep learning

- The perceptron
  - The perceptron learning rule and its convergence
  - The capacity of the perceptron

- Generalization and the VC dimension
Classification problems

Figure 4: (Left) Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). (Right) Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

Reached (super) human performance in last decade.
Other important developments:

- 1980. Statistical physics theory enters as a way to analyze neural networks
The Perceptron

\(\xi = (\xi_1, \ldots, \xi_d)\) is an array of \(d\) input pixels, ie. \(\xi\) is the image. 
\(\phi_i(\xi), i = 1, \ldots, n\) is an array of (given) features computed for each image \(\xi\). 
\(w = (w_0, w_1, \ldots w_n)\) is an array of adaptable parameters.

Given \(\xi, w\) compute

\[y = \text{sign}\left(\sum_{i=1}^{n} w_i \phi_i(\xi) + w_0\right)\]

where

\[\text{sign}(a) = \begin{cases} 
+1, & a \geq 0 \\
-1, & a < 0.
\end{cases}\]

Figure 3.10. The perceptron network used a fixed set of processing elements, denoted \(\phi_j\), followed by a layer of adaptive weights \(w_j\) and a threshold activation function \(g(\cdot)\). The processing elements \(\phi_j\) typically also had threshold activation functions, and took inputs from a randomly chosen subset of the pixels of the input image.
The Perceptron

Learning problem: Given a training set of images $\xi^\mu, \mu = 1, \ldots, P$ where each image belongs to one of two classes labeled by $t^\mu = \pm 1$. Find $w$ such that

$$\text{sign} \left( \sum_{i=0}^{n} w_i \phi_i^\mu \right) = t^\mu, \quad \mu = 1, \ldots, P,$$

where we write $\phi_i^\mu = \phi_i(\xi^\mu)$.

\[ \text{9We have defined } \phi_0 = 1 \text{ so that } \sum_{i=0}^{n} w_i \phi_i = \sum_{i=1}^{n} w_i \phi_i + w_0. \]
Equivalently, we demand

\[
\text{sign} \left( \sum_{i=0}^{n} w_i^{\mu} \phi_i^{\mu} t^\mu \right) = 1 \quad \text{or} \quad w^T \cdot x^\mu > 0
\]

with \( x_i^\mu = \phi_i^{\mu} t^\mu \).

\(^{10}\)In the left picture, the labels on the axes should read \( \phi_1, \phi_2 \).
Linear separation

Classification depends on sign of $w \cdot x$. Thus, decision boundary is hyper plane:

$$0 = \sum_{j=1}^{n} w_j \phi_j + w_0$$

All points of one class should be on one side of the hyper plane, and all others on the other side. Thus, the perceptron can solve so-called linearly separable problems.

AND problem is linearly separable.

XOR problem and linearly dependent inputs not linearly separable.
Perceptron learning rule

Learning successful when

\[ w \cdot x^\mu > 0, \quad \text{all patterns } \mu \]

Consider a learning mechanism, where the weights are updated when a single input pattern \( \phi^\mu \) with its corresponding (presumably correct) output label \( t^\mu \).

1. Compute \( x_i^\mu = \phi_i^\mu t^\mu \).
2. When \( w \) is such that \( w \cdot x^\mu > 0 \) do nothing.
3. else \( w := w + \eta x^\mu \). \( \eta > 0 \) is called the learning rate.  

\[ 11 \]

\[ ^{11} \text{Note, that } w \cdot x^\mu \text{ increases: } w \cdot x^\mu := (w + \eta x^\mu) \cdot x^\mu = w \cdot x^\mu + \eta \| x^\mu \|^2 > w \cdot x^\mu \]
Perceptron learning rule

Thus

\[
\begin{align*}
    w_{i}^{\text{new}} &= w_{i}^{\text{old}} + \Delta w_{i} \\
    \Delta w_{i} &= \eta \Theta(-w \cdot x^{\mu})x_{i}^{\mu}
\end{align*}
\]

Example, learn with two-dimensional patterns \(x^1, x^2, x^3\) the weight vector \(w\) and \(\eta = 1\).

\(^{12}\Theta(x) = 1\) if \(x > 0\) and \(\Theta(x) = 0\) otherwise.
Convergence of the perceptron learning rule

Depending on the data, there may be many or few solutions to the learning problem (or none at all)

Any $w$ for which $w \cdot x^\mu > 0$ for all $\mu$ is a solution to the learning problem, or

$$\min_{\mu} w \cdot x^\mu > 0$$

Note, that the perceptron does not depend on the magnitude of $w$. Thus, we define
\[ D(w) = \frac{1}{\|w\|} \min_{\mu} w \cdot x^{\mu} \]

Better solutions have a larger \( D(w) \).

\(^{13}\|w\|^2 \text{ is the Euclidean norm: } \|w\|^2 = \sum_{i=0}^{n} w_i^2. \)
Convergence of the perceptron learning rule

We define the best solution as $D_{\text{max}} = \max_w D(w)$.

When $D_{\text{max}} > 0$ the problem is linearly separable. Otherwise it is not.
Convergence of Perceptron rule

Assume that the problem is linearly separable, so that there is a solution $w^*$ with $D(w^*) > 0$.

Suppose we have done $t$ iterations of the learning rule. In each iteration, a pattern $\mu$ was presented that either changed $w$ or not.

Denote $M^\mu$ denote the number of times pattern $\mu$ has been presented and caused a non-zero update of $w$. If we start with the initial value $w = 0$, after $t$ iterations

$$w = \eta \sum_{\mu} M^\mu x^\mu$$

Denote $M = \sum_{\mu} M^\mu$ the total number of non-zero updates of $w$. We will prove that

$$A(w) = \frac{w \cdot w^*}{||w||||w^*||} \geq O(\sqrt{M}),$$

If the learning rule does not converge, $w$ will be updated forever, $M \to \infty$, and $A \to \infty$. 

Bert Kappen ML 78
However, $A(w) = \cos \theta \leq 1$ and cannot grow forever. The conclusion is that $M$ must stay finite. This means that after a finite number of updates of $w$ there are no more changes.

Thus, the perceptron learning rule converges in a finite number of steps when the problem is linearly separable.
Convergence of Perceptron rule

We can bound

\[ w \cdot w^* = \eta \sum_{\mu} M^\mu x^\mu \cdot w^* \geq \eta \sum_{\mu} M^\mu \min y^\nu \cdot w^* = \eta M \min y^\nu \cdot w^* = \eta M D(w^*) \|w^*\| \]

Consider the change of \( \|w\|^2 \) in when updating with a single pattern \( \mu \):

\[ \Delta \|w\|^2 = \|w + \eta x^\mu\|^2 - \|w\|^2 = 2\eta w \cdot x^\mu + \eta^2 \|x^\mu\|^2 \leq \eta^2 \|x^\mu\|^2 \]

where in the inequality we used that \( w \cdot x^\mu < 0 \), because otherwise there would have been no update and \( \delta \|w\|^2 = 0 \). Thus, at iteration \( t \):

\[ \|w\|^2 = \sum_{\mu} M^\mu \Delta \|w\|^2 \leq \eta^2 \sum_{\mu} M^\mu \|x^\mu\|^2 \leq \eta^2 MB \]

\[ \|w\| \leq \eta \sqrt{MB} \]
where we define $B = \max_{\mu} \|x^\mu\|^2$, which is a constant. Thus,

$$A(w) = \frac{w \cdot w^*}{\|w\|\|w^*\|} \geq \sqrt{\frac{M}{B}} D(w^*)$$

We can invert this relation to bound the total number of weight updates:

$$M \leq \frac{B}{D^2(w^*)} \propto \frac{n}{D^2(w^*)}$$

since $B = \max_{\mu} \|x^\mu\|^2 = O(n)$. The convergence is slower for higher dimensional problems and also for harder problems, for which $D(w^*) \gtrsim 0$. 
Capacity of the Perceptron

The perceptron can only learn linearly separable problems.

Is this good or bad? How often will one encounter a linearly separable problem in practice?

In the following analysis we will consider random problems for which we can compute some answers.
Capacity of the Perceptron

Consider $P$ patterns in $n$ dimensions

$$
\phi^\mu = (\phi_1^\mu, \ldots, \phi_n^\mu), \quad \mu = 1, \ldots, P
$$

We colour each pattern $\phi^\mu$ randomly either black ($t^\mu = 1$) or white ($t^\mu = -1$). This defines a classification problem where we want to separate the white from the black. What is the probability that this problem is linearly separable?

---

$^{14}$We are interested in the large $n$ behaviour and therefore we ignore the component $\phi_0 = 1$ and $w_0$. 
Capacity of the Perceptron

We assume that the $n$-dimensional vectors $\phi^\mu$ are in general position, which avoids situations such as in fig b, where a subset of patterns are exactly on a linear subspace. When $\phi^\mu$ are generated at random, they are in general position with probability one.
Capacity of the Perceptron

Each pattern can be either black or white. So each set of $P$ patterns defines $2^P$ classification problems.

The perceptron defines a hyper plane in $n$ dimensions through the origin $\sum_{i=1}^n w_i \phi_i = 0$: Points on one side of the hyper plane ($\sum_{i=1}^n w_i \phi_i > 0$) are classified as black and points on the other side of the hyper plane as white. Which fraction of the $2^P$ possible classification problems $s$ is linearly separable?

Examples

- $n = 2, p = 2$. The total number of problems is $2^p = 4$. They are all linearly separable.

- $n = 2, p = 3$. The total number of problems is $2^p = 8$. There are 2 problems that are not linearly separable.

- $n = 3, p = 3$. The total number of problems is $2^p = 8$. They are all linearly separable.

- ...
Capacity of the Perceptron

Define \( n = N \).

Thm: The number of linearly separable colorings on \( P \) points in \( N \) dimensions, with separability plane through the origin is (Cover 1966):

\[
C(P, N) = 2 \sum_{i=0}^{N-1} \binom{P - 1}{i}
\]

In the exercise you will show:
When \( P \leq N \), then \( C(P, N) = 2^P \). So all problems are linearly separable.
When \( P = 2N \), then \( C(P, N) = 2^{P-1} \). So 50 % of all problems are linearly separable.
When $P, N \to \infty$ the transition is at $P = 2N$. 
Proof of Thm

Proof of the Thm is by induction.

Suppose the number of linearly separable problem with $P$ patterns in $N$ dimensions is $C(P, N)$. (In the left figure $P = 4$, the two blue and red points. Ignore the color and the line for the moment).

We add one point $X$ (the green point) and wish to compute $C(P + 1, N)$. The set $C(P, N)$ consists of linear separable problems, some of which have
- a separating hyper plane that can be drawn through $X$ (A). For each of these problems we can define two new linearly separable problems by coloring the green point either red or blue
- rest (B). For each of these problems we can define only one separable problem.
Thus,

\[ C(P + 1, N) = 2A + B = (A + B) + A = C(P, N) + A \]

The set \( A \) is the set of linearly separable problems on \( P \) points in \( N \) dimensions, where the separating hyper plane goes through the origin \( O \) and through the point \( X \). So, one of the dimensions is fixed (by the line \( O - X \)) and only the dimensions of the hyper plane and the points orthogonal to \( O - X \) matter. Therefore,

\[ A = C(P, N - 1) \]

Thus

\[ C(P + 1, N) = C(P, N) + C(P, N - 1) \]
Yields

\[ C(P, N) = 2 \sum_{i=0}^{N-1} \binom{P-1}{i} \]
Generalization

Given inputs $x = (x_1, \ldots, x_n)$ and binary output $y$. Suppose we wish to learn a given, but unknown, function $\bar{f} : x \to y$ from data. Denote our solution as $f : x \to y$.

An example is that $\bar{f}$ is some linearly separable classifier from which we have some data. $f$ is our perceptron solution.

Define the generalization performance of any function $f$ as

$$g(f) = \text{Prob}(f(x) = \bar{f}(x))$$

$x$ is uniform random

Also, define the performance of $f$ on a training set of $P$ patterns

$$g_P(f) = \text{Prob}(f(x) = \bar{f}(x))$$

$x$ is uniform from training set of size $P$

We do not know $g(f)$ but we do know $g_P(f)$ because we have the training set.
Generalization

There exists a theorem [Vapnik, 1982] that allows us to estimate $g(f)$ from $g_P(f)$:

$$\text{Prob} \left( \max_f |g_P(f) - g(f)| > \epsilon \right) \leq 4m(2P)e^{-\epsilon^2P/8}$$

So, if we can make the rhs small, say $0.01$ and we find perfect performance on the training set $g_P(f) = 1$, we know that the generalization performance $g(f) > 1 - \epsilon$ with 99% probability.

$m(P)$ is called the growth function and counts the number of different binary functions on $P$ inputs that can be realized by our learning architecture. In the case of the perceptron $m(P) = C(N, P)$ is the number of linearly separable functions.
Generalization

In general $m(P) = 2^P$ for $P \leq d_{VC}$ and becomes polynomial in $P$ for $P > d_{VC}$. $d_{VC}$ is called the VC (Vapnik-Cervonenkis) dimension.

Note, that we need $P > d_{VC}$ in order to make the rhs $m(2P)e^{-\epsilon^2P/8}$ of the bound small.

