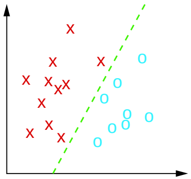
Probabilistic and Unsupervised Learning

Peter Orbanz • Fall 2020

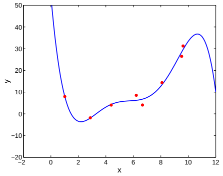
Version: 4 November 2020

SUPERVISED LEARNING

Here are two types of learning problems you may have encountered before:



classifiction



regression

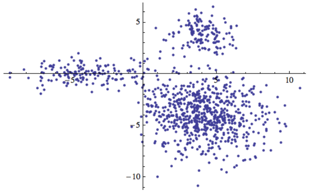
- These are both supervised learning problems.
- In both cases we are fitting a function to data—statistically speaking we are performing regression. The function takes finitely many values (classification) or is continuous-valued (regression).
- The term *supervised* indicates that we have examples of (possibly noisy) solutions, namely input and output values.

Three learning problems

- Predicting new outcomes: generalising.
    - **Supervised learning.** Observe input/output pairs ("teaching"):
      $$(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), \ldots$$
      Predict the correct y^* for new input x^*.
- Systematising (noisy) observations: discovering structure.
    - **Unsupervised learning.** Observe (sensory) input alone:
      $$x_1, x_2, x_3, x_4, \ldots$$
      Describe pattern of data $[p(x)]$, identify and extract underlying structural variables $[x_i \to y_i]$.
- Choosing actions wisely.
    - **Reinforcement learning.** Rewards or payoffs (and possibly also inputs) depend on actions:
      $$x_1 : a_1 \to r_1, x_2 : a_2 \to r_2, x_3 : a_3 \to r_3 \ldots$$
      Find a policy for action choice that maximises payoff.

Supervised vs Unsupervised

Consider a data source generating point in $\mathbb{R}^2$. Each point belongs to one of three groups. Here is a sample:



We are asked to assign data points to groups. There is a supervised and an unsupervised versions of this problem.

Classification

- Given: The data above, and the group assignment of each point.
- Task: Assign *new* points generated by the same source to their groups.
- Supervised problem.
- Jargon: Groups are called "classes".
- Solution is typically a function.

Clustering

- Given: The data above only.
- Task: Assign each point *in the data* to a group.
- Unsupervised problem.
- Jargon: Groups are called "clusters".
- Solution is typically a distribution.

PENDULUM
(WORK OF MARC DEISENROTH AND CARL EDWARD RASMUSSEN)

Task
Balance the pendulumn 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).

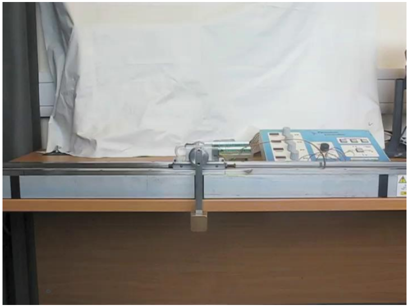
Formalization

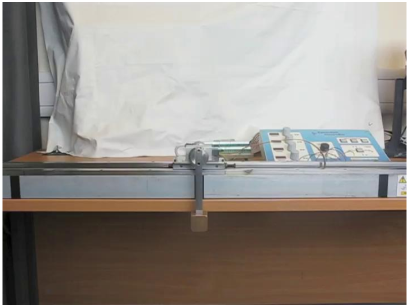
State = 4 variables (sled location, sled velocity, angle, angular velocity)
Actions = sled movements

The system can be described by a function

$$\begin{aligned} f: \quad \mathcal{S} \times \mathcal{A} &\to \mathcal{S} \\ (\text{state}, \text{action}) &\mapsto \text{state} \end{aligned}$$

Pendulum





Our Topics

In Part I

- Models and learning approaches
  Review of Bayesian learning, maximum likelihood estimation, Gaussian distributions
- Optimization
  Arguably the most widely used way to fit a model to data is by optimization.
- Latent variable models
  Latent variable models, free energies, the EM algorithm
- Markov and hidden Markov models
  A special class of latent variable models for sequence data.
- Sampling algorithms and MCMC
  Fitting and evaluating models by randomized simulation can be an alternative to optimization.

In Part II

- (Approximate) inference in graphical models.

BASICS

Representing a data source

Our approach to learning starts with a model of data production:

$$P(\text{data}|\text{parameters}) \qquad P(x|\theta) \text{ or } P(y|x,\theta)$$

Machine learning jargon also calls this a generative model.

In more detail

- We observe data $\mathcal{D} = (x_1, \dots, x_n)$ from a data source, in a sample space $\mathcal{X}$.
- Our mathematical description of the source is a probability distribution P on $\mathcal{X}$.
- A (statistical) model is a set $\mathcal{M}$ of probability distributions. "I assume model $\mathcal{M}$" means "I assume P is contained in $\mathcal{M}$".
- We typically index the elements of $\mathcal{M}$ by the elements of a parameter space $\mathcal{T}$. The model then takes the form
$$\mathcal{M} = \{P(\,\bullet\,|\theta) \,|\, \theta \in \mathcal{T}\}$$
- Example: $\mathcal{M} = \{\text{Gaussian distributions with variance 1 and mean } \theta \,|\, \theta \in \mathbb{R}\}$

Terminology varies, and the term "model" may refer to other mathematical objects.

Basic rules of probability

- Probabilities are non-negative $P(x) \geq 0 \ \forall x$.
- Probabilities normalise: $\sum_{x \in \mathcal{X}} P(x) = 1$ for distributions if x is a discrete variable and $\int_{\mathcal{X}} p(x)dx = 1$ for probability densities over continuous variables
- The joint probability of x and y is: $P(x, y)$.
- The marginal probability of x is: $P(x) = \sum_y P(x, y)$, assuming y is discrete.
- The conditional probability of x given y is: $P(x|y) = P(x, y)/P(y)$
- Bayes' Formula: Since $P(x, y) = P(x)P(y|x) = P(y)P(x|y)$, we have

$$P(y|x) = \frac{P(x|y)P(y)}{P(x)} \qquad \text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}$$

Warning: We will not be obsessively careful in our use of p and P for probability density and probability distribution. Should be obvious from context.

Independence

Recall that two random variables X_1 with (marginal) distribution P_1 and X_2 with distribution P_2 are independent if their joint distribution P factorizes,

$$P(x_1, x_2) = P_1(x_1)P_2(x_2) \quad \text{or equivalently} \quad P(x_1|x_2) = P_1(x_1) \; .$$

Informally:

Two random variables are independent if knowing the value of one does not provide any information about the outcome of the other.

Random variables are called iid (independent and identically distributed if they are independent and have identical marginal distributions: $P(x_1, \dots, x_n) = \prod_{i=1}^{n} P(x_i)$

Exponential Families

A model $\mathcal{M} = \{p(\,\bullet\,|\theta) \mid \theta \in \mathcal{T}\}$ is an exponential family if

$$p(x|\theta) = f(x)g(\theta)e^{\phi(\theta)^{\mathsf{T}}\mathbf{T}(x)} \qquad \text{for all } x \in \mathcal{X} \text{ and } \theta \in \mathcal{T}\,.$$

The components of the model are:

- A function $\mathbf{T} : \mathcal{X} \to \mathbb{R}^m$, for some dimension $m \in \mathbb{N}$. This function is the sufficient statistic of the model.
- A function $\phi : \mathcal{T} \to \mathbb{R}^m$. Its values $\phi(\theta)$ are called natural parameters.
- Functions $f : \mathcal{X} \to \mathbb{R}_{\geq 0}$ and $g : \mathcal{T} \to \mathbb{R}_{>0}$.

Note g is completely determined by $\mathbf{T}$, f and ϕ.

Relevance

- These are arguably the most common types of distributions in machine learning.
- They often arise as building blocks of more complicated models.
- The class of exponential families includes most elementary probability distributions. As we will see, it is also *very* convenient to work with.

EXAMPLES OF EXPONENTIAL FAMILIES

Model	Sample space	Sufficient statistic
Gaussian	$\mathbb{R}^d$	$\mathbf{T}(\mathbf{x}) = (\mathbf{x}\mathbf{x}^t, \mathbf{x})$
Gamma	$\mathbb{R}_+$	$\mathbf{T}(x) = (\ln(x), x)$
Poisson	$\mathbb{N}_0$	$\mathbf{T}(x) = x$
Multinomial	$\{1, \dots, K\}$	$\mathbf{T}(x) = x$
Wishart	Positive definite matrices	(requires more details)
Mallows	Rankings (permutations)	(requires more details)
Beta	$[0, 1]$	$\mathbf{T}(x) = (\ln(x), \ln(1 - x))$
Dirichlet	Probability distributions on d events	$\mathbf{T}(\mathbf{x}) = (\ln x_1, \dots, \ln x_d)$
Bernoulli	$\{0, 1\}$	$\mathbf{T}(x) = x$

A non-example: Student's t-distribution, and other heavy-tailed distributions.

Informally

- A given sample space $\mathcal{X}$ often has a "natural" statistic $\mathbf{T}$. (A statistic is a function of the data.)
- The exponential family defined by $\mathbf{T}$ is often the "natural" distribution for simple sources generating data in $\mathcal{X}$.

Learning parameters

We distinguish two approaches:

- **Point estimation** of a parameter θ assumes there is a true but unknown value θ_0. An estimate $\hat{\theta}$ is a function of observed data that approximates θ_0.
- **Bayesian inference** represents lack of knowledge as randomness; the parameter is a random variable Θ. This means we have to specify the distribution of Θ, called the prior distribution. We ask not for a true value of the parameter, but for its distribution $p(\theta|\mathcal{D})$ given the observed data, called the posterior distribution.
  (We assume that you are familiar with Bayes' equation and basics of Bayesian inference.)

Specific approaches we will discuss:

- **Maximum likelihood estimation** is the most common form of point estimation.
- Bayesian inference may compute a posterior distribution from data (e.g. **conjugate posteriors**), approximate complicated posteriors by simpler distributions (e.g. **variational inference**), or use **sampling algorithms** to generate random draws from the posterior, which are then processed further.
- The **MAP** estimate (**m**aximum **a** **p**osteriori estimate) is the parameter value most probable under the posterior,

  $$\theta^{\text{MAP}} = \arg\max P(\theta|\mathcal{D}) = \arg\max P(\theta)P(\mathcal{D}|\theta)\,.$$

  Although it is defined in terms of a posterior, it is a point estimate.

