COURSE ADMIN
TERM TIMELINE

First class
Sep 5/6

Midterm

Final class
Dec 7/11

Python tutorial

TensorFlow tutorial

Final project due

Peter Orbanz

John Cunningham

Dates

<table>
<thead>
<tr>
<th>Event</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Python tutorial</td>
<td>12/13 September</td>
</tr>
<tr>
<td>TensorFlow tutorial</td>
<td>24/25 October</td>
</tr>
<tr>
<td>Midterm exam</td>
<td>19/23 October</td>
</tr>
<tr>
<td>Final project due</td>
<td>11 December</td>
</tr>
</tbody>
</table>
ASSISTANTS AND GRADING

Teaching Assistants

Ian Kinsella  and  Wenda Zhou

Office Hours  Mon/Tue 5:30-7:30pm, Room 1025, Dept of Statistics, 10th floor SSW

Class Homepage

https://wendazhou.com/teaching/AdvancedMLFall17/

Homework

• Some homework problems and final project require coding
• Coding: Python
• Homework due: Tue/Wed at 4pm – no late submissions
• You can drop two homeworks from your final score

Grade

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Homework</td>
<td>20%</td>
</tr>
<tr>
<td>Midterm Exam</td>
<td>40%</td>
</tr>
<tr>
<td>Final Project</td>
<td>40%</td>
</tr>
</tbody>
</table>
Email

All email to the TAs, please.

The instructors will not read your email unless it is forwarded by a TA.

Problems with exam/project

If you cannot take the exam or finish the project: You must let us know

at least one week before

the midterm exam/project due date.
The relevant course material are the slides.

Books (optional)

See class homepage for references.
Part I (Orbanz)

- Neural networks (basic definitions and training)
- Graphical models (ditto)
- Sampling algorithms
- Variational inference
- Optimization for GMs and NNs

Part II (Cunningham)

- NN software
- Convolutional NNs and computer vision
- Recurrent NNs
- Reinforcement learning
- Dimension reduction and autoencoders
Introduction
**Historical origins: Artificial intelligence and engineering**

Machines need to...

- recognize patterns (e.g. vision, language)
- make decisions based on experience (= data)
- predict
- cope with uncertainty

**Modern applications: (A few) Examples**

- medical diagnosis
- face detection/recognition
- speech and handwriting recognition
- web search
- recommender systems
- bioinformatics
- natural language processing
- computer vision

**Today**

Machine learning and statistics have become hard to tell apart.
Task
Balance the pendulum upright by moving the sled left and right.

- The computer can control only the motion of the sled.
- Available data: Current state of system (measured 25 times/second).
Learning and Statistics

Formalization

State = 4 variables (sled location, sled velocity, angle, angular velocity)
Actions = sled movements

The system can be described by a function

\[ f : S \times A \rightarrow S \]

(state, action) \mapsto state
After each run
Fit a function

\[ f : S \times A \rightarrow S \]
\[(\text{state}, \text{action}) \mapsto \text{state} \]

to the data obtained in previous runs.

Running the system involves:

1. The function \( f \), which tells the system “how the world works”.
2. An optimization method that uses \( f \) to determine how to move towards the optimal state.

Note well

Learning how the world works is a regression problem.
Our main topics

Neural networks
- Define *functions*
- Represented by directed graph
  - Each vertex represents a function
  - Incoming edges: Function arguments
  - Outgoing edges: Function values
  - Learning: Differentiation/optimization

Graphical models
- Define *distributions*
- Represented by directed graph
  - Each vertex represents a distribution
  - Incoming edges: Conditions
  - Outgoing edges: Draws from distribution
  - Learning: Estimation/inference
Supervised learning

Unsupervised learning

Problems
Classification
Regression
Clustering (mixture models)
HMMs
Dimension reduction (PCA)

Solutions
Functions
Distributions
Neural networks

\[ y_2 = \phi(v^T x) \]

Graphical models
Neural networks
- Representation of function using a graph
- Layers:
  \[ x \rightarrow g \rightarrow f \]
  - Symbolizes: \[ f(g(x)) \]
    "\( f \) depends on \( x \) only through \( g \)"

Graphical models
- Representation of a distribution using a graph
- Layers:
  \[ X \rightarrow Y \rightarrow Z \]
  - Symbolizes: \[ p(x, z, y) = p(z|y)p(y|x)p(x) \]
    "\( Z \) is conditionally independent of \( X \) given \( Y \)"
Grouping dependent variables into layers is a good thing.
**Historical perspective: McCulloch-Pitts Neuron Model (1943)**

A neuron is modeled as a “thresholding device” that combines input signals:

\[ y = \mathbb{I}\{\langle v, x \rangle > c \} \quad \text{for some } c \in \mathbb{R} . \]

**McCulloch-Pitts model**

- Collect the input signals \( x_1, x_2, x_3 \) into a vector \( x = (x_1, x_2, x_3) \in \mathbb{R}^3 \)
- Choose fixed vector \( v \in \mathbb{R}^3 \) and constant \( c \in \mathbb{R} \).
- Compute:

\[ y = \mathbb{I}\{\langle v, x \rangle > c \} \quad \text{for some } c \in \mathbb{R} . \]
Recall: Linear classifier

\[ f(x) = \text{sgn}(\langle v, x \rangle - c) \]
**Linear Classification**

$$f(x) = \text{sgn}(\langle v, x \rangle - c)$$
• The “neural network” represents a linear two-class classifier (on $\mathbb{R}^3$).
• It does not specify the training method.
• To train the classifier, we need a cost function and an optimization method.
For parameter estimation by optimization, we need an optimization target.

Idea: Choose 0-1 loss as simplest loss for classification.

Minimize empirical risk (on training data) under this loss.

Figure 5.11: Four learning criteria as a function of weights in a linear classifier. At the upper left is the total number of patterns misclassified, which is piecewise constant and hence unacceptable for gradient descent procedures. At the upper right is the Perceptron criterion (Eq. 16), which is piecewise linear and acceptable for gradient descent. The lower left is squared error (Eq. 32), which has nice analytic properties and is useful even when the patterns are not linearly separable. The lower right is the square error with margin (Eq. 33). A designer may adjust the margin \( b \) in order to force the solution vector to lie toward the middle of the \( b = 0 \) solution region in hopes of improving generalization of the resulting classifier.

Thus, the batch Perceptron algorithm for finding a solution vector can be stated very simply: the next weight vector is obtained by adding some multiple of the sum of the misclassified samples to the present weight vector. We use the term “batch” training to refer to the fact that (in general) a large group of samples is used when computing each weight update. (We shall soon see alternate methods based on single samples.) Figure 5.12 shows how this algorithm yields a solution vector for a simple two-dimensional example with \( a(1) = 0 \), and \( \eta(k) = 1 \). We shall now show that it will yield a solution for any linearly separable problem.
**The Perceptron Criterion**

- Piece-wise constant function not suitable for numerical optimization.
- Approximate by piece-wise linear function \( \rightarrow \) perceptron cost function

![Perceptron criterion diagram](image)
“Train” McCulloch-Pitts model (that is: estimate \((c, v)\)) by applying gradient descent to the function

\[
C_p(c, v) := \sum_{i=1}^{n} \mathbb{I}\{\text{sgn}(\langle v, \tilde{x}_i \rangle - c) \neq \tilde{y}_i\} \left| \left| \begin{pmatrix} c \\ v \end{pmatrix}, \begin{pmatrix} 1 \\ \tilde{x}_i \end{pmatrix} \right| \right|
\]

called the **Perceptron cost function**.
Part I (Orbanz)

- Neural networks (basic definitions and training)
- Graphical models (ditto)
- Sampling algorithms
- Variational inference
- Optimization for GMs and NNs

Part II (Cunningham)

- NN software
- Convolutional NNs and computer vision
- Recurrent NNs
- Reinforcement learning
- Dimension reduction and autoencoders
TOOLS: LOGISTIC REGRESSION
**SIGMOIDS**

Sigmoid function

\[ \sigma(x) = \frac{1}{1 + e^{-x}} \]

**Note**

\[ 1 - \sigma(x) = \frac{1 + e^{-x} - 1}{1 + e^{-x}} = \frac{1}{e^x + 1} = \sigma(-x) \]

**Derivative**

\[ \frac{d\sigma}{dx}(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = \sigma(x)(1 - \sigma(x)) \]

Sigmoid (blue) and its derivative (red)
• In linear classification: Decision boundary is a discontinuity
• Boundary is represented either by indicator function $\mathbb{I}\{\bullet > c\}$ or sign function $\text{sign}(\bullet - c)$
• These representations are equivalent:
  Note $\text{sign}(\bullet - c) = 2 \cdot \mathbb{I}\{\bullet > c\} - 1$

The most important use of the sigmoid function in machine learning is as a smooth approximation to the indicator function.

Given a sigmoid $\sigma$ and a data point $x$, we decide which side of the approximated boundary we are own by thresholding

$$\sigma(x) \geq \frac{1}{2}$$
We can add a scale parameter by defining

\[ \sigma_\theta(x) := \sigma(\theta x) = \frac{1}{1 - e^{-\theta x}} \quad \text{for } \theta \in \mathbb{R} \]

**Influence of \( \theta \)**

- As \( \theta \) increases, \( \sigma_\theta \) approximates \( I \) more closely.
- For \( \theta \to \infty \), the sigmoid converges to \( I \) pointwise, that is: For every \( x \neq 0 \), we have
  \[ \sigma_\theta(x) \to I\{x > 0\} \quad \text{as } \theta \to +\infty. \]
- Note \( \sigma_\theta(0) = \frac{1}{2} \) always, regardless of \( \theta \).
Approximating a Linear Classifier

So far, we have considered $\mathbb{R}$, but linear classifiers usually live in $\mathbb{R}^d$.

The decision boundary of a linear classifier in $\mathbb{R}^2$ is a discontinuous ridge:

- This is a linear classifier of the form $\mathbb{I}\{\langle \mathbf{v}, \mathbf{x} \rangle - c \}$.
- Here: $\mathbf{v} = (1, 1)$ and $c = 0$.

We can “stretch” $\sigma$ into a ridge function on $\mathbb{R}^2$:

- This is the function $\mathbf{x} = (x_1, x_2) \mapsto \sigma(x_1)$.
- The ridge runs parallel to the $x_2$-axes.
- If we use $\sigma(x_2)$ instead, we rotate by 90 degrees (still axis-parallel).
STEERING A SIGMOID

Just as for a linear classifier, we use a normal vector $\mathbf{v} \in \mathbb{R}^d$.

- The function $\sigma(\langle \mathbf{v}, \mathbf{x} \rangle - c)$ is a sigmoid ridge, where the ridge is orthogonal to the normal vector $\mathbf{v}$, and $c$ is an offset that shifts the ridge “out of the origin”.
- The plot on the right shows the normal vector (here: $\mathbf{v} = (1, 1)$) in black.
- The parameters $\mathbf{v}$ and $c$ have the same meaning for $\mathbb{I}$ and $\sigma$, that is, $\sigma(\langle \mathbf{v}, \mathbf{x} \rangle - c)$ approximates $\mathbb{I}\{\langle \mathbf{v}, \mathbf{x} \rangle \geq c\}$. 
**Logistic Regression**

*Logistic regression* is a classification method that approximates decision boundaries by sigmoids.

**Setup**

- Two-class classification problem
- Observations \( \mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d \), class labels \( y_i \in \{0, 1\} \).

**The logistic regression model**

We model the conditional distribution of the class label given the data as

\[
P(y|\mathbf{x}) := \text{Bernoulli}(\sigma(\langle \mathbf{v}, \mathbf{x} \rangle - c))
\]

- Recall \( \sigma(\langle \mathbf{v}, \mathbf{x} \rangle - c) \) takes values in \([0, 1]\) for all \( \theta \), and value \( \frac{1}{2} \) on the class boundary.
- The logistic regression model interprets this value as the probability of being in class \( y \).
Since the model is defined by a parametric distribution, we can apply maximum likelihood.

**Notation**
Recall from Statistical Machine Learning: We collect the parameters in a vector $\mathbf{w}$ by writing

$$
\mathbf{w} := \begin{pmatrix} \mathbf{v} \\ -c \end{pmatrix} \quad \text{and} \quad \tilde{\mathbf{x}} := \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix}
$$

so that $\langle \mathbf{w}, \tilde{\mathbf{x}} \rangle = \langle \mathbf{v}, \mathbf{x} \rangle - c$.

**Likelihood function of the logistic regression model**

$$
\prod_{i=1}^{n} \sigma(\langle \mathbf{w}, \tilde{\mathbf{x}}_i \rangle)^{y_i} (1 - \sigma(\langle \mathbf{w}, \tilde{\mathbf{x}}_i \rangle))^{1-y_i}
$$

**Negative log-likelihood**

$$
L(\mathbf{w}) := -\sum_{i=1}^{n} \left( y_i \log \sigma(\langle \mathbf{w}, \tilde{\mathbf{x}}_i \rangle) + (1 - y_i) \log(1 - \sigma(\langle \mathbf{w}, \tilde{\mathbf{x}}_i \rangle)) \right)
$$
Maximum likelihood

\[ \nabla L(w) = \sum_{i=1}^{n} (\sigma(w^t \tilde{x}_i) - y_i) \tilde{x}_i \]

Note

- Each training data point \( x_i \) contributes to the sum proportionally to the approximation error \( \sigma(w^t \tilde{x}_i) - y_i \) incurred at \( x_i \) by approximating the linear classifier by a sigmoid.

Maximum likelihood

- The ML estimator \( \hat{w} \) for \( w \) is the solution of
  \[ \nabla L(w) = 0 . \]
  - For logistic regression, this equation has no solution in closed form.
  - To find \( \hat{w} \), we use numerical optimization.
  - The function \( L \) is convex (= \( \cup \)-shaped).
Recall from Statistical Machine Learning

- If $f$ is differentiable, we can apply gradient descent:

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - \nabla f(\mathbf{x}^{(k)})$$

where $\mathbf{x}^{(k)}$ is the candidate solution in step $k$ of the algorithm.

- If the Hessian matrix $H_f$ of partial second derivatives exists and is invertible, we can apply Newton’s method, which converges faster:

$$\mathbf{x}^{(k+1)} := \mathbf{x}^{(k)} - H_f^{-1}(\mathbf{x}^{(k)}) \cdot \nabla f(\mathbf{x}^{(k)})$$

- Recall that the Hessian matrix of a (twice continuously differentiable) function $f : \mathbb{R}^d \to \mathbb{R}$ is

$$H_f(\mathbf{x}) := \left( \frac{\partial^2 f}{\partial x_i \partial x_j} \right)_{i,j \leq n}$$

Since $f$ is twice differentiable, each $\partial^2 f / \partial x_i \partial x_j$ exists; since it is twice continuously differentiable, $\partial^2 f / \partial x_i \partial x_j = \partial^2 f / \partial x_j \partial x_i$, so $H_f$ is symmetric.

- The inverse of $H_f(\mathbf{x})$ exists if and only if the matrix is positive definite (semidefinite does not suffice), which in turn is true if and only if $f$ is strictly convex.
Applying Newton

\[ w^{(k+1)} := w^{(k)} - H^{-1}_L(w^{(k)}) \cdot \nabla L(w^{(k)}) \]

Matrix notation

\[ \tilde{X} := \begin{pmatrix} 1 & (\tilde{x}_1)_1 & \ldots & (\tilde{x}_1)_j & \ldots & (\tilde{x}_1)_d \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & (\tilde{x}_i)_1 & \ldots & (\tilde{x}_i)_j & \ldots & (\tilde{x}_i)_d \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & (\tilde{x}_n)_1 & \ldots & (\tilde{x}_n)_j & \ldots & (\tilde{x}_n)_d \end{pmatrix} \]

\[ D_{\sigma} = \begin{pmatrix} \sigma(w^t\tilde{x}_1)(1 - \sigma(w^t\tilde{x}_1)) & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \sigma(w^t\tilde{x}_n)(1 - \sigma(w^t\tilde{x}_n)) \end{pmatrix} \]

\( \tilde{X} \) is the data matrix (or design matrix) you know from linear regression. \( \tilde{X} \) has size \( n \times (d + 1) \) and \( D_{\sigma} \) is \( n \times n \).