One can prove that the polynomial growth of $m(P)$ is bounded as

$$m(P) \leq \left( \frac{eP}{d_{VC}} \right)^{d_{VC}} \quad P > d_{VC}$$

\[ ^{15} \text{For details, see extra material "Learnability and VC dimension".} \]
Generalization for the perceptron

For the perceptron $m(P) = C(N, P)$ and $d_{VC} = N$.

We can then estimate that when $P, N$ are large, we need

$$P \gtrsim \frac{N}{\epsilon^2} \quad \rightarrow \quad |g_P(f) - g(f)| < \epsilon$$

Thus we need $O(N)$ training samples for good generalization of the perceptron (but with a pretty large prefactor $1/\epsilon^2$!)

Proof: Using $d_{VC} = N$, the bound on the growth factor is $m(2P) \leq \left(\frac{2eP}{N}\right)^N$. The rhs of the generalization bound

$$4m(2P) \exp\left(-\frac{\epsilon^2 P}{8}\right) \leq 4 \exp\left(N\left(\log(2e\alpha) - \frac{\alpha \epsilon^2}{8}\right)\right)$$

with $\alpha = P/N$. The rhs is small when the term in the exponent becomes negative, which occurs for sufficiently large $\alpha$. An analysis of the function $f(\alpha) = \log(2e\alpha) - \frac{\alpha \epsilon^2}{8}$.
\frac{\alpha \epsilon^2}{8} \text{ shows that it crosses zero when } \alpha = \alpha_c \propto \frac{1}{\epsilon^2}. \text{ Thus}

\alpha \gtrsim \alpha_c \quad P_c = N\alpha_c \gtrsim \frac{N}{\epsilon^2}
Summary

Any learning machine has a capacity, which is its VC dimension. By definition, the number of problems grows exponential as $2^P$ for $P \leq d_{VC}$ and grows slower polynomially for $P > d_{VC}$. For the perceptron the capacity is $d_{VC} = N$.

For $P < d_{VC}$ learning is easy. In the case of the perceptron it means the problem is linearly separable with overwhelming probability and that there exists many solutions that have zero training error. Because of these many solution, it is expected that the (average) generalization error of these solutions is large.

For $P > d_{VC}$ learning random instances becomes impossible with high probability. However, in practice the data are non-random and have structure. For instance, in the case of the perceptron, the data may have been generated by some unknown (teacher) perceptron. In that case, the problem is still linearly separable. The VC bound shows that in that regime the generalization error is effectively bounded. So good generalization requires many samples.
Avoid the use of loops in your code

To make your code run efficient it is absolutely necessary to use the fast treatment of array operations.

I am not a Python programmer, but use Matlab. But the concepts are similar.

Avoid for loops. do_not_use_for_loops_matrix_multiplication.m
A faster parallel version of the perceptron learning rule

The perceptron learning rule is sequential, updating one pattern at the time, and requires a for loop.

for mu =1: P
    if (x(mu,:) * w)<0
        w=w+eta*x(mu,:);
    end;
end;

You can make a much faster parallel version, updating all 'wrong' patterns at once.

1. Find the set $S$ of training patterns for which $x^\mu \cdot w < 0$.

2. Update $w := w + \eta \sum_{\mu \in S} x^\mu$

klad=(x*w'<>0);
w=w+eta*sum(x(find(klad),:),1);
Lecture 4

• Learning rules
  – Gradient descent, momentum, Newton’ method, line search, conjugate gradient descent
  – Stochastic gradient descent, Robbins-Munro

• Feedforward networks
  – Linear regression, logistic regression
  – Multi-layered perceptrons, error backpropagation
  – Regularization
  – MLPs are universal approximators
Network training: regression

Data: \{ (x^\mu, t^\mu), \mu = 1, \ldots, P \}. \; x^\mu = (x_1^\mu, \ldots, x_n^\mu) is the vector of (continuous) inputs. \; t^\mu is a one dimensional continuous output.

Objective is to learn a model, which is a parametrized function \( y(x, w) \). Find best parameters \( w \) by minimizing

\[
E(w) = \frac{1}{2} \sum_{\mu=1}^{P} (y(x^\mu, w) - t^\mu)^2 = -\log \prod_{\mu=1}^{P} \mathcal{N}(t^\mu | y(x^\mu, w), \beta^{-1} = 1) + \ldots
\]

Equivalent to maximum likelihood, assuming a Gaussian model.
Network training: 2 class classification

Data: \{ (x^{\mu}, t^{\mu}), \mu = 1, \ldots, P \}. \ x^{\mu} = (x_{1}^{\mu}, \ldots, x_{n}^{\mu}) \ is \ the \ vector \ of \ (continuous) \ inputs. \ t^{\mu} = 0, 1

Model \ y(x, w) = p(t = 1 | x, w) \ is \ probability \ to \ belong \ to \ class \ t = 1. \n
\[
L(w) = \prod_{\mu=1}^{P} p(t^{\mu} | x^{\mu}, w) = \prod_{\mu=1}^{P} y(x^{\mu}, w)^{t^{\mu}} (1 - y(x^{\mu}, w))^{1-t^{\mu}}
\]

or

\[
E(w) = - \sum_{\mu=1}^{P} (t^{\mu} \log y(x^{\mu}, w) + (1 - t^{\mu}) \log(1 - y(x^{\mu}, w)))
\]
Network training: more classes

More than two classes: consider network with $K$ outputs. $t^\mu_k = 1$ if $x^\mu$ belongs to class $k$ and zero otherwise. $y_k(x^\mu, w), k = 1, \ldots, K$ is the network output.

Define the probability vector:

$$ p_k(x, w) = \frac{\exp(y_k(x, w))}{\sum_{k'=1}^{K} \exp(y_{k'}(x, w))} $$

The maximum likelihood objective becomes:

$$ L(w) = \prod_{\mu=1}^{P} \prod_{k=1}^{K} p_k(x^\mu, w)^{t^\mu_k} \quad \leftrightarrow \quad E(w) = - \sum_{\mu=1}^{P} \sum_{k=1}^{K} t^\mu_k \log p_k(x^\mu, w) $$
Parameter optimization

$E(w)$ is minimal when $\nabla E(w) = 0$, but not vice versa!

As a consequence, gradient based methods find a local minimum, not necessarily the global minimum.
Gradient descent optimization

The simplest procedure to optimize $E$ is to start with a random $w$ and iterate

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

t labels the learning iterations. This is called batch learning, where all training data are included in the computation of $\nabla E$.

Does this algorithm converge? Yes, if $\eta$ is "sufficiently small" and $E$ bounded from below.

Proof: Denote $\Delta w = -\eta \nabla E$.

$$E(w + \Delta w) \approx E(w) + \sum_i \Delta w_i \frac{\partial E}{\partial w_i} = E(w) - \eta \sum_i \left(\frac{\partial E}{\partial w_i}\right)^2 \leq E(w)$$

In each gradient descent step the value of $E$ is lowered. Since $E$ bounded from below, the procedure must converge asymptotically.
Convergence of gradient descent in a quadratic well

\[
E(w) = \frac{1}{2} \sum_i \lambda_i w_i^2
\]

\[
\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = -\eta \lambda_i w_i
\]

\[
w_i^{\text{new}} = w_i + \Delta w_i = (1 - \eta \lambda_i)w_i
\]

Convergence when \(|1 - \eta \lambda_i| < 1\). Oscillations when \(1 - \eta \lambda_i < 0\).

Optimal \(\eta\) depends on curvature of each dimension: \(\eta \lesssim \frac{1}{\lambda_i}\).
Learning with momentum

One solution is adding momentum term, with $0 \leq \alpha < 1$.

$$
\Delta w^{t+1} = -\eta \nabla E(w^t) + \alpha \Delta w^t \\
= -\eta \nabla E(w^t) + \alpha \left(-\eta \nabla E(w^{t-1}) + \alpha \left(-\eta \nabla E(w^{t-2}) + \ldots\right)\right) \\
= -\eta \sum_{k=0}^{t} \alpha^k \nabla E(w^{t-k})
$$

$\nabla E$ may overshoot in some directions (oscillations) and not in other directions.

**No oscillations.** Rough approximation is that all derivative are equal:

$$
\Delta w^{t+1} \approx -\eta \nabla E \sum_{k=0}^{t} \alpha^k \approx -\frac{\eta}{1-\alpha} \nabla E
$$

results in acceleration

\[16\sum_{k=0}^{t} \alpha^k = \frac{1-\alpha^{t+1}}{1-\alpha}. \text{ Proof: Define } I = \sum_{k=0}^{t} \alpha^k. \text{ Then } (1-\alpha)I = 1 - \alpha^{t+1}, \text{ end of proof. Thus } \sum_{k=0}^{\infty} \alpha^k = \frac{1}{1-\alpha}.\]
Oscillations: approximate all derivatives are equal but have opposite sign:

$$\Delta w^{t+1} \approx -\eta \nabla E \sum_{k=0}^{t} (-\alpha)^k \approx -\frac{\eta}{1 + \alpha} \nabla E$$

results in deceleration
Newton's method

One can also use Hessian information for optimization. As an example, consider a quadratic approximation to $E$ around $w_0$:

$$E(w) \approx E(w_0) + b^T(w - w_0) + \frac{1}{2}(w - w_0)^T H(w - w_0)$$

$$b_i = \frac{\partial E(w_0)}{\partial w_i} \quad H_{ij} = \frac{\partial^2 E(w_0)}{\partial w_i \partial w_j}$$

$$\nabla E(w) \approx b + H(w - w_0)$$

We can solve $\nabla E(w) = 0$ and obtain

$$w = w_0 - H^{-1}\nabla E(w_0)$$

This is called Newton's method. Is expensive when $n$ is large.

Quadratic approximation is exact when $E$ is quadratic, so convergence in one step.

Quasi-Newton: Consider only diagonal of $H$. 
Line search

Another solution is line optimisation:

\[ w_1 = w_0 + \lambda_0 d_0, \quad d_0 = -\nabla E(w_0) \]

\( \lambda_0 > 0 \) is found by a one dimensional optimisation

\[ 0 = \frac{\partial}{\partial \lambda_0} E(w_0 + \lambda_0 d_0) = d_0 \cdot \nabla E(w_1) = d_0 \cdot d_1 \]

Therefore, subsequent search directions are orthogonal.
Line search

The learning objective is to find $w$ such that $\nabla E(w) = 0$. This is a vector equation.

Line search starts in $w_0$ and achieves

- $w_1$ such that $\nabla E(w_1) \cdot d_0 = 0$
- $w_2$ such that $\nabla E(w_2) \cdot d_1 = 0$
- ... 

We would like to find a method that finds $\nabla E(w_2) \cdot d_1 = 0$ and $\nabla E(w_2) \cdot d_0 = 0$. 
Conjugate gradient descent

We choose as new direction a combination of the gradient and the old direction

\[ d'_1 = -\nabla E(w_1) + \beta d_0 \]

Line optimisation \( w_2 = w_1 + \lambda_1 d'_1 \) yields \( \lambda_1 > 0 \) such that \( d'_1 \cdot \nabla E(w_2) = 0 \).

The direction \( d'_1 \) is found by demanding that \( \nabla E(w_2) \approx 0 \) also in the 'old' direction \( d_0 \):

\[ 0 = d_0 \cdot \nabla E(w_2) \approx d_0 \cdot (\nabla E(w_1) + \lambda_1 H(w_1)d'_1) \]

or

\[ d_0 H(w_1)d'_1 = 0 \]

The subsequent search directions \( d_0, d'_1 \) are said to be conjugate.
Polak-Ribiere rule

The conjugate directions can be computed without computing the Hessian matrix, for instance using the Polak-Ribiere rule:\footnote{17}{We need $0 = d_0^T H(w_1)d'$. We use $\nabla E(w_0) \approx \nabla E(w_1) + (w_0 - w_1)^T H(w_1) = \nabla E(w_1) - \lambda_0 d_0^T H(w_1)$. Then
\[
0 = \lambda_0 d_0^T H(w_1)d'_1 = (\nabla E(w_1) - \nabla E(w_0)) \cdot (-\nabla E(w_1) + \beta d_0) = - (\nabla E(w_1) - \nabla E(w_0)) \cdot \nabla E(w_1) + \beta \|\nabla E(w_0)\|^2
\]
where in the last step we used that $d_0 \cdot \nabla E(w_1) = 0$.}

\[
\beta = \frac{(\nabla E(w_1) - \nabla E(w_0)) \cdot \nabla E(w_1)}{\|\nabla E(w_0)\|^2}
\]

For quadratic problems, it can be proven that this rule keeps the last $n$ directions all mutually conjugate

\[
d_i^T H d_j = 0 \quad i, j = 1, \ldots, n
\]

\footnote{18}{Conjugate gradient is method of choice for solving large linear systems $Ax = b$ as $\min_x \|Ax - b\|$\cite{Press2016}}
Stochastic gradient descent

One can also consider on-line learning, where only one or a subset of training patterns is considered for computing $\nabla E$.

$$E(w) = \sum \mu E_\mu(w) \quad \nabla E(w) = \sum \mu \nabla E_\mu(w)$$

$$w^{t+1} = w^t - \eta_t \nabla E_\mu(w^t)$$

Is efficient for large data sets. This results in a stochastic dynamics in $w$ that can help to escape local minima.
Consider the $n$-dimensional problem to find $x = (x_1, \ldots, x_n)$ such that

$$M_i(x) = a_i, \quad M_i(x) = \sum_{\mu} p_{\mu} N_{i\mu}(x)$$

$N_{i\mu}(x)$ some non-linear function, $p_{\mu} > 0$ is a probability distribution ($\sum_{\mu} p_{\mu} = 1$) and $a_i$ a constant.