BASIC BAYESIAN LEARNING

Modeling assumption

- Model $\mathcal{M} = \{p(\bullet|\theta)|\theta \in \mathcal{T}\}$. Each element $p(\bullet|\theta)$ is the distribution of a *single* random element X of $\mathcal{X}$.
- As prior, we choose a distribution π on $\mathcal{T}$.
- We assume data is generated as follows: Generate

$$\begin{aligned} \Theta &\sim \pi \\ X_1, \dots, X_n|\Theta &\sim_{\text{iid}} p(\bullet|\Theta) . \end{aligned}$$

  and explain data $\mathcal{D} = \{x_1, \dots, x_n\}$ as the values assumed by $X_1, \dots, X_n$.
- $X_1, \dots, X_n$ are conditionally iid (iid = independently identically distributed).

The term Bayesian model often refers to the pair $(\mathcal{M}, \pi)$.

Posterior

The independence assumption implies $p(\mathcal{D}|\theta) = \prod_{i=1}^n p(x_i|\theta)$. Bayes' rule then takes the form

$$\pi(\theta|\mathcal{D}) = \frac{\prod_{i=1}^n p(x_i|\theta)}{\int_{\mathcal{T}} \prod_{i=1}^n p(x_i|\theta)\pi(\theta)d\theta}\pi(\theta) \propto \prod_{i=1}^n p(x_i|\theta)\pi(\theta)$$

Bayesian inference without the conditional i.i.d. assumption is also possible, but typically more complicated.

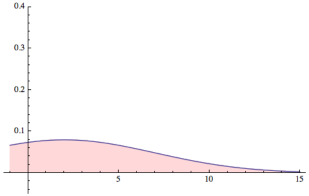
Example: Unknown Gaussian Mean

Model

- Observations are generated by a Gaussian, with unknown mean θ and fixed, known standard deviation σ.
- That means our model is $\mathcal{M} = \{p(\,\bullet\,|\theta,\sigma)|\theta \in \mathbb{R}\}$, where p is the Gaussian density on the real line.

Prior

- We choose a Gaussian prior $\pi(\,\bullet\,|\mu,\xi)$ with known mean μ and standard deviation ξ.
- In the figure, $\mu = 2$ and $\xi = 5$. Hence, we assume $\mu = 2$ is the most probable value of θ, and that $\theta \in [-3, 7]$ with a probability ~ 0.68.



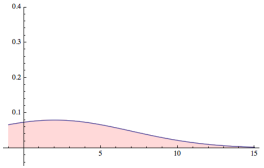
Posterior

Application of Bayes' formula shows the posterior is again a Gaussian,

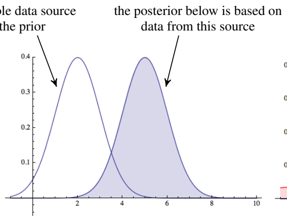
$$\pi(\theta|x_{1:n},\mu,\xi) = \pi(\theta|\mu_n,\xi_n) \quad \text{where} \quad \mu_n := \frac{\sigma^2\mu + \xi^2\sum_{i=1}^n x_i}{\sigma^2 + n\xi^2} \text{ and } \xi_n^2 := \frac{\sigma^2\xi^2}{\sigma^2 + n\xi^2}$$

Example: Unknown Gaussian Mean

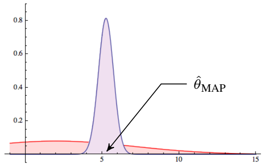
Model



prior

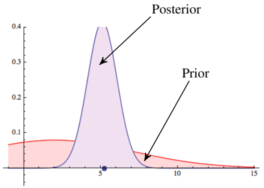


observation model

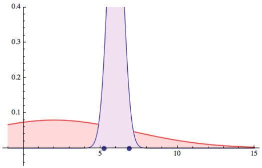


posterior

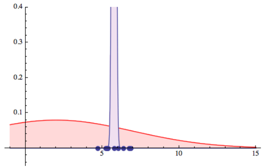
Posterior under increasing sample size



$n = 1$



$n = 2$



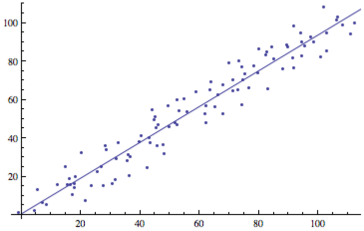
$n = 10$

A Slightly Different Perspective

Parameters

Intuitively, we can think of θ as the common pattern underlying the data:

$$P(X|\theta) = \text{Probability}(\text{data}|\text{pattern})$$



Inference idea

$$\text{data} \quad = \quad \text{underlying pattern} \quad + \quad \text{independent randomness}$$

Broadly speaking, the goal of inference is to extract the underlying pattern from the data.
Bayesian statistics models the pattern as a random quantity.

Bayesian learning and model selection

Choosing a model $\mathcal{M}$ from a set $\mathcal{M}_1, \mathcal{M}_2, \dots$ of candidate models is called model selection. Bayesian inference can be extended to model selection.

- Problem specification:

  Models: $\mathcal{M}_i = \{P(x|\theta_i, \mathcal{M}_i) | \theta_i \in \mathcal{T}_i\}$.
  Prior probability of models: $P(\mathcal{M}_i)$.
  Prior probabilities of model parameters: $P(\theta_i|\mathcal{M}_i)$

- Data probability (likelihood)

$$P(\mathcal{D}|\theta_i, \mathcal{M}_i) = \prod_{j=1}^{n} P(x_j|\theta_i, \mathcal{M}_i) \equiv \mathcal{L}(\theta_i)$$

  Note we are assuming the data is conditionally i.i.d. given the model and parameter.

- Parameter learning (posterior):

$$P(\theta_i|\mathcal{D}, \mathcal{M}_i) = \frac{P(\mathcal{D}|\theta_i, \mathcal{M}_i)P(\theta_i|\mathcal{M}_i)}{P(\mathcal{D}|\mathcal{M}_i)}; \quad P(\mathcal{D}|\mathcal{M}_i) = \int d\theta_i \, P(\mathcal{D}|\theta_i, \mathcal{M}_i)P(\theta_i|\mathcal{M}_i)$$

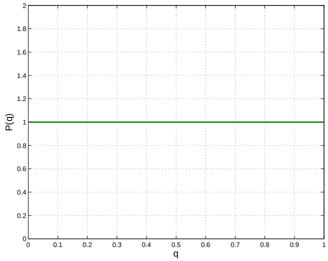
  $P(\mathcal{D}|\mathcal{M}_i)$ is called the marginal likelihood or evidence for $\mathcal{M}_i$. It is proportional to the posterior probability model $\mathcal{M}_i$ being the one that generated the data.

- Model selection:

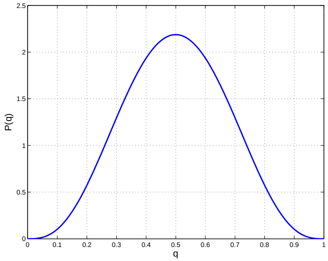
$$P(\mathcal{M}_i|\mathcal{D}) = \frac{P(\mathcal{D}|\mathcal{M}_i)P(\mathcal{M}_i)}{P(\mathcal{D})}$$

BAYESIAN LEARNING: A COIN TOSS EXAMPLE

Coin toss: One parameter q — the probability of obtaining *heads*
So our space of models is the set of distributions over $q \in [0,1]$.
Learner A believes (Bayesian) model $\mathcal{M}_{\text{A}}$: all values of q are equally plausible;
Learner B believes (Bayesian) model $\mathcal{M}_{\text{B}}$: “fair” coin ($q \approx 0.5$) more plausible than “biased”.



A: $\alpha_1 = \alpha_2 = 1.0$



B: $\alpha_1 = \alpha_2 = 4.0$

Both prior beliefs can be described by the Beta distribution:

$$p(q|\alpha_1,\alpha_2) = \frac{q^{(\alpha_1-1)}(1-q)^{(\alpha_2-1)}}{B(\alpha_1,\alpha_2)} = \text{Beta}(q|\alpha_1,\alpha_2)$$

Bayesian learning: The coin toss (cont)

Now we observe a toss. Two possible outcomes:

$$p(\mathrm{H}|q) = q \qquad\qquad p(\mathrm{T}|q) = 1 - q$$

Suppose our single coin toss comes out *heads*

The probability of the observed data (likelihood) is:

$$p(\mathrm{H}|q) = q$$

Using Bayes' formula, we multiply the prior, $p(q)$ by the likelihood and renormalise to get the posterior probability:

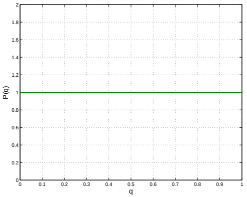
$$\begin{aligned} p(q|\mathrm{H}) &= \frac{p(q)p(\mathrm{H}|q)}{p(\mathrm{H})} \propto q\, \mathsf{Beta}(q|\alpha_1, \alpha_2) \\ &\propto q\, q^{(\alpha_1-1)}(1-q)^{(\alpha_2-1)} = \mathsf{Beta}(q|\alpha_1+1, \alpha_2) \end{aligned}$$

Bayesian learning: The coin toss (cont)

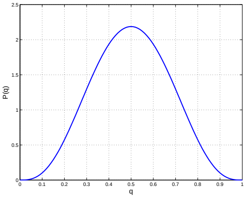
A

B

Prior

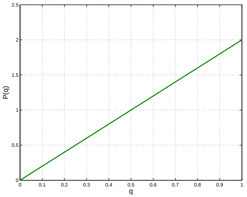


$\text{Beta}(q|1,1)$

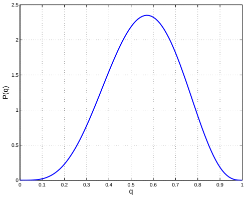


$\text{Beta}(q|4,4)$

Posterior



$\text{Beta}(q|2,1)$



$\text{Beta}(q|5,4)$

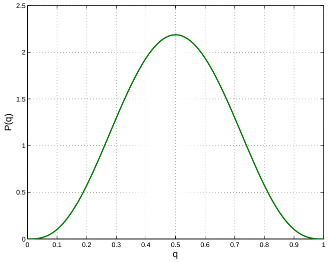
Bayesian learning: The coin toss (cont)