Newton step

\[ w^{(k+1)} = (\tilde{X}^tD_{\sigma}\tilde{X})^{-1}\tilde{X}^tD_{\sigma} \left( \tilde{X}w^{(k)} - D_{\sigma} \begin{pmatrix} \sigma(\langle w^{(k)}, \tilde{x}_1 \rangle) \\ \vdots \\ \sigma(\langle w^{(k)}, \tilde{x}_n \rangle) \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \right) \]
\[
\mathbf{w}^{(k+1)} = (\tilde{\mathbf{X}}^tD_\sigma \tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}^tD_\sigma \left( \tilde{\mathbf{X}}\mathbf{w}^{(k)} - D_\sigma \begin{pmatrix} \sigma(\langle \mathbf{w}^{(k)}, \tilde{x}_1 \rangle) \\ \vdots \\ \sigma(\langle \mathbf{w}^{(k)}, \tilde{x}_n \rangle) \end{pmatrix} - \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \right) = (\tilde{\mathbf{X}}^tD_\sigma \tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}^tD_\sigma \mathbf{u}^{(k)}
\]

\[
=: \mathbf{u}^{(k)}
\]

Compare this to the least squares solution of a linear regression problem:

\[
\hat{\beta} = (\tilde{\mathbf{X}}^t\tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}^t\mathbf{y}
\]
\[
\mathbf{w}^{(k+1)} = (\tilde{\mathbf{X}}^tD_\sigma \tilde{\mathbf{X}})^{-1}\tilde{\mathbf{X}}^tD_\sigma \mathbf{u}^{(k)}
\]

Differences:

- The vector \(\mathbf{y}\) of regression responses is substituted by the vector \(\mathbf{u}^{(k)}\) above.
- The matrix \(\tilde{\mathbf{X}}^t\tilde{\mathbf{X}}\) is substituted by the matrix \(\tilde{\mathbf{X}}^tD_\sigma \tilde{\mathbf{X}}\).
- Note that matrices of product form \(\tilde{\mathbf{X}}^t\tilde{\mathbf{X}}\) are positive semidefinite; since \(D_\sigma\) is diagonal with non-negative entries, so is \(\tilde{\mathbf{X}}^tD_\sigma \tilde{\mathbf{X}}\).

**Iteratively Reweighted Least Squares**

- At each step, the algorithm solves a least-squares problem “rewighted” by the matrix \(D_\sigma\).
- Since this happens at each step of an iterative algorithm, Newton’s method applied to the logistic regression log-likelihood is also known as **Iteratively Reweighted Least Squares**.
Newton: Cost

- The size of the Hessian is \((d + 1) \times (d + 1)\).
- In high-dimensional problems, inverting \(H_L\) can become problematic.

Other methods

Maximum likelihood only requires that we minimize the negative log-likelihood; we can choose any numerical method, not just Newton. Alternatives include:

- Pseudo-Newton methods (only invert \(H_L\) once, for \(w^{(1)}\), but do not guarantee quadratic convergence).
- Gradient methods.
- Approximate gradient methods, like stochastic gradient.
Recall from Statistical Machine Learning

- If we increase the length of $v$ without changing its direction, the sign of $\langle v, x \rangle$ does not change, but the value changes.
- That means: If $v$ is the normal vector of a classifier, and we scale $v$ by some $\theta > 0$, the decision boundary does not move, but $\langle \theta v, x \rangle = \theta \langle v, x \rangle$.

Effect inside a sigmoid

$$\sigma(\langle \theta v, x \rangle) = \sigma(\theta \langle v, x \rangle) = \sigma_{\theta}(\langle v, x \rangle)$$

As the length of $v$ increases, $\sigma(\langle v, x \rangle)$ becomes more similar to $\mathbb{I}\{\langle v, x \rangle > 0\}$. 
Effect on ML for logistic regression

Recall each training data point $\mathbf{x}_i$ contributes an error term $\sigma(\mathbf{w}^T \tilde{x}_i) - y_i$ to the log-likelihood.

By increasing the lengths of $\mathbf{w}$, we can make $\sigma(\mathbf{w}^T \tilde{x}_i) - y_i$ arbitrarily small without moving the decision boundary.
Consequence for linearly separable data

- Once the decision boundary is correctly located between the two classes, the maximization algorithm can increase the log-likelihood arbitrarily by increasing the length of $w$.
- That does not move the decision boundary, but the logistic function looks more and more like the indicator $I$.
- That may fit the training data more tightly, but can lead to bad generalization (e.g. for similar reasons as for the perceptron, where the decision boundary may end up very close to a training data point).

That is a form of overfitting.

Data that is not linearly separable

- If the data is not separable, sufficiently many points on the “wrong” side of the decision boundary prevent overfitting (since making $w$ larger increases error contributions of these points).
- For large data sets, overfitting can still occur if the fraction of such points is small.

Solutions

- Overfitting can be addressed by including an additive penalty of the form $L(w) + \lambda \|w\|$. 
Logistic Regression for Multiple Classes

Bernoulli and multinomial distributions

- The multinomial distribution of $N$ draws from $K$ categories with parameter vector $(\theta_1, \ldots, \theta_K)$ (where $\sum_{k \leq K} \theta_k = 1$) has probability mass function

$$P(m_1, \ldots, m_K|\theta_1, \ldots, \theta_K) = \frac{N!}{m_1! \cdots m_K!} \prod_{k=1}^{K} \theta_k^{m_k} \text{ where } m_k = \# \text{ draws in category } k$$

- Note that Bernoulli($p$) = Multinomial($p, 1-p; N=1$).

Logistic regression

- Recall two-class logistic regression is defined by $P(Y|x) = \text{Bernoulli}(\sigma(w^t x))$.

- Idea: To generalize logistic regression to $K$ classes, choose a separate weight vector $w_k$ for each class $k$, and define $P(Y|x)$ by

  $$\text{Multinomial}(\tilde{\sigma}(w_1^t x), \ldots, \tilde{\sigma}(w_K^t x))$$

  where $\tilde{\sigma}(w_k^t x) = \frac{\sigma(w_k^t x)}{\sum_k \sigma(w_k^t x)}$.  

Logistic regression for $K$ classes

The label $y$ now takes values in $\{1, \ldots, K\}$.

\[
P(y|x) = \prod_{k=1}^{K} \tilde{\sigma}(w_k^T \tilde{x}) \mathbb{I}\{y=k\}
\]

The negative log-likelihood becomes

\[
L(w_1, \ldots, w_K) = - \sum_{i \leq n, k \leq K} \mathbb{I}\{y = k\} \log \tilde{\sigma}(w_k^T \tilde{x}_i)
\]

This can again be optimized numerically.

Comparison to two-class case

- Recall that $1 - \sigma(x) = \sigma(-x)$.
- That means

  \[
  \text{Bernoulli}(\sigma(\langle v, x \rangle - c)) \equiv \text{Multinomial}(\sigma(w^T x), \sigma((-1)w^T x))
  \]

  - That is: Two-class logistic regression as above is equivalent to multiclass logistic regression with $K = 2$ provided we choose $w_2 = -w_1$. 

Graphical Models
A graphical model represents the dependence structure within a set of random variables as a graph.

**Overview**

Roughly speaking:

- Each random variable is represented by a vertex.
- If $Y$ depends on $X$, we draw an edge $X \rightarrow Y$.
- For example:

```
X  Z  Y
  ^   |
   X→  Y
```

This says: “$X$ depends on $Z$, and $Y$ depends on $Z$”.

- We have to be careful: The above does not imply that $X$ and $Y$ are independent. We have to make more precise what *depends on* means.
We will use the notation:

\[ L(X) = \text{distribution of the random variable } X \]
\[ L(X|Y) = \text{conditional distribution of } X \text{ given } Y \]

(\( L \) means “law”.)

**Reason**

- If \( X \) is discrete, \( L(X) \) is usually given by a mass function \( P(x) \).
- If it is continuous, \( L(X) \) is usually given by a density \( p(x) \).
- With the notation above, we do not have to distinguish between discrete and continuous variables.
Dependence and Independence

Dependence between random variables $X_1, \ldots, X_n$ is a property of their joint distribution $\mathcal{L}(X_1, \ldots, X_n)$.

**Recall**

Two random variables are *stochastically independent*, or *independent* for short, if their joint distribution factorizes:

$$\mathcal{L}(X, Y) = \mathcal{L}(X) \mathcal{L}(Y)$$

For densities/mass functions:

$$P(x, y) = P(x)P(y) \quad \text{or} \quad p(x, y) = p(x)p(y)$$

*Dependent* means *not independent*.

**Intuitively**

$X$ and $Y$ are dependent if knowing the outcome of $X$ provides any information about the outcome of $Y$.

More precisely:

- If someone draws $(X, Y)$ simultaneously, and only discloses $X = x$ to you, does that change your mind about the distribution of $Y$? (If so: Dependence.)
- Once $X$ is given, the distribution of $Y$ is the conditional $\mathcal{L}(Y|X = x)$.
- If that is still $\mathcal{L}(Y)$, as before $X$ was drawn, the two are independent. If $\mathcal{L}(Y|X = x) \neq \mathcal{L}(Y)$, they are dependent.
CONDITIONAL INDEPENDENCE

Definition

Given random variables $X$, $Y$, $Z$, we say that $X$ is **conditionally independent of $Y$ given $Z$** if

\[ \mathcal{L}(X, Y | Z = z) = \mathcal{L}(X | Z = z) \mathcal{L}(Y | Z = z) . \]

That is equivalent to

\[ \mathcal{L}(X | Y = y, Z = z) = \mathcal{L}(X | Z = z) . \]

Notation

\[ X \perp \! \! \! \perp Z \ Y \]

Intuitively

$X$ and $Y$ are dependent given $Z = z$ if, although $Z$ is known, knowing the outcome of $X$ provides additional information about the outcome of $Y$. 
Factorizing a joint distribution

The joint probability of random variables $X_1, \ldots, X_n$ can always be factorized as

$$
\mathcal{L}(X_1, \ldots, X_n) = \mathcal{L}(X_n|X_1, \ldots, X_{n-1}) \mathcal{L}(X_{n-1}|X_1, \ldots, X_{n-2}) \cdots \mathcal{L}(X_1). 
$$

Note that we can re-arrange the variables in any order.

If there are conditional independencies, we can remove some variables from the conditionals:

$$
\mathcal{L}(X_1, \ldots, X_n) = \mathcal{L}(X_n|\mathcal{X}_n) \mathcal{L}(X_{n-1}|\mathcal{X}_{n-1}) \cdots \mathcal{L}(X_1),
$$

where $\mathcal{X}_i$ is the subset of $X_1, \ldots, X_n$ on which $X_i$ depends.

**Definition**

Let $X_1, \ldots, X_n$ be random variables. A *(directed) graphical model* represents a factorization of joint distribution $\mathcal{L}(X_1, \ldots, X_n)$ as follows:

- Factorize $\mathcal{L}(X_1, \ldots, X_n)$.
- Add one vertex for each variable $X_i$.
- For each variable $X_i$, add an edge from each variable $X_j \in \mathcal{X}_i$ to $X_i$.

That is: An edge $X_j \rightarrow X_i$ is added if $\mathcal{L}(X_1, \ldots, X_n)$ contains the factor $\mathcal{L}(X_i|X_j)$. 

---

**Advanced Machine Learning 49 / 212**
Lack of uniqueness

The factorization is usually not unique, since e.g.

\[ \mathcal{L}(X, Y) = \mathcal{L}(X|Y)\mathcal{L}(Y) = \mathcal{L}(Y|X)\mathcal{L}(X). \]

That means the direction of edges is not generally determined.

Remark

- If we use a graphical model to \textit{define} a model or visualize a model, we decide on the direction of the edges.
- Estimating the direction of edges from data is a very difficult (and very important) problem. This is one of the main subjects of a research field called \textit{causal inference} or \textit{causality}. 
A simple example

$$X \perp_{Z} Y$$

An example with layers

All variables in the $$(k + 1)$$st layer are conditionally independent given the variables in the $$k$$th layer.
**Words of Caution I**

**Important**

- $X$ and $Y$ are *not* independent, independence holds only conditionally on $Z$.
- In other words: If we do not observe $Z$, $X$ and $Y$ are dependent, and we have to change the graph:

$$X \perp_{Z} Y$$
Words of Caution II

Conditioning on $Z$ makes $X$ and $Y$ dependent.

Example

- Suppose we start with two independent normal variables $X$ and $Y$.
- $Z = X + Y$.

If we know $Z$, and someone reveals the value of $Y$ to us, we know everything about $X$.

This effect is known as explaining away. We will revisit it later.
Hidden Markov models
Sigmoid belief networks
A graphical model in which each node is a binary random variable and each conditional probability is a logistic regression model is called a sigmoid belief network.

Terminology: “Belief network” or “Bayes net” are alternative names for graphical models.

Deep belief networks
A deep belief network is a layered directed graphical model that looks like this.

Two tasks for deep belief nets are:

- **Sampling**: Draw $X_{1:d}$ from $\mathcal{L}(X_{1:d} | \Theta_{1:d})$.
  
  Note: Variables in each layer conditionally independent given previous layer.

- **Inference**: Estimate $\mathcal{L}(\Theta_{1:d} | X_{1:d} = x_{1:d})$ when data $x_{1:d}$ is observed.

  Problem: Conditioning a layer on the following one makes variables dependent.

(More details later.)
MARKOV RANDOM FIELDS
A graphical model is undirected when its dependency graph is undirected; equivalently, if each edge in the graph is either absent, or present in both directions.

An undirected graphical model is more commonly known as a Markov random field.

Markov random fields are special cases of (directed) graphical models, but have distinct properties. We treat them separately.

We will consider the undirected case first.
We start with an undirected graph:

A random variable $\Theta_i$ is associated with each vertex. Two random variables interact if they are neighbors in the graph.
We define a **neighborhood graph**, which is a weighted, undirected graph:

\[ \mathcal{N} = (V_{\mathcal{N}}, W_{\mathcal{N}}) \]

The vertices \( v_i \in V_{\mathcal{N}} \) are often referred to as **sites**.

- The edge weights are scalars \( w_{ij} \in \mathbb{R} \). Since the graph is undirected, the weights are symmetric (\( w_{ij} = w_{ji} \)).
- An edge weight \( w_{ij} = 0 \) means "no edge between \( v_i \) and \( v_j \)."

**Neighborhoods**

The set of all neighbors of \( v_j \) in the graph,

\[ \partial(i) := \{ j \mid w_{ij} \neq 0 \} \]

is called the **neighborhood** of \( v_j \).
Given a neighborhood graph $\mathcal{N}$, associate with each site $v_i \in V_\mathcal{N}$ a RV $\Theta_i$.

**The Markov property**

We say that the joint distribution $P$ of $(\Theta_1, \ldots, \Theta_n)$ satisfies the Markov property with respect to $\mathcal{N}$ if

$$\mathcal{L}(\Theta_i | \Theta_j, j \neq i) = \mathcal{L}(\Theta_i | \Theta_j, j \in \partial(i)) .$$

The set $\{\Theta_j, j \in \partial(i)\}$ of random variables indexed by neighbors of $v_i$ is called the Markov blanket of $\Theta_i$.

**In words**

The Markov property says that each $\Theta_i$ is conditionally independent of the remaining variables given its Markov blanket.

**Definition**

A distribution $\mathcal{L}(\Theta_1, \ldots, \Theta_n)$ which satisfies the Markov property for a given graph $\mathcal{N}$ is called a Markov random field.
Probabilities and energies
A (strictly positive) density $p(x)$ can always be written in the form

$$p(x) = \frac{1}{Z} \exp(-H(x)) \quad \text{where} \quad H : \mathbf{X} \rightarrow \mathbb{R}_+$$

and $Z$ is a normalization constant.

The function $H$ is called an energy function, or cost function, or a potential.

MRF energy
In particular, we can write a MRF density for RVs $\Theta_{1:n}$ as

$$p(\theta_1, \ldots, \theta_n) = \frac{1}{Z} \exp(-H(\theta_1, \ldots, \theta_n))$$
Graphical models factorize over the graph. How does that work for MRFs?

A **clique** in a graph is a fully connected subgraph. In undirected graphs:

The cliques in this graph are:

i) The triangles (1, 2, 3), (1, 3, 4).

ii) Each pair of vertices connected by an edge (e.g. (2, 6)).
Theorem

Let $\mathcal{N}$ be a neighborhood graph with vertex set $V_{\mathcal{N}}$. Suppose the random variables \( \{\Theta_i, i \in V_{\mathcal{N}}\} \) take values in $\mathcal{T}$, and their joint distribution has probability mass function $P$, so there is an energy function $H$ such that

$$P(\theta_1, \ldots, \theta_n) = \frac{e^{-H(\theta_1, \ldots, \theta_n)}}{\sum_{i \leq n} \sum_{\theta_i \in \mathcal{T}} e^{-H(\theta_1, \ldots, \theta_n)}}.$$

The joint distribution has the Markov property if and only if

$$H(\theta_1, \theta_2, \ldots) = \sum_{C \in C} H_C(\theta_i, i \in C),$$

where $C$ is the set of cliques in $\mathcal{N}$, and each $H_C$ is a non-negative function with $|C|$ arguments. Hence,

$$P(\theta_1, \ldots, \theta_n) = \prod_{C \in C} \frac{e^{-H_C(\theta_i,i \in C)}}{\sum_{C \in C} \sum_{\theta_i \in \mathcal{T}} e^{-H_C(\theta_i,i \in C)}}$$

Markov random fields factorize over cliques.
USE OF MRFs

In general

- Modeling systems of dependent RVs is one of the hardest problems in probability.
- MRFs model dependence, but break it down to a limited number of interactions to make the model tractable.

MRFs on grids

- By far the most widely used neighborhood graphs are 2-dimensional grids.
- MRFs on grids are used in spatial statistics to model spatial interactions between RVs.
- Hammersley-Clifford for grids: The only cliques are the edges!