Method of *stochastic approximation* originally due to Robbins and Monro 1951:

- Initialize $x^0$ random
- For $t = 0, \ldots$, Choose $\mu \sim p_{\mu}$; Update

$$x_{i}^{t+1} = x_{i}^{t} + \eta_{t}(a_{i} - N_{i\mu}(x^{t}))$$

If $M_i(x) = \nabla_i E$ and $E$ is convex and $x^*$ the unique solution ($M_i(x^*) = a_i$). Then one can prove that $\|x^t - x^*\|^2 \to 0$, provided that

$$\sum_{t=1}^{\infty} \eta_{t} = \infty \quad \sum_{t=1}^{\infty} \eta_{t}^{2} < \infty \quad (\eta_{t} = 1/t)$$
Stochastic gradient descent

Denote training error

\[ E(w) = \frac{1}{P} \sum_\mu E_\mu(w) \]

we wish to find solution of

\[ \nabla E(w) = \frac{1}{P} \sum_\mu \nabla E_\mu(w) = 0 \]

This is an instance of the Robbins-Monro problem with

\[ p_\mu = \frac{1}{P} \quad a_i = 0 \quad N_{i\mu}(w) = \nabla_i E_\mu(w) \]

Thus, SGD converges to the optimal solution provided \( \eta_t \) is lowered appropriately and \( E \) is convex.

Extensions of SGD and comparisons see [Sohl-Dickstein et al., 2013].
Linear regression

\[ y(x, w) = \sum_j w_j x_j \quad E(w) = \frac{1}{2} \sum_\mu \left( t^\mu - \sum_j w_j x_i^\mu \right)^2 \]

\[ \frac{\partial E}{\partial w_i} = - \sum_\mu \left( t^\mu - \sum_j w_j x_i^\mu \right) x_i^\mu \]

\[ H_{ij} = \frac{\partial^2 E}{\partial w_i \partial w_j} = \sum_\mu x_i^\mu x_j^\mu \]

Note, that in this case we can directly solve \( \frac{\partial E}{\partial w_i} = 0 \) as

\[ w = H^{-1} b \quad b_i = \sum_\mu t^\mu x_i^\mu \]

\[^{19}\text{Use } x_0 = 1 \text{ as with the perceptron.}\]
Logistic regression

\[ y(x, w) = \sigma \left( \sum_j w_j x_j \right) \]

\[ E(w) = -\frac{1}{P} \sum_{\mu} (t^{\mu} \log y^{\mu} + (1 - t^{\mu}) \log(1 - y^{\mu})) \]

with \( y^{\mu} = y(x^{\mu}, w) \) and \( \sigma \) is the sigma function \( \sigma(x) = (1 + e^{-x})^{-1} \)

The gradient and Hessian are \(^{21}\)

\[ \nabla_i E(w) = \frac{\partial E}{\partial w_i} = \frac{1}{P} \sum_{\mu} (y^{\mu} - t^{\mu}) x_i^{\mu} \]

\[ H_{ij}(w) = \frac{\partial^2 E}{\partial w_i \partial w_j} = \frac{1}{P} \sum_{\mu} x_i^{\mu} y^{\mu} (1 - y^{\mu}) x_j^{\mu} \]

\(^{20}\)The \( 1/P \) term is just to make the quantities of order 1 for numerical stability.

\(^{21}\)Use here that \( \sigma'(x) = \frac{d\sigma(x)}{dx} = \sigma(x)(1 - \sigma(x)) \). Thus \( \frac{\partial y^{\mu}}{\partial w_i} = x_i^{\mu} \sigma'(\sum_j w_j x_j) = x_i^{\mu} y^{\mu}(1 - y^{\mu}) \).
Multi-layered neural networks

Two layers with \( M \) hidden neurons

\[
y_k(x, w) = h_2 \left( w_{k0}^{(2)} + \sum_{j=1}^{M} w_{kj}^{(2)} h_1 \left( w_{j0}^{(1)} + \sum_{i=1}^{n} w_{ji}^{(1)} x_i \right) \right)
\]

\( h_{1,2} \) is scalar functions such as \( x, \sigma(x) \) or ReLu\((x)\). \footnote{ReLu\((x) = x \) when \( x > 0 \) and ReLu\((x) = 0 \) otherwise.}

Left) Two layer architecture. Right) general feed-forward network with skip-layer connections. \( D = n, K = 1 \)
Two layer NN with 3 'tanh' hidden units and linear output can approximate many functions. $x \in [-1, 1]$, 50 equally spaced points. From left to right: $f(x) = x^2$, $\sin(x)$, $|x|$, $\Theta(x)$. Dashed lines are outputs of the 3 hidden units.

Two layer NN with two inputs and 2 'tanh' hidden units and sigmoid output for classification. Dashed lines are hidden unit activities.

Feed-forward neural networks have good approximation properties.
Error backpropagation

Error is sum of error per pattern

\[ E(w) = \sum_\mu E_\mu(w) \quad E_\mu(w) = \frac{1}{2} \sum_k (y_k(x^\mu, w) - t_k^\mu)^2 \]

\[ y_k = h \left( w_k^{(2)} + \sum_{j=1}^M w_{kj}^{(2)} h \left( w_j^{(1)} + \sum_{i=1}^D w_{ji}^{(1)} x_i \right) \right) \]

\[ = h \left( \sum_j w_{kj}^{(2)} z_j^{(1)} \right) \quad z_j^{(1)} = h \left( \sum_i w_{ji}^{(1)} x_i \right) \]

\[ \frac{\partial}{\partial w_{kj}^{(2)}} \frac{1}{2} \sum_{k'} (y_{k'} - t_{k'}^\prime)^2 = (y_k - t_k) \frac{\partial y_k}{\partial w_{kj}^{(2)}} = (y_k - t_k) h'(a_k^{(2)}) z_j^{(1)} = \delta_k^{(2)} z_j^{(1)} \]
Error backpropagation

Similarly,

\[ y_k = h \left( \sum_j w_{kj}^{(2)} h \left( \sum_i w_{ji}^{(1)} z_i^{(0)} \right) \right) \]

\[
\frac{\partial}{\partial w_{ji}^{(1)}} \frac{1}{2} \sum_k (y_k - t_k)^2 = \sum_{k=1}^K (y_k - t_k) \frac{\partial y_k}{\partial w_{ji}^{(1)}} = \sum_{k=1}^K \left( \frac{y_k - t_k}{\delta_k^{(2)}} \right) w_{kj}^{(2)} h'(a_j^{(1)}) z_i^{(0)} = \delta_j^{(1)} z_i^{(0)}
\]

with

\[
\delta_j^{(1)} = h'(a_j^{(1)}) \sum_{k=1}^K \delta_k^{(2)} w_{kj}^{(2)}
\]
The back propagation extends to $T$ layers. For each pattern $\mu$:

1. Compute forward activities $a^{\mu,(t)}_j = \sum_i w^{(t)}_{ji} z^{\mu,(t-1)}_i$ and $z^{\mu,(t)}_j = h(a^{\mu,(t)}_j)$, $t = 1, \ldots, T$.
   With $z^{\mu,(0)}_i = x^{\mu}_i$.

2. Compute the errors $\delta^{\mu,(t-1)}_j = h'(a^{\mu,(t-1)}_j) \sum_k w^{(t)}_{kj} \delta^{\mu,(t)}_k$, $t = T, \ldots, 2$.
   With $\delta^{\mu,(T)}_k = (y^{\mu}_k - t^{\mu}_k) h'(a^{\mu,(T)}_k)$.

3. $\frac{\partial E_\mu}{\partial w^{(t)}_{ji}} = \delta^{\mu,(t)}_j z^{\mu,(t-1)}_i$

The gradient is $\frac{\partial E}{\partial w^{(t)}_{ji}} = \sum_\mu \frac{\partial E_\mu}{\partial w^{(t)}_{ji}}$
$E$ is a function of $O(|w|)$ variables. In general, the computation of $E$ requires $O(|w|)$ operations. The computation of $\nabla E$ would thus require $O(|w|^2)$ operations.

The backpropagation method allows to compute $\nabla E$ efficiently, in $O(|w|)$ operations.
Complexity of neural network solution is controlled by number of hidden units.

Sum squared test error for different number of hidden units and different weight initializations. Error is also affected by local minima.
Part of the cause of local minima is the saturation of the sigmoid functions \( \tanh(\sum w_{ij}x_j) \). When \( w_{ij} \) becomes large, any change in its value hardly affects the output, implying \( \nabla_{ij}E = 0 \).

One can partly prevent this from happening by

- choose \( \tanh \) instead of \( \sigma \) transfer functions and scaling of inputs and outputs with mean zero and standard deviation one
- choose ReLU
- proper initialisation of \( w_{ij} \) with mean zero and standard deviation of order \( 1/\sqrt{n_1} \), with \( n_1 \) the number of inputs to neuron \( i \).
- add regularizer such as \( \sum_i w_i^2 \) to cost keeps weights small
Early stopping is related to weight decay

Early stopping is to stop training when error on test set starts increasing.

Parameters in flat directions learn slower than in steep directions. Thus, early stopping with small initial weights yields $w_i = w_i^*$ in steep directions and $w_i \ll w_i^*$ is flat directions.

Weight decay with $\lambda_1 \ll \lambda \ll \lambda_2$

$$E(w) = \lambda_1(w_1 - w_1^*)^2 + \lambda_2(w_2 - w_2^*)^2 + \lambda(w_1^2 + w_2^2)$$

$$\frac{\partial E}{\partial w_i} = \lambda_i(w_i - w_i^*) + \lambda w_i = 0, \quad i = 1, 2$$

$$w_i = \frac{\lambda_i}{\lambda_i + \lambda} w_i^* \quad w_1 \approx \frac{\lambda_1}{\lambda_1 w_1^*} \quad w_2 \approx \frac{\lambda_2}{w_2^*}.$$
MLPs are universal approximators

Consider $2^n$ binary patterns in $n$ dimensions and two classes:

$$x^\mu \rightarrow c^\mu = \pm 1, \quad x_i^\mu = \pm 1$$

Use $2^n$ hidden units, labeled $j = 0, \ldots, 2^n - 1$, $i$ labels input. Set

$$w_{ji} = b \quad \text{if } \text{ith digit in binary repr. of } j \text{ is 1}$$
$$w_{ji} = -b \quad \text{else}$$

<table>
<thead>
<tr>
<th>$j$</th>
<th>binary</th>
<th>$w_{j1}$</th>
<th>$w_{j2}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>00</td>
<td>-b</td>
<td>-b</td>
</tr>
<tr>
<td>1</td>
<td>01</td>
<td>-b</td>
<td>b</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>b</td>
<td>-b</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>b</td>
<td>b</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$\sum_i w_{0i}x_i$</th>
<th>$w_{1i}x_i$</th>
<th>$w_{2i}x_i$</th>
<th>$w_{3i}x_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
<td>2b</td>
<td>0</td>
<td>0</td>
<td>-2b</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>2b</td>
<td>-2b</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-2b</td>
<td>2b</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-2b</td>
<td>0</td>
<td>0</td>
<td>2b</td>
</tr>
</tbody>
</table>
MLPs are universal approximators

Use threshold of \( (n - 1)b \) at each hidden unit. \( z_j = \Theta[\sum_i w_{ji}x_i - (n - 1)b] \). The remaining problem has \( p = 2^n \) patterns in \( 2^n \) dimensions and is linearly separable.

Define \( c = \text{sign}[\sum_{j=0}^3 w_jz_j] \).

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>( z_0 )</th>
<th>( z_1 )</th>
<th>( z_2 )</th>
<th>( z_3 )</th>
<th>( c )</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>sign([w_0])</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>sign([w_1])</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>sign([w_2])</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>sign([w_3])</td>
</tr>
</tbody>
</table>

The combination of linear summation and non-linear functions can create many different functions.

- The MLP with a single hidden layer can map any continuous function [Cybenko, 1989, Hornik et al., 1989, Barron, 1989]
Lecture 5

Deep learning

- Convolutional networks
- Imagenet competition
- Drop out, batch normalization
- Why do deep networks generalize well?
  - regularization, drop out, batch normalization
  - Properties of GD: implicit regularization and memorization.
Units in a convolutional layer are organized in feature maps.

Each unit in a feature map is connected to local patches in the feature maps of the previous layer through a *common* set of weights called a filter bank (weight sharing). The result of this local weighted sum is then passed through a non-linearity such as a ReLU. Different feature maps in a layer use different filter banks.