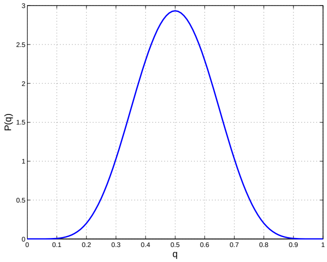
What about multiple tosses? Suppose we observe $\mathcal{D} = \{\,\mathsf{H\ H\ T\ H\ T\ T}\,\}$:

$$p(\{\,\mathsf{H\ H\ T\ H\ T\ T}\,\}|q) = qq(1-q)q(1-q)(1-q) = q^3(1-q)^3$$

This is still straightforward:

$$\begin{aligned} p(q|\mathcal{D}) &= \frac{p(q)p(\mathcal{D}|q)}{p(\mathcal{D})} \propto q^3(1-q)^3\ \mathsf{Beta}(q|\alpha_1,\alpha_2) \\ &\propto \mathsf{Beta}(q|\alpha_1+3,\alpha_2+3) \end{aligned}$$





Conjugate priors

Updating the prior to form the posterior was particularly easy in these examples. This is because we used a conjugate prior for an exponential family likelihood.
Recall that exponential family distributions take the form:

$$P(x|\theta) \;=\; g(\theta)f(x)e^{\boldsymbol{\phi}(\theta)^{\mathsf{T}}\mathbf{T}(x)}$$

with $g(\theta)$ the normalising constant. Given n conditionally iid observations,

$$P(\{x_i\}|\theta) \;=\; \prod_i P(x_i|\theta) \;=\; g(\theta)^n e^{\boldsymbol{\phi}(\theta)^{\mathsf{T}}\left(\sum_i \mathbf{T}(x_i)\right)} \prod_i f(x_i)$$

A conjugate prior for an exponentially family is of the form

$$P(\theta) \;=\; F(\boldsymbol{\tau},\nu)g(\theta)^{\nu}e^{\boldsymbol{\phi}(\theta)^{\mathsf{T}}\boldsymbol{\tau}} \;,$$

with normaliser $F(\boldsymbol{\tau},\nu)$. The posterior is then

$$P(\theta|\{x_i\}) \propto \; P(\{x_i\}|\theta)P(\theta) \;\propto\; g(\theta)^{\nu+n}e^{\boldsymbol{\phi}(\theta)^{\mathsf{T}}\left(\boldsymbol{\tau}+\sum_i \mathbf{T}(x_i)\right)}$$

with the normaliser given by $F\big(\boldsymbol{\tau}+\sum_i \mathbf{T}(x_i), \nu+n\big)$.

CONJUGATE PRIORS

The posterior given an exponential family likelihood and conjugate prior is:

$$P(\theta|\{x_i\}) = F\big(\boldsymbol{\tau} + \sum_i \mathsf{T}(x_i), \nu + n\big) g(\theta)^{\nu+n} \exp\Big[\boldsymbol{\phi}(\theta)^{\mathsf{T}}\Big(\boldsymbol{\tau} + \sum_i \mathsf{T}(x_i)\Big)\Big]$$

Here,

$\boldsymbol{\tau} \in \mathcal{T}$	specifies the expected value of the prior.
$\nu \in \mathbb{R}_{>0}$	is the concentration or scale of the prior (roughly: larger scale $\rightarrow$ smaller variance).

As new data come in, each one increments the sufficient statistics vector and the scale to define the posterior.

Interpretation

- The posterior effectively interpolates between the prior assumption $\boldsymbol{\tau}$ and the observational evidence $\sum_i \mathsf{T}(x_i)$.
- The concentration ν specifies how much weight we assign to the prior belief $\boldsymbol{\tau}$. (Large ν $\rightarrow$ strong prior assumption.)
- Some authors interpret $\boldsymbol{\tau}$ as $\boldsymbol{\tau} = \sum \mathsf{T}(x_i')$ for "fictitious observations" or "pseudo-observations" $x_1', \ldots, x_\nu'$. (If ν is an integer, which it need not be.)

Conjugacy in the coin flip

Distributions are not always written in their natural exponential form.

The Bernoulli distribution (a single coin flip) with parameter q and observation $x \in \{0, 1\}$, can be written:

$$\begin{aligned} P(x|q) &= q^x(1-q)^{(1-x)} \\ &= e^{x \log q + (1-x) \log(1-q)} \\ &= e^{\log(1-q) + x \log(q/(1-q))} = (1-q)e^{\log(q/(1-q))x} \end{aligned}$$

So the natural parameter is the log odds $\log(q/(1-q))$, and the sufficient stats (for multiple tosses) is the number of heads. The conjugate prior is

$$\begin{aligned} P(q) &= F(\tau,\nu)\ (1-q)^{\nu} e^{\log(q/(1-q))\tau} \\ &= F(\tau,\nu)\ (1-q)^{\nu} e^{\tau \log q - \tau \log(1-q)} = F(\tau,\nu)\ (1-q)^{\nu-\tau} q^{\tau} \end{aligned}$$

which has the form of the Beta distribution $\Rightarrow F(\tau,\nu) = 1/B(\tau+1, \nu-\tau+1)$.

In general, then, the posterior will be $P(q|\{x_i\}) = \mathsf{Beta}(\alpha_1, \alpha_2)$, with

$$\alpha_1 = 1 + \tau + \textstyle\sum_i x_i \qquad\qquad \alpha_2 = 1 + (\nu + n) - \left(\tau + \textstyle\sum_i x_i\right)$$

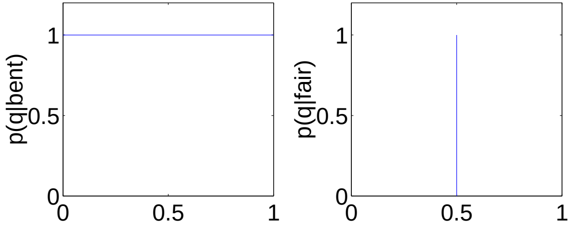
If we observe a head, we add 1 to the sufficient statistic $\sum x_i$, and also 1 to the count n. This increments α_1. If we observe a tail we add 1 to n, but not to $\sum x_i$, incrementing α_2.

Bayesian coins – comparing models

We have seen how to update posteriors within each model. To study the choice of model, consider two more extreme models: "fair" and "bent". A priori, we may think that "fair" is more probable, eg:

$$p(\text{fair}) = 0.8, \qquad p(\text{bent}) = 0.2$$

For the bent coin, we assume all parameter values are equally likely, whilst the fair coin has a fixed probability:



We make 10 tosses, and get: $\mathcal{D} = (\text{T H T H T T T T T T})$.

Bayesian coins – comparing models

Which model should we prefer *a posteriori* (i.e. after seeing the data)?

The evidence for the fair model is:

$$P(\mathcal{D}|\text{fair}) \;=\; (1/2)^{10} \approx 0.001$$

and for the bent model is:

$$P(\mathcal{D}|\text{bent}) \;=\; \int dq\; P(\mathcal{D}|q,\text{bent})p(q|\text{bent}) \;=\; \int dq\; q^2(1-q)^8 \;=\mathrm{B}(3,9) \approx 0.002$$

Thus, the posterior for the models, by Bayes' formula:

$$P(\text{fair}|\mathcal{D}) \propto 0.0008, \qquad P(\text{bent}|\mathcal{D}) \propto 0.0004,$$

ie, a two-thirds probability that the coin is fair.

How do we make predictions? Could choose the fair model (model selection).
Or could weight the predictions from each model by their probability (model averaging).
Probability of H at next toss is:

$$P(\text{H}|\mathcal{D}) = P(\text{H}|\mathcal{D},\text{fair})P(\text{fair}|\mathcal{D}) + P(\text{H}|\mathcal{D},\text{bent})P(\text{bent}|\mathcal{D}) = \frac{1}{2}\times\frac{2}{3}+\frac{3}{12}\times\frac{1}{3} = \frac{5}{12}.$$

MAXIMUM LIKELIHOOD ESTIMATION

Problem specification

- Data $\mathcal{D} = \{x_1, \dots, x_n\}$ is generated iid from model $\mathcal{M} = \{p(x|\theta) \mid \theta \in \mathcal{T}\}$.
  We assume iid rather than conditionally iid since the parameter is not assumed to be random.
- Objective: Find the distribution in $\mathcal{M}$ that best explains the data.
  That means we have to identify a "best" parameter value $\hat{\theta}$.

Maximum Likelihood approach

The maximum likelihood estimator (MLE) is defined as

$$\hat{\theta}_{\text{ML}} := \arg\max_{\theta \in \mathcal{T}} p(x_1, \dots, x_n|\theta) \ .$$

We hence assume the data is best explained by that distribution in $\mathcal{M}$ under which it is most likely to occur (has the highest probability or density value).

If differentiability holds, $\hat{\theta}_{\text{ML}}$ is the solution of the maximum likelihood equation

$$\nabla_\theta p(\mathcal{D}|\theta) \ = \ \nabla_\theta \left(\prod_{i=1}^{n} p(x_i|\theta) \right) \ = \ 0$$

Logarithm Trick

Instead of the likelihood $p(\mathcal{D}|\theta)$, we often work with the log-likelihood $\log p(\mathcal{D}|\theta)$.

Recall two properties of the logarithm function:

- It turns products into sums: $\log\left(\prod_i f_i\right) = \sum_i \log(f_i)$
- Since it is monotonically increasing on $\mathbb{R}_+$, it does not change the *location* of maxima and minima:

$$\max_y \log(g(y)) \neq \max_y g(y) \qquad \text{The } \textit{value} \text{ changes.}$$

$$\arg\max_y \log(g(y)) = \arg\max_y g(y) \qquad \text{The } \textit{location} \text{ does not change.}$$

MLE with the log-likelihood: Since

$$\hat{\theta}_{\text{ML}} = \arg\max_\theta \prod_{i=1}^n p(x_i|\theta) = \arg\max_\theta \log\left(\prod_{i=1}^n p(x_i|\theta)\right) = \arg\max_\theta \sum_{i=1}^n \log p(x_i|\theta) \;,$$

the MLE $\hat{\theta}_{\text{ML}}$ is the solution of

$$\sum_{i=1}^n \nabla_\theta \log p(x_i|\theta) \;=\; 0$$

BACKGROUND: LAW OF LARGE NUMBERS

For iid random variables $X_1, X_2, \ldots$ and any function f with $\mathbb{E}[|f(X_1)|] \leq \infty$,

$$\frac{1}{n}\sum_{i=1}^{n} f(X_i) \xrightarrow{n\to\infty} \mathbb{E}[f(X_1)] \qquad \text{with probability } 1\ .$$

In short: We know how to estimate expectations.