2-dimensional grid graph with 4-neighborhoods

MRFs on grids factorize over edges.
**The Potts Model**

**Definition**
Suppose $\mathcal{N} = (V_N, W_N)$ a neighborhood graph with $n$ vertices and $\beta > 0$ a constant. Then

$$p(\theta_{1:n}) := \frac{1}{Z(\beta, W_N)} \exp \left( \beta \sum_{i,j} w_{ij} \mathbb{I}\{\theta_i = \theta_j\} \right)$$

defines a joint distribution of $n$ random variables $\Theta_1, \ldots, \Theta_n$. This distribution is called the **Potts model**.

**Interpretation**

- If $w_{ij} > 0$: The overall probability *increases* if $\Theta_i = \Theta_j$.
- If $w_{ij} < 0$: The overall probability *decreases* if $\Theta_i = \Theta_j$.
- If $w_{ij} = 0$: No interaction between $\Theta_i$ and $\Theta_j$.

Positive weights encourage *smoothness*. 
Ising model

The simplest choice is \( w_{ij} = 1 \) if \((i, j)\) is an edge.

\[
p(\theta_1: n) = \frac{1}{Z(\beta)} \exp \left( \sum_{(i,j) \text{ is an edge}} \beta \mathbb{1}\{\theta_i = \theta_j\} \right)
\]

If \( \mathcal{N} \) is a \( d \)-dim. grid, this model is called the **Ising model**.

**Example**

Samples from an Ising model on a \( 56 \times 56 \) grid graph.

Increasing \( \beta \) \( \rightarrow \)
We consider a spatial problem with observations $X_i$. Each $i$ is a location on a grid.

**Spatial model**

Suppose we model each $X_i$ by a distribution $\mathcal{L}(X|\Theta_i)$, i.e. each location $i$ has its own parameter variable $\Theta_i$. This model is Bayesian (the parameter is a random variable). We use an MRF as prior distribution.

We can think of $\mathcal{L}(X|\Theta_i)$ as an emission probability, similar to an HMM.

**Spatial smoothing**

- We can define the joint distribution $(\Theta_1, \ldots, \Theta_n)$ as a MRF on the grid graph.
- For positive weights, the MRF will encourage the model to explain neighbors $X_i$ and $X_j$ by the same parameter value. $\rightarrow$ Spatial smoothing.
Mixture model

- A BMM can be used for image segmentation.
- The BMM prior on the component parameters is a natural conjugate prior $q(\theta)$.
- In the spatial setting, we index the parameter of each $X_i$ separately as $\theta_i$. For $K$ mixture components, $\theta_{1:n}$ contains only $K$ different values.
- The joint BMM prior on $\theta_{1:n}$ is

$$q_{\text{BMM}}(\theta_{1:n}) = \prod_{i=1}^{n} q(\theta_i).$$

Smoothing term

We multiply the BMM prior $q_{\text{BMM}}(\theta)$ with an MRF prior

$$q_{\text{MRF}}(\theta_{1:n}) = \frac{1}{Z(\beta)} \exp \left( \beta \sum_{w_{ij} \neq 0} \mathbb{I}\{\theta_i = \theta_j\} \right)$$

This encourages spatial smoothness of the segmentation.
MRFs pose two main computational problems.

Problem 1: Sampling

Generate samples from the joint distribution of $(\Theta_1, \ldots, \Theta_n)$.

Problem 2: Inference

If the MRF is used as a prior, we have to compute or approximate the posterior distribution.

Solution

- MRF distributions on grids are not analytically tractable. The only known exception is the Ising model in 1 dimension.
- Both sampling and inference are based on Markov chain sampling algorithms.
Sampling Algorithms
Sampling Algorithms

In general

- A **sampling algorithm** is an algorithm that outputs samples $X_1, X_2, \ldots$ from a given distribution $P$ or density $p$.
- Sampling algorithms can for example be used to approximate expectations:

$$\mathbb{E}_p[f(X)] \approx \frac{1}{n} \sum_{i=1}^{n} f(X_i)$$

Inference in Bayesian models

Suppose we work with a Bayesian model whose posterior $\hat{Q}_n := \mathcal{L}(\Theta | X_{1:n})$ cannot be computed analytically.

- We will see that it can still be possible to *sample* from $\hat{Q}_n$.
- Doing so, we obtain samples $\Theta_1, \Theta_2, \ldots$ distributed according to $\hat{Q}_n$.
- This reduces posterior estimation to a density estimation problem (i.e. estimate $\hat{Q}_n$ from $\Theta_1, \Theta_2, \ldots$).
Posterior expectations

If we are only interested in some statistic of the posterior of the form \( \mathbb{E}_{\hat{Q}_n}[f(\Theta)] \) (e.g. the posterior mean), we can again approximate by

\[
\mathbb{E}_{\hat{Q}_n}[f(\Theta)] \approx \frac{1}{m} \sum_{i=1}^{m} f(\Theta_i) .
\]

Example: Predictive distribution

The **posterior predictive distribution** is our best guess of what the next data point \( x_{n+1} \) looks like, given the posterior under previous observations. In terms of densities:

\[
p(x_{n+1}|x_{1:n}) := \int_T p(x_{n+1}|\theta) \hat{Q}_n(d\theta|x_{1:n} = x_{1:n}) .
\]

This is one of the key quantities of interest in Bayesian statistics.

Computation from samples

The predictive is a posterior expectation, and can be approximated as a sample average:

\[
p(x_{n+1}|x_{1:n}) = \mathbb{E}_{\hat{Q}_n}[p(x_{n+1}|\Theta)] \approx \frac{1}{m} \sum_{i=1}^{m} p(x_{n+1}|\Theta_i)
\]
Say we are interested in a probability density $p$ on the interval $[a, b]$.

### Key observation

Suppose we can define a uniform distribution $U_A$ on the blue area $A$ under the curve. If we sample

$$(X_1, Y_1), (X_2, Y_2), \ldots \sim_{iid} U_A$$

and discard the vertical coordinates $Y_i$, the $X_i$ are distributed according to $p$,

$$X_1, X_2, \ldots \sim_{iid} p.$$ 

**Problem:** Defining a uniform distribution is easy on a rectangular area, but difficult on an arbitrarily shaped one.
**Solution: Rejection sampling**

We can enclose $p$ in box, and sample uniformly from the box $B$.

- We can sample $(X_i, Y_i)$ uniformly on $B$ by sampling
  
  $$X_i \sim \text{Uniform}[a, b] \quad \text{and} \quad Y_i \sim \text{Uniform}[0, c].$$

- If $(X_i, Y_i) \in A$, keep the sample.
  That is: If $Y_i \leq p(X_i)$.

- Otherwise: Discard it ("reject" it).

Result: The remaining (non-rejected) samples are uniformly distributed on $A$. 
This strategy still works if we scale the vertically by some constant $k > 0$.

We simply draw $Y_i \sim \text{Uniform}[0, kc]$ instead of $Y_i \sim \text{Uniform}[0, c]$.

**Consequence**

For sampling, it is sufficient if $p$ is known only up to normalization (only the shape of $p$ is known).
Sampling methods usually assume that we can evaluate the target distribution \( p \) up to a constant. That is:

\[
p(x) = \frac{1}{\tilde{Z}} \tilde{p}(x),
\]

and we can compute \( \tilde{p}(x) \) for any given \( x \), but we do not know \( \tilde{Z} \).

We have to pause for a moment and convince ourselves that there are useful examples where this assumption holds.

**Example 1: Simple posterior**

For an arbitrary posterior computed with Bayes’ theorem, we could write

\[
\Pi(\theta|x_{1:n}) = \frac{\prod_{i=1}^{n} p(x_i|\theta)q(\theta)}{\tilde{Z}} \quad \text{with} \quad \tilde{Z} = \int_{\Theta} \prod_{i=1}^{n} p(x_i|\theta)q(\theta) d\theta.
\]

Provided that we can compute the numerator, we can sample without computing the normalization integral \( \tilde{Z} \).
Example 2: Bayesian Mixture Model

Recall that the posterior of the BMM is (up to normalization):

\[
\hat{q}_n(c_{1:K}, \theta_{1:K} | x_{1:n}) \propto \prod_{i=1}^{n} \left( \sum_{k=1}^{K} c_k p(x_i | \theta_k) \right) \left( \prod_{k=1}^{K} q(\theta_k | \lambda, y) \right) q_{\text{Dirichlet}}(c_{1:K})
\]

We already know that we can discard the normalization constant, but can we evaluate the non-normalized posterior \( \tilde{q}_n \)?

- The problem with computing \( \tilde{q}_n \) (as a function of unknowns) is that the term \( \prod_{i=1}^{n} \left( \sum_{k=1}^{K} \ldots \right) \) blows up into \( K^n \) individual terms.

- If we evaluate \( \tilde{q}_n \) for specific values of \( c, x \) and \( \theta \), \( \sum_{k=1}^{K} c_k p(x_i | \theta_k) \) collapses to a single number for each \( x_i \), and we just have to multiply those \( n \) numbers.

So: Computing \( \tilde{q}_n \) as a formula in terms of unknowns is difficult; evaluating it for specific values of the arguments is easy.
Example 3: Markov random field

In a MRF, the normalization function is the real problem.

For example, recall the Ising model:

\[ p(\theta_1:n) = \frac{1}{Z(\beta)} \exp\left( \sum_{(i,j) \text{ is an edge}} \beta \mathbb{I}\{\theta_i = \theta_j\} \right) \]

The normalization function is

\[ Z(\beta) = \sum_{\theta_1:n \in \{0,1\}^n} \exp\left( \sum_{(i,j) \text{ is an edge}} \beta \mathbb{I}\{\theta_i = \theta_j\} \right) \]

and hence a sum over \(2^n\) terms. The general Potts model is even more difficult.

On the other hand, evaluating

\[ \tilde{p}(\theta_1:n) = \exp\left( \sum_{(i,j) \text{ is an edge}} \beta \mathbb{I}\{\theta_i = \theta_j\} \right) \]

for a given configuration \(\theta_1:n\) is straightforward.
If we are not on the interval, sampling uniformly from an enclosing box is not possible (since there is no uniform distribution on all of $\mathbb{R}$ or $\mathbb{R}^d$).

**Solution: Proposal density**

Instead of a box, we use another distribution $r$ to enclose $p$:

To generate $B$ under $r$, we apply similar logic as before backwards:
- Sample $X_i \sim r$.
- Sample $Y_i|X_i \sim \text{Uniform}[0, r(X_i)]$.

$r$ is always a simple distribution which we can sample and evaluate.
Choose a simple distribution \( r \) from which we know how to sample.

Scale \( \tilde{p} \) such that \( \tilde{p}(x) < r(x) \) everywhere.

Sampling: For \( i = 1, 2, \ldots, \):

1. Sample \( X_i \sim r \).
2. Sample \( Y_i | X_i \sim \text{Uniform}[0, r(X_i)] \).
3. If \( Y_i < \tilde{p}(X_i) \), keep \( X_i \).
4. Else, discard \( X_i \) and start again at (1).

The surviving samples \( X_1, X_2, \ldots \) are distributed according to \( p \).
The rejection step can be interpreted in terms of probabilities and densities.

**Factorization**

We factorize the target distribution or density $p$ as

$$p(x) = r(x) \cdot A(x)$$

where $X' \sim r$ and $Z|X' \sim \text{Bernoulli}(A(X'))$.

**Sampling from the factorization**

$$X = X' \cdot Z$$

**Sampling Bernoulli variables with uniform variables**

$$Z|X' \sim \text{Bernoulli}(A(X')) \iff Z = \mathbb{I}\{U < A(X')\} \quad \text{where} \quad U \sim \text{Uniform}[0, 1] .$$
If we draw proposal samples $X_i$ i.i.d. from $r$, the resulting sequence of accepted samples produced by rejection sampling is again i.i.d. with distribution $p$. Hence:

Rejection samplers produce i.i.d. sequences of samples.

**Important consequence**

If samples $X_1, X_2, \ldots$ are drawn by a rejection sampler, the sample average

$$\frac{1}{m} \sum_{i=1}^{m} f(X_i)$$

(for some function $f$) is an unbiased estimate of the expectation $\mathbb{E}_p[f(X)]$. 

The fraction of accepted samples is the ratio $\frac{|A|}{|B|}$ of the areas under the curves $\tilde{p}$ and $r$.

If $r$ is not a reasonably close approximation of $p$, we will end up rejecting a lot of proposal samples.
An important bit of imprecise intuition

Example figures for sampling methods tend to look like this.

A high-dimensional distribution of correlated RVs will look rather more like this.

Sampling is usually used in multiple dimensions. Reason, roughly speaking:

- Intractable posterior distributions arise when there are several interacting random variables. The interactions make the joint distribution complicated.
- In one-dimensional problems (1 RV), we can usually compute the posterior analytically.
- Independent multi-dimensional distributions factorize and reduce to one-dimensional case.

**Warning**: Avoid sampling if you can solve analytically.
Why is not every sampler a rejection sampler?

We can easily end up in situations where we accept only one in $10^6$ (or $10^{10}$, or $10^{20}$, . . .) proposal samples. Especially in higher dimensions, we have to expect this to be not the exception but the rule.
The rejection problem can be fixed easily if we are only interested in approximating an expectation $\mathbb{E}_p[f(X)]$.

**Simple case: We can evaluate $p$**

Suppose $p$ is the target density and $q$ a proposal density. An expectation under $p$ can be rewritten as

$$\mathbb{E}_p[f(X)] = \int f(x)p(x)dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx = \mathbb{E}_q \left[ \frac{f(X)p(X)}{q(X)} \right]$$

**Importance sampling**

We can sample $X_1, X_2, \ldots$ from $q$ and approximate $\mathbb{E}_p[f(X)]$ as

$$\mathbb{E}_p[f(X)] \approx \frac{1}{m} \sum_{i=1}^{m} f(X_i) \frac{p(X_i)}{q(X_i)}$$

There is no rejection step; all samples are used.

This method is called **importance sampling**. The coefficients $\frac{p(X_i)}{q(X_i)}$ are called **importance weights**.
importance sampling

General case: We can only evaluate $\tilde{p}$

In the general case,

$$p = \frac{1}{Z_p} \tilde{p} \quad \text{and} \quad q = \frac{1}{Z_q} \tilde{q},$$

and $Z_p$ (and possibly $Z_q$) are unknown. We can write $\frac{Z_p}{Z_q}$ as

$$\frac{Z_p}{Z_q} = \frac{\int \tilde{p}(x)dx}{Z_q} = \frac{\int \tilde{p}(x)\frac{q(x)}{q(x)}dx}{Z_q} = \int \tilde{p}(x)\frac{q(x)}{Z_q \cdot q(x)}dx = \mathbb{E}_q \left[ \frac{\tilde{p}(X)}{\tilde{q}(X)} \right]$$

Approximating the constants

The fraction $\frac{Z_p}{Z_q}$ can be approximated using samples $x_{1:m}$ from $q$:

$$\frac{Z_p}{Z_q} = \mathbb{E}_q \left[ \frac{\tilde{p}(X)}{\tilde{q}(X)} \right] \approx \frac{1}{m} \sum_{i=1}^{m} \frac{\tilde{p}(X_i)}{\tilde{q}(X_i)}$$

Approximating $\mathbb{E}_p[f(X)]$

$$\mathbb{E}_p[f(X)] \approx \frac{1}{m} \sum_{i=1}^{m} f(X_i) \frac{p(X_i)}{q(X_i)} = \frac{1}{m} \sum_{i=1}^{m} f(X_i) \frac{Z_q \tilde{p}(X_i)}{Z_p \tilde{q}(X_i)} = \sum_{i=1}^{m} \frac{f(X_i) \frac{\tilde{p}(X_i)}{\tilde{q}(X_i)}}{\sum_{j=1}^{m} \frac{\tilde{p}(X_j)}{\tilde{q}(X_j)}}$$
**IMPORTANCE SAMPLING IN GENERAL**

**Conditions**

- Given are a target distribution \( p \) and a proposal distribution \( q \).
- \( p = \frac{1}{Z_p} \tilde{p} \) and \( q = \frac{1}{Z_q} \tilde{q} \).
- We can evaluate \( \tilde{p} \) and \( \tilde{q} \), and we can sample \( q \).
- The objective is to compute \( \mathbb{E}_p[f(X)] \) for a given function \( f \).

**Algorithm**

1. Sample \( X_1, \ldots, X_m \) from \( q \).
2. Approximate \( \mathbb{E}_p[f(X)] \) as

\[
\mathbb{E}_p[f(X)] \approx \frac{\sum_{i=1}^m f(X_i) \frac{\tilde{p}(X_i)}{\tilde{q}(X_i)}}{\sum_{j=1}^m \frac{\tilde{p}(X_j)}{\tilde{q}(X_j)}}
\]
Markov Chain Monte Carlo
Suppose we rejection-sample a distribution like this:

Once we have drawn a sample in the narrow region of interest, we would like to continue drawing samples within the same region. That is only possible if each sample depends on the location of the previous sample.

Proposals in rejection sampling are i.i.d. Hence, once we have found the region where $p$ concentrates, we forget about it for the next sample.
Recall: Markov chain

- A sufficiently nice Markov chain (MC) has an invariant distribution $P_{\text{inv}}$.
- Once the MC has converged to $P_{\text{inv}}$, each sample $X_i$ from the chain has marginal distribution $P_{\text{inv}}$.