- (max) pooling: down sample each feature map (not adaptive). Each pooling unit computes the max of a local patch in a feature map. Induces translational invariance.
- repeat many times.
Convolution networks

[Fukushima and Miyake, 1982, LeCun et al., 1990]

In one dimension input image $x_j$ and one dimensional feature array $y_i$. Normally $y_i = h\left(\sum_j w_{ij}x_j\right)$. Weight sharing

$$y_i = h\left(\sum_{j=i-K}^{i+K} w_{j-i}x_j\right)$$

So $x_i$ and $y_j$ have same connection as $x_{i+a}$ and $y_{j+a}$ (which is $w_{j-i}$).

In two dimension input image $x_{j,j'}$, and two dimensional feature array

$$y_{i,i'} = h\left(\sum_{j=i-K}^{i+K} \sum_{j'=i'-K}^{i'+K} w_{j-i,j'-i'}x_{j,j'}\right)$$
Imagenet 2012 competition [Krizhevsky et al., 2012]

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Imagenet is an annual computer vision competition. Classify images in 1000 classes. 1.2 million training images, 50,000 validation images, and 150,000 testing images.
The first 5 layers of weights are convolutional. The last three layers are fully connected.

The first convolutional layer filters the $224 \times 224 \times 3$ input image with 96 kernels (feature maps) of size $11 \times 11 \times 3$ with a stride of 4 pixels (this is the distance between the receptive field centers of neighboring neurons in a kernel map).

The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with 256 kernels of size $5 \times 5 \times 48$.

---

23 See paper for details.
The third convolutional layer has 384 kernels of size $3 \times 3 \times 256$

The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$

The fifth convolutional layer has 256 kernels of size $3 \times 3 \times 192$

(No pooling or normalization).

In layers 2,4,5 kernel computation on each GPU takes input only from own GPU.
The fully-connected layers have 4096 neurons each.

The output of the last fully-connected layer is fed to a 1000-way softmax which produces a distribution over the 1000 class labels.

All non-linearities are ReLU.

The network is trained to maximize the multinomial logistic regression objective.

60 million parameters and 650,000 neurons.
Imagenet 2012 competition [Krizhevsky et al., 2012]

Data augmentation: During training replace each $256 \times 256$ image by 2048 images of size $224 \times 224$ (shifting and reflection).
At test time compute output for 5 images of size $224 \times 224$ (four corners and center) and average result.

Dropout in the first two fully-connected layers. Without dropout, there is substantial overfitting. Dropout roughly doubles the number of iterations required to converge.

![Figure 1: A four-layer convolutional neural network with ReLUs (solid line) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (dashed line). The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.](image)

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-1</th>
<th>Top-5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparse coding [2]</td>
<td>47.1%</td>
<td>28.2%</td>
</tr>
<tr>
<td>SIFT + FVs [24]</td>
<td>45.7%</td>
<td>25.7%</td>
</tr>
<tr>
<td>CNN</td>
<td>37.5%</td>
<td>17.0%</td>
</tr>
</tbody>
</table>

Table 1: Comparison of results on ILSVRC-2010 test set. In *italics* are best results achieved by others.
Imagenet 2012 competition [Krizhevsky et al., 2012]

Figure 4: (Left) Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). (Right) Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.
Dropout [Krizhevsky et al., 2012]

Combining the predictions of many different models is a very successful way to reduce test errors [Breiman, 2001].

‘dropout’ mimicks model averaging by setting to zero the output of each hidden neuron with probability 0.5. The neurons which are dropped out in this way do not contribute to the forward pass and do not participate in backpropagation. So every time an input is presented, the neural network samples a different architecture, but all these architectures share weights.

This technique reduces complex co-adaptations of neurons, since a neuron cannot rely on the presence of particular other neurons. It is, therefore, forced to learn more robust features that are useful in conjunction with many different random subsets of the other neurons.

At test time, either

- for each test pattern, average the output for random dropout realisations
- use all the neurons but multiply their outputs by 0.5
Batch normalization [Ioffe and Szegedy, 2015]

A $T$ layer network:

$$y_k = h \left( \sum_{j_T} w_{j_T j_{T-1}}^{(T)} h \left( \sum_{j_{T-1}} w_{j_{T-1} j_{T-2}}^{(T-1)} h \left( \ldots h \left( \sum_{j_1} w_{j_1 j_0}^{(1)} x_{j_0} \right) \right) \right) \ldots \right)$$

with $h$ a non-linear activation function. Or,

$$x^{(t)} = w^{(t)} a^{(t)} \quad a^{(t+1)} = h \left( x^{(t)} \right) \quad t = 1, \ldots, T$$

with $x^{(t)}$, $a^{(t)}$ the vectors of summed input and ouput neural activity and $w^{(t)}$ the $t$-th layer weight matrix.

**Vanishing gradient problem:** Empirically it is observed that $w^{(t)}$ tends to grow as the result of learning. Then $h \left( x^{(t)} \right)$ saturates. The gradient $h' \left( x^{(t)} \right) \to 0$ and this slows down learning. The problem is largest in early layers of deep networks.

**Co-dependence of learning in different layers:** Since $x^{(t)} = w^{(t)} h \left( w^{(t-1)} a^{(t-1)} \right)$, the optimal value of $w^{(t)}$ depends on optimal values of $w^{(t-1)}$. 
Batch normalization (Ioffe and Szegedy, 2015)

Batch normalization: for each neuron $i$ consider the distribution of values $x_i$ in a mini batch of training samples.

Normalize this distribution to zero mean and variance one as $\hat{x}_i = \frac{x_i - \mu_i}{\sigma_i}$, with $\mu_i, \sigma_i^2$ the mean and variance of $x_i$ in the mini-batch. Note, that in this way $\hat{x}_i$ becomes largely insensitive to changes previous layer weights.

Introduce new parameters $\beta_i, \gamma_i$ and define the batch normalization as

$$\text{BN}(x_i) = \gamma_i \hat{x}_i + \beta_i$$

Note, that $\text{BN}(x_i)$ depends on all values of $x_i$ in the mini-batch (through $\mu_i, \sigma_i^2$). $\beta_i, \gamma_i$ are additional parameters that are learned along with the rest of the network parameters through gradient descent.

Since $\beta_i, \gamma_i$ are layer specific, previous layer weight changes have less effect on the current layer and learning of layers is more independent, and therefore faster.
Batch normalization [Ioffe and Szegedy, 2015]

Test on MNIST problem with \((28 \times 28), 100, 100, 100, 10\) architecture.

![Graph showing test accuracy of MNIST network with and without Batch Normalization](image)

Figure 1: (a) The test accuracy of the MNIST network trained with and without Batch Normalization, vs. the number of training steps. Batch Normalization helps the network train faster and achieve higher accuracy. (b, c) The evolution of input distributions to a typical sigmoid, over the course of training, shown as \(\{15, 50, 85\}\)th percentiles. Batch Normalization makes the distribution more stable and reduces the internal covariate shift.

BN can accelerate 14 times on imagenet and improves accuracy at the same time.
Batch normalization \cite{Ioffe and Szegedy, 2015}

Test on Imagenet classification task with a version of the Inception network.

![Graph showing single crop validation accuracy of Inception and its batch-normalized variants vs. the number of training steps.](image)

<table>
<thead>
<tr>
<th>Model</th>
<th>Steps to 72.2%</th>
<th>Max accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception</td>
<td>$3.1 \cdot 10^6$</td>
<td>72.2%</td>
</tr>
<tr>
<td>BN-Baseline</td>
<td>$1.3 \cdot 10^6$</td>
<td>72.7%</td>
</tr>
<tr>
<td>BN-x5</td>
<td>$2.1 \cdot 10^6$</td>
<td>73.0%</td>
</tr>
<tr>
<td>BN-x30</td>
<td>$2.7 \cdot 10^6$</td>
<td>74.8%</td>
</tr>
<tr>
<td>BN-x5-Sigmoid</td>
<td></td>
<td>69.8%</td>
</tr>
</tbody>
</table>

Figure 3: For Inception and the batch-normalized variants, the number of training steps required to reach the maximum accuracy of Inception (72.2%), and the maximum accuracy achieved by the network.

Results in 2-10 times faster convergence. \footnote{Increase learning rate, increase learning rate decay, remove dropout, reduce $L_2$ weight regularization, ...}
How to understand the good generalization performance of deep learning?

Traditionally, good generalization is explained by:

- the structure in the data matches the structure in the model.
- sufficient data

Deep networks can essentially learn anything and often have insufficient data. How can we understand their good generalization performance?

Here we look at

- regularization, data augmentation, batch normalization
- Implicit regularization of GD and memorization

---

25For instance, the data is linearly separable and the model is a perceptron.
Deep neural networks easily fit random labels

[Zhang et al., 2016]

Figure 1: Fitting random labels and random pixels on CIFAR10. (a) shows the training loss of various experiment settings decaying with the training steps. (b) shows the relative convergence time with different label corruption ratio. (c) shows the test error (also the generalization error since training error is 0) under different label corruptions.

Regularization not always prevents overfitting: Adding regularization to MLP and

26Cifar10 consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

27Inception architecture obtained 5.6 % top-5 test error and 21 % top-1 test error on Imagenet in 2016

[Szegedy et al., 2016]
Inception on Cifar10 still yields zero training error.
Regularization helps, but generalization without regularization is also very good.
- Cifar10 (table 1) Inception achieves 85.7 % versus 89 % top-5 accuracy
- Imagenet (table 2) Inception achieves 80.4 % top-5 accuracy without regularization, compared to 83.6 % the winner of ILSVRC 2012 (Krizhevsky et al., 2012).
Data augmentation may yield significantly better generalization (Left figure blue versus yellow).
Early stopping works in some cases (left) but not in others (right)
Batch normalization (BN) may significantly improve convergence and generalization on Cifar10 (right and table 1).
Implicit regularization of gradient descend [Zhang et al., 2016]

When number of samples $N$ is smaller than the number of parameters in the network and is trained with (stochastic) gradient descend (S)GD, one observes *memorization*: the network explicitly represents the training data in some form.

We illustrate this for the problem of linear regression $y = \sum_{i=1}^{d} w_i x_i$.

Given the data $\{x_i^\mu, t^\mu, \mu = 1, \ldots, N\}$ and cost

$$L(w) = \sum_{\mu} (y^\mu - t^\mu)^2 \quad y^\mu = \sum_{i=1}^{d} w_i x_i^\mu$$

A solution $w$ with $L(w) = 0$ satisfies

$$t^\mu = y^\mu = \sum_{i} w_i x_i^\mu \quad t = Xw$$

\[\text{28The problem can be easily generalized to higher dimensional output, non-linear output, and a more general cost function, where } y_j = \phi(\sum_{i=1}^{d} w_{ij} x_i) \text{ with } \phi \text{ an invertible function (such as sigmoid for instance) and } L(w) = \sum_{\mu} F(y^\mu, t^\mu) \text{ with } F(y, t) \geq 0 \text{ and such that } F(y, t) = 0 \text{ if } y = t \text{ and } y_j^\mu = \phi(\sum_{i=1}^{d} w_{ij} x_i^\mu).\]
with $X_{(N \times d)} = x_i^\mu$ and $t_{(N \times 1)} = t^\mu$.

We have $N$ equations and $d$ unknowns. When $N < d$ there are many solutions.
Implicit regularization of gradient descend [Zhang et al., 2016]

The gradient is a linear combination of the training data:

\[
\frac{\partial L}{\partial w_i} = 2 \sum_{\mu} (y^\mu - t^\mu)x_i^\mu \quad \Delta w_i^t = -\eta_t \frac{\partial L}{\partial w_i}
\]

Therefore, starting at \( w = w^0 \) the (S)GD solution is a linear combination of the training data

\[
w_i = w_i^0 + \sum_{t=1}^{T} \Delta w_i^t = w_i^0 + \sum_{\mu} \alpha_\mu x_i^\mu \quad w = w_0 + X^T \alpha
\]

with \( \alpha_{(N \times 1)} = \alpha^\mu \). Since also \( Xw = t \), we obtain

\[
XX^T \alpha = t - Xw_0
\]

Since \( N < d \), \( XX^T \) is full rank and invertible. Thus, gradient descend finds a unique solution

\[
w = w_0 + X^T \alpha \quad \alpha = \left(XX^T\right)^{-1} (t - Xw_0)
\]
but depends on the initialization $w_0$.

The idea is easily extended to single layer networks with multiple outputs by observing that each output defines an independent non-linear regression problem.
Implicit regularization of gradient descend [Zhang et al., 2016]

When \( w_0 = 0 \), the (S)GD solution is also the minimum norm solution:

\[
\min_w \frac{1}{2} \|w\|^2 \quad \text{s.t} \quad Xw = t
\]

Introducing Lagrange multipliers we get

\[
C = \frac{1}{2} \|w\|^2 + \sum_{\mu} \lambda^\mu \left( t^\mu - \sum_i w_i x_i^\mu \right)
\]

\[
\frac{\partial C}{\partial w_i} = w_i - \sum_{\mu} \lambda^\mu x_i^\mu = 0
\]

Thus, \( w = X^T \lambda \) and \( Xw = t \). The solution is identical to the solution GD solution with \( \lambda = \alpha \) and \( w_0 = 0 \).