Consequences for parameter estimation

Idea: Reduce parameter estimation to the problem of estimating an expectation.

- Suppose θ_0 is the parameter value that has generated the data.
- Find some function g such that

$$\mathbb{E}_{P_{\theta_0}}[g(X,\theta)] = 0 \quad \text{if and only if } \theta = \theta_0$$

- By the law of large numbers, we can then estimate θ_0 as the solution of

$$\frac{1}{n}\sum_{i=1}^{n} g(X_i,\theta) = 0$$

MLE DERIVED FROM THE LAW OF LARNGE NUMBERS

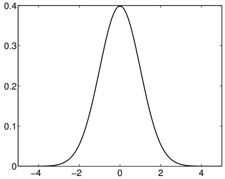
- It can be shown (under some conditions on the model) that the function
$$g(x,\theta) := \nabla_\theta \log p(x,\theta)$$
satsifies the property above. That is,
$$\mathbb{E}_{P_{\theta_0}}[\nabla_\theta \log p(X,\theta)] = 0 \quad \text{if and only if } \theta = \theta_0$$
- The function $\theta \mapsto \nabla_\theta \log p(x,\theta)$ is called the score function, or the Fisher score.
- Substituting the law of large numbers estimate for the expectation gives
$$\frac{1}{n}\sum_{i=1}^{n} \nabla_\theta \log p(X_i,\theta) = 0 \; .$$
The solution of the equation is exactly the maximum likelihood estimator.

TOOLS: GAUSSIAN DISTRIBUTIONS

Gaussian Distribution

Gaussian density in one dimension

$$p(x;\mu,\sigma) := \frac{1}{\sqrt{2\pi}\sigma}\exp\Big(-\frac{(x-\mu)^2}{2\sigma^2}\Big)$$



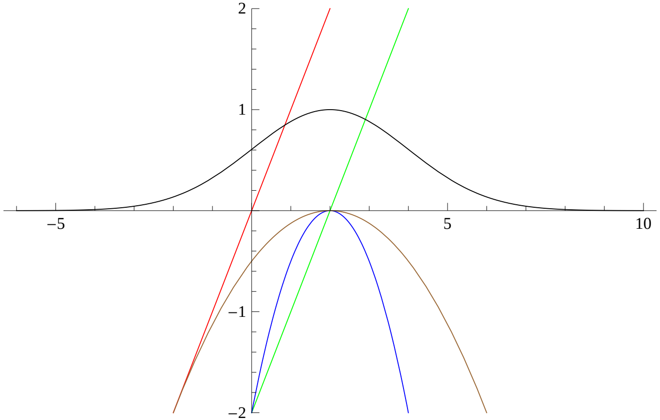
- μ = expected value of x, σ^2 = variance, σ = standard deviation
- The quotient $\frac{x-\mu}{\sigma}$ measures deviation of x from its expected value in units of σ (i.e. σ defines the length scale)

Recall: Standard deviation around the mean

- Recall that the interval $[\mu-\sigma,\mu+\sigma]$ ("one standard deviation") always contains the same amount of probability mass (ca. 68.27%), regardless of the choice of μ and σ.
- Similarly, the intervall $[\mu-2\sigma,\mu+2\sigma]$ contains $\sim 95.45\%$ of the mass, and $[\mu-3\sigma,\mu+3\sigma]$ contains $\sim 99.73\%$.

COMPONENTS OF A 1D GAUSSIAN

$\mu = 2, \sigma = 2$



- Red: $x \mapsto x$
- Green: $x \mapsto x - \mu$
- Blue: $x \mapsto -\frac{1}{2}(x-\mu)^2$
- Brown: $x \mapsto -\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2$
- Black: $x \mapsto \exp\left(-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right)$

Covariance Matrices

Recall: Covariance

The covariance of two random variables X_1, X_2 is

$$\mathrm{Cov}[X_1, X_2] = \mathbb{E}[(X_1 - \mathbb{E}[X_1])(X_2 - \mathbb{E}[X_2])] \; .$$

If $X_1 = X_2$, the covariance is the variance: $\mathrm{Cov}[X, X] = \mathrm{Var}[X]$.

Covariance matrix

If $X = (X_1, \ldots, X_m)$ is a random vector with values in $\mathbb{R}^m$, the matrix of all covariances

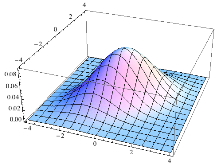
$$\mathrm{Cov}[X] := (\mathrm{Cov}[X_i, X_j])_{i,j} = \begin{pmatrix} \mathrm{Cov}[X_1, X_1] & \cdots & \mathrm{Cov}[X_1, X_m] \\ \vdots & & \vdots \\ \mathrm{Cov}[X_m, X_1] & \cdots & \mathrm{Cov}[X_m, X_m] \end{pmatrix}$$

is called the **covariance matrix** of X.

Notation

It is customary to denote the covariance matrix $\mathrm{Cov}[X]$ by Σ.

GAUSSIAN IN MULTIPLE DIMENSIONS



Gaussian density in m dimensions

The quadratic function

$$-\frac{(x-\mu)^2}{2\sigma^2} \quad = \quad -\frac{1}{2}(x-\mu)(\sigma^2)^{-1}(x-\mu)$$

is replaced by a quadratic form:

$$p(\mathbf{x};\boldsymbol{\mu},\Sigma) := \frac{1}{\sqrt{(2\pi)^d\,|\Sigma|}}\exp\Big(-\tfrac{1}{2}\Big\langle(\mathbf{x}-\boldsymbol{\mu}),\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\Big\rangle\Big)$$

for a positive definite matrix Σ.

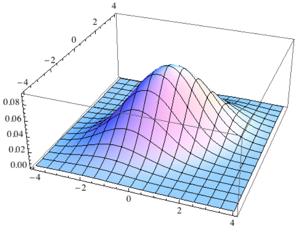
Covariance matrix of a Gaussian

If a random vector $X \in \mathbb{R}^m$ has Gaussian distribution with density $p(\mathbf{x};\mu,\Sigma)$, its covariance matrix is $\mathrm{Cov}[X] = \Sigma$. In other words, a Gaussian is parameterized by its covariance.

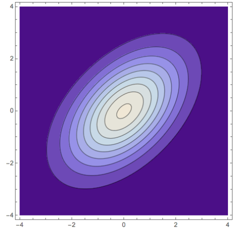
Assuming a multivariate Gaussian model means we assume that all stochastic dependence between dimensions is captured by the covariance.

GAUSSIAN DENSITY: EXAMPLE

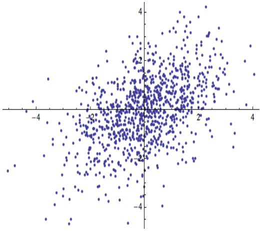
$$p(\mathbf{x};\boldsymbol{\mu},\Sigma) \quad \text{with} \quad \boldsymbol{\mu}=(0,0) \quad \text{and} \quad \Sigma=\begin{pmatrix}2 & 1\\ 1 & 2\end{pmatrix}$$



density

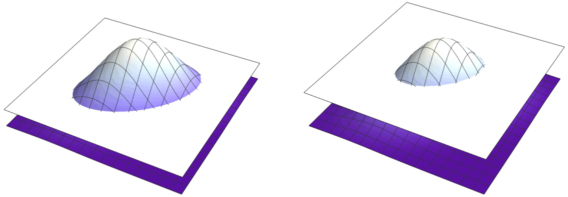


contour lines

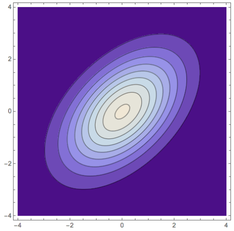


1000 sample points

CONTOUR LINES



Intersect density with a horizontal plane, draw intersection as a curve, and project it down onto the plane.



Each elliptical line is such a contour, for planes at different heights.

Contours and standard deviation

- Each ellipse consists of all points $\mathbf{x} \in \mathbb{R}^2$ that satisfy the equation

$$\left\langle \mathbf{x}, \Sigma^{-1}\mathbf{x} \right\rangle = c \qquad \text{for some fixed } c > 0 \; .$$

  Changing c changes the size of the ellipse.

- The ellipses play the same role as intervals around the mean for 1D Gaussians: The ellipse with $\langle \mathbf{x}, \Sigma^{-1}\mathbf{x} \rangle = 1$ contains $\sim 68.27\%$ of the probability mass, etc.
- That is: The area within the ellipse given by $\langle \mathbf{x}, \Sigma^{-1}\mathbf{x} \rangle = k$ corresponds to k standard deviations.

TOOLS:
SPECTRA AND GEOMETRY OF GAUSSIANS

EIGENVALUES

The properties of covariance matrices are summarized by their spectral properties (their eigenvalues and eigenvectors). That makes spectral properties the key to understanding Gaussian distributions.

Recall the definition

We consider a square matrix $A \in \mathbb{R}^{m \times m}$.

A vector $\xi \in \mathbb{R}^m$ is an eigenvector of A if there is a scalar λ such that

$$A\xi = \lambda\xi \ .$$

λ is called an eigenvalue of A for the eigenvector ξ.

In words: The direction of ξ does not change under application of A. Only its length changes, by a factor λ.

The set of eigenvalues of a matrix is called its spectrum.

Properties

- In general, eigenvalues are complex numbers $\lambda \in \mathbb{C}$.
- The class of matrices with the nicest eigen-structure are symmetric matrices, for which all eigenvalues are real and the eigenvectors are mutually orthogonal.

Eigenstructure of symmetric matrices

If a matrix is symmetric:

- There are $\text{rank}(A)$ distinct eigendirections.
- The eigenvectors are pair-wise orthogonal.
- If $\text{rank}(A) = m$, the eigenvectors form an orthogonal basis of $\mathbb{R}^m$. We can normalise each eigenvector (which again produces an eigenvector) and obtain an ortho*normal* basis of $\mathbb{R}^m$.