Markov chain Monte Carlo

We want to sample from a distribution with density $p$. Suppose we can define a MC with invariant distribution $P_{\text{inv}} \equiv p$. If we sample $X_1, X_2, \ldots$ from the chain, then once it has converged, we obtain samples

$$X_i \sim p.$$  

This sampling technique is called **Markov chain Monte Carlo (MCMC)**.

**Note:** For a Markov chain, $X_{i+1}$ can depend on $X_i$, so at least in principle, it is possible for an MCMC sampler to "remember" the previous step and remain in a high-probability location.
The Markov chains we discussed so far had a finite state space \( X \). For MCMC, state space now has to be the domain of \( p \), so we often need to work with continuous state spaces.

**Continuous Markov chain**
A continuous Markov chain is defined by an initial distribution \( P_{\text{init}} \) and conditional probability \( t(y|x) \), the **transition probability** or **transition kernel**.

In the discrete case, \( t(y = i|x = j) \) is the entry \( p_{ij} \) of the transition matrix \( p \).

**Example: A Markov chain on \( \mathbb{R}^2 \)**
We can define a very simple Markov chain by sampling

\[
X_{i+1}|X_i = x_i \sim g(., |x_i, \sigma^2)
\]

where \( g(x|\mu, \sigma^2) \) is a spherical Gaussian with fixed variance. In other words, the transition distribution is

\[
t(x_{i+1}|x_i) := g(x_{i+1}|x_i, \sigma^2) .
\]
Invariant Distribution

Recall: Finite case

- The invariant distribution \( P_{\text{inv}} \) is a distribution on the finite state space \( X \) of the MC (i.e. a vector of length \(|X|\)).
- "Invariant" means that, if \( X_i \) is distributed according to \( P_{\text{inv}} \), and we execute a step \( X_{i+1} \sim t(.|x_i) \) of the chain, then \( X_{i+1} \) again has distribution \( P_{\text{inv}} \).
- In terms of the transition matrix \( \mathbf{p} \):

\[
\mathbf{p} \cdot P_{\text{inv}} = P_{\text{inv}}
\]

Continuous case

- \( X \) is now uncountable (e.g. \( X = \mathbb{R}^d \)).
- The transition matrix \( \mathbf{p} \) is substituted by the conditional probability \( t \).
- A distribution \( P_{\text{inv}} \) with density \( p_{\text{inv}} \) is invariant if

\[
\int_X t(y|x)p_{\text{inv}}(x)dx = p_{\text{inv}}(y)
\]

This is simply the continuous analogue of the equation \( \sum_i p_{ij}(P_{\text{inv}})_i = (P_{\text{inv}})_j \).
We run the Markov chain $n$ for steps. Each step moves from the current location $x_i$ to a new $x_{i+1}$.

We "forget" the order and regard the locations $x_1:n$ as a random set of points.

If $p$ (red contours) is both the invariant and initial distribution, each $X_i$ is distributed as $X_i \sim p$.

Problems we need to solve

1. We have to construct a MC with invariant distribution $p$.
2. We cannot actually start sampling with $X_1 \sim p$; if we knew how to sample from $p$, all of this would be pointless.
3. Each point $X_i$ is marginally distributed as $X_i \sim p$, but the points are not i.i.d.
Given is a continuous target distribution with density $p$.

**Metropolis-Hastings (MH) kernel**

1. We start by defining a conditional probability $q(y|x)$ on $X$.
   $q$ has nothing to do with $p$. We could e.g. choose $q(y|x) = g(y|x, \sigma^2)$, as in the previous example.

2. We define a **rejection kernel** $A$ as
   \[
   A(x_{n+1}|x_n) := \min\left\{ 1, \frac{q(x_i|x_{i+1})p(x_{i+1})}{q(x_{i+1}|x_i)p(x_i)} \right\}
   \]
   The normalization of $p$ cancels in the quotient, so knowing $\tilde{p}$ is again enough.

3. We define the transition probability of the chain as
   \[
   t(x_{i+1}|x_i) := q(x_{i+1}|x_i)A(x_{i+1}|x_i) + \delta_{x_i}(x_{i+1})c(x_i)
   \]
   where
   \[
   c(x_i) := \int q(y|x_i)(1-A(y|x_i))dy
   \]

**Sampling from the MH chain**

At each step $i + 1$, generate a proposal $X^* \sim q(\cdot|x_i)$ and $U_i \sim \text{Uniform}[0, 1]$.

- If $U_i \leq A(x^*|x_i)$, accept proposal: Set $x_{i+1} := x^*$.
- If $U_i > A(x^*|x_i)$, reject proposal: Set $x_{i+1} := x_i$. 
Recall: Fundamental theorem on Markov chains

Suppose we sample $X_1 \sim P_{\text{init}}$ and $X_{i+1} \sim t(. \mid x_i)$. This defines a distribution $P_i$ of $X_i$, which can change from step to step. If the MC is nice (recall: recurrent and aperiodic), then

$$P_i \rightarrow P_{\text{inv}} \quad \text{for} \quad i \rightarrow \infty.$$ 

Note: Making precise what aperiodic means in a continuous state space is a bit more technical than in the finite case, but the theorem still holds. We will not worry about the details here.

Implication

- If we can show that $P_{\text{inv}} \equiv p$, we do not have to know how to sample from $p$.
- Instead, we can start with any $P_{\text{init}}$, and will get arbitrarily close to $p$ for sufficiently large $i$. 

Burn-In and Mixing Time

The number $m$ of steps required until $P_m \approx P_{\text{inv}} \equiv p$ is called the mixing time of the Markov chain. (In probability theory, there is a range of definitions for what exactly $P_m \approx P_{\text{inv}}$ means.)

In MC samplers, the first $m$ samples are also called the burn-in phase. The first $m$ samples of each run of the sampler are discarded:

\[ X_1, \ldots, X_{m-1}, X_m, X_{m+1}, \ldots \]

\begin{align*}
\text{Burn-in; discard.} & & \text{Samples from (approximately) } p; \text{ keep.}
\end{align*}

Convergence diagnostics

In practice, we do not know how large $j$ is. There are a number of methods for assessing whether the sampler has mixed. Such heuristics are often referred to as convergence diagnostics.
PROBLEM 2: SEQUENTIAL DEPENDENCE

Even after burn-in, the samples from a MC are not i.i.d.

Strategy

- Estimate empirically how many steps $L$ are needed for $x_i$ and $x_{i+L}$ to be approximately independent. The number $L$ is called the lag.
- After burn-in, keep only every $L$th sample; discard samples in between.

Estimating the lag

The most common method uses the autocorrelation function:

$$\text{Auto}(x_i, x_j) := \frac{\mathbb{E}[x_i - \mu_i] \cdot \mathbb{E}[x_j - \mu_j]}{\sigma_i \sigma_j}$$

We compute $\text{Auto}(x_i, x_{i+L})$ empirically from the sample for different values of $L$, and find the smallest $L$ for which the autocorrelation is close to zero.
There are about half a dozen popular convergence criteria; the one below is an example.

### Gelman-Rubin criterion

- Start several chains at random. For each chain $k$, sample $X^k_i$ has a marginal distribution $P^k_i$.
- The distributions of $P^k_i$ will differ between chains in early stages.
- Once the chains have converged, all $P_i = P_{inv}$ are identical.
- Criterion: Use a hypothesis test to compare $P^k_i$ for different $k$ (e.g. compare $P^2_i$ against null hypothesis $P^1_i$). Once the test does not reject anymore, assume that the chains are past burn-in.

The Metropolis-Hastings rejection kernel was defined as:

\[ A(x_{n+1}|x_n) = \min \left\{ 1, \frac{q(x_i|x_{i+1})p(x_{i+1})}{q(x_{i+1}|x_i)p(x_i)} \right\}. \]

Hence, we certainly accept if the second term is larger than 1, i.e. if

\[ q(x_i|x_{i+1})p(x_{i+1}) > q(x_{i+1}|x_i)p(x_i). \]

That means:

- We always accept the proposal value \( x_{i+1} \) if it increases the probability under \( p \).
- If it decreases the probability, we still accept with a probability which depends on the difference to the current probability.

**Hill-climbing interpretation**

- The MH sampler somewhat resembles a gradient ascent algorithm on \( p \), which tends to move in the direction of increasing probability \( p \).
- However:
  - The actual steps are chosen at random.
  - The sampler can move "downhill" with a certain probability.
  - When it reaches a local maximum, it does not get stuck there.
Everyone’s favorite example: Two Gaussians

- $\text{Var}[q]$ too large: Will overstep $p$; many rejections.
- $\text{Var}[q]$ too small: Many steps needed to achieve good coverage of domain.

If $p$ is unimodal and can be roughly approximated by a Gaussian, $\text{Var}[q]$ should be chosen as smallest covariance component of $p$.

More generally

For complicated posteriors (recall: small regions of concentration, large low-probability regions in between) choosing $q$ is much more difficult. To choose $q$ with good performance, we already need to know something about the posterior.

There are many strategies, e.g. mixture proposals (with one component for large steps and one for small steps).
**Summary: MH Sampler**

- MCMC samplers construct a MC with invariant distribution $p$.
- The MH kernel is one generic way to construct such a chain from $p$ and a proposal distribution $q$.
- Formally, $q$ does not depend on $p$ (but arbitrary choice of $q$ usually means bad performance).
- We have to discard an initial number $m$ of samples as burn-in to obtain samples (approximately) distributed according to $p$.
- After burn-in, we keep only every $L$th sample (where $L = \text{lag}$) to make sure the $x_i$ are (approximately) independent.

\[ X_1, \ldots, X_{m-1}, X_m, X_{m+1}, \ldots, X_{m+L-1}, X_{m+L}, X_{m+L+1}, \ldots, X_{m+2L-1}, X_{m+2L}, \ldots \]

- **Burn-in; discard.**
- **Samples correlated with $X_j$; discard.**
- **Samples correlated with $X_{j+L}$; discard.**
THE GIBBS SAMPLER
By far the most widely used MCMC algorithm is the Gibbs sampler.

**Full conditionals**
Suppose \( L(X) \) is a distribution on \( \mathbb{R}^D \), so \( X = (X_1, \ldots, X_D) \). The conditional probability of the entry \( X_d \) given all other entries,

\[
L(X_d | X_1, \ldots, X_{d-1}, X_{d+1}, \ldots, X_D)
\]

is called the **full conditional** distribution of \( X_d \).

On \( \mathbb{R}^D \), that means we are interested in a density

\[
p(x_d | x_1, \ldots, x_{d-1}, x_{d+1}, \ldots, x_D)
\]

**Gibbs sampling**
The Gibbs sampler is the special case of the Metropolis-Hastings algorithm defined by

\[
\text{propsoal distribution for } X_d \quad = \quad \text{full conditional of } X_d.
\]

- Gibbs sampling is only applicable if we can compute the full conditionals for each dimension \( d \).
- If so, it provides us with a *generic* way to derive a proposal distribution.
Proposal distribution
Suppose $p$ is a distribution on $\mathbb{R}^D$, so each sample is of the form $X_i = (X_{i,1}, \ldots, X_{i,D})$. We generate a proposal $X_{i+1}$ coordinate-by-coordinate as follows:

$$X_{i+1,1} \sim p(\cdot | x_{i,2}, \ldots, x_{i,D})$$

$$\vdots$$

$$X_{i+1,d} \sim p(\cdot | x_{i+1,1}, \ldots, x_{i+1,d-1}, x_{i,d+1}, \ldots, x_{i,D})$$

$$\vdots$$

$$X_{i+1,D} \sim p(\cdot | x_{i+1,1}, \ldots, x_{i+1,D-1})$$

Note: Each new $X_{i+1,d}$ is immediately used in the update of the next dimension $d + 1$.

A Metropolis-Hastings algorithms with proposals generated as above is called a **Gibbs sampler**.

No rejections
It is straightforward to show that the Metropolis-Hastings acceptance probability for each $x_{i+1,d+1}$ is 1, so *proposals in Gibbs sampling are always accepted*. 

---

**The Gibbs Sampler**
**Example: MRF**

In a MRF with $D$ nodes, each dimension $d$ corresponds to one vertex.

**Full conditionals**

In a grid with 4-neighborhoods for instance, the Markov property implies that

$$p(\theta_d | \theta_1, \ldots, \theta_{d-1}, \theta_{d+1}, \ldots, \theta_D) = p(\theta_d | \theta_{\text{left}}, \theta_{\text{right}}, \theta_{\text{up}}, \theta_{\text{down}})$$

**Specifically: Potts model with binary weights**

Recall that, for sampling, knowing only $\tilde{p}$ (unnormalized) is sufficient:

$$\tilde{p}(\theta_d | \theta_1, \ldots, \theta_{d-1}, \theta_{d+1}, \ldots, \theta_D) =$$

$$\exp\left(\beta (\mathbb{I}\{\theta_d = \theta_{\text{left}}\} + \mathbb{I}\{\theta_d = \theta_{\text{right}}\} + \mathbb{I}\{\theta_d = \theta_{\text{up}}\} + \mathbb{I}\{\theta_d = \theta_{\text{down}}\})\right)$$

This is clearly very efficiently computable.
Sampling the Potts model
Each step of the sampler generates a sample

$$\theta_i = (\theta_{i,1}, \ldots, \theta_{i,D})$$

where $D$ is the number of vertices in the grid.

Gibbs sampler
Each step of the Gibbs sampler generates $n$ updates according to

$$\theta_{i+1,d} \sim p(\cdot | \theta_{i+1,1}, \ldots, \theta_{i+1,d-1}, \theta_{i,d+1}, \ldots, \theta_{i,D})$$

$$\propto \exp \left( \beta (\mathbb{I}\{\theta_{i+1,d} = \theta_{\text{left}}\} + \mathbb{I}\{\theta_{i+1,d} = \theta_{\text{right}}\} + \mathbb{I}\{\theta_{i+1,d} = \theta_{\text{up}}\} + \mathbb{I}\{\theta_{i+1,d} = \theta_{\text{down}}\}) \right)$$
MRFs as "segmentation" priors

• MRFs where introduced as tools for image smoothing and segmentation by D. and S. Geman in 1984.

• They sampled from a Potts model with a Gibbs sampler, discarding 200 iterations as burn-in.

• Such a sample (after 200 steps) is shown above, for a Potts model in which each variable can take one out of 5 possible values.

• These patterns led computer vision researchers to conclude that MRFs are "natural" priors for image segmentation, since samples from the MRF resemble a segmented image.
E. Sudderth ran a Gibbs sampler on the same model and sampled after 200 iterations (as the Geman brothers did), and again after 10000 iterations:

- The "segmentation" patterns are not sampled from the MRF distribution $p \equiv P_{\text{inv}}$, but rather from $P_{200} \neq P_{\text{inv}}$.
- The patterns occur not because MRFs are "natural" priors for segmentations, but because the sampler’s Markov chain has not mixed.
- MRFs are smoothness priors, not segmentation priors.
VARIATIONAL INFERENCE
Problem
We have to solve an inference problem where the correct solution is an “intractable” distribution with density $p^*$ (e.g. a complicated posterior in a Bayesian inference problem).

Variational approach
Approximate $p^*$ as

$$ q^* := \arg \min_{q \in Q} \phi(q, p^*) $$

where $Q$ is a class of simple distributions and $\phi$ is a cost function (small $\phi$ means good fit). That turns the inference problem into a constrained optimization problem

$$ \min \phi(q, p^*) $$

s.t. $q \in Q$

Variational inference approximates a complicated distribution by minimizing the distance (or discrepancy) to a class of tractable distributions.
**Background: Variational Methods**

**Recall: Optimization approach to problems**
Formulate your problem such that the solution \( x^* \in \mathbb{R}^d \) is the minimum of some function \( f \), and solve

\[
x^* := \arg \min_{x \in \mathbb{R}^d} f(x)
\]

possibly under constraints.

Examples: Support vector machines, linear regression, logistic regression, . . .

**Inference problem as above**

\[
q^* := \arg \min_{q \in Q} \phi(q, p^*)
\]

- \( q \) is now a function (a density), not a point in \( \mathbb{R}^d \).
- We have to optimize over a space of functions. Such spaces are in general infinite-dimensional.
- Often: \( Q \) is a parametric model, with parameter space \( \mathcal{T} \subset \mathbb{R}^d \)
  \( \rightarrow \) reduces to optimization over \( \mathbb{R}^d \).
- However: Optimization over infinite-dimensional spaces is in principle possible.
Let $\mathcal{F}$ be a space of functions (e.g. all continuous functions on $\mathbb{R}$).

A function $\phi : \mathcal{F} \to \mathbb{R}$ (a function whose arguments are functions) is called a functional.

Examples: (1) The integral of a function. (2) The differential entropy of a density.

Recall: Derivatives (on $\mathbb{R}$)

The differential of $f : \mathbb{R}^d \to \mathbb{R}$ at point $x$ is

$$
\delta f(x) = \lim_{\varepsilon \searrow 0} \frac{f(x + \varepsilon) - f(x)}{\varepsilon}
$$

if $d = 1$ or

$$
\delta f(x) = \lim_{\|\tilde{x}\| \searrow 0} \frac{f(x + \tilde{x}) - f(x)}{\|\tilde{x}\|}
$$

in general.