The minimum norm solution is the most 'smooth' solution consistent with training error zero. Therefore, (S)GD on overparametrized problems has an implicit regularization.
Gradient descent solution memorizes the training set

The final solution is

\[ y = \sum_i w_i x_i = \sum_{i,\mu} \alpha^\mu x_i^\mu x_i \]

By construction, when \( x = x^\mu \) the output \( y = t^\mu \).

For \( x \approx x^\mu \), the output \( y \approx t^\mu \).

The solution 'memorizes' the training set and is expected to give good generalization for data (very) near individual training samples and bad generalization everywhere else.

So the answer to the question: why do deep neural networks generalize well is that they are tested on data near the training samples.
Memorization in overparametrized auto encoders

[Radhakrishnan et al., 2018]

We can further understand the properties of deep networks by studying iterated autoencoders.

Consider an autoencoder defined by a function $f : \mathbb{R}^d \rightarrow \mathbb{R}^d$. After successful training, all training samples satisfy $f(x^\mu) = x^\mu$. 
Memorization in overparametrized auto encoders

[Radhakrishnan et al., 2018]

Consider the dynamical system by iterating the autoencoder multiple times: $x_{t+1} = f(x_t)$. The training examples are local attractors, i.e. stable fixed points of the dynamics, if for $x_0$ in a neighborhood of $x^\mu$ we have

$$x_0 \to x_1 = f(x_0) \to x_2 = f(x_1) \to \ldots \to x_\infty = x^\mu$$

Local stability requires that all eigenvalues $\lambda_i$ of the Jacobian

$$J_{ij} = \frac{\partial f_i}{\partial x_j}(x^\mu)$$

satisfy $|\lambda_i| < 1$. Otherwise, the point is unstable.
Memorization in overparametrized auto encoders

[Radhakrishnan et al., 2018]

(Top left) Interpolating solution of an overparameterized autoencoder (blue line) that perfectly fits the training data (stars). Such an arbitrary solution is possible, but the solution found by a shallow overparameterized autoencoder is contractive towards the training examples (top right), which is illustrated by iteration (bottom left). (Bottom right) The solution learned by a deep overparameterized autoencoder is strongly attractive towards training examples.
When $f$ is an overparameterized autoencoder, trained with (S)GD, the training examples are not only fixed points, but also attractors.

A fully-connected autoencoder (10 layers, 512 hidden neurons) was trained to convergence on 100 samples (red points) from the 3D Swiss Roll dataset. The trajectories of 1000 uniformly spaced points (green points) were computed by iterating the autoencoder over the points. From left to right, the plots represent the results after 0, 11, 111, and 1111 iterations. Most of the trajectories converge to training examples, thereby demonstrating that the autoencoder is contractive towards training examples.
Memorization in overparametrized auto encoders

[Radhakrishnan et al., 2018]

When $f$ is an overparameterized autoencoder, trained with (S)GD, the training examples are not only fixed points, but also attractors.

(a) A fully connected autoencoder (7 layers, 128 hidden neurons) was trained to convergence on 100 samples from the MNIST dataset. We show the trajectories of random test examples obtained by iterating the autoencoder over the images together with the nearest neighbor (NN) training image. (b) A fully connected autoencoder (14 layers, 128 hidden neurons) was trained to convergence on 20 samples from the MNIST dataset. Here the training examples are superattractors: the autoencoder maps arbitrary input images directly to a training example.
Memorization in overparametrized auto encoders

[Radhakrishnan et al., 2018]

Memorization increases with network width and depth.

$R_t$ measures the number of training patterns that are attractors after $t$ iterations.

$$R_t := \mathbb{P}_{x_0 \sim P_{\text{train}}} \left\{ \min_{x \in D_{\text{test}}} \| f^t(x) - x_0 \| < \epsilon \right\},$$

Autoencoders of varying width (a) and depth (b) were trained on 1000 samples from the MNIST dataset. X-axis: number of iterations. Y-axis: recovery probability of training examples.

Autoencoders of varying width (a) and depth (b) were trained on 1000 samples from the MNIST dataset. X-axis: number of iterations. Y-axis: number of attractive training examples.
Memorization in overparametrized auto encoders

[Radhakrishnan et al., 2018]

Same holds for convolutional networks.

(a) A convolutional autoencoder was trained to convergence on 100 samples from the CIFAR dataset. We show the trajectories of random test examples obtained by iterating the autoencoder over the images together with the nearest neighbor (NN) training image. (b) When trained on 10 examples, one from each class of CIFAR10, the nonlinear autoencoder maps arbitrary input images to training examples. The downsampling architecture used is from Figure 1a in Supplementary Material G modified with leaky ReLU activations.
Memorization and generalization coexist

[Radhakrishnan et al., 2018]

**Memorization**: overparameterized autoencoders exhibit a form of data-dependent self-regularization that encourages solutions that concentrate around the training examples.

Memorization coexists with generalization (slope close to 1) and no overfitting.

The conclusion is that deep learning in the overparametrized setting learns locally linear or constant functions around the training points. This somewhat explains the good generalization when tested on similar data.
Lecture 6

Graphical models

- Directed graphical models, conditional independence
- Time-series models

Promedas medical diagnostic expert system

Inference in graphical models, junction tree.

Learning graphical models

- convex problem when all variables are observed
- non-convex problem when some variables are hidden
Directed graphical models

Given $p(x_1, \ldots, x_n|\theta)$. How can we

- represent this distribution?
- infer some variables given some others?
- learn this distribution?

One can always write

$$p(x_1, \ldots, x_n) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \ldots p(x_n|x_1, \ldots, x_{n-1})$$

Problem is that $p(x_i|x_1, \ldots, x_{i-1})$ becomes a very big table for $i$ large. If $x_i$ has $K$ states we need $K^{i-1}(K - 1) \approx K^i$ parameters.

Parametrizing $p(x_i|x_1:i-1)$ may help. For instance for $x_i = \pm 1$ (binary data)

$$p(x_i|x_1:i-1) = \sigma \left( x_i \left( \sum_{j=0}^{i-1} w_j x_j \right) \right)$$
reduces $2^i$ to $i$ parameters.
Conditional independence

Conditional independence of $x$ and $y$ in the context $z$:

$$p(x, y|z) = p(x|z)p(y|z) \quad \text{(common cause)}$$

For instance when $t$ denotes time, and the future is independent of the past given the present:

$$p(x_1, \ldots, x_n) = \prod_{t=1}^n p(x_t|x_{t-1}) \quad p(x_t|x_1, \ldots, x_{t-1}) = p(x_t|x_{t-1})$$

Other structures:

$$p(x, y, z) = p(x|y)p(y|z)p(z) \quad \text{(chain)}$$
$$p(x, y, z) = p(z|x, y)p(x)p(y) \quad \text{(multiple cause)}$$
Conditional independence

In general $p(x_t|x_{1:t-1})$ may only depend on a sub set of $x_{1:t-1}$ which we denote as the parents of node $x_t$. A graphical model as a graphical representation of these (in)dependencies

\[
p(x_1, x_2, x_3, x_4, x_5) = p(x_1)p(x_2|x_1)p(x_3|x_1)p(x_4|x_2, x_3)p(x_5|x_3)
\]

\[
p(x_1, \ldots, x_n) = \prod_{t=1}^{n} p(x_t|x_{Pa(t)})
\]

---

Bert Kappen

ML 166
Naive Bayes classifier

\[ y \text{ is a class label } y = 1, \ldots, K. \ x_j \text{ is a feature. The naive Bayes model assumes that the features are independent given a class.} \]

\[ p(x_1, \ldots, x_n, y) = p(y) \prod_{j=1}^{n} p(x_j|y) \]

The parameters of this model can be easily estimated from data (see Murphy chapter 3.5). \[^{29}\]

It can be used for classification by using Bayes rule:

\[ p(y|x_1, \ldots, x_n) = \frac{p(x_1, \ldots, x_n, y)}{\sum_y p(x_1, \ldots, x_n, y)} \]

\[^{29}\text{The log likelihood is } L = \sum_\mu \log p(x_1^{\mu}, \ldots, x_n^{\mu}, y^{\mu}|\theta) = \sum_\mu \log p(y^{\mu}|\theta_0) + \sum_j \sum_\mu \log p(x_j^{\mu}|y^{\mu}, \theta_j) \text{ with } \theta = (\theta_0, \theta_1, \ldots, \theta_n). \]

Note, that the parameters \(\theta_0, \theta_1, \ldots, \theta_n\) can all be determined independently of each other.
Markov models

First order Markov model: \( p(x_{1:T}) = p(x_1) \prod_{t=2}^{T} p(x_t|x_{t-1}) \)

Second order Markov model is \( p(x_{1:T}) = p(x_1, x_2) \prod_{t=3}^{T} p(x_t|x_{t-1}, x_{t-2}) \)

Low order Markov models may fail to capture long range correlations (in the data). High order Markov models have too many parameters. An often used solution is the hidden Markov model (HMM)

\[
p(z_1, \ldots, z_T, x_1, \ldots, x_T) = p(z_1) \prod_{t=2}^{T} p(z_t|z_{t-1}) \prod_{t=1}^{T} p(x_t|z_t)
\]

Typically, hidden variable \( z_t \) much lower dimensional than observed \( x_t \). \( z_t \) may encode ‘words’ someone is speaking while \( x_t \) encodes the spectral wave forms.

\(^{30}\)You can think of classical physics as a second order Markov model. Position \( x_t \) and velocity \( v_t = \frac{x_t - x_{t-dt}}{dt} \) determine the next position \( x_{t+dt} \). Noise is zero in this case: \( p(x_{t+dt}|x_t, x_{t-dt}) = \delta(x_{t+dt} - f(x_t, v_t)) \) and \( p(x_1, x_2) \) determine the initial position and velocity.
State estimation: estimate $p(z_t|x_{1:t}, \theta)$
Learning: estimate $\theta$ in $p(x_t|z_t, \theta)$ and $p(z_t|z_{t-1}, \theta)$ from data (Murphy chapter 17).
Medical diagnosis

$h$ (any type) denote conditions (given)
$d$ (binary) denote diagnoses (unknown or given)
$t$ (any type) denote tests (unknown or given)

\[ p(t_1, \ldots, t_m, d_1, \ldots, d_n) = p(d_1) \ldots p(d_n) \]
\[ p(t_1|d_{Pa(1)}) \ldots p(t_m|d_{Pa(m)}) \]

All terms may depend on $h$ (age, gender, ...).

$p(d_j = 1)$ is the prevalence of the disease.
$p(t_i|d_{Pa(i)})$ explains symptom $t_i$ in terms of the relevant diagnoses $d_{Pa(i)}$. 
Differential diagnosis

Given some patient findings $t_{1:m}$, the probability of diagnose $j$ is given by Bayes formula:

\[
p(d_j = 1 | t_{1:m}) = \frac{p(d_j = 1, t_{1:m})}{p(t_{1:m})}
\]

\[
p(d_j, t_{1:m}) = \sum_{d_{1:n} \setminus j} p(t_{1:m}, d_{1:n})
\]

\[
p(t_{1:m}, d_{1:n}) = \prod_{j=1}^{n} p(d_j) \prod_{i=1}^{m} p(t_i | d_{\text{Pa}(i)})
\]

$d_{1:n}$ is the union of the parents of $t_{1:m}$.

The differential diagnosis is a sorted list.
One diagnose \( d = 0, 1 \) and one continuous test \( t \).

- \( p(x|d = 0) \) models temperature distribution of healthy person
- \( s(x|d = 1) \) models temperature distribution of person with flue
- Prevalence \( p(d = 1) = 0.01 \)
Multiple causes and explaining away

Suppose patient data $t_{1:m}$:

$$p(d_j = 1 | t_{1:m}) = \frac{p(d_j = 1, t_{1:m})}{p(d_j = 1, t_{1:m}) + p(d_j = 0, t_{1:m})}$$

Competition between:
$p(d_j = 1, t_{1:m})$: the probability that the $t_{1:m}$ can be explained by diagnose $j$
$p(d_j = 0, t_{1:m})$: the probability that the $t_{1:m}$ can be explained by another diagnose

Higher $p(d_j = 0, t_{1:m})$ makes $p(d_j = 1 | t_{1:m})$ smaller.

Specific tests have few parents: low $p(d_j = 0, t_{1:m})$, high posterior probabilities.
Non-specific tests have many parents: high $p(d_j = 0, t_{1:m})$, low posterior probabilities

Useful DD requires accurate modelling of this effect.
Finding most likely explanation

Posterior

\[
p(d_{1:n}|t_{1:m}) \propto \prod_{j=1}^n p(d_j) \prod_{i=1}^m p(t_i|d_{Pa(i)})
\]

has multiple ‘peaks’, each corresponding to a sub-set of diagnoses that may explain the patient data:

\[
t_{1:m} \rightarrow (d_1), (d_2, d_3), (d_1, d_3), (d_2, d_3, d_4), \ldots
\]

Differential diagnosis is the marginal probability of each diagnosis in the posterior. Computing \( p(d_j, t_{1:m}) \) is intractable.
Promedas for internal medicine

The current system contains:

- 3714 diagnoses of which 1437 medications with their side effects
- 1586 tests
- 1910 conditions (features such as ’africa visited’ or ’sushi chef’) which affect the prior probabilities of some diagnoses
- 56298 relations

The model is far from complete in terms of both variables and relations.
Example case

A 47-year old woman who was in the surgical ward because of a mastectomy 2 days earlier because of breast cancer, developed acute pain of the thorax, with a high pulse rate and breathlessness. A CT angio confirmed the presence of pulmonary embolism as the cause.
Example case

age: 47
gender: female

SYMPTOMS AND TESTS:
1 dyspnoea (short of breath, breathlessness) true
2 white cell count (total leukocytes) (x 10e9/l) above
3 pulse rate (n/min) above

CONDITIONS:
acute history true
previously: surgery true
previously: prolonged immobility true

computing meddle differential diagnosis...