Definiteness

type	if ...
positive definite	all eigenvalues > 0
positive semi-definite	all eigenvalues ≥ 0
negative semi-definite	all eigenvalues ≤ 0
negative definite	all eigenvalues < 0
indefinite	none of the above

Orthonormal Bases

Recall that a basis $\{v_1, \dots, v_m\}$ of $\mathbb{R}^m$ is called an orthonormal basis (ONB) if

$$\langle v_i, v_j \rangle = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

That is: The v_i are pairwise orthogonal and each of length 1.

Orthogonal matrices

A matrix is orthogonal precisely if its rows form an ONB. Any two ONBs can be transformed into each other by an orthogonal matrix.

Transforming between ONBs

If $\mathcal{V} = \{v_1, \dots, v_m\}$ and $\mathcal{W} = \{w_1, \dots, w_m\}$ are ONBs, there is an orthogonal matrix O such that

$$A_{[\mathcal{V}]} = O A_{[\mathcal{W}]} O^{-1}$$

for any matrix A. By $A_{[\mathcal{V}]}$, we denote the representation of A in $\mathcal{V}$.

Eigenvector ONB

Setting

- Suppose A symmetric, $\xi_1, \dots, \xi_m$ are eigenvectors and form an ONB.
- $\lambda_1, \dots, \lambda_m$ are the corresponding eigenvalues.

How does A act on a vector $v \in \mathbb{R}^m$?

1. Represent v in basis $\xi_1, \dots, \xi_m$:
$$v = \sum_{j=1}^{m} v_j^{\mathrm{A}} \xi_j \qquad \text{where } v_j^{\mathrm{A}} \in \mathbb{R}$$
2. Multiply by A: Eigenvector definition (recall: $A\xi_j = \lambda \xi_j$) yields
$$Av = A\Big(\sum_{j=1}^{m} v_j^{\mathrm{A}} \xi_j\Big) = \sum_{j=1}^{m} v_j^{\mathrm{A}} A\xi_j = \sum_{j=1}^{m} v_j^{\mathrm{A}} \lambda_j \xi_j$$

A symmetric matrix acts by scaling along the directions of its eigenvectors.

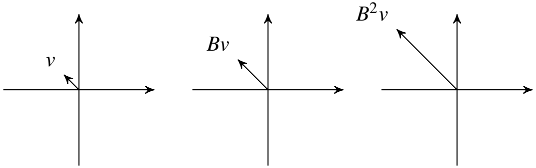
ILLUSTRATION

Suppose we *repeatedly* apply a symmetric matrix B to a vector $v \in \mathbb{R}^m$: We compute

$$Bv, \quad B(Bv) = B^2 v, \quad B(B(Bv))) = B^3 v, \quad \dots$$

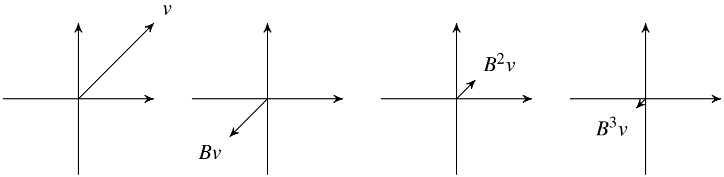
How does v change?

Example 1: v is an eigenvector with eigenvalue 2



The direction of v does not change, but its length doubles with each application of B.

Example 2: v is an eigenvector with eigenvalue $-\frac{1}{2}$



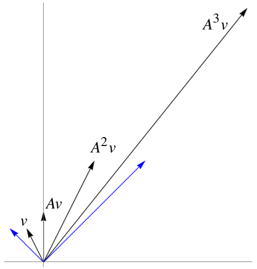
EFFECT OF A SYMMETRIC MATRIX

If v is an arbitrary vector, we can represent it as a linear combination $v = \sum_j v_j^{\mathrm{B}} \xi_j$ of eigenvectors of B, and obtain

$$B^n v = \sum_{j=1}^{m} v_j^{\mathrm{B}} \lambda_j^n \xi_j$$

The weight λ_j^n grows most rapidly for eigenvalue with largest absolute value.

The direction of $B^n v$ converges to the direction of the eigenvector with largest eigenvalue as n grows large.



$$A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix} \quad v = \begin{pmatrix} -\frac{1}{2} \\ 1 \end{pmatrix}$$

eigenvectors shown in blue
length proportional to eigenvalue

Quadratic Forms

In applications, symmetric matrices often occur in quadratic forms.

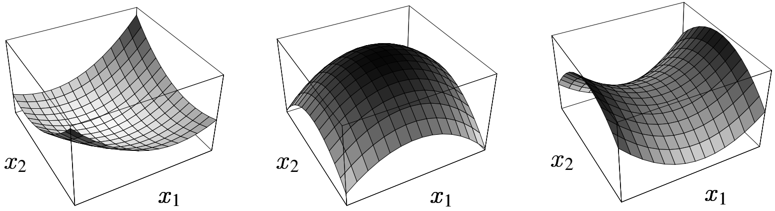
Definition

The quadratic form defined by a symmetric matrix A is the function

$$\begin{aligned} q_A : \mathbb{R}^m &\to \mathbb{R} \\ x &\mapsto \langle x, Ax \rangle \end{aligned}$$

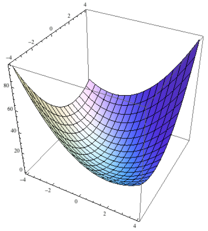
Intuition

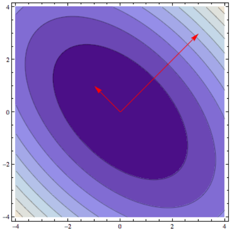
A quadratic form is the m-dimensional analogue of a quadratic function ax^2, with a vector substituted for the scalar x and the matrix A substituted for the scalar $a \in \mathbb{R}$.



Quadratic Forms

Here is the quadratic form for the matrix $A = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$:



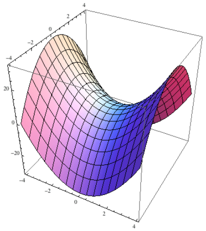


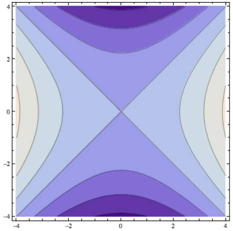
- Left: The function value q_A is graphed on the vertical axis.
- Right: Contours. Each line in $\mathbb{R}^2$ corresponds to a constant function value of q_A.
  Dark color = small values.
- The red lines are eigenvector directions of A. Their lengths represent the (absolute) values of the eigenvalues.
- In this case, both eigenvalues are positive. If all eigenvalues are positive, the contours are ellipses. So:

$$\text{positive/negative definite matrices} \quad \leftrightarrow \quad \text{elliptic quadratic forms}$$

QUADRATIC FORMS

In this plot, the eigenvectors are axis-parallel, and one eigenvalue is negative:





The matrix here is $A = \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}$.

Intuition

- If we change the sign of one of the eigenvalue, the quadratic function along the corresponding eigen-axis flips.
- There is a point which is a minimum of the function along one axis direction, and a maximum along the other. Such a point is called a *saddle point*.

APPLICATION: COVARIANCE MATRIX

Recall: Covariance

The covariance of two random variables X_1, X_2 is

$$\mathrm{Cov}[X_1, X_2] = \mathbb{E}[(X_1 - \mathbb{E}[X_1])(X_2 - \mathbb{E}[X_2])] \ .$$

If $X_1 = X_2$, the covariance is the variance: $\mathrm{Cov}[X, X] = \mathrm{Var}[X]$.

Covariance matrix

If $X = (X_1, \dots, X_m)$ is a random vector with values in $\mathbb{R}^m$, the matrix of all covariances

$$\mathrm{Cov}[X] := (\mathrm{Cov}[X_i, X_j])_{i,j} = \begin{pmatrix} \mathrm{Cov}[X_1, X_1] & \cdots & \mathrm{Cov}[X_1, X_m] \\ \vdots & & \vdots \\ \mathrm{Cov}[X_m, X_1] & \cdots & \mathrm{Cov}[X_m, X_m] \end{pmatrix}$$

is called the **covariance matrix** of X.

Notation

It is customary to denote the covariance matrix $\mathrm{Cov}[X]$ by Σ.

GEOMETRY OF GAUSSIANS

Covariance matrix of a Gaussian

If a random vector $X \in \mathbb{R}^m$ has Gaussian distribution with density $p(\mathbf{x}; \mu, \Sigma)$, its covariance matrix is $\text{Cov}[X] = \Sigma$. In other words, a Gaussian is parameterized by its covariance.

Observation

Since $\text{Cov}[X_i, X_j] = \text{Cov}[X_j, X_i]$, the covariance matrix is symmetric.

What is the eigenstructure of Σ?

- We know: Σ symmetric $\Rightarrow$ there is an eigenvector ONB
- Call the eigenvectors in this ONB $\xi_1, \dots, \xi_m$ and their eigenvalues $\lambda_1, \dots, \lambda_m$
- We can rotate the coordinate system to $\xi_1, \dots, \xi_m$. In the new coordinate system, Σ has the form

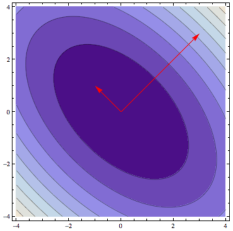
$$\Sigma_{[\xi_1,\dots,\xi_n]} = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_m \end{pmatrix} = \text{diag}(\lambda_1, \dots, \lambda_m)$$

Example

Quadratic form

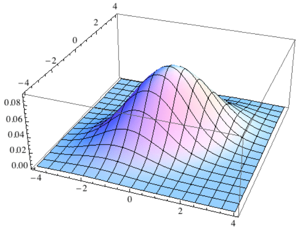
$$\langle \mathbf{x}, \Sigma\mathbf{x} \rangle \quad \text{with} \quad \Sigma = \begin{pmatrix} 2 & 1 \\ 1 & 2 \end{pmatrix}$$

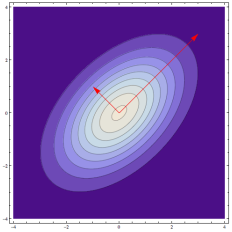
The eigenvectors are $(1,1)$ and $(-1,1)$ with eigenvalues 3 and 1.

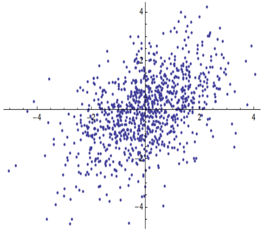


Gaussian density

$p(\mathbf{x}; \boldsymbol{\mu}, \Sigma)$ with $\boldsymbol{\mu} = (0,0)$.