The $d$-dimensional case works by reducing to the 1-dimensional case using a norm.

Derivatives of functionals

If $\mathcal{F}$ is a function space and $\| \cdot \|$ a norm on $\mathcal{F}$, we can apply the same idea to $\phi : \mathcal{F} \to \mathbb{R}$:

$$
\delta \phi(f) := \lim_{\|\tilde{f}\| \searrow 0} \frac{\phi(f + \tilde{f}) - \phi(f)}{\|\tilde{f}\|}
$$

$\delta \phi(f)$ is called the Fréchet derivative of $\phi$ at $f$.

$f$ is a minimum of a Fréchet-differentiable functional $\phi$ only if $\delta \phi(f) = 0$. 
Optimization of Functionals

Optimization

We can *in principle* find a minimum of \( \phi \) by gradient descent: Add increment functions \( \Delta f_k \) “in the direction of” \( \delta \phi(f_k) \) to the current solution candidate \( f_k \).

The maximum entropy problem is often cited as an example.

Horseshoes

- We have to represent the infinite-dimensional quantities \( f_k \) and \( \Delta f_k \) in some way.
- Many interesting functionals \( \phi \) are not Fréchet-differentiable as functionals on \( F \). They only become differentiable when constrained to a much smaller subspace.

One solution is “variational calculus”, an analytic technique that addresses both problems. (We will not need the details.)

Recall: Maximum entropy principle

- The maximum entropy principle chooses a distribution within some set \( \mathcal{P} \) of candidates by selecting the one with the largest entropy.
- That is: It solves the optimization problem

\[
\max \mathbb{H}(p) \\
\text{s.t. } p \in \mathcal{P}
\]

- For example, if \( \mathcal{P} \) are all those distributions under which some given statistic \( S \) takes a given expected value, we obtain exponential family distributions with sufficient statistic \( S \).
Optimization of Functionals

Maximum entropy as functional optimization

- The entropy \( H \) assigns a scalar to a distribution \( \rightarrow \) functional!
- Problem: The entropy as a functional e.g. on all distributions on \( \mathbb{R} \) is concave, but it is \textit{not} differentiable; it is not even continuous.
- The solution for exponential families can be determined using variational calculus.

Functional optimization in machine learning

- We will be interested in problems of the form

\[
\min_q \phi(q)
\]

s.t. \( q \in Q \)

where \( Q \) is a parametric family.
- That means each element of \( Q \) is of the form \( q(\bullet | \theta) \), for \( \theta \in \mathbb{T} \subset \mathbb{R}^d \).
- The problem then reduces back to optimization in \( \mathbb{R}^d \):

\[
\min_{\theta} \phi(q(\bullet | \theta))
\]

s.t. \( \theta \in \mathbb{T} \)

- We can apply gradient descent, Newton, etc.
KULLBACK-LEIBLER DIVERGENCE

Recall

The information in observing $X = x$ under a probability mass function $P$ is

$$J_P(x) := \log \frac{1}{P(x)} = - \log P(x).$$

Its expectation $\mathbb{H}(P) := \mathbb{E}_P[J_P(X)]$ is the entropy of $P$.

The Kullback-Leibler divergence of $P$ and $Q$ is

$$D_{KL}(P\|Q) := \mathbb{E}_P[J_Q(X)] - \mathbb{H}(P) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

Entropy and KL divergence for densities

If $p$ and $q$ are probability densities, then

$$\mathbb{H}(p) := - \int p(x) \log p(x) dx \quad \text{and} \quad D_{KL}(p\|q) := \int p(x) \log \frac{p(x)}{q(x)} dx$$

are the differential entropy of $p$ and the Kullback-Leibler divergence of $p$ and $q$.

Be careful

- The differential entropy does not behave like the entropy (e.g. it can be negative).
- The KL divergence for densities has properties analogous to the mass function case.
Recall VI optimization problem

\[ q^* := \arg \min_{q \in \mathcal{Q}} \phi(q, p^*) \]

We have to choose a cost function.

The term “variational inference” in machine learning typically implies \( \phi \) is a KL divergence,

\[ q^* := \arg \min_{q \in \mathcal{Q}} D_{KL}(q, p^*) \]

Order of the arguments

Recall that \( D_{KL} \) is not symmetric, so

\[ D_{KL}(q, p^*) \neq D_{KL}(p^*, q) \]

Which order should we use?

- Recall \( D_{KL}(p \| q) \) is an expectation with respect to \( p \).
- \( D_{KL}(p^* \| q) \) emphasizes regions where the “true” model \( p^* \) has high probability. That is what we should use if possible.
- We use VI because \( p^* \) is intractable, so we can usually not compute expectations under it.
- We use the expectation \( D_{KL}(q, p^*) \) under the approximating simpler model instead.

We have to understand the implications of this choice.
Approximating a Gaussian by a spherical Gaussian

What VI would do if possible

What VI does

\[ D_{KL}(p^* || q) = D_{KL}(\text{green} || \text{red}) \]

\[ D_{KL}(q || p^*) = D_{KL}(\text{red} || \text{green}) \]
Approximating a Gaussian mixture by a single Gaussian

What VI would do if possible

$$D_{KL}(p^* \| q) = D_{KL}(\text{blue} \| \text{red})$$

What VI does

$$D_{KL}(q \| p^*) = D_{KL}(\text{red} \| \text{blue})$$

Illustration: Bishop (2006)
Often: Target distribution is a posterior of a parameter or latent variable $Z$, given data $x$.

**Basic approximation problem**

If the posterior density is $p^*(z) = p(z|x) = \frac{p(x|z)p(z)}{p(x)}$, then

$$q^*(\cdot) = \arg\min_{q \in Q} D_{KL}(q(\cdot)||p(\cdot|x)) .$$

**Transforming the objective function**

$$D_{KL}(q(\cdot)||p(\cdot|x)) = \mathbb{E}\left[ \log \frac{q(Z)}{p(Z|x)} \right]$$

$$= \mathbb{E}[\log q(Z)] - \mathbb{E}[\log p(Z|x)]$$

$$= \mathbb{E}[\log q(Z)] - \mathbb{E}[\log p(Z, x)] + \log p(x)$$

- The evidence $p(x)$ is hard to compute (it is an integral over $p^*(x|z)$).
- It depends only on $x$, so it is an additive constant w.r.t. the optimization problem.
- Dropping it from the objective function does not change the location of the minimum.

$$F(q) := \mathbb{E}[\log q(Z)] - \mathbb{E}[\log p(Z, x)]$$
Summary: VI approximation

\[
\min_{q \in \mathcal{Q}} \quad F(q) \\
\text{s.t.} \\
\text{where} \\
F(q) = \mathbb{E}[\log q(Z)] - \mathbb{E}[\log p(Z, x)]
\]

Terminology

- The function \( F \) is called a **free energy** in statistical physics.
- Since there are different forms of free energies, various authors attach different adjectives (variational free energy, Helmholtz free energy, etc).
- Parts of the machine learning literature have renamed \( F \), by maximizing the objective function \(-F\) and calling it an **evidence lower bound**, since

\[
-F(q) + D_{\text{KL}}(q \| p(\cdot | x)) = \log p(x) \\
\text{hence} \\
e^{-F(q)} \leq p(x),
\]

and \( p(x) \) is the “evidence” in the Bayes equation.
**Mean Field Approximation**

**Definition**
A variational approximation of a probability distribution $p$ on a $d$-dimensional space

$$q^* := \arg\min_{q \in Q} D_{KL}(q, p^*)$$

is called **mean field** approximation if each element of $Q$ is factorial,

$$q(z) = q_1(z_1) \cdots q_d(z_d).$$

In previous example

**Mean field (Gaussian spherical)**

**Not a mean field**
Model
We consider a MRF distribution for $X_1, \ldots, X_n$ with values in $\{-1, +1\}$, given by

$$P(X_1, \ldots, X_n) = \frac{1}{Z(\beta)} \exp(-\beta H(X_1, \ldots, X_n))$$

where $H = -\sum_{i,j=1}^{n} w_{ij} X_i X_j - \sum_{i=1}^{n} h_i X_i$

"external field"

$w_{ij}$ and $h_i$ are constant weights.

Physicists call this a “Potts model with an external magnetic field”.

Variational approximation
We choose $Q$ as the family

$$Q := \left\{ \prod_{i=1}^{n} Q_{m_i} \right\} \bigg| m_i \in [-1, 1] \bigg\} \quad \text{where} \quad Q_m(X) := \begin{cases} \frac{1+m}{2} & X = 1 \\ \frac{1-m}{2} & X = -1 \end{cases}$$

Each factor is a Bernoulli($\frac{1+m}{2}$), except that the range is $\{-1, 1\}$ not $\{0, 1\}$. 
**Example: Mean Field for the Potts Model**

**Optimization Problem**

\[
\min D_{KL}\left(\prod_{i=1}^{n} Q_{m_i} \parallel P\right)
\]

s.t. \(m_i \in [-1, 1]\) for \(i = 1, \ldots, n\)

**Mean field solution**

The mean field approximation is given by the parameter values \(m_i\) satisfying the equations

\[
m_i = \tanh\left(\beta \left(\sum_{j=1}^{n} w_{ij} m_j + h_i\right)\right).
\]

That is: For given values of \(w_{ij}\) and \(h_i\), we have to solve for the values \(m_i\).
**Example: Mean Field for the Potts Model**

We have approximated the MRF

\[ P(X_1, \ldots, X_n) \text{ by } \prod_{i=1}^{n} \text{Bernoulli}(m_i) \text{ satisfying } m_i = \tanh\left( \beta \left( \sum_{j=1}^{n} w_{ij} m_j + h_i \right) \right) \]

(where we interpret a 0 generated by the Bernoulli as a \(-1\)).

**Interpretation**

- In the MRF \( P \), the random variables \( X_i \) interact.
- There is no interaction in the approximation.
- Instead, the effect of interactions is approximated by encoding them in the parameters.
- This is somewhat like a single effect ("field") acting on all variables simultaneously ("mean field").

**How accurate is the approximation?**

- In physics, \( P \) is used to model a ferromagnet with an external magnetic field.
- In this case, \( \beta \) is the inverse temperature.
- These systems exhibit a phenomenon called *spontaneous magnetization* at certain temperatures. The mean field solution predicts *spontaneous magnetization*, but at the wrong temperature values.
DIRECTED GRAPHICAL MODELS: MIXTURES AND ADMIXTURES
We will consider two variations on finite mixtures:

- *Bayesian mixture models* (mixtures with priors).
- *Admixtures*, in which the generation of each observation (e.g. document) can be influenced by several components (e.g. topics).
- One particular admixture model, called *latent Dirichlet allocation*, is one of the most successful machine learning models of the past ten years.
FINITE MIXTURE AS A GRAPHICAL MODELS

\[ \pi(x) = \sum_{k \leq K} c_k p(x|\theta_k) \]

Sampling from this model

1. Fix \( c_k \) and \( \theta_k \) for \( k = 1, \ldots, K \).
2. Generate \( Z_i \sim \text{Multinomial}(c_1, \ldots, c_K) \).
3. Generate \( X_i|Z_i \sim p(\cdot|\theta_{Z_i}) \).

As a graphical model

Box notation indicates \( c \) and \( \theta \) are not random.
If variables are sampled repeatedly in a graphical model, we enclose these variables in “plate”.

This means: Draw $n$ (conditionally) independent samples from $X$.

Finite mixture with plate notation
Bayesian Mixture Model

Recall: Mixing distribution of a FMM

\[ \pi(x) = \sum_{k=1}^{K} c_k p(x|\theta_k) = \int_{T} p(x|\theta)m(\theta)d\theta \quad \text{with} \quad m := \sum_{k=1}^{K} c_k \delta_{\theta_k} \]

All parameters are summarized in the mixing distribution \( m \).

Bayesian mixture model: Idea

In a Bayesian model, parameters are random variables. Here, that means a random mixing distribution:

\[ M(.) = \sum_{k=1}^{K} C_k \delta_{\Theta_k}(.) \]
How can we define a random distribution?
Since $M$ is discrete with finitely many terms, we only have to generate the random variables $C_k$ and $\Theta_k$:

$$M(. \,) = \sum_{k=1}^{K} C_k \delta_{\Theta_k}(. \,)$$

More precisely
Specifically, the term BMM implies that all priors are natural conjugate priors. That is:

- The mixture components $p(x|\theta)$ are an exponential family model.
- The prior on each $\Theta_k$ is a natural conjugate prior of $p$.
- The prior of the vector $(C_1, \ldots, C_K)$ is a Dirichlet distribution.

Explanation: Dirichlet distribution

- When we sample from a finite mixture, we choose a component $k$ from a multinomial distribution with parameter vector $(c_1, \ldots, c_k)$.
- The conjugate prior of the multinomial is the Dirichlet distribution.
Bayesian Mixture Models

Definition
A model of the form

$$
\pi(x) = \sum_{k=1}^{K} C_k p(x|\Theta_k) = \int_{T} p(x|\theta)M(\theta)d\theta
$$

is called a Bayesian mixture model if \( p(x|\theta) \) is an exponential family model and \( M \) a random mixing distribution, where:

- \( \Theta_1, \ldots, \Theta_K \sim_{iid} q(\cdot | \lambda, y) \), where \( q \) is a natural conjugate prior for \( p \).
- \( (C_1, \ldots, C_K) \) is sampled from a \( K \)-dimensional Dirichlet distribution.
Bayesian Mixture as a Graphical Model

Sampling from a Bayesian Mixture

1. Draw $C = (C_1, \ldots, C_k)$ from a Dirichlet prior.
2. Draw $\Theta_1, \ldots, \Theta_K \sim_{iid} q$, where $q$ is the conjugate prior of $p$.
3. Draw $Z_i|C \sim \text{Multinomial}(C)$.
4. Draw $X_i|Z_i, \Theta \sim p(\cdot|\Theta_{Z_i})$.

As a graphical model
Bayesian Mixture: Inference

Posterior distribution
The posterior density of a BMM under observations $x_1, \ldots, x_n$ is (up to normalization):

$$
\Pi(c_{1:K}, \theta_{1:K}|x_{1:n}) \propto \prod_{i=1}^{n} \left( \sum_{k=1}^{K} c_k p(x_i|\theta_k) \right) \left( \prod_{k=1}^{K} q(\theta_k|\lambda, y) \right) q_{\text{Dirichlet}}(c_{1:K})
$$

The posterior is analytically intractable

- Thanks to conjugacy, we can evaluate each term of the posterior.
- However: Due to the $\prod_{k=1}^{K} \left( \sum_{i=1}^{n} \ldots \right)$ bit, the posterior has $K^n$ terms!
- Even for 10 clusters and 100 observations, that is impossible to compute.
Gibbs Sampler for the BMM

This Gibbs sampler is a bit harder to derive, so we skip the derivation and only look at the algorithm.

Recall: Bayesian mixture model

- Exponential family likelihood \( p(x|\theta_k) \) for each cluster \( k = 1, \ldots, K \).
- Natural conjugate prior \( q \) for all \( \theta_k \).
- Dirichlet prior \( \text{Dirichlet}(\alpha, g) \) for the mixture weights \( c_{1:K} \).

Assignment probabilities

Each step of the Gibbs sampler computes an assignment matrix:

\[
\mathbf{a} = \begin{pmatrix}
a_{11} & \cdots & a_{1K} \\
\vdots & & \vdots \\
an_{1} & \cdots & a_{nK}
\end{pmatrix} = \left( \text{Pr}\{x_i \text{ in cluster } k\} \right)_{ik}
\]

Entries are computed as they are in the EM algorithm:

\[
a_{ik} = \frac{C_k p(x_i | \Theta_k)}{\sum_{l=1}^{K} C_l p(x_i | \Theta_l)}
\]

In contrast to EM, the values \( C_k \) and \( \Theta_k \) are random.
GIBBS FOR BMM: ALGORITHM

In each iteration $j$, the algorithm cycles through these steps:

1. For each $x_i$, sample an assignment

   \[ Z_i^j \sim \text{Multinomial}(a_{i1}^j, \ldots, a_{iK}^j) \]

   where \[ a_{ik}^j = \frac{C_k^{j-1} p(x_i|\Theta_k^{j-1})}{\sum_{l=1}^K C_l^{j-1} p(x_i|\Theta_l^{j-1})} \]

   exactly as in EM

2. For each cluster $k$, sample a new value for $\Theta_k^j$ from the conjugate posterior $\Pi(\Theta_k)$ under the observations currently assigned to $k$:

   \[ \Theta_k^j \sim \Pi\left(\Theta \left| \lambda + \sum_{i=1}^n \mathbb{I}\{Z_i^j = k\}, y + \sum_{i=1}^n \mathbb{I}\{Z_i^j = k\} S(x_i)\right.\right) \]

   # points currently assigned to $k$

   aggregate $S(x_i)$ over cluster $k$

3. Sample new cluster proportions $C_{1:K}^j$ from the Dirichlet posterior (under all $x_i$):

   \[ C_{1:K}^j \sim \text{Dirichlet}(\alpha + n, g_{1:K}^j) \]

   where \[ g_k^j = \frac{\alpha \cdot g_k + \sum_{i=1}^n \mathbb{I}\{Z_i^j = k\}}{\alpha + n} \]

   normalization
The BMM Gibbs sampler looks very similar to the EM algorithm, with maximization steps (in EM) substituted by posterior sampling steps:

<table>
<thead>
<tr>
<th>Representation of assignments</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>EM Assignment probabilities $a_{i,1:K}$</td>
<td>$a_{ik}$-weighted MLE</td>
</tr>
<tr>
<td>K-means $m_i = \text{arg max}<em>k(a</em>{i,1:K})$</td>
<td>MLE for each cluster</td>
</tr>
<tr>
<td>Gibbs for BMM $m_i \sim \text{Multinomial}(a_{i,1:K})$</td>
<td>Sample posterior for each cluster</td>
</tr>
</tbody>
</table>
TOOLS: THE DIRICHLET DISTRIBUTION
THE DIRICHLET DISTRIBUTION

Recall: Probability simplex
The set of all probability distributions on \( K \) events is the simplex

\[
\Delta_K := \{(c_1, \ldots, c_k) \in \mathbb{R}_+^K \mid c_k \geq 0 \text{ and } \sum_k c_K = 1 \}.
\]

Dirichlet distribution
The **Dirichlet distribution** is the distribution on \( \Delta_K \) with density

\[
q_{\text{Dirichlet}}(c_1:K \mid \alpha, g_1:K) := \frac{1}{\mathcal{Z}(\alpha, g_1:K)} \exp\left(\sum_{k=1}^K (\alpha g_k - 1) \log(c_k)\right)
\]

Parameters:
- \( g_1:K \in \Delta_K \): Mean parameter, i.e. \( \mathbb{E}[c_1:K] = g_1:K \).
  - Note \( g_1:K \) is a probability distribution on \( K \) categories.
- \( \alpha \in \mathbb{R}_+ \): Concentration.
  - Larger \( \alpha \rightarrow \) sharper concentration around \( g_1:K \).
THE DIRICHLET DISTRIBUTION

In all plots, $g_{1:K} = \left( \frac{1}{3}, \frac{1}{3}, \frac{1}{3} \right)$. Light colors = large density values.