MEDDL DIFFERENTIAL DIAGNOSIS 1 2 3
0.522852 acute appendicitis x
0.359756 pulmonary embolism x x
0.225452 streptococcus pneumoniae pneumonia (lobar pneumonia) x x x
0.123028 small/medium pulmonary embolism x x
0.101444 chronic obstructive pulmonary disease (copd) x x
0.067427 multiple recurrent pulmonary embolism x x
0.066730 atrial fibrillation (af) x x x
0.049631 acute pyelonephritis x
0.044869 atrial flutter x x x
0.031735 anaphylaxis (anaphylactic reaction) x x
0.028490 smoking x
0.018188 aortic regurgitation x x
0.014886 biventricular cardiac failure (congestive heart failure) x x
0.012509 massive pulmonary embolism x x
0.008003 group b beta haemolytic streptococcus x x
0.006688 extrinsic allergic alveolitis x x
0.005761 ischaemic colitis x
0.004573 otitis media x
0.004389 cholesterol crystal embolism (purple toe syndrome) x
0.004165 ascariasis (roundworm infection) x x

Promedas correctly gives pulmonary embolism as option 2 and 4. pneumonia, COPD, anaphylaxis are reasonable items in the differential diagnosis.
Example case

A complicated case that presented in the emergency room. A 71-year old woman, very obese, with shortness of breath, lethargy and general malaise, chronic diarrhea and palpitations. She also complained of chest pain. She used a lot of medication, including acetazolamide. Because of her general malaise and loss of appetite she had not been eating and drinking well lately, so she was dehydrated. In her history she had had a myocardial infarction, hypertension, type 2 diabetes mellitus, obstructive sleep apnea syndrome and more. Physical examination showed a very obese woman, with a BMI of > 45, an irregular pulse, and shallow breathing. Blood results: high white blood cell count, very low pH in her arterial blood gas, with an increased pCO2 and low bicarbonate. Her creatinine was strongly increased; her glucose was normal. ECG showed ST segment depression and atrial fibrillation. She was admitted to hospital for analysis, the final conclusion was chronic hypoventilation and obstructive sleep apnoe syndrome due to her extreme obesity (also known as obesity hypoventilation syndrome or Pickwickian syndrome) - and her arterial blood gas abnormalities (very low pH) were further affected by her diarrhea, worsening of her renal function (due to dehydration) and her continued use of acetazolamide. She also had atrial fibrillation, as the cause of her palpitations and irregular pulse.
Example case

### Case:
- **Age:** 71
- **Gender:** Female

#### Symptoms and Tests:
- Anorexia (loss of appetite)
- Chronic diarrhoea
- White cell count (total leukocytes) (x 10⁶/μl)
- Palpitations
- Chest pain like pressure or heaviness
- ECG: ST segment depression, elevation
- BMI (body mass index) (kg/m²)
- Irregular pulse
- Dyspnoea (short of breath, breathlessness)
- Lethargy
- Serum bicarbonate blood gases arterial (mm hg)
- Creatinine (μmol/l)
- PCO₂ blood gases arterial (mm hg)
- PO₂ blood gases arterial
- Sleep disturbance
- Glucose fasting (mmol/l)

#### Conditions:
- Chronic history
- Previously: myocardial infarction
- Previously: hypertension (treated)

#### Computer model differential diagnosis:

<table>
<thead>
<tr>
<th>Condition</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atrial fibrillation (af)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Bi-ventricular cardiac failure (congestive heart failure)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Atrial flutter</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Aortic stenosis</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Obesity</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Alcohol consumption (excess)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Chronic renal failure (crf)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Anaphylaxis (anaphylactic reaction)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Chronic obstructive pulmonary disease (copd)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Hyperkalaemia due to alcohol</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Chronic widespread pain (previously Fibromyalgia)</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Chronic fatigue syndrome</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Obstructive sleep apnoea</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Hypoaldosterone side effects</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Lactic acidosis</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Dementia</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Menopause</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Depressive conduct disorder</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Hypercalcemia severe</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Bisphosphonate side effects</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
</tbody>
</table>

Atrial fibrillation is correct. Obesity correctly explains extreme high BMI and dyspnoea. The chronic renal failure is correct as well. Promedas does not contain the diagnosis obesity hypoventilation syndrome (or Pickwickian syndrome) yet, so could not produce that yet.
Promedas for emergency medicine

Old Promedas discontinued in 2012. Problems:
- too large domain
- hard to find accurate probabilities in literature
- intractable computation

Since 2020, restart in collaboration with Erasmus MC Rotterdam.
- 100s of diagnoses instead of 1000s
- better model and inference method
Inference

A typical operation on a probability distribution is to compute the statistics of some variables given some other variables. Separate all variables in three subsets $x = (x_a, x_b, x_c)$ with $a, b, c$ a partition of the indices $1, \ldots, n$. Then

$$p(x_a | x_b) = \frac{p(x_a, x_b)}{p(x_b)} \quad p(x_a, x_b) = \sum_{x_c} p(x_a, x_b, x_c) \quad p(x_b) = \sum_{x_a} p(x_a, x_b)$$

For discrete variables with $K$ values, we need $O(K^n)$ operations. The junction tree algorithm reduces this to $O(nK^w)$ operations with $w \leq n$ the width of the tree (Murphy chapter 20).

- When the graph is a tree or chain, $w = 1$ and computation time is linear in $n$
- When graph is sparse $w \ll n$ and junction tree efficient
- When graph is dense $w \approx n$ and junction tree useless

\[^{31}\] For instance, in medical diagnosis, $x_a = d_j$ is a diagnose variable, $x_b = t_{1:m}$ a subset of known symptoms and tests, and $x_c$ is the collection of all other variables (diagnoses and tests) in the model.

\[^{32}\] When $p(x_a, x_b, x_c)$ is a multivariate Gaussian distribution all these operations can be done in $O(n^3)$ time.
Junction tree algorithm

Convert the graph to a tree of cliques.

Inference in trees is linear in the number of nodes.

Junction tree is exponential in the size of the largest clique.
Learning

Learning is defined by maximum posterior: given data \( x^\mu, \mu = 1, \ldots, P \), find \( \theta \) such that

\[
L(\theta, D) = p(\theta) \prod_{\mu=1}^{P} p(x^\mu|\theta) \quad \text{log } L(\theta) = \sum_{\mu=1}^{P} \log p(x^\mu|\theta) + \log p(\theta)
\]

is maximal.

\( p(\theta) = \prod_{i=1}^{n} p_i(\theta_i) \) is the prior on \( \theta \). If \( p(\theta) = \text{const.} \) we obtain the maximum likelihood estimate.

Note, that we treat \( \theta \) also as a random variable. \( L(\theta, D) \) is then a naive Bayes graphical model. It says that given \( \theta \) the different data samples \( x^1, \ldots, x^P, \theta \) are independent. We can consider learning as Bayesian inference and infer \( p(\theta|D) \) by Bayes rule.
Learning complete data

When $p(x|\theta)$ is a DAG, we can write

$$p(x|\theta) = \prod_{i=1}^{n} p(x_i|x_{\text{Pa}(i)}, \theta_i)$$

Each factor $i$ has its own parameters $\theta_i$.

With data on all variables, the log likelihood factorizes in a sum of independent terms:\[33\]

$$L(\theta) = \sum_{i=1}^{n} \sum_{\mu} \log p(x_i^\mu|x_{\text{Pa}(i)}^\mu, \theta_i) + \sum_{i} \log p_i(\theta_i) = \sum_{i} L_i(\theta_i)$$

with $\theta_i$ the parameters of factor $i$.

So we can optimize the parameters of each term in the DAG independently.

\[33\] We assume that the prior factorizes over the components of the DAG $p(\theta) = \prod_i p_i(\theta_i)$. 
Tabular case

Treat each table entry $p(x_i|x_{Pa(i)})$ as a parameters $\theta_{x_i|x_{Pa(i)}}$. The index $x_i|x_{Pa(i)}$ labels the different parameters. Then maximize $L$ (no prior) with respect to $p(x_i|x_{Pa(i)})$ with Lagrange multiplier for normalization:

$$L = \sum_{i=1}^{n} \sum_{\mu} \log p(x_i^\mu|x_{Pa(i)}^\mu) + \sum_{i=1}^{n} \sum_{x_{Pa(i)}} \lambda_i(x_{Pa(i)}) \left( \sum_{x_i} p(x_i|x_{Pa(i)}) - 1 \right)$$

$$\frac{\partial L}{\partial p(x_i|x_{Pa(i)})} = \frac{1}{p(x_i|x_{Pa(i)})} \sum_{\mu} \delta_{(x_i,x_{Pa(i)})=(x_i^\mu,x_{Pa(i)}^\mu)} + \lambda_i(x_{Pa(i)})$$

$$= \frac{N(x_i, x_{Pa(i)})}{p(x_i|x_{Pa(i)})} + \lambda_i(x_{Pa(i)}) = 0$$

with $N(x_i, x_{Pa(i)})$ the number of times the combination $x_i, x_{Pa(i)}$ occur in the data. The solution is

$$p(x_i|x_{Pa(i)}) = \frac{N(x_i, x_{Pa(i)})}{-\lambda_i(x_{Pa(i)})} = \frac{N(x_i, x_{Pa(i)})}{\sum_{x_i} N(x_i, x_{Pa(i)})}$$
Gaussian case

To illustrate the parametrized case, consider $x \in \mathbb{R}^n$ and Gaussian conditional models. Define

$$p(x_i | x_{\text{Pa}(i)}) = \mathcal{N}(x_i | a_i, \sigma^2)$$

$$a_i = \sum_{j=0}^{i-1} w_{ij} x_j$$

with fixed $\sigma$ and adaptive parameters $w_{ij}$. Note, that $x_i$ depends only on $x_j$ for $j < i$ to ensure a DAG. The log likelihood is

$$L(w) = \sum_{i=1}^{n} \sum_{\mu} \log \mathcal{N}(x_i^\mu | a_i^\mu, \sigma^2)$$

$$a_i^\mu = \sum_{j=0}^{i-1} w_{ij} x_j^\mu$$

$$= \sum_{i=1}^{n} \sum_{\mu} -\frac{1}{2\sigma^2} \left( x_i^\mu - \sum_{j=0}^{i-1} w_{ij} x_j^\mu \right)^2 + \text{const.}$$
The gradient is:

$$\frac{\partial L}{\partial w_{ij}} = \frac{1}{\sigma^2} \sum_{\mu} \left( x_i^\mu - \sum_{j'=0}^{i-1} w_{ij'} x_{j'}^\mu \right) x_j^\mu \quad j < i$$

Define

$$b_j = \sum_{\mu} x_i^\mu x_j^\mu \quad C_{jj'} = \sum_{\mu} x_j^\mu x_{j'}^\mu \quad j, j' = 0, \ldots, i - 1$$

Suppress the $i$ index temporarily, so that $w_{ij} = w_j$. Then

$$\frac{\partial L}{\partial w_{ij}} = b - wC = 0 \quad w = C^{-1}b.$$
Learning with missing and/or latent variables

Sofar, we saw

\[ p(x) = \prod_{i=1}^{n} p(x_i | x_{Pa(i)}) \]
\[ \log L = \sum_{i=1}^{n} \sum_{\mu} \log p(x^\mu_i | x^\mu_{Pa(i)}) \]

The parameters of each term \( p(x_i | x_{Pa(i)}) \) can be optimized independent of the other terms.

In many cases there are latent, or hidden, variables that are not observed in the data. Write \( x = (x_v, x_h) \) we only have data \( x^\mu_v \). Then

\[ L(\theta) = \sum_{\mu} \log p(x^\mu_v | \theta) + \log p(\theta) \]
\[ p(x_v | \theta) = \sum_{x_h} p(x_v, x_h | \theta) = \sum_{x_h} \prod_{i=1}^{n} p(x_i | x^\mu_{Pa(i)}, \theta_i) \]

The \( \log \sum_{x_h} \) prevents to write \( \log L \) as a sum of terms, each involving one factor.

In this case learning is more difficult and uses the EM algorithm.
Example: the naive Bayes model

\[ x_v = (x_1, \ldots, x_n) \text{ and } x_h = y. \]

\[ p(x_1, \ldots, x_n, y) = p(y) \prod_{j=1}^{n} p(x_j|y) \]

Data \( x^\mu = (x_1^\mu, \ldots, x_n^\mu) \). The log likelihood is

\[ \log L = \sum_\mu \log \left( \sum_y p(y) \prod_{j=1}^{n} p(x_j^\mu|y) \right) \]

is not a sum of terms, each involving one of the factors \( p(y) \) or \( p(x_i|y) \).
Data likelihood is convex for observed data

Consider a model of the exponential family

\[ p(x, k|\theta) = \frac{1}{Z(\theta)} e^{\sum_{r=1}^{R} \theta_r \Phi_r(x, k)} \]

\[ Z(\theta) = \sum_{x, k} e^{\sum_{r=1}^{R} \theta_r \Phi_r(x, k)} \]

The characteristic of these models is that the term in the exponent is linear in \( \theta \) (aka log linear models). Most distributions (Gaussian, Dirichlet, multinomial) are in the exponential family.