Density graph


Density contour


1000 sample points

INTERPRETATION

The ξ_i as random variables

Write $e_1, \ldots, e_m$ for the ONB of axis vectors. We can represent each ξ_i as

$$\xi_i = \sum_{j=1}^{m} \alpha_{ij} e_j$$

Then $O = (\alpha_{ij})$ is the orthogonal transformation matrix between the two bases.
We can represent random vector $X \in \mathbb{R}^m$ sampled from the Gaussian in the eigen-ONB as

$$X_{[\xi_1,\ldots,\xi_m]} = (X'_1, \ldots, X'_m) \qquad \text{with} \qquad X'_i = \sum_{j=1}^{m} \alpha_{ij} X_j$$

Since the X_j are random variables (and the α_{ij} are fixed), each X'_i is a scalar random variable.

INTERPRETATION

Meaning of the random variables ξ_i

For any Gaussian $p(\mathbf{x}; \boldsymbol{\mu}, \Sigma)$, we can

1. shift the origin of the coordinate system into $\boldsymbol{\mu}$
2. rotate the coordinate system to the eigen-ONB of Σ.

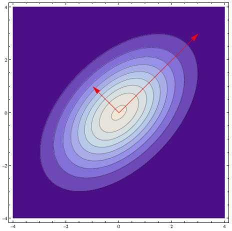
In this new coordinate system, the Gaussian has covariance matrix

$$\Sigma_{[\xi_1,\dots,\xi_m]} = \mathrm{diag}(\lambda_1, \dots, \lambda_m)$$

where λ_i are the eigenvalues of Σ.

Gaussian in the new coordinates

A Gaussian vector $X_{[\xi_1,\dots,\xi_m]}$ represented in the new coordinates consists of *m independent* 1D Gaussian variables X'_i. Each X'_i has mean 0 and variance λ_i.



A multidimensional Gaussian consists of independent, orthogonal scalar normal variables in *some* coordinate system.

MLE for Gaussians

Consider data $\mathcal{D} = (\mathbf{x}_1, \dots, \mathbf{x}_n)$ in $\mathbb{R}^d$, and the Gaussian model

$$\mathcal{M} = \{ g(\,\bullet\,|\mu, \Sigma) \mid \mu \in \mathbb{R}^d, \Sigma \in \mathbb{R}^{d \times d} \text{ positive definite } \}$$

where g denotes the Gaussian density on $\mathbb{R}^d$.

Ansatz

ML estimation treats the two parameters separately. I will write θ to denote either $\theta = \mu$ (for fixed Σ), or $\theta = \Sigma$ (for fixed μ), depending on which parameter we estimate.

To find the maximum likelihood estimator, we have to solve

$$0 \; = \; \nabla_\theta \log \prod_{i=1}^{n} g(\mathbf{x}_i|\theta) \; = \; \nabla_\theta \sum_{i=1}^{n} \log\Big(\frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \Big) - \frac{1}{2} (\mathbf{x_i} - \mu)^{\mathsf{T}} \Sigma^{-1} (\mathbf{x}_i - \mu))$$

Maximum likelihood estimators

The solutions are

$$\hat{\mu}_{\text{ML}} := \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i \quad \text{and} \quad \hat{\Sigma}_{\text{ML}} := \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^{\mathsf{T}} \; .$$

Note $\hat{\mu}$ does not depend on Σ. We can estimate $\hat{\mu}$ first, then plug in $\hat{\mu}$ for μ in $\hat{\Sigma}_{\text{ML}}$.

Refresher – Matrix derivatives of scalar forms

We will use the following facts:

$$\mathbf{x}^{\mathsf{T}}A\mathbf{y} = \mathbf{y}^{\mathsf{T}}A^{\mathsf{T}}\mathbf{x} = \mathsf{Tr}\left[\mathbf{x}^{\mathsf{T}}A\mathbf{y}\right] \quad \text{(scalars equal their own transpose and trace)}$$

$$\mathsf{Tr}\,[A] = \mathsf{Tr}\left[A^{\mathsf{T}}\right] \qquad \mathsf{Tr}\,[ABC] = \mathsf{Tr}\,[CAB] = \mathsf{Tr}\,[BCA]$$

$$\frac{\partial}{\partial A_{ij}}\mathsf{Tr}\left[A^{\mathsf{T}}B\right] = \frac{\partial}{\partial A_{ij}}\sum_n [A^{\mathsf{T}}B]_{nn} = \frac{\partial}{\partial A_{ij}}\sum_n\sum_m A^{\mathsf{T}}_{nm}B_{mn} = \frac{\partial}{\partial A_{ij}}\sum_{mn}A_{mn}B_{mn} = B_{ij}$$

$$\Rightarrow \frac{\partial}{\partial A}\mathsf{Tr}\left[A^{\mathsf{T}}B\right] = B$$

$$\begin{aligned}
\frac{\partial}{\partial A}\mathsf{Tr}\left[A^{\mathsf{T}}BAC\right] &= \frac{\partial}{\partial A}\mathsf{Tr}\left[F_1(A)^{\mathsf{T}}BF_2(A)C\right] \text{ with } F_1 \text{ and } F_2 \text{ both identity maps} \\
&= \frac{\partial}{\partial F_1}\mathsf{Tr}\left[F_1^{\mathsf{T}}BF_2C\right]\frac{\partial F_1}{\partial A} + \frac{\partial}{\partial F_2}\mathsf{Tr}\left[F_1^{\mathsf{T}}BF_2C\right]\frac{\partial F_2}{\partial A} \\
&= \frac{\partial}{\partial F_1}\mathsf{Tr}\left[F_1^{\mathsf{T}}BF_2C\right]\frac{\partial F_1}{\partial A} + \frac{\partial}{\partial F_2}\mathsf{Tr}\left[CF_1^{\mathsf{T}}BF_2\right]\frac{\partial F_2}{\partial A} \\
&= \frac{\partial}{\partial F_1}\mathsf{Tr}\left[F_1^{\mathsf{T}}BF_2C\right]\frac{\partial F_1}{\partial A} + \frac{\partial}{\partial F_2}\mathsf{Tr}\left[F_2^{\mathsf{T}}B^{\mathsf{T}}F_1C^{\mathsf{T}}\right]\frac{\partial F_2}{\partial A} \\
&= BF_2C + B^{\mathsf{T}}F_1C^{\mathsf{T}} = BAC + B^{\mathsf{T}}AC^{\mathsf{T}}
\end{aligned}$$

$$\frac{\partial}{\partial A_{ij}}\log|A| = \frac{1}{|A|}\frac{\partial}{\partial A_{ij}}|A| = \frac{1}{|A|}\frac{\partial}{\partial A_{ij}}\sum_k(-1)^{i+k}A_{ik}\,|[A]_{ik}| = \frac{1}{|A|}(-1)^{i+j}\,|[A]_{ij}|$$

$$\Rightarrow \frac{\partial}{\partial A}\log|A| = (A^{-1})^{\mathsf{T}}$$

GAUSSIAN DERIVATIVES

$$
\begin{aligned}
\frac{\partial(-\ell)}{\partial \boldsymbol{\mu}} &= \frac{\partial}{\partial \boldsymbol{\mu}} \left[\frac{N}{2} \log |2\pi\Sigma| + \frac{1}{2} \sum_n (\mathbf{x}_n - \boldsymbol{\mu})^{\mathsf{T}} \Sigma^{-1} (\mathbf{x}_n - \boldsymbol{\mu}) \right] \\
&= \frac{1}{2} \sum_n \frac{\partial}{\partial \boldsymbol{\mu}} \left[(\mathbf{x}_n - \boldsymbol{\mu})^{\mathsf{T}} \Sigma^{-1} (\mathbf{x}_n - \boldsymbol{\mu}) \right] \\
&= \frac{1}{2} \sum_n \frac{\partial}{\partial \boldsymbol{\mu}} \left[\mathbf{x}_n^{\mathsf{T}} \Sigma^{-1} \mathbf{x}_n + \boldsymbol{\mu}^{\mathsf{T}} \Sigma^{-1} \boldsymbol{\mu} - 2\boldsymbol{\mu}^{\mathsf{T}} \Sigma^{-1} \mathbf{x}_n \right] \\
&= \frac{1}{2} \sum_n \frac{\partial}{\partial \boldsymbol{\mu}} \left[\boldsymbol{\mu}^{\mathsf{T}} \Sigma^{-1} \boldsymbol{\mu} \right] - 2 \frac{\partial}{\partial \boldsymbol{\mu}} \left[\boldsymbol{\mu}^{\mathsf{T}} \Sigma^{-1} \mathbf{x}_n \right] \\
&= \frac{1}{2} \sum_n \left[2\Sigma^{-1} \boldsymbol{\mu} - 2\Sigma^{-1} \mathbf{x}_n \right] \\
&= N \Sigma^{-1} \boldsymbol{\mu} - \Sigma^{-1} \sum_n \mathbf{x}_n
\end{aligned}
$$

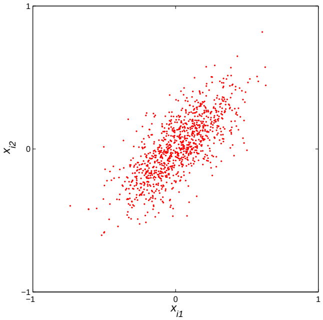
$$
= 0 \Rightarrow \hat{\boldsymbol{\mu}} = \frac{1}{N} \sum_n \mathbf{x}_n
$$

Gaussian Derivatives

$$\begin{aligned}
\frac{\partial(-\ell)}{\partial\Sigma^{-1}} &= \frac{\partial}{\partial\Sigma^{-1}}\left[\frac{N}{2}\log|2\pi\Sigma| + \frac{1}{2}\sum_n (\mathbf{x}_n-\boldsymbol{\mu})^{\mathsf{T}}\Sigma^{-1}(\mathbf{x}_n-\boldsymbol{\mu})\right] \\
&= \frac{\partial}{\partial\Sigma^{-1}}\left[\frac{N}{2}\log|2\pi I|\right] - \frac{\partial}{\partial\Sigma^{-1}}\left[\frac{N}{2}\log|\Sigma^{-1}|\right] \\
&\qquad + \frac{1}{2}\sum_n \frac{\partial}{\partial\Sigma^{-1}}\left[(\mathbf{x}_n-\boldsymbol{\mu})^{\mathsf{T}}\Sigma^{-1}(\mathbf{x}_n-\boldsymbol{\mu})\right] \\
&= -\frac{N}{2}\Sigma^{\mathsf{T}} + \frac{1}{2}\sum_n (\mathbf{x}_n-\boldsymbol{\mu})(\mathbf{x}_n-\boldsymbol{\mu})^{\mathsf{T}} \\
= 0 \Rightarrow \hat{\Sigma} &= \frac{1}{N}\sum_n (\mathbf{x}_n-\boldsymbol{\mu})(\mathbf{x}_n-\boldsymbol{\mu})^{\mathsf{T}}
\end{aligned}$$

Multivariate Linear Regression

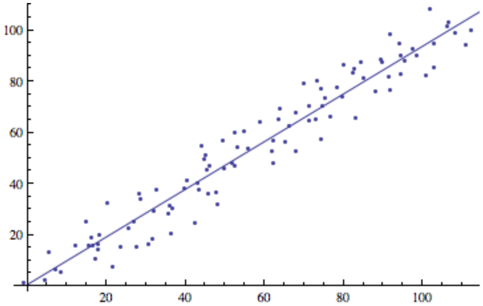
The relationship between variables can also be modelled as a conditional distribution.