Density plots

$\alpha = 1.8$

As heat maps

$\alpha = 0.8$
Large density values at extreme points

$\alpha = 1$
Uniform distribution on $\Delta_K$

$\alpha = 1.8$
Density peaks around its mean

$\alpha = 10$
Peak sharpens with increasing $\alpha$
**Multinomial-Dirichlet Model**

**Model**
The Dirichlet is the natural conjugate prior on the multinomial parameters. If we observe $h_k$ counts in category $k$, the posterior is

$$\Pi(c_1:K|h_1, \ldots, h_k) = q_{\text{Dirichlet}}(c_1:K|\alpha + n, (\alpha g_1 + h_1, \ldots, \alpha g_K + h_K))$$

where $n = \sum_k h_k$ is the total number of observations.

**Illustration: One observation**
Suppose $K = 3$ and we obtain a single observation in category 3.

Prior: Mean at the center.

Posterior: Shifted mean, increased concentration.

This extreme point corresponds to $k = 3$. 
THE DIRICHLET IS SPECIAL

- Rule of thumb: In a Bayesian model, stochastic dependence between dimensions in the prior amplifies in the posterior.
- To keep inference feasible: Keep variables in the prior “as independent as possible”.

Defining random probability distributions on $K$ categories

- If $(\Theta_1, \ldots, \Theta_K)$ is a random probability distribution, the $\Theta_k$ cannot be independent.
- How do we define $\Theta_1:K$ so that components are as independent as possible?
- Idea: Start with independent variables $X_1, \ldots, X_K$ in $(0, \infty)$. If we define

\[ \bar{X}_k := \frac{X_k}{\sum_{j=1}^K X_j} \]

then

\[ (\bar{X}_1, \ldots, \bar{X}_K) \in \Delta_K \]

Dirichlet and gamma distributions

Suppose $X_1, \ldots, X_K$ are independent random variables. If

\[ X_k \sim \text{Gamma}(\gamma_k, 1) \]

then

\[ (\bar{X}_1, \ldots, \bar{X}_K) \sim \text{Dirichlet}\left(\sum_k \gamma_k; \frac{\gamma_1}{\sum_j \gamma_j}, \ldots, \frac{\gamma_K}{\sum_j \gamma_j}\right) \]
Dependence in the prior
In general: Even if $X_1, \ldots, X_K$ are independent,

\[
\frac{X_k}{\sum_j X_j} \quad \text{and} \quad \sum_j X_j
\]

are stochastically dependent.

So: Components of the prior couple (1) through normalization and (2) through the latent variable $\sum_j X_j$.

Lukacs’ theorem: The gamma distribution is special
If $X$ and $Y$ are independent random variables in $(0, \infty)$ (and not constant), then

\[
\frac{X}{X + Y} \perp \perp X + Y
\]

if and only if $X, Y$ are gamma with the same shape parameter.

Consequence: The Dirichlet is special

- In the Dirichlet, components couple only through the normalization constraint.
- Any other random probability defined by normalizing independent variables introduces more dependence.
Recall: Multinomial text clustering

We assume the corpus is generated by a multinomial mixture model of the form

$$\pi(H) = \sum_{k=1}^{K} c_k P(H|\theta_k) ,$$

where $P(H|\theta_k)$ is multinomial.

- A document is represented by a histogram $H$.
- Topics $\theta_1, \ldots, \theta_K$.
- $\theta_{kj} = \Pr\{\text{word } j \text{ in topic } k\}$.

Problem

Each document is generated by a single topic; that is a very restrictive assumption.
Parameters
Suppose we consider a corpus with \( K \) topics and a vocabulary of \( d \) words.

- \( \phi \in \Delta_K \) topic proportions \( (\phi_k = \text{Pr}\{ \text{topic } k \}) \).
- \( \theta_1, \ldots, \theta_K \in \Delta_d \) topic parameter vectors \( (\theta_{kj} = \text{Pr}\{ \text{word } j \text{ in topic } k \}) \).

**Note:** For random generation of documents, we assume that \( \phi \) and the topic parameters \( \theta_k \) are given (they properties of the corpus). To train the model, they have to be learned from data.

**Model 1: Multinomial mixture**
To sample a document containing \( M \) words:

1. Sample topic \( Z \sim \text{Multinomial}(\phi) \).
2. For \( i = 1, \ldots, M \): Sample word \( i | Z \sim \text{Multinomial}(\theta_Z) \).

The entire document is sample from topic \( Z \).
Mixtures of topics

Whether we sample words or entire documents makes a big difference.

- When we sample from the multinomial mixture, we choose a topic at random, then sample the entire document from that topic.
- For several topics to be represented in the document, we have to sample each word individually (i.e. choose a new topic for each word).
- Problem: If we do that in the mixture above, every document has the same topic proportions.

Model 2: Admixture model

Each document explained as a mixture of topics, with mixture weights $C_{1:K}$.

Fix a matrix $\theta$ of size $\#\text{topics} \times \#\text{words}$, where

\[
\theta_{kj} := \text{probability that word } j \text{ occurs under topic } k
\]

1. Sample topic proportions $c_{1:K} \sim \text{Dirichlet}(\phi)$.
2. For $i = 1, \ldots, M$:
   2.1 Sample topic for word $i$ as $Z_i|C_{1:K} \sim \text{Multinomial}(C_{1:K})$.
   2.2 Sample word $i|Z_i \sim \text{Multinomial}(\theta_{Z_i})$.

This model is known as Latent Dirichlet Allocation (LDA).
Observation

LDA is *almost* a Bayesian mixture model: Both use multinomial components and a Dirichlet prior on the mixture weights. However, they are not identical.

Comparison

<table>
<thead>
<tr>
<th>Bayesian MM</th>
<th>Admixture (LDA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sample $c_{1:K} \sim \text{Dirichlet}(\phi)$.</td>
<td>Sample $c_{1:K} \sim \text{Dirichlet}(\phi)$.</td>
</tr>
<tr>
<td>Sample topic $k \sim \text{Multinomial}(c_{1:K})$.</td>
<td>For $i = 1, \ldots, M$:</td>
</tr>
<tr>
<td>For $i = 1, \ldots, M$:</td>
<td>Sample topic $k_i \sim \text{Multinomial}(c_{1:K})$.</td>
</tr>
<tr>
<td>Sample word$_i \sim \text{Multinomial}(\theta_k)$.</td>
<td>Sample word$<em>i \sim \text{Multinomial}(\theta</em>{k_i})$.</td>
</tr>
</tbody>
</table>

In admixtures:

- $c_{1:K}$ is generated at random, *once for each document*.
- Each word is sampled from its own topic.

What do we learn in LDA?

LDA explains each document by a separate parameter $c_{1:K} \in \Delta_K$. That is, LDA models documents as *topic proportions*. 
Bayesian mixture
(for $M$ documents of $N$ words each)
• The parameter matrix $\theta$ is of size 
  
  $\text{#topics} \times |\text{vocabulary}|$.

• Meaning: $\theta_{ki} = \text{probability that term i is observed under topic k}$.

• Note entries of $\theta$ are non-negative and each row sums to 1.

• To learn the parameters $\theta$ along with the other parameters, we add a Dirichlet prior with parameters $\eta$. The rows are drawn i.i.d. from this prior.
**Variational Inference for LDA**

**Target distribution**

Posterior \( \mathcal{L}(Z, C, \Theta | \text{words}, \alpha, \eta) \)

Recall: \( Z|C \sim \text{Multinomial}(C) \) and \( C, \Theta_1, \ldots, \Theta_K \sim \text{Dirichlet} \).

**Variational approximation**

\[ Q = \{ q(z, c, \theta | \lambda, \phi, \gamma) | \lambda, \phi, \gamma \} \]

where

\[
q(z, c, \theta | \lambda, \phi) := \prod_{k=1}^K p(\theta_k | \lambda) \prod_{m=1}^M \left( q(c_m | \gamma_m) \prod_{n=1}^N p(z_{mn} | \phi_{mn}) \right)
\]

We have introduced a new set of “variational” parameters \( \lambda, \gamma, \phi \).

**Variational inference problem**

\[
\min \quad D_{KL}(q(z, c, \theta | \lambda, \phi)\|p(z, c, \theta | \alpha, \eta))
\]

s.t. \( \quad q \in Q \)
Variational Inference for LDA

$$q(z, c, \theta | \lambda, \phi) := \prod_{k=1}^{K} p(\theta_k | \lambda) \prod_{m=1}^{M} \left( q(c_m | \gamma_m) \prod_{n=1}^{N} p(z_{mn} | \phi_{mn}) \right)$$

LDA

Variational approximation

"global" parameters

"local" parameters
Algorithmic solution
We solve the minimization problem

\[
\min \ D_{\text{KL}}(q(z, c, \theta | \lambda, \phi) \| p(z, c, \theta | \alpha, \eta)) \\
\text{s.t.} \quad q \in Q
\]

by applying gradient descent to the resulting free energy.

VI algorithm
It can be shown that gradient descent amounts to the following algorithm:

repeat (using update equations on next page)

- update local parameters
- update global parameters

until free energy has converged
In iteration $t$ of the algorithm:

**Local updates**

$$
\phi_{mn}^{(t+1)} := \frac{\tilde{\phi}_{mn}}{\sum_n \tilde{\phi}_{mn}}
$$

where

$$
\tilde{\phi}_{mn} = \exp\left( \mathbb{E}_q[\log(C_{m1}), \ldots, \log(C_{md})|\gamma^{(t)}] + \mathbb{E}_q[\log(\Theta_{1,w_n}), \ldots, \log(\Theta_{d,w_n})|\lambda^{(t)}] \right)
$$

$$
\gamma^{(t+1)} := \alpha + \sum_{n=1}^{N} \phi_n^{(t+1)}
$$

**Global updates**

$$
\lambda_k^{(t+1)} = \eta + \sum_m \sum_n \text{word}_{mn} \phi_{mn}^{(t+1)}
$$
**Example: Mixture of Topics**

<table>
<thead>
<tr>
<th>“Arts”</th>
<th>“Budgets”</th>
<th>“Children”</th>
<th>“Education”</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEW</td>
<td>MILLION</td>
<td>CHILDREN</td>
<td>SCHOOL</td>
</tr>
<tr>
<td>FILM</td>
<td>TAX</td>
<td>WOMEN</td>
<td>STUDENTS</td>
</tr>
<tr>
<td>SHOW</td>
<td>PROGRAM</td>
<td>PEOPLE</td>
<td>SCHOOLS</td>
</tr>
<tr>
<td>MUSIC</td>
<td>BUDGET</td>
<td>CHILD</td>
<td>EDUCATION</td>
</tr>
<tr>
<td>MOVIE</td>
<td>BILLION</td>
<td>YEARS</td>
<td>TEACHERS</td>
</tr>
<tr>
<td>PLAY</td>
<td>FEDERAL</td>
<td>FAMILIES</td>
<td>HIGH</td>
</tr>
<tr>
<td>MUSICAL</td>
<td>YEAR</td>
<td>WORK</td>
<td>PUBLIC</td>
</tr>
<tr>
<td>BEST</td>
<td>SPENDING</td>
<td>PARENTS</td>
<td>TEACHER</td>
</tr>
<tr>
<td>ACTOR</td>
<td>NEW</td>
<td>SAYS</td>
<td>BENNETT</td>
</tr>
<tr>
<td>FIRST</td>
<td>STATE</td>
<td>FAMILY</td>
<td>MANIGAT</td>
</tr>
<tr>
<td>YORK</td>
<td>PLAN</td>
<td>WELFARE</td>
<td>NAMPHY</td>
</tr>
<tr>
<td>OPERA</td>
<td>MONEY</td>
<td>MEN</td>
<td>STATE</td>
</tr>
<tr>
<td>THEATER</td>
<td>PROGRAMS</td>
<td>PERCENT</td>
<td>PRESIDENT</td>
</tr>
<tr>
<td>ACTRESS</td>
<td>GOVERNMENT</td>
<td>CARE</td>
<td>ELEMENTARY</td>
</tr>
<tr>
<td>LOVE</td>
<td>CONGRESS</td>
<td>LIFE</td>
<td>HAITI</td>
</tr>
</tbody>
</table>

The William Randolph Hearst Foundation will give $1.25 million to Lincoln Center, Metropolitan Opera Co., New York Philharmonic and Juilliard School. “Our board felt that we had a real opportunity to make a mark on the future of the performing arts with these grants an act every bit as important as our traditional areas of support in health, medical research, education and the social services,” Hearst Foundation President Randolph A. Hearst said Monday in announcing the grants. Lincoln Center’s share will be $200,000 for its new building, which will house young artists and provide new public facilities. The Metropolitan Opera Co. and New York Philharmonic will receive $400,000 each. The Juilliard School, where music and the performing arts are taught, will get $250,000. The Hearst Foundation, a leading supporter of the Lincoln Center Consolidated Corporate Fund, will make its usual annual $100,000 donation, too.
Restricted Boltzmann Machines
Definition
A Markov random field distribution of variables $X_1, \ldots, X_n$ with values in $\{0, 1\}$ is called a Boltzmann machine if its joint law is

$$P(x_1, \ldots, x_n) = \frac{1}{Z} \exp \left( - \sum_{i < j \leq n} w_{ij} x_i x_j - \sum_{i \leq n} c_i x_i \right),$$

where $w_{ij}$ are the edge weights of the MRF neighborhood graph, and $c_1, \ldots, c_n$ are scalar parameters.

Remarks
- The Markov blanket of $X_i$ are those $X_j$ with $w_{ij} \neq 0$.
- For $\mathbf{x} \in \{-1, 1\}^n$ instead: Potts model with external magnetic field.
- This is an exponential family with sufficient statistics $X_i X_j$ and $X_i$.
- As an exponential family, it is also a maximum entropy model.
Weight matrix

\[ P(x_1, \ldots, x_n) = \frac{1}{Z} \exp \left( - \sum_{i < j \leq n} w_{ij} x_i x_j - \sum_{i \leq n} c_i x_i \right) \]

Matrix representation

We collect the parameters in a matrix \( W := (w_{ij}) \) and a vector \( c = (c_1, \ldots, c_n) \), and write equivalently:

\[ P(x_1, \ldots, x_n) = \frac{1}{Z(W, c)} e^{-x' W x - c' x} \]

- The matrix \( W \) is the adjacency matrix of the MRF neighborhood graph.
- Because the MRF is undirected, the matrix is symmetric.
With observations
If some vertices represent observation variables $Y_i$:

$$P(x_1, \ldots, x_n, y_1, \ldots, y_m) = \frac{e^{-(x,y)'W(x,y) - c'x - \tilde{c}'y}}{Z(W, c, \tilde{c})}$$

Recall our hierarchical design approach

- Only permit layered structure.
- Obvious grouping: One layer for $X$, one for $Y$.
- As before: No connections within layers.
- Since the graph is undirected, that makes it bipartite.
Bipartite graphs

A graph is **bipartite** graph is a graph whose vertex set $V$ can be subdivided into two sets $A$ and $B = V \setminus A$ such that all edges have one end in $A$ and one end in $B$.

Definition

A **restricted Boltzmann machine (RBM)** is a Boltzmann machine whose neighborhood graph is bipartite.