In the case of \( N \) observed data \( x^\mu, k^\mu \), the log likelihood is

\[ L(\theta) = \sum_{\mu} \log p(x^\mu, k^\mu|\theta) = \sum_{r} \theta_r \sum_{\mu} \Phi_r(x^\mu, k^\mu) - N \log Z(\theta) \]

\( Z(\theta) \) is a convex function (Boyd 2004). Difference of linear and convex function is concave. Thus \( L(\theta) \) has unique maximum.
Data likelihood is non-convex for latent variables

In the case of $N$ observed data $x^\mu$, the log likelihood is

$$L(\theta) = \sum_\mu \log \sum_k p(x^\mu, k|\theta) = \sum_\mu \log \sum_k e^{\sum_r \theta_r \phi_r(x^\mu, k)} - N \log Z(\theta)$$

$Z(\theta)$ is a convex function and $\log \sum \exp$ is convex (Boyd 2004). Thus $L(\theta)$ is a difference of convex functions and is not convex.
Lecture week 7

Mixture models

Clustering

The expectation maximization algorithm (EM)

Variational EM
Mixture models and the EM algorithm

There are two approaches to model correlations in data:

- use observable variables only, with complex interactions between them.
- use additional latent variables (latent variable models, LVMs).

Latent variable models have less parameters, require less data to learn, but harder to learn.

Left: $3 + 8 + 2 + 2 + 2 = 17$. Right: $3 + 8 + 16 + 32 = 59$. 
Clustering

Consider a model

\[ p(x, k) = p(k)p(x|k) \]

with \( x \) observed (continuous or real) and \( k = 1, \ldots, K \) a discrete latent variable. The log likelihood is

\[ \log L = \sum_{\mu} \log p(x^\mu) \quad p(x^\mu) = \sum_k p(x^\mu, k) \]

The sum expresses that different clusters contribute to the probability of the data point \( x^\mu \). In (hard) clustering, the sum is replaced by the single most contributing term \[34\].

\[ p(x^\mu) = \sum_k p(x^\mu, k) \approx p(x^\mu, k^\mu) \quad k^\mu = \arg\max_k p(x^\mu, k) \]

\[ \log L \approx \sum_{\mu} \log p(k^\mu) + \sum_{\mu} \log p(x^\mu|k^\mu) \]

\[34\text{NB: } k^\mu \text{ depends on the parameters of the model and changes during the algorithm} \]
Mixtures of multinomials

A distribution over $d$ dimensional binary vectors $x = (x_1, \ldots, x_d)$ with $x_i = 0, 1$.

$$p(x|\theta) = \sum_{k=1}^{K} \pi_k p(x|k) \quad p(x|k) = \prod_{j=1}^{d} \mu_{jk}^{x_j}(1 - \mu_{jk})^{1-x_j}$$

with $\theta = \{\pi_k, \mu_{jk}\}$ and $\sum_k \pi_k = 1$ and $0 \leq \mu_{jk} \leq 1$.

We derive a clustering algorithm by adding Lagrange multipliers:

$$\log L \approx \sum_{\mu} \log \pi_{k\mu} + \sum_{\mu, j} \left( x_j^{\mu} \log \mu_{jk\mu} + (1 - x_j^{\mu}) \log(1 - \mu_{jk\mu}) \right) + \lambda \left( \sum_k \pi_k - 1 \right)$$

with $k^{\mu} = \text{argmax}_k p(x^{\mu}, k)$. 
Mixtures of multinomials

Setting the derivatives to zero, we obtain:

$$\frac{\partial \log L}{\partial \pi_k} = \frac{1}{\pi_k} \sum_{\mu} \delta_{k,\mu} + \lambda = 0 \quad \rightarrow \quad \pi_k = \frac{N_k}{N}$$

with $N_k = \sum_{\mu} \delta_{k,\mu}$ the number of data samples that are assigned to cluster $k$ and $N = \sum_k N_k$ the total number of data samples.

$$\frac{\partial \log L}{\partial \mu_{jk}} = \frac{1}{\mu_{jk}} \sum_{\mu} x_{j}^{\mu} \delta_{k,\mu} - \frac{1}{1 - \mu_{jk}} \sum_{\mu} (1 - x_{j}^{\mu}) \delta_{k,\mu}$$

$$= N_k \left( \frac{m_{jk}}{\mu_{jk}} - \frac{1 - m_{jk}}{1 - \mu_{jk}} \right) = 0 \quad \rightarrow \quad \mu_{jk} = m_{jk}$$

with

$$m_{jk} = \frac{1}{N_k} \sum_{\mu} x_{j}^{\mu} \delta_{k,\mu} = \frac{1}{N_k} \sum_{\mu \in k} x_{j}^{\mu}$$

the mean of the data that is assigned to cluster $k$. 
Mixtures of multinomials

Algorithm:

1. Initialize $\pi_k, \mu_{jk}$ random.

2. For $t = 1, 2, \ldots$,
   
   (a) For $\mu = 1, \ldots, N$ compute $k^{\mu} = \arg\max_k p(x^{\mu}, k)$
   
   (b) For $k = 1, \ldots, K$ compute $N_k$. Set $\pi_k := N_k / N$
   
   (c) For $k = 1, \ldots, K$, $j = 1, \ldots, d$ compute $m_{jk}$. Set $\mu_{jk} = m_{jk}$
Mixtures of multinomials

MNIST data $N = 60000, K = 10, d = 784$. Estimated $\pi_k$ and $\mu_{jk}$ from data.

Result is not very good:
- some digits are not represented, others are double.
- Result is not reproducible: different initializations yield different results.
- Model is too simple: assumes all pixels independent given the cluster.
- Some digits might require multiple clusters (different ways to write a 4).
Mixtures of Gaussians

A distribution over $d$ dimensional continuous vectors.

$$p(x|\theta) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$$

with $\theta = \{\pi_k, \mu_k, \Sigma_k\}$.

Left: mixture of $K = 3$ three Gaussians in $d = 2$.
Right: $d = 7$ dimensional genetic data and $K = 16$ estimated cluster centers $\mu_k$. 
The original EM method was introduced in [Baum et al., 1970].

Consider a model $p(x, k|\theta)$. The problem is to find $\theta$ that maximizes the data likelihood on the observed data $x$:

$$L(\theta) = \sum_x D(x) \log p(x|\theta) = \sum_x D(x) \log \sum_k p(x, k|\theta)$$

with $D(x)$ the empirical distribution of $x$\textsuperscript{35}. The log $\sum$ makes optimisation hard.

Instead of maximizing $L$ we compute a lower bound and maximize that. For given $x$:

$$L_x(\theta) = \log \sum_k p(x, k|\theta) = \log \sum_k q_x(k) \frac{p(x, k|\theta)}{q_x(k)} \geq \sum_k q_x(k) \log \frac{p(x, k|\theta)}{q_x(k)} = Q_x(\theta, q)$$

This is called a variational or Jensen bound. The bound can be understood in\textsuperscript{35}

---

\textsuperscript{35}The empirical distribution is the distribution implied by the data set: $D(x) = \frac{1}{N} \sum_{\mu=1}^{N} \delta_{x,x^\mu}$ and $L(\theta) = \frac{1}{N} \sum_{\mu} \log p(x^\mu|\theta)$. We use $D(x)$ instead of $q(x)$ because we reserve the latter for the variational distribution in the EM algorithm.
terms of KL divergence:

\[
L_x(\theta) - Q_x(\theta, q) = \log p(x|\theta) - \sum_k q_x(k) \log \frac{p(x, k|\theta)}{q_x(k)} \\
= \sum_k q_x(k) \log \frac{q_x(k)}{p(k|x, \theta)} = KL(q_x|p(\cdot|x, \theta))
\]

The bound \( L_x(\theta) \geq Q_x(\theta, q) \) thus follows also from \( KL \geq 0 \).

**E step:** Maximizing \( Q_x(\theta_t, q) \) (or minimizing \( KL \)) with respect to \( q_x \) gives

\[
q_x^*(k) = p(k|x, \theta_t) \quad KL(q_x^*(k)|p(k|x, \theta_t)) = 0
\]

with \( \theta_t \) the parameters at iteration \( t \) of the algorithm. \( r^\mu_k = q_x^*(k) \) is called the responsability of cluster \( k \) for data point \( x^\mu \).  

NB: we have a different distribution \( q_x \) for each data point \( x \).  

NB: \( KL(q_x^*(k)|p(k|x, \theta_t)) = 0 \) can only be obtained when variational model \( q \) is sufficiently expressive.
Substitution gives

\[ Q_x(\theta, q^*_x) = \sum_k p(k|x, \theta_t) \log \frac{p(x, k|\theta)}{p(k|x, \theta_t)} \]

Note, that \(36\)

\[ Q(\theta_t, q^*) = L(\theta_t) \]

**M step:** maximize \(Q(\theta) = \sum_x D(x)Q_x(\theta, q^*_x)\) wrt \(\theta\).

\[ \theta_{t+1} = \arg\max_{\theta} \sum_x D(x)Q_x(\theta, q^*_x) = \arg\max_{\theta} \sum_x D(x) \sum_k p(k|x, \theta_t) \log p(x, k|\theta) \]

Note that the \(\sum_k\) now appears outside the log, whereas in the original log likelihood \(L(\theta) = \sum_x D(x) \log \sum_k p(x, k|\theta)\) it appears inside the log.

\(36\)

\[
Q(\theta_t, q^*) = \sum_{x,k} D(x)p(k|x, \theta_t) \log \frac{p(x, k|\theta_t)}{p(k|x, \theta_t)} = \sum_{x,k} D(x)p(k|x, \theta_t) \log p(x|\theta_t) = \sum_x D(x) \log p(x|\theta_t) = L(\theta_t)
\]
\[ \log p(x|\theta) \] is concave in \( \theta \) because \( p(x, k|\theta) \) is an exponential model. As a result, the maximization in the M step can be easily computed (for exponential models).
Convexity of $\log p(x|\theta)$

Here we show that $\log p(x|\theta)$ is concave in $\theta$ for an exponential model $p(x|\theta)$.

Consider an exponential model

$$p(x|\theta) = \exp\left(\sum_a \theta_a \phi_a(x) - \log Z(\theta)\right) \quad Z = \sum_x \exp\left(\sum_a \theta_a \phi_a(x)\right)$$

It is easy to show that

$$\frac{\partial \log Z}{\partial \theta_a} = \langle \phi_a \rangle \quad \frac{\partial^2 \log Z}{\partial \theta_a \partial \theta_b} = \langle \phi_a \phi_b \rangle - \langle \phi_a \rangle \langle \phi_b \rangle = C_{ab}$$

The covariance matrix $C$ is positive semi definite ($\sum_{ab} v_a C_{ab} v_b \geq 0$ for all vectors $v_a$). Therefore $\log Z(\theta)$ is convex in $\theta$. Since $\log p(x|\theta) = \sum_a \theta_a \phi_a(x) - \log Z(\theta)$ we conclude that $p(x|\theta)$ is concave in $\theta$. 

EM increases the data log likelihood

\[ L(\theta_t) = Q(\theta_t, q^*_t) \leq Q(\theta_{t+1}, q^*_t) \leq Q(\theta_{t+1}, q^*_{t+1}) = L(\theta_{t+1}) \]

First and second equality because the model \( q^* \) is sufficiently expressive. First inequality because of M-step/maximization. Second inequality because of E-step/Jensen’s bound.

Blue: \( Q(\theta, q^*_t) \) with \( Q(\theta_t, q^*_t) = L(\theta_t) \) (E step). Maximization yields \( \theta_{t+1} \) (M step).

Green: \( Q(\theta, q^*_t) \) with \( Q(\theta_{t+1}, q^*_t) = L(\theta_{t+1}) \). Maximization yields \( \theta_{t+2} \).
Maximum likelihood via EM for mixture of Gaussians

The model is

\[ p(x, k) = p(k)p(x | k) \quad p(k) = \pi_k \quad p(x | k) = \mathcal{N}(x | \mu_k, \Sigma_k) \]

with parameters \( \pi_k, \mu_k, \Sigma_k \).