- data $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1) \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$
- each $\mathbf{x}_i$ ($\mathbf{y}_i$) is a vector of D_x (D_y) features,
- $\mathbf{y}_i$ is conditionally independent of all else, given $\mathbf{x}_i$.

A simple form of supervised (predictive) learning: model $\mathbf{y}$ as a **linear** function of $\mathbf{x}$, with **Gaussian** noise.

$$p(\mathbf{y}|\mathbf{x}, \mathsf{W}, \Sigma_y) = |2\pi\Sigma_y|^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(\mathbf{y} - \mathsf{W}\mathbf{x})^{\mathsf{T}}\Sigma_y^{-1}(\mathbf{y} - \mathsf{W}\mathbf{x})\right\}$$



100
80
60
40
20
20
40
60
80
100

Multivariate Linear Regression – ML estimate

ML estimates are obtained by maximising the (conditional) likelihood, as before:

$$\begin{aligned}
\ell &= \sum_i \log p(\mathbf{y}_i|\mathbf{x}_i, \mathsf{W}, \Sigma_y) \\
&= -\frac{N}{2}\log|2\pi\Sigma_y| - \frac{1}{2}\sum_i(\mathbf{y}_i - \mathsf{W}\mathbf{x}_i)^\mathsf{T}\Sigma_y^{-1}(\mathbf{y}_i - \mathsf{W}\mathbf{x}_i) \\
\frac{\partial(-\ell)}{\partial\mathsf{W}} &= \frac{\partial}{\partial\mathsf{W}}\left[\frac{N}{2}\log|2\pi\Sigma_y| + \frac{1}{2}\sum_i(\mathbf{y}_i - \mathsf{W}\mathbf{x}_i)^\mathsf{T}\Sigma_y^{-1}(\mathbf{y}_i - \mathsf{W}\mathbf{x}_i)\right] \\
&= \frac{1}{2}\sum_i\frac{\partial}{\partial\mathsf{W}}\left[(\mathbf{y}_i - \mathsf{W}\mathbf{x}_i)^\mathsf{T}\Sigma_y^{-1}(\mathbf{y}_i - \mathsf{W}\mathbf{x}_i)\right] \\
&= \frac{1}{2}\sum_i\frac{\partial}{\partial\mathsf{W}}\left[\mathbf{y}_i^\mathsf{T}\Sigma_y^{-1}\mathbf{y}_i + \mathbf{x}_i^\mathsf{T}\mathsf{W}^\mathsf{T}\Sigma_y^{-1}\mathsf{W}\mathbf{x}_i - 2\mathbf{x}_i^\mathsf{T}\mathsf{W}^\mathsf{T}\Sigma_y^{-1}\mathbf{y}_i\right] \\
&= \frac{1}{2}\sum_i\left[\frac{\partial}{\partial\mathsf{W}}\mathrm{Tr}\left[\mathsf{W}^\mathsf{T}\Sigma_y^{-1}\mathsf{W}\mathbf{x}_i\mathbf{x}_i^\mathsf{T}\right] - 2\frac{\partial}{\partial\mathsf{W}}\mathrm{Tr}\left[\mathsf{W}^\mathsf{T}\Sigma_y^{-1}\mathbf{y}_i\mathbf{x}_i^\mathsf{T}\right]\right] \\
&= \frac{1}{2}\sum_i\left[2\Sigma_y^{-1}\mathsf{W}\mathbf{x}_i\mathbf{x}_i^\mathsf{T} - 2\Sigma_y^{-1}\mathbf{y}_i\mathbf{x}_i^\mathsf{T}\right] \\
= 0 \Rightarrow \widehat{\mathsf{W}} &= \sum_i \mathbf{y}_i\mathbf{x}_i^\mathsf{T}\Big(\sum_i \mathbf{x}_i\mathbf{x}_i^\mathsf{T}\Big)^{-1}
\end{aligned}$$

Multivariate Linear Regression – Posterior

Let y_i be scalar (so that W is a row vector) and write $\mathbf{w}$ for the column vector of weights.

A conjugate prior for $\mathbf{w}$ is

$$P(\mathbf{w}|\mathsf{A}) = \mathcal{N}\left(\mathbf{0}, \mathsf{A}^{-1}\right)$$

Then the log posterior on $\mathbf{w}$ is

$$\begin{aligned}\log P(\mathbf{w}|\mathcal{D}, \mathsf{A}, \sigma_y) &= \log P(\mathcal{D}|\mathbf{w}, \mathsf{A}, \sigma_y) + \log P(\mathbf{w}|\mathsf{A}, \sigma_y) - \log P(\mathcal{D}|\mathsf{A}, \sigma_y) \\ &= -\frac{1}{2}\mathbf{w}^\mathsf{T}\mathsf{A}\mathbf{w} - \frac{1}{2}\sum_i (y_i - \mathbf{w}^\mathsf{T}\mathbf{x}_i)^2 \sigma_y^{-2} + \text{const} \\ &= -\frac{1}{2}\mathbf{w}^\mathsf{T}(\mathsf{A} + \sigma_y^{-2}\sum_i \mathbf{x}_i\mathbf{x}_i^\mathsf{T})\mathbf{w} + \mathbf{w}^\mathsf{T}\sum_i (y_i\mathbf{x}_i)\sigma_y^{-2} + \text{const} \\ &= -\frac{1}{2}\mathbf{w}^\mathsf{T}\Sigma_w^{-1}\mathbf{w} + \mathbf{w}^\mathsf{T}\Sigma_w^{-1}\Sigma_w\sum_i (y_i\mathbf{x}_i)\sigma_y^{-2} + \text{const} \\ &= \log\ \mathcal{N}\left(\Sigma_w \textstyle\sum_i (y_i\mathbf{x}_i)\sigma_y^{-2}, \Sigma_w\right)\end{aligned}$$

MAP and ML for linear regression

As the posterior is Gaussian, the MAP and posterior mean weights are the same:

$$\mathbf{w}^{\text{MAP}} = \underbrace{\left(A + \frac{\sum_i \mathbf{x}_i \mathbf{x}_i^\top}{\sigma_y^2}\right)^{-1}}_{\Sigma_w} \frac{\sum_i y_i \mathbf{x}_i}{\sigma_y^2} = \left(A\sigma_y^2 + \sum_i \mathbf{x}_i \mathbf{x}_i^\top\right)^{-1} \sum_i y_i \mathbf{x}_i$$

Compare this to the (transposed) ML weight vector for scalar outputs:

$$\mathbf{w}^{\text{ML}} = \widehat{\mathsf{W}}^\top = \left(\sum_i \mathbf{x}_i \mathbf{x}_i^\top\right)^{-1} \sum_i y_i \mathbf{x}_i$$

- The prior acts to “inflate” the apparent covariance of inputs.
- As A is positive (semi)definite, shrinks the weights towards the prior mean (here $\mathbf{0}$).
- If $A = \alpha I$ this is known as the ridge regression estimator.
- The MAP/shrinkage/ridge weight estimate often has lower squared error (despite bias) and makes more accurate predictions on test inputs than the ML estimate.
- An example of prior-based regularisation of estimates.

Remarks

- Models the *conditional* $P(\mathbf{y}|\mathbf{x})$.
- If we also model $P(\mathbf{x})$, then learning is indistinguishable from unsupervised. In particular if $P(\mathbf{x})$ is Gaussian, and $P(\mathbf{y}|\mathbf{x})$ is linear-Gaussian, then $\mathbf{x}, \mathbf{y}$ are *jointly Gaussian*.

REFERENCE POINTERS

- It is very important that you *understand* all the material in the following cribsheet:
  http://www.gatsby.ucl.ac.uk/teaching/courses/ml1/cribsheet.pdf

- The following notes by (the late) Sam Roweis are quite useful:
  - Matrix identities and matrix derivatives:
    http://www.cs.nyu.edu/~roweis/notes/matrixid.pdf
  - Gaussian identities:
    http://www.cs.nyu.edu/~roweis/notes/gaussid.pdf

- Here is a useful statistics / pattern recognition glossary:
  http://alumni.media.mit.edu/~tpminka/statlearn/glossary/

- Tom Minka's in-depth notes on matrix algebra:
  http://research.microsoft.com/en-us/um/people/minka/papers/matrix/

OPTIMIZATION

Motivation

The way machine learning applies models to data can roughly be categorized into two ways:

Optimization:

- The solution is a specific value.
- We define mathematically what a “good” solution is and search for the best one.

Simulation:

- The solution is given by distribution (e.g. the distribution itself, an expectation, etc)
- We approximate the solution by samples drawn from the distribution.

We will discuss optimization methods next, and simulation methods later in the course.

Extremal Principles

	Purpose	Objective function
Maximum likehood estimation	Fit a statistical model	Likelihood
Empirical risk minimization	Fit a classifier/predictor	Error rate (more generally: empirical risk)
Variational inference	Approximate a posterior	Distance between approximate and true posterior

The list is far from exhaustive.