That defines two layers. We usually think of one of these layers as observed, and one as unobserved.
Consider a probability of the form

$$P(x) = \frac{1}{Z} e^{-\theta x} \quad \text{for } x \in \{0, 1\} \text{ and a fixed } \theta \in \mathbb{R}.$$ 

This is what the probability of a single variable in a RBM looks like, if the distribution factorizes.

$$P(x = 1) = \frac{e^{-\theta}}{e^{-\theta} + e^0} = \frac{1}{1 + e^\theta} = \sigma(-\theta)$$

where $\sigma$ again denotes the sigmoid function.

**Consequence**

For random variables in $\{0, 1\}$:

$$P(x) = \frac{1}{Z} e^{-\theta x} \quad \Leftrightarrow \quad X \sim \text{Bernoulli}(\sigma(-\theta))$$
Full conditionals: General case

\[ P(X = x) = \frac{e^{x^t W x + c^t x}}{Z(W, c)} \]

\[ P(X_i = 1 | x_{(i)}) = \sigma(W_i^t x + c_i) \]

Full conditionals: RBM

- Variables in \( X \)-layer are conditionally independent given \( Y \)-layer and vice versa
- Two groups of conditionals: \( X|Y \) and \( Y|X \)
- Blocked Gibbs samplers

\[ P(X = x | Y = y) = \sigma(W^t y + \tilde{c}) \]
\[ P(Y = y | X = x) = \sigma(W^t x + c) \]
Deep Belief Networks
DIRECTED GRAPHICAL MODEL

\[ \Theta_1 \quad \Theta_2 \quad \cdots \quad \Theta_N \]

\[ X_1 \quad X_2 \quad \cdots \quad X_N \]

Input layer

Data
**Bayesian View**

\[
\begin{align*}
L(\Theta_{1:N}) &= \text{Prior} \\
L(X_{1:N}|\Theta_{1:N}) &= \text{Likelihood}
\end{align*}
\]
**Inference Problem**

- Task: Given $X_1, \ldots, X_N$, find $\mathcal{L}(\Theta_{1:N}|X_{1:N})$.
- Problem: Recall “explaining away”.

Conditioning on a common outcome makes variables dependent.

- That means: Although each layer is conditionally independent given the previous one, conditioning on the subsequent one creates dependence within the layer.
The prior $\mathcal{L}(\Theta_{1:N})$ can itself be represented as a directed (layered) graphical model:

Combining this prior with the likelihood stacks the two networks on top of each other:

![Graphical Models](image-url)
Complementary Prior: Idea

**Idea**

- Invent a prior such that the dependencies in the prior and likelihood “cancel out”.
- Such a prior is called a *complementary prior*.
- We will see how to construct a complementary prior below.
Consider a layered directed graphical model. Denote the vector of variables in the $k$th layer $X^{(k)}$, so

$$X^{(k)} = (X_1^{(k)}, \ldots, X_N^{(k)})$$

**Observation**

$X^{(1)}, X^{(2)}, \ldots, X^{(K)}$ is a Markov chain.

**Complementary prior idea**

- Suppose Markov chain is reversible.
- Then all arrows can be reversed.
- Now: Inference easy.
Consider a layered directed graphical model. Denote the vector of variables in the $k$th layer $X^{(k)}$, so

$$X^{(k)} = (X^{(k)}_1, \ldots, X^{(k)}_N)$$

**Observation**

$X^{(1)}, X^{(2)}, \ldots, X^{(K)}$ is a Markov chain.

**Complementary prior idea**

- Suppose Markov chain is reversible.
- Then all arrows can be reversed.
- Now: Inference easy.
• Find reversible Markov chain with $P^{(1)} = P_{\infty}$

• Let $p_T$ be its transition kernel

• Choose $P^{(k+1)}(\bullet | X^{(k)} = x) = p_T(\bullet | x)$

• Then $P^{(2)} = \ldots = P^{(k)} = P_{\infty}$

• Since chain is reversible, $P^{(k)}(\bullet | X^{(k+1)} = x) = p_T(\bullet | x)$

and edges flip.

**Building a Complementary Prior**

- Find reversible Markov chain with
  \[ P^{(1)} = P_\infty \]
- Let \( p_T \) be its transition kernel
- Choose
  \[ P^{(k+1)}(\bullet | X^{(k)} = x) = p_T(\bullet | x) \]
- Then \( P^{(2)} = \ldots = P^{(k)} = P_\infty \)
- Since chain is reversible,
  \[ P^{(k)}(\bullet | X^{(k+1)} = x) = p_T(\bullet | x) \]
  and edges flip.

WHERE DO WE GET THE MARKOV CHAIN?

Start with an RBM

\[
\begin{array}{c}
X \\
Y
\end{array}
\]

Blocked Gibbs sampling alternates between \(X\) and \(Y\)

\[
X^{(1)} \rightarrow Y^{(1)} \rightarrow X^{(2)} \rightarrow Y^{(2)} \rightarrow \ldots \rightarrow X^{(K)} \rightarrow Y^{(K)}
\]

“Roll off” this chain into a graphical model

The Gibbs sampler for the RBM becomes the model for the directed network.
Where do we get the Markov chain?

For $K \to \infty$, we have

$$Y^{(\infty)} \overset{d}{=} Y$$

Now suppose we use $\mathcal{L}(Y^{(\infty)})$ as our prior distribution. Then:

Using an “infinitely deep” graphical model given by the Gibbs sampler for the RBM as a prior is equivalent to using the $Y$-layer of the RBM as a prior.
Deep Belief Networks

First two layers: RBM (undirected)

Remaining layers: Directed
Deep Belief Networks

Summary

• The RBM consisting of the first two layers is equivalent to an infinitely deep directed network representing the Gibbs samplers. (The rolled-off Gibbs sampler on the previous slide.)

• That network is infinitely deep because a draw from the actual RBM distribution corresponds to a Gibbs sampler that has reached its invariant distribution.

• When we draw from the RBM, the second layer is distributed according to the invariant distribution of the Markov chain given by the RBM Gibbs sampler.

• If the transition from each layer to the next in the directed part is given by the Markov chain’s transition kernel $p_T$, the RBM is a complementary prior.

• We can then reverse all edges between the $\Theta$-layer and the $X$-layer.
Neural Networks: Basics
A neural network represents a function $f : \mathbb{R}^{d_1} \rightarrow \mathbb{R}^{d_2}$. 
\[ f : \mathbb{R}^3 \rightarrow \mathbb{R}^3 \quad \text{with input} \quad x = \begin{pmatrix} x_1 \\ x_2 \\ x_2 \end{pmatrix} \]

\[ f_1(x) = \phi_1(w_1^t x) \quad f_2(x) = \phi_2(w_2^t x) \quad f_3(x) = \phi_3(w_3^t x) \]

\[ f(x) = \begin{pmatrix} f_1(x) \\ f_2(x) \\ f_3(x) \end{pmatrix} \quad \text{with} \quad f_i(x) = \phi_i \left( \sum_{j=1}^{3} w_{ij} x_j \right) \]
\[
\begin{align*}
  f(x) &= f^{(K)}(\ldots f^{(2)}(f^{(1)}(x))) = f^{(K)} \circ \ldots \circ f^{(1)}(x)
\end{align*}
\]
Layers

- Each function $f^{(k)}$ is of the form
  \[ f^{(k)} : \mathbb{R}^{d_k} \to \mathbb{R}^{d_{k+1}} \]

- $d_k$ is the number of nodes in the $k$th layer. It is also called the width of the layer.
- We mostly assume for simplicity: $d_1 = \ldots = d_K =: d$.

Layered feed-forward network

A feed-forward network one organized into successive layers as on previous slide,

\[ f(x) = f^{(K)} \circ \ldots \circ f^{(1)}(x) \]

Each layer represents a function $f^{(k)}$. These functions are of the form:

\[
f^{(k)}(\cdot) = \begin{pmatrix}
\phi^{(k)}_1(\langle w^{(k)}_1, \cdot \rangle) \\
\vdots \\
\phi^{(k)}_d(\langle w^{(k)}_d, \cdot \rangle)
\end{pmatrix}
\]

typically:

\[
\phi(x) = \begin{cases}
\sigma(x) & \text{(sigmoid)} \\
\mathbb{I}\{\pm x > \tau\} & \text{(threshold)} \\
c & \text{(constant)} \\
x & \text{(linear)}
\end{cases}
\]
Hidden units

- Any nodes (or “units”) in the network that are neither input nor output nodes are called hidden.
- Every network has an input layer and an output layer.
- If there any additional layers (which hence consist of hidden units), they are called hidden layers.

Linear and nonlinear networks

- If a network has no hidden units, then
  \[ f_i(x) = \phi_i(\langle w_i, x \rangle) \]
  That means: \( f \) is a linear functions, except perhaps for the final application of \( \phi \).
- For example: In a classification problem, a two layer network can only represent linear decision boundaries.
- Networks with at least one hidden layer can represent nonlinear decision surfaces.
Three-layer neural network. We now turn to the crucial problem of setting the weights. We have just seen that any function from input to output can be implemented as a single-layer network. This is related to the training patterns in a very complicated way. All in all, then, these methods are not particularly practical because for most problems we know ahead of time neither the number of hidden units required, nor the proper weight values. Even if there is a way to choose the weights, it is not clear how.

6.3 Backpropagation algorithm

While we can be confident that a complete set of functions, such as all polynomial functions, can represent any function, it is nevertheless a fact that a three-, four-, or higher-layer network can also represent any function. In the absence of information suggesting otherwise, we generally use a single functional form for the transfer functions. While these latter constructions show that any desired function can be implemented by a three-layer network, they are not particularly practical because for most of the problems we know ahead of time neither the number of hidden units required, nor the proper weight values.

Two-layer networks — their main benefit for pattern recognition (Fig. 6.3).
### The XOR Problem

- Two ridges at different locations are subtracted from each other.
- That generates a region bounded on both sides.
- A linear classifier cannot represent this decision region.
- Note this requires at least one hidden layer.
Each hidden unit emits a nonlinear function $\Xi$ of its total input; the output unit merely emits the sum of the contributions of the hidden units.

Unfortunately, the relationship of Kolmogorov's theorem to practical neural networks is a bit tenuous, for several reasons. In particular, the functions $\Xi_j$ and $\psi_{ij}$ are not the simple weighted sums passed through nonlinearities favored in neural networks. In fact those functions can be extremely complex; they are not smooth, and indeed for subtle mathematical reasons they cannot be smooth. As we shall soon see, smoothness is important for gradient descent learning. Most importantly, Kolmogorov's Theorem tells us very little about how to find the nonlinear functions based on data — the central problem in network based pattern recognition.

A more intuitive proof of the universal expressive power of three-layer nets is inspired by Fourier's Theorem that any continuous function $g(x)$ can be approximated arbitrarily closely by a (possibly infinite) sum of harmonic functions (Problem 2). One can imagine networks whose hidden units implement such harmonic functions. Proper hidden-to-output weights related to the coefficients in a Fourier synthesis would then enable the full network to implement the desired function. Informally speaking, we need not build up harmonic functions for Fourier-like synthesis of a desired function. Instead a sufficient number of "bumps" at different input locations, of different amplitude and sign, can be put together to give our desired function. Such localized bumps might be implemented in a number of ways, for instance by sigmoidal transfer functions grouped appropriately (Fig. 6.2). The Fourier analogy and bump constructions are conceptual tools, they do not explain the way networks in fact function. In short, this is not how neural networks "work" — we never find that through training (Sect. 6.3) simple networks build a Fourier-like representation, or learn to group sigmoids to get component bumps.

Features

- Raw measurement data is typically not used directly as input for a learning algorithm. Some form of preprocessing is applied first.
- We can think of this preprocessing as a function, e.g.

\[ F : \text{raw data space} \rightarrow \mathbb{R}^d \]

(\(\mathbb{R}^d\) is only an example, but a very common one.)
- If the raw measurements are \(m_1, \ldots, m_N\), the data points which are fed into the learning algorithm are the images \(x_n := F(m_n)\).

Terminology

- \(F\) is called a feature map.
- Its dimensions (the dimensions of its range space) are called features.
- The preprocessing step (= application of \(F\) to the raw data) is called feature extraction.
This is what a typical processing pipeline for a supervised learning problem might look like.
Where does learning start?

- It is often a matter of definition where feature extraction stops and learning starts.
- If we have a perfect feature extractor, learning is trivial.
- For example:
  - Consider a classification problem with two classes.
  - Suppose the feature extractor maps the raw data measurements of class 1 to a single point, and all data points in class to to a single distinct point.
  - Then classification is trivial.
  - That is of course what the classifier is supposed to do in the end (e.g. map to the points 0 and 1).

Multi-layer networks and feature extraction

- An interesting aspect of multi-layer neural networks is that their early layers can be interpreted as feature extraction.
- For certain types of problems (e.g. computer vision), features were long “hand-tuned” by humans.
- Features extracted by neural networks give much better results.
- Several important problems, such as object recognition and face recognition, have basically been solved in this way.
• The network on the right is a classifier $f : \mathbb{R}^d \rightarrow \{0, 1\}$.

• Suppose we subdivide the network into the first $K - 1$ layer and the final layer, by defining

$$F(x) := f^{(K-1)} \circ \ldots \circ f^{(1)}(x)$$

• The entire network is then

$$f(x) = f^{(K)} \circ F(x)$$

• The function $f^{(K)}$ is a two-class logistic regression classifier.

• We can hence think of $f$ as a feature extraction $F$ followed by linear classification $f^{(K)}$.

$$f^{(K)}(\bullet) = \sigma(\langle w^{(K)}, \bullet \rangle)$$
6.6. BACKPROPAGATION, BAYES THEORY AND PROBABILITY

Sample training patterns
Learned input-to-hidden weights

Figure 6.14: The top images represent patterns from a large training set used to train a 64-2-3 sigmoidal network for classifying three characters. The bottom figures show the input-to-hidden weights (represented as patterns) at the two hidden units after training. Note that these learned weights indeed describe feature groupings useful for the classification task. In large networks, such patterns of learned weights may be difficult to interpret in this way.

6.6.1 Bayes discriminants and neural networks

As we saw in Chap. ?? Sect. ??, the LMS algorithm computed the approximation to the Bayes discriminant function for two-layer nets. We now generalize this result in two ways: to multiple categories and to nonlinear functions implemented by three-layer neural networks. We use the network of Fig. 6.4 and let $g_k(x; w)$ be the output of the $k$th output unit — the discriminant function corresponding to category $\omega_k$.

Recall first Bayes' formula,

$$P(\omega_k | x) = \frac{P(x | \omega_k) P(\omega_k)}{\sum_{i=1}^c P(x | \omega_i) P(\omega_i)} = P(x, \omega_k) P(x)$$

and the Bayes decision for any pattern $x$: choose the category $\omega_k$ having the largest discriminant function $g_k(x) = P(\omega_k | x)$.

Suppose we train a network having $c$ output units with a target signal according to:

$$t_k(x) = \begin{cases} 1 & \text{if } x \in \omega_k \\ 0 & \text{otherwise} \end{cases}$$

(In practice, teaching values of $\pm 1$ are to be preferred, as we shall see in Sect. 6.8; we use the values 0–1 in this derivation for computational simplicity.) The contribution to the criterion function based on a single output unit $k$ for finite number of training samples $x$ is:

$$J(w) = \sum_x [g_k(x; w) - t_k]^2$$

Problem: Classify characters into three classes (E, F and L).

• Each digit given as a $8 \times 8 = 64$ pixel image

• Neural network: 64 input units (=pixels)

• 2 hidden units

• 3 binary output units, where $f_i(x) = 1$ means image is in class $i$.

• Each hidden unit has 64 input weights, one per pixel. The weight values can be plotted as $8 \times 8$ images.
A Simple Example

- Dark regions = large weight values.
- Note the weights emphasize regions that distinguish characters.
- We can think of weight (= each pixel) as a feature.
- The features with large weights for $h_1$ distinguish \{E,F\} from L.
- The features for $h_2$ distinguish \{E,L\} from F.
A neural network represents a (typically) complicated function $f$ by simple functions $\phi_i^{(k)}$.

**What functions can be represented?**

A well-known result in approximation theory says: Every continuous function $f : [0, 1]^d \rightarrow \mathbb{R}$ can be represented in the form

$$f(x) = \sum_{j=1}^{2d+1} \xi_j \left( \sum_{i=1}^d \tau_{ij}(x_i) \right)$$

where $\xi_i$ and $\tau_{ij}$ are functions $\mathbb{R} \rightarrow \mathbb{R}$. A similar result shows one can approximate $f$ to arbitrary precision using specifically sigmoids, as

$$f(x) \approx \sum_{j=1}^M w_j^{(2)} \sigma \left( \sum_{i=1}^d w_{ij}^{(1)} x_i + c_i \right)$$

for some finite $M$ and constants $c_i$.

Note the representations above can both be written as neural networks with three layers (i.e. with one hidden layer).
**Width vs Depth**

**Depth rather than width**

- The representations above can achieve arbitrary precision with a single hidden layer (roughly: a three-layer neural network can represent any continuous function).
- In the first representation, $\xi_j$ and $\tau_{ij}$ are “simpler” than $f$ because they map $\mathbb{R} \to \mathbb{R}$.
- In the second representation, the functions are more specific (sigmoids), and we typically need more of them ($M$ is large).
- That means: The price of precision are many hidden units, i.e. the network grows wide.
- The last years have shown: We can obtain very good results by limiting layer width, and instead increasing depth (= number of layers).
- There is no theory to properly explain this yet.