**E step:** For each data point compute the responsability

\[ q^*_\mu(k) = p(k | x^\mu, \theta_t) = \frac{\pi_k \mathcal{N}(x^\mu | \mu_k, \Sigma_k)}{\sum_{k'} \pi_{k'} \mathcal{N}(x^\mu | \mu_{k'}, \Sigma_{k'})} = r_{\mu k} \quad k = 1, \ldots, K \]

\( p(k | x^\mu, \theta_t) \) does a 'soft assignment' of data point \( x^\mu \) to Gaussian component \( k \).
Maximum likelihood via EM for mixture of Gaussians

M step:

$$\theta_{t+1} = \arg\max_{\theta} \sum_x D(x) \sum_k q^*_x(k) \log p(x, k|\theta)$$

$$= \arg\max_{\theta} \sum_{\mu, k} r_{\mu k} (\log \pi_k + \log \mathcal{N}(x^\mu|\mu_k, \Sigma_k))$$

with $\theta_{t+1} = \{\pi_k, \mu_k, \Sigma_k\}_{t+1}$. Define $r_k = \sum_{\mu} r_{\mu k}$. The solution is

$$\pi_k = \frac{r_k}{N}, \quad (\mu_k)_i = \frac{\sum_{\mu} r_{\mu k} x_i^\mu}{r_k}, \quad (\Sigma_k)_{ij} = \frac{\sum_{\mu} r_{\mu k} x_i^\mu x_j^\mu}{r_k} - (\mu_k)_i(\mu_k)_j$$

This solution makes sense: the cluster mean and covariance are weighted sums of the points assigned to the cluster.
Maximum likelihood via EM for mixture of Gaussians

(a) Data and initial $\theta_0$ (b) E step computes responsibilities for $\theta_0$ (c) M step computes $\theta_1$ (d) after 3 iterations (e) after 5 iterations (f) after 16 iterations.
Variational EM

For complex models, finding a 'tabular' solution \( q(z) = p(z|x, \theta_t) \) in the E step may not always be possible. An example is the variational autoencoder.

Suppose we have some high dimensional data \( x \) (images for instance) that we want to compress into latent variables \( z \). We assume a model \( p(x, z|\theta) = p(z|\theta)p(x|z, \theta) \). This is a generative model that generates images \( x \):

\[
\begin{align*}
  z &\sim p(z|\theta) & \quad x &\sim p(x|z, \theta) \quad \text{(decoder)}
\end{align*}
\]

The problem is to estimate \( \theta \) such that \( p \) generates images that are similar to a given data set by maximizing the log likelihood \( \sum_{\mu} \log p(x^{\mu}|\theta) \).
So we can use the EM method. The variational distribution \( q(z|x, \phi) \) approximates the responsibilities \( p(z|x, \theta) \) as before, but with important differences:
- \( q(z|x) = q(z|x, \phi) \) is a parametrized distribution with parameters \( \phi \).
- In the E step, \( q(z|x, \phi) \) is optimized not for each \( x \) separately, but for all \( x \) simultaneously.

\[
Q_x(\theta, q) = \sum_k q_x(k) \log \frac{p(x, k|\theta)}{q_x(k)} \quad \rightarrow \quad Q(\theta, \phi) = \frac{1}{N} \sum_{\mu} \sum_z q(z|x^\mu, \phi) \log \frac{p(x^\mu, z|\theta)}{q(z|x^\mu, \phi)}
\]

Solid lines denote the generative model \( p(z|\theta)p(x|z, \theta) \) (decoder), dashed lines denote the variational approximation \( q(z|x, \phi) \) (encoder). The variational parameters \( \phi \) are learned jointly with the generative model parameters \( \theta \).
Variational EM

In simple tabular EM, the E step yielded the solution that saturated the bound:

\[ q^*_x(k) = p(k|x, \theta_t) \quad Q(\theta_t, q^*_x) = L_x(\theta_t), \]

In variational EM this is no longer true:

\[ q(z|x, \phi^*) \neq p(z|x, \theta_t) \quad Q_x(\theta_t, \phi^*) < L(\theta_t) \]

The consequence is that the property that each EM step increases the likelihood is no longer guaranteed (But if not, the bound gets better!)

Instead, one performs so-called coordinate ascent on \( Q(\theta, \phi) \), by alternating a gradient step in \( \phi \) for fixed \( \theta \) (E) and a gradient step for \( \theta \) for fixed \( \phi \) (M).

\[^{37}F(q, \theta)\] in the figures is \( Q(\theta, \phi) \) in my notation.
Variational Auto encoder (VAE) [Kingma and Welling, 2013]

Model the distribution $p(x)$ of binary MNIST data.

**Decoder:** Generate $z \sim N(z|0, 1)$ and $p(x|z, \theta)$ a MLP with one hidden layer:

$$p(x|z, \theta) = \prod_{i=1}^{n_x} \mu_i^x(1-\mu_i)^{1-x_i} \quad \mu_i = \sigma \left( \sum_{j=1}^{n_h} w^{(1)}_{ij} h_j + w^{(1)}_{i0} \right) \quad h_j = \tanh \left( \sum_{k=1}^{n_z} w^{(0)}_{jk} z_k + w^{(0)}_{j0} \right)$$

with $n_x, n_z$ the dimensions of the $x, z$ spaces and $n_h$ the number of hidden units and $\theta = \{w^{(0)}_{jk}, w^{(1)}_{ij}\}$. 

\[ \phi \quad \cdots \quad \theta \quad N \]
Variational Auto encoder (VAE) [Kingma and Welling, 2013]

Model the distribution $p(x)$ of continuous Frey face data.

Decoder: Generate $z \sim \mathcal{N}(z|0, 1)$ and $p(x|z, \theta)$ two MLPs with one hidden layer:

$$p(x|z, \theta) = \prod_{i=1}^{n_x} \mathcal{N}(x_i|\mu_i, \sigma^2_i) \quad \mu_i = \sum_{j=1}^{n_h} w^{(1)}_{ij} h_j + w^{(1)}_{i0}$$

$$\log \sigma^2_i = \sum_{j=1}^{n_h} w^{(2)}_{ij} h_j + w^{(2)}_{i0} \quad h_j = \tanh \left( \sum_{k=1}^{n_z} w^{(0)}_{jk} z_k + w^{(0)}_{j0} \right)$$

with $\theta = \{w^{(0)}_{jk}, w^{(1)}_{ij}, w^{(2)}_{ij}\}$. 
Variational Auto encoder (VAE) [Kingma and Welling, 2013]

Encoder:

\[ q(z|x, \phi) = \prod_{i=1}^{n_z} \mathcal{N}(z_i | \mu_i, \sigma^2_i) \]

\[ \mu_i = \sum_{j=1}^{n_h} w_{ij}^{(3)} h_j + w_{i0}^{(3)} \]

\[ \log \sigma^2_i = \sum_{j=1}^{n_h} w_{ij}^{(4)} h_j + w_{i0}^{(4)} \]

\[ h_j = \tanh \left( \sum_{k=1}^{n_z} w_{jk}^{(5)} z_k + w_{j0}^{(5)} \right) \]

with \( \phi = \{ w_{ij}^{(3)}, w_{ij}^{(4)}, w_{jk}^{(5)} \} \).

Bert Kappen ML 214
Variational Auto encoder (VAE) [Kingma and Welling, 2013]

Training with stochastic gradient descent. Weight decay (prior on $\theta$). Number of hidden units is 500 (MNIST) and 200 (Frey face).

Variational lower bound $Q(\theta, \phi)$ versus training iteration.

Bound improves with larger latent space dimension. No overfitting.
Variational Auto encoder (VAE) [Kingma and Welling, 2013]

Visualisation of latent $z$ space in case of $n_z = 2$. $z \sim \mathcal{N}(z|0, 1)$ and $x \sim p(x|z, \theta)$ with $\theta$ the parameters after training.
Variational Auto encoder (VAE) [Kingma and Welling, 2013]

Generated images in case of MNIST for increasing $n_z$. $z \sim \mathcal{N}(z|0, 1)$ and $x \sim p(x|z, \theta)$ with $\theta$ the parameters after training.


Marginal and conditional Gaussians

Consider a multivariate Gaussian distribution in \(d\) dimensions

\[
\mathcal{N}(x|\mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp \left( -\frac{1}{2} (x - \mu)' \Sigma^{-1} (x - \mu) \right)
\]

\[
\mu = \mathbb{E}x = \int x \mathcal{N}(x|\mu, \Sigma) dx
\]

\[
\Sigma = \mathbb{E}(x - \mu)(x - \mu)' = \int (x - \mu)(x - \mu)' \mathcal{N}(x|\mu, \Sigma) dx
\]

with \(\cdot\) is transpose and \(\mathbb{E}\) denotes expectation value.

Consider two sub sets of variables \(x_a\) and \(x_b\) with \(x = (x_a, x_b)'\) and

\[
\mu = (\mu_a, \mu_b)'
\]

\[
\Sigma = \begin{pmatrix}
\Sigma_{aa} & \Sigma_{ab} \\
\Sigma_{ba} & \Sigma_{bb}
\end{pmatrix}
\]
The marginal distribution of a Gaussian distribution is Gaussian

Marginal \( p(x_a) = \int dx_b N(x_a, x_b) \) is \( p(x_a) = N(x_a | \mu_a, \Sigma_{aa}) \).

Proof: It is easy to see that \( p(x_a) \) is Gaussian because it is an integral of a Gaussian. Since \( p(x_a) \) is Gaussian, it is fully specified by its mean and covariance:

\[
\mathbb{E}x_a = \int dx_a x_a p(x_a) = \int dx_a dx_b N(x_a, x_b) x_a = \mu_a
\]

\[
\mathbb{E}(x_a - \mu_a)(x_a - \mu_a)' = \int dx_a (x_a - \mu_a)(x_a - \mu_a)' p(x_a)
\]

\[
= \int dx_a dx_b (x_a - \mu_a)(x_a - \mu_a)' N(x_a, x_b) = \Sigma_{aa}
\]

where the last identities follow from the definition of the multivariate Gaussian on the previous slide. This completes the proof.
The conditional distribution of a Gaussian distribution is Gaussian

The conditional \( p(x_a|x_b) = \frac{N(x_a, x_b)}{N(x_b)} \) is Gaussian and

\[
\begin{align*}
p(x_a|x_b) &= N(x_a|\mu_a|x_b, \Sigma_a|x_b) \\
\Sigma_a|x_b &= \Sigma_a - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba} \\
\mu_a|x_b &= \mu_a + \Sigma_{ab}\Sigma_{bb}^{-1}(x_b - \mu_b)
\end{align*}
\]
Proof

The conditional \( p(x_a|x_b) \propto p(x_a, x_b) \) for constant \( x_b \).

Note, that

\[
 p(x_a, x_b) \propto \exp\left( -\frac{1}{2}(x - \mu)'K(x - \mu) \right)
\]

with \( x = (x_a, x_b) \) and where we define the *precision matrix* \( K = \Sigma^{-1} \).

The exponent can be written as (ignoring all constant terms):

\[
-\frac{1}{2}(x - \mu)'K(x - \mu) = -\frac{1}{2}x_a'K_{aa}x_a + x_a'K_{aa}\mu_a - x_a'K_{ab}(x_b - \mu_b)
\]

\[
= \frac{1}{2}x_a'K_{aa}x_a + x_a'K_{aa}(\mu_a - K_a^{-1}K_{ab}(x_b - \mu_b)) = \frac{1}{2}x_a'^{-1}\Sigma^{-1}_{a\mid b}x_a + x_a'^{-1}\Sigma^{-1}_{a\mid b}\mu_{a\mid b}
\]

\[
= \frac{1}{2}(x_a - \mu_{a\mid b})'\Sigma^{-1}_{a\mid b}(x_a - \mu_{a\mid b}) \quad \Sigma_{a\mid b} = K_a^{-1} \quad \mu_{a\mid b} = \mu_a - K_a^{-1}K_{ab}(x_b - \mu_b)
\]
Proof

We now need to relate $K_{aa}^{-1}, K_{ab}$ to the components $\Sigma$. Consider a matrix made up of block matrices $A, B, C, D$. Then

$$
\begin{pmatrix}
A & B \\
C & D
\end{pmatrix}^{-1} = \begin{pmatrix}
M & -MBD^{-1} \\
-D^{-1}CM & D^{-1} + D^{-1}CMBD^{-1}
\end{pmatrix}
$$

with $M = (A - BD^{-1}C)^{-1}$.

Since $K = \Sigma^{-1}$:

$$
\begin{pmatrix}
K_{aa} & K_{ab} \\
K_{ba} & K_{bb}
\end{pmatrix} = \begin{pmatrix}
\Sigma_{aa} & \Sigma_{ab} \\
\Sigma_{ba} & \Sigma_{bb}
\end{pmatrix}^{-1} = \begin{pmatrix}
M & -M\Sigma_{ab}\Sigma_{bb}^{-1} \\
-\Sigma_{bb}^{-1}\Sigma_{ba}M & \Sigma_{bb}^{-1} + \Sigma_{bb}^{-1}\Sigma_{ba}M\Sigma_{ab}\Sigma_{bb}^{-1}
\end{pmatrix}
$$

with $M = (\Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba})^{-1}$. Thus, $K_{aa} = M$ and

$$
\begin{align*}
\Sigma_{a|b} &= K_{aa}^{-1} = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba} \\
\mu_{a|b} &= \mu_a - K_{aa}^{-1}K_{ab}(x_b - \mu_b) = \mu_a + M^{-1}M\Sigma_{ab}\Sigma_{bb}^{-1}(x_b - \mu_b) \\
&= \mu_a + \Sigma_{ab}\Sigma_{bb}^{-1}(x_b - \mu_b)
\end{align*}
$$