TERMINOLOGY

Min and argmin

$$\min_x f(x) = \text{ smallest value of } f(x) \text{ for any } x$$
$$\arg\min_x f(x) = \text{ value of } x \text{ for which } f(x) \text{ is minimal}$$

Minimum with respect to subset of arguments

$$\min_x f(x, y) = \text{ smallest value of } f(x, y) \text{ for any } x \text{ if } y \text{ is kept fixed}$$

Optimization problem

For a given function $f : \mathbb{R}^d \to \mathbb{R}$, a problem of the form

$$\text{find} \quad x^* := \arg\min_x f(x)$$

is called a **minimization problem**. If $\arg\min$ is replaced by $\arg\max$, it is a **maximization problem**. Minimization and maximization problems are collectively referred to as **optimization problems**.

Minimization vs Maximization

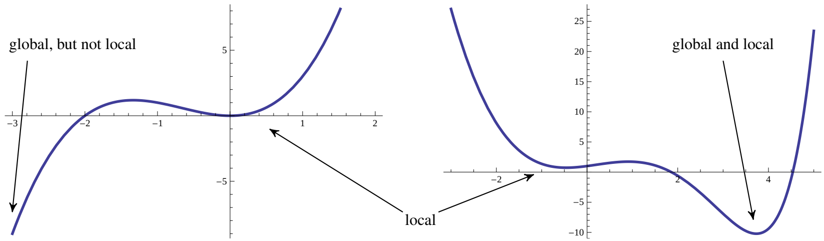
For any function f, we have

$$\min f(x) = -\max(-f(x)) \qquad \text{and} \qquad \arg\min f(x) = \arg\max(-f(x))$$

That means:

- If we know how to minimize, we also know how to maximize, and vice versa.
- We do not have to solve both problems separately; we can just generically discuss minimization.

Types of Minima



Local and global minima

A minimum of f at x is called:

- **Global** if f assumes no smaller value on its domain.
- **Local** if there is some open interval (a, b) containing x such that $f(x)$ is a global minimum of f restricted to that interval.

Solving Optimization Problems

Analytic criteria for local minima

Recall that x is a local minimum of f if

$$f'(x) = 0 \qquad \text{and} \qquad f''(x) > 0 \; .$$

In $\mathbb{R}^d$,

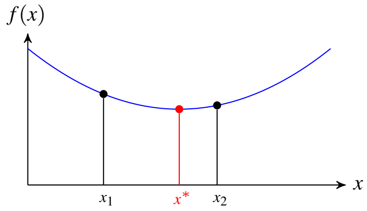
$$\nabla f(x) = 0 \qquad \text{and} \qquad H_f(x) = \Big(\frac{\partial f}{\partial x_i \partial x_j}(x)\Big)_{i,j=1,\ldots,n} \text{ positive definite.}$$

The $d \times d$-matrix $H_f(x)$ is called the **Hessian matrix** of f at x.

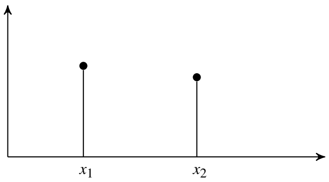
Typical situation

- Given is a function $f : \mathbb{R}^d \to \mathbb{R}$.
- The dimension d is usually very large.
  (In neural network training problems: Often in the millions.)
- We cannot plot or “look at” the function.
- We can only evaluate its value $f(x)$ point by point.

One-dimensional Illustration



- Our goal is to find x^*.
- We can evaluate the function at points of our choice, say x_1 and x_2.



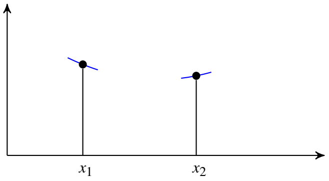
- However, we cannot "see" the function.
- All we know are values at a few points.

Task

Based on the values we know, we have to:

- Either make a decision what x^* is.
- Or gather more information, by evaluating f at additional points. In that case, we have to decide which point to evaluate next.

NUMERICAL OPTIMIZATION



- If we can compute the derivatives $f'(x_1)$ and $f'(x_2)$, we have (the slope of) linear approximations to f at both points that are locally exact.
- That is: We can substitute the derivatives for the two short blue lines in the figure.
- We can tell from the sign of the derivative in which direction the function decreases.
- We also know that $f'(x) = 0$ if x is a minimum.

Minimization Strategy

Basic idea

Start with some point x_0. Compute the derivative $f'(x_0)$ at x. Then:

- "Move downhill": Choose some $c > 0$, and set $x_1 = x_0 + c$ if $f'(x_0) < 0$ and $x_1 = x_0 - c$ if $f'(x_0) > 0$.
- Compute $f'(x_1)$. If it is 0 (possibly a minimum), stop.
- Otherwise, move downhill from x_1, etc.

Observations

- Since the sign of f' is determined by whether f increases or decreases, we can summarize the case distinction above by setting

$$x_1 = x_0 - \text{sign}(f'(x_0)) \cdot c$$

- If f changes rapidly, it may be a good strategy to make a large step (choose a large c), since we presumably are still far from the minimum. If f changes slowly, c should be small.
- One way of doing so is to choose c as the magnitude of f', since $|f'|$ has exactly this property. In that case:

$$x_1 = x_0 - \text{sign}(f'(x_0)) \cdot |f'(x_0)| = x_0 - f'(x_0)$$

The algorithm obtained by applying this step repeatedly is called **gradient descent**.

GRADIENT DESCENT

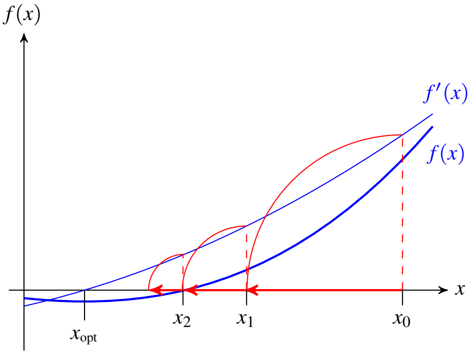
Gradient descent searches for a minimum of a differentiable function f.

Algorithm

Start with some point $x_0 \in \mathbb{R}$ and fix a precision $\varepsilon > 0$.

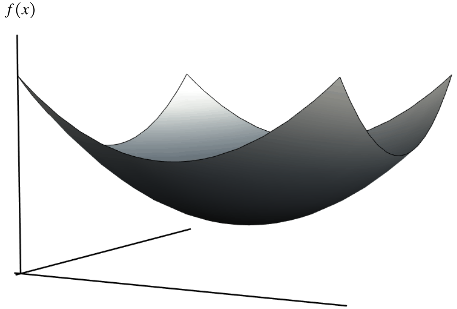
Repeat for $n = 1, 2, \ldots$:

1. Check whether $|f'(x_n)| < \varepsilon$. If so, report the solution $x^* := x_n$ and terminate.
2. Otherwise, set
$$x_{n+1} := x_n - f'(x_n)$$



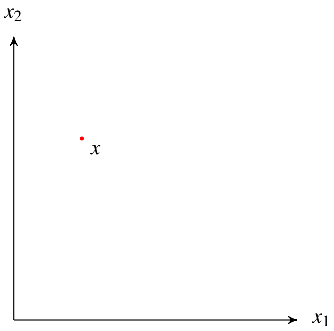
Mention numerical evaluation.

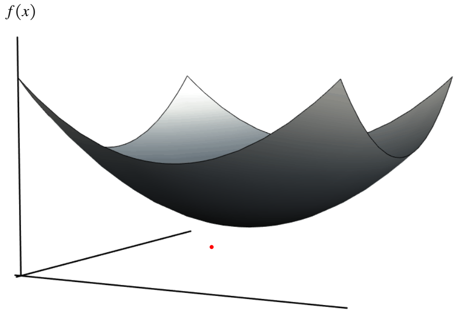
Derivatives in Multiple Dimensions



- We now ask how to define a derivative in multiple dimensions.
- Consider a function $f : \mathbb{R}^d \to \mathbb{R}$. What is the derivative of f at a point x?
- For simplicity, we assume $d = 2$ (so that we can plot the function).

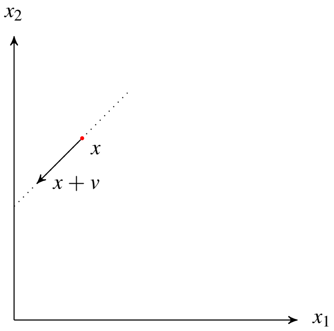
Derivatives in Multiple Dimensions

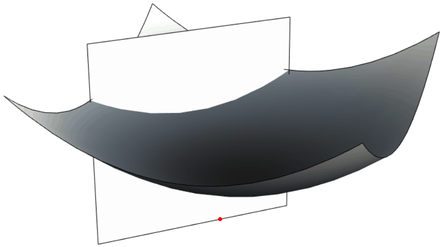




- We fix a point $x = (x_1, x_2)$ in $\mathbb{R}^2$, marked red above.
- We will try to turn this into a 1-dimensional problem, so that we can use the definition of a derivative we already know.

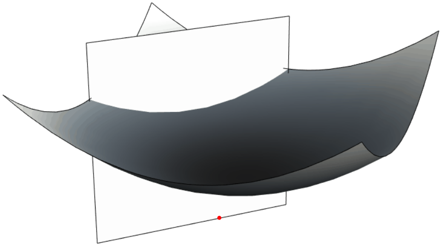
REDUCING TO ONE DIMENSION

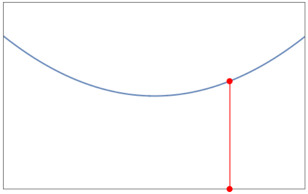




- To make the problem 1-dimensional, fix some vector $v \in \mathbb{R}_2$, and draw a line through x in direction of v.
- Then intersect f with a plane given by this line: In the coordinate system of f, choose the plane that contains the line and is orthogonal to $\mathbb{R}^2$.
- The plane contains the point x.
- Note we can do that even if $d > 2$. We still obtain a plane.

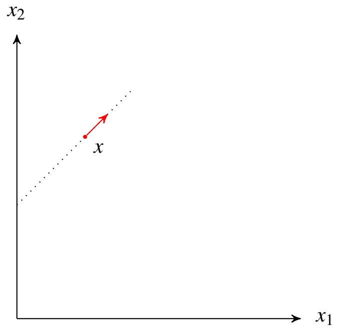
Reducing to one Dimension