**Limiting width**

- Limiting layer width means we limit the degrees of freedom of each function $f^{(k)}$.
- That is a notion of parsimony.
- Again: There seem to be a lot of interesting questions to study here, but so far, we have no answers.
An example for the effect of layer are autoencoders.

- An **autoencoder** is a neural network that is trained on its own input: If the network has weights $W$ and represents a function $f_W$, training solves the optimization problem

$$\min_W \|x - f_W(x)\|^2$$

or something similar for a different norm.

- That seems pointless at first glance: The network tries to approximate the identity function using its (possibly nonlinear) component functions.

- However: If the layers in the middle have much fewer nodes that those at the top and bottom, the network learns to *compress the input.*
Autoencoders

- Train network on many images.
- Once trained: Input an image $x$.
- Store $x' := f^{(2)}(x)$. Note $x'$ has fewer dimensions than $x \rightarrow$ compression.
- To decompress $x'$: Input it into $f^{(3)}$ and apply the remaining layers of the network $\rightarrow$ reconstruction $f(x) \approx x$ of $x$. 
**Autoencoders**

![Diagram of Autoencoders](image)

Error measure

- We assume each training data point \( x \) comes with a label \( y \).
- We specify an error measure \( D \) that compares \( y \) to a prediction.

Typical error measures

- Classification problem:
  \[
  D(\hat{y}, y) := y \log \hat{y}
  \]
- Regression problem:
  \[
  D(\hat{y}, y) := \|y - \hat{y}\|^{2}
  \]

Training as an optimization problem

- Given: Training data \( (x_1, y_1), \ldots, (x_N, y_N) \) with labels \( y_i \).
- We specify an error measure \( D \) that compares \( y \) to a prediction.

\[
J(w) := \sum_{i=1}^{N} D(f_w(x_i), y_i)
\]
Neural network training optimization problem

\[
\min_w J(w)
\]

The application of gradient descent to this problem is called *backpropagation*.

Backpropagation is gradient descent applied to \( J(w) \) in a feed-forward network.

**Deriving backpropagation**

- We have to evaluate the derivative \( \nabla_w J(w) \).
- Since \( J \) is additive over training points, \( J(w) = \sum_n J_n(w) \), it suffices to derive \( \nabla_w J_n(w) \).
Recall from calculus: Chain rule
Consider a composition of functions \( f \circ g(x) = f(g(x)) \).

\[
\frac{d(f \circ g)}{dx} = \frac{df}{dg} \frac{dg}{dx}
\]

If the derivatives of \( f \) and \( g \) are \( f' \) and \( g' \), that means: \( \frac{d(f \circ g)}{dx}(x) = f'(g(x))g'(x) \)

Neural network
Let \( w^{(k)} \) denote the weights in layer \( k \). The function represented by the network is

\[
f_w(x) = f_w^{(K)} \circ \cdots \circ f_w^{(1)}(x) = f_w^{(K)}(w^{(K)}(\cdots(v^{(1)}(x))))
\]

To solve the optimization problem, we have to compute derivatives of the form

\[
\frac{d}{dw} D(f_w(x_n), y_n) = \frac{dD(\cdot, y_n)}{df_w} \frac{df_w}{dw}
\]
CONSIDERING THE DERIVATIVES

- We will compute the derivates layer by layer.
- Suppose we are only interested in the weights of layer $k$, and keep all other weights fixed. The function $f$ represented by the network is then

$$f_{w(k)}(x) = f^{(K)} \circ \cdots \circ f^{(k+1)} \circ f_{w(k)}^{(k)} \circ f^{(k-1)} \circ \cdots \circ f^{(1)}(x)$$

- The first $k - 1$ layers enter only as the function value of $x$, so we define

$$z^{(k)} := f^{(k-1)} \circ \cdots \circ f^{(1)}(x)$$

and get

$$f_{w(k)}(x) = f^{(K)} \circ \cdots \circ f^{(k+1)} \circ f_{w(k)}^{(k)}(z^{(k)})$$

- If we differentiate with respect to $w^{(k)}$, the chain rule gives

$$\frac{d}{dw(k)} f_{w(k)}(x) = \frac{df^{(K)}}{df^{(K-1)}} \cdots \frac{df^{(k+1)}}{df^{(k)}} \cdot \frac{df_{w(k)}^{(k)}}{dw(k)}$$
Within a Single Layer

- Each $f^{(k)}$ is a vector-valued function $f^{(k)} : \mathbb{R}^{d_k} \to \mathbb{R}^{d_{k+1}}$.
- It is parametrized by the weights $w^{(k)}$ of the $k$th layer and takes an input vector $z \in \mathbb{R}^{d_k}$.
- We write $f^{(k)}(z, w^{(k)})$.

Layer-wise derivative

Since $f^{(k)}$ and $f^{(k-1)}$ are vector-valued, we get a Jacobian matrix

$$
\frac{df^{(k+1)}}{df^{(k)}} = \begin{pmatrix}
\frac{\partial f_1^{(k+1)}}{\partial f_1^{(k)}} & \cdots & \frac{\partial f_1^{(k+1)}}{\partial f_{d_k}^{(k)}} \\
\vdots & \ddots & \vdots \\
\frac{\partial f_{d_k+1}^{(k+1)}}{\partial f_1^{(k)}} & \cdots & \frac{\partial f_{d_k+1}^{(k+1)}}{\partial f_{d_k}^{(k)}}
\end{pmatrix} =: \Delta^{(k)}(z, w^{(k+1)})
$$

- $\Delta^{(k)}$ is a matrix of size $d_{k+1} \times d_k$.
- The derivatives in the matrix quantify how $f^{(k+1)}$ reacts to changes in the argument of $f^{(k)}$ if the weights $w^{(k+1)}$ and $w^{(k)}$ of both functions are fixed.
Let $w^{(1)}, \ldots, w^{(K)}$ be the current settings of the layer weights. These have either been computed in the previous iteration, or (in the first iteration) are initialized at random.

**Step 1: Forward pass**

We start with an input vector $x$ and compute

$$z^{(k)} := f^{(k)} \circ \ldots \circ f^{(1)}(x)$$

for all layers $k$.

**Step 2: Backward pass**

- Start with the last layer. Update the weights $w^{(K)}$ by performing a gradient step on

$$D\left(f^{(K)}(z^{(K)}, w^{(K)}), y\right)$$

regarded as a function of $w^{(K)}$ (so $z^{(K)}$ and $y$ are fixed). Denote the updated weights $\tilde{w}^{(K)}$.

- Move backwards one layer at a time. At layer $k$, we have already computed updates $\tilde{w}^{(K)}, \ldots, \tilde{w}^{(k+1)}$. Update $w^{(k)}$ by a gradient step, where the derivative is computed as

$$\Delta^{(K-1)}(z^{(K-1)}, \tilde{w}^{(K)}) \cdot \ldots \cdot \Delta^{(k)}(z^{(k)}, \tilde{w}^{(k+1)}) \frac{df^{(k)}}{dw^{(k)}}(z, w^{(k)})$$

On reaching level 1, go back to step 1 and recompute the $z^{(k)}$ using the updated weights.
Backpropagation is a gradient descent method for the optimization problem

$$\min_w J(w) = \sum_{i=1}^{N} D(f_w(x_i), y_i)$$

$D$ must be chosen such that it is additive over data points.

- It alternates between forward passes that update the layer-wise function values $z^{(k)}$ given the current weights, and backward passes that update the weights using the current $z^{(k)}$.

- The layered architecture means we can (1) compute each $z^{(k)}$ from $z^{(k-1)}$ and (2) we can use the weight updates computed in layers $K, \ldots, k + 1$ to update weights in layer $k$. 
MACHINE ARITHMETIC AND PSEUDORANDOM NUMBERS
Binary Representation of Numbers

Binary numbers

Our usual (decimal) representation of integers represents a number $x \in \mathbb{N}_0$ by digits $d_1, d_2, \ldots \in \{0, \ldots, 9\}$ as

$$[x]_{10} = d_i \cdot 10^i + d_{i-1} \cdot 10^{i-1} + \ldots + d_1 \cdot 10^0$$

where $i$ is the largest integer with $10^i \leq x$.

The binary representation of $x$ is similarly

$$[x]_2 = 2^j \cdot 1^i + b_{j-1} \cdot 2^{j-1} + \ldots + b_1 \cdot 2^0$$

where $b_1, b_2, \ldots \in \{0, 1\}$ and $j$ is the largest integer with $2^j \leq x$.

Non-integer numbers

Binary numbers can have fractional digits just as decimal numbers. The post-radix digits correspond to inverse powers of two:

$$[10.125]_{10} = [101.001]_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2 + 0 \cdot \frac{1}{2} + 0 \cdot \frac{1}{2^2} + 1 \cdot \frac{1}{2^3}$$

Numbers that look “simple” may become more complicated in binary representation:

$$[0.1]_{10} = [0.00011]_2 = [0.000110011001100110011 \ldots]_2$$
Basic FP representation

\[ x = (-1)^s \cdot 2^e \cdot m \]

where:

- \( s \), the **sign**, is a single bit, \( s \in \{0, 1\} \).
- \( e \), the **exponent**, is a (binary) non-negative integer \( e \in \{-2^{n-1} + 1, \ldots, 2^{n-1}\} \).
- \( m \), the **mantissa**, is a (binary) integer \( m \in \{0, \ldots, 2^k - 1\} \).

There are \( 2^n \) possible values for \( e \), and \( 2^k \) possible values for \( m \).

**The IEEE 754 standard**

- The standard floating point system is specified by an industry standard called IEEE 754.
- This standard is implemented by all (well-written) modern software.
- The standard was first published in 1985. There were no proper, widely implemented rules for numerical computation before then.
Single and Double Precision

Single precision numbers (in IEEE 754)

\[
\begin{array}{ccccccc}
  s & e_1 & \cdots & e_n & m_1 & \cdots & m_k \\
  0 & 1 & 8 & 9 & 31 \\
\end{array}
\]

- A single precision number is represented by 32 bits.
- It is customary to enumerate starting with 0, as 0, \ldots, 31.
- 8 bits are invested in the exponent, 1 bit is needed for the sign; the remainder form the mantissa.

Double precision numbers (in IEEE 754)

\[
\begin{array}{ccccccc}
  s & e_1 & \cdots & e_n & m_1 & \cdots & m_k \\
  0 & 1 & 11 & 12 & 63 \\
\end{array}
\]

- A single precision number is represented by 64 bits.
- 1 bit for the sign, 11 for the exponent, 52 for the mantissa.
- This is the FP number format used in most numerical computations.
Rounding

- Choose a set of FP numbers $M$, (say double precision numbers).
- Start with a number $x \in \mathbb{R}$. Unless $x$ happens to be in the finite set $M$, we cannot represent it. It gets rounded to the closest FP number,

$$\hat{x} := \arg\min_{\hat{y} \in M} |x - \hat{y}|$$

- Suppose we multiply to machine numbers $\hat{x}, \hat{y} \in M$. The product $z = \hat{x} \cdot \hat{y}$ has twice as many digits, and is typically not in $M$. It gets rounded to $\hat{z}$.

Rounding errors

We distinguish two types of errors:

- The absolute rounding error $\hat{z} - z$.
- The relative rounding error \( \frac{\hat{z} - z}{z} \).

FP arithmetic is based on the premise that the relative rounding error is more important.
Distribution of FP numbers
The distance between two neighboring FP numbers depends on the value of the exponent. That means:

An interval of length 1 (or any fixed length) contains more FP numbers if it is close to 0 than if it is far away from 0.

In other words, the relative rounding error does not depend on the size of a number, but the absolute rounding error is larger for large numbers.

Normalized FP number
FP numbers are not unique: Note that

\[ 2 = (-1)^0 \cdot 2^1 \cdot 1 = (-1)^0 \cdot 2^0 \cdot 2 = \ldots \]

A machine number is normalized if the mantissa represents a number with exactly one digit before the radix, and this digit is 1:

\[ (-1)^s \cdot 2^e \cdot [1.m_1 \ldots m_k]_2 \]

IEEE 754 only permits normalized numbers. That means every number is unique.
Error guarantees
IEEE 754 guarantees that
\[ x \oplus y = (x \oplus y)(1 + r) \]
satisfies \[ |r| < \text{eps} \]
where
- \( r \) is the relative rounding error.
- \( \text{eps} \) is machine precision.
- \( \oplus \) is one of the operations +, −, × or /, and \( \hat{\oplus} \) its machine version.

Machine precision
The machine precision, denoted \( \text{eps} \), is the smallest machine number for which \( \hat{1} \oplus \text{eps} \) is distinguishable from 1:
\[ \text{eps} = \min_{\hat{1} \oplus e > \hat{1}} \{ e \in M \} \]
This is not the same as the smallest representable number. In double precision:
- Machine precision is \( \text{eps} = 2^{-52} \)
- The smallest (normalized) FP number is \( 2^{-1022} \).
That is so because the intervals between FP numbers around 1 are much larger than around \( 2^{-1022} \) : If we evaluate \( 1 + 2^{-1022} \), the result is 1.
Cancellation Errors

Cancellation

- Suppose two machine numbers $\hat{x} = 1.2345e0$ and $\hat{y} = 1.2346e0$ are obtained by some numerical computation.
- That means they have already been rounded, perhaps several times.
- Previous rounding means the smallest digits in the mantissa carry errors.
- If we substract the numbers, $\hat{x} - \hat{y} = -0.0001e0$, only the those small digits remain.
- Effectively, we have deleted the reliable information and only kept the errors.

The example illustrates a fundamental problem: The FP principle (keep the relative error under control) does not work well with substraction.

Example: Linear equations

- One application where cancellation errors are very problematic are linear equation systems.
- If a matrix is inverted by Gaussian elimination, a lot of terms have to be scaled and substracted.
- If the input matrices are not well-conditioned, Gaussian elimination can produce results that consist only of “noise”.
- The Householder and Givens algorithms that most computer packages used have been developed to avoid this problem.
**RANDOM NUMBERS**

**Pseudorandom numbers**

- All implementations of sampling algorithms, random variables, randomized algorithms, etc on a computer require the generation of random numbers.
- Computers are deterministic machines and have no access to “real” randomness.
- The “random” numbers we use on computers are not random; rather, they are generated in a way that makes the look like random numbers, in the sense that they do not contain any obvious deterministic “patterns”. Such numbers are called **pseudorandom numbers**.

**Reproducibility**

- Scientific experiments are required to be reproducible.
- Using actual random numbers in a computer experiment or simulation would make it non-reproducible.
- In this sense, pseudorandomness is an advantage rather than disadvantage.

---

The so-called random numbers generated by a computer are not random.

If you restart the generation process of a pseudorandom sequence with the same initial conditions, you reproduce exactly the same sequence.
Recursive generation
Most PRN generators use a form of recursion: A sequence $x_1, x_2, \ldots$ of PRNs is generated as

$$x_{n+1} = f(x_n)$$

Modulo operation
The $\text{mod}$ operation outputs the “remainder after division”:

$$m \mod n = m - kn$$

for largest $k \in \mathbb{N}_0$ such that $kn < m$

For example: $13 \mod 5 = 3$

Linear Congruence Generator
An example are generators of the form

$$x_{n+1} = C \cdot x_n \mod D$$

for fixed constants $C, D \in \mathbb{N}$.

On a 32-bit machine, a simple generator would be

$$x_{n+1} = 16807 \cdot x_n \mod (2^{32} - 1)$$
Period length

- For a recursive generator \( x_{n+1} = f(x_n) \), each \( x_{n+1} \) depends only on \( x_n \).
- Once \( x_n \) takes the same value as \( x_1 \), the sequence \((x_1, \ldots, x_{n-1})\) repeats:

\[
(x_1, \ldots, x_{n-1}) = (x_n, \ldots, x_{2n-1}) = \ldots
\]

- If so, \( n - 1 \) is called the **period length**.
- Note a period need not start at 1: Once a value reoccurs for the first time, the generator has become periodic.
- Since there are finitely many possible numbers, that must happen sooner or later.

Mersenne Twisters

- Almost all software tools we use (Python, Matlab etc) use a generator called a **Mersenne twister**.
- This algorithm also uses a recursion, but the recursion is matrix-valued. That makes the algorithm a little more complicated.
- Mersenne twisters have very long period lengths (e.g. \( 2^{19937} - 1 \) for the standard implementation for 32-bit machines).
Random seed of a generator

- A recursive generator $x_{n+1} = f(x_n)$ can be started at any initial value $x_0$ (so the first generated number is $x_1 = f(x_0)$).
- This initial value $x_0$ is called the random seed.

In practice

- Every time a generator is started or reset (e.g. when you start Matlab), it starts with a default seed.
- For example, Matlab’s default generator is a Mersenne twister with seed 0.
- The user can usually pass a different seed to the generator.
- When you run experiments or simulations, always re-run them several times with different seeds.
- To ensure reproducibility, record the seeds you used along with the code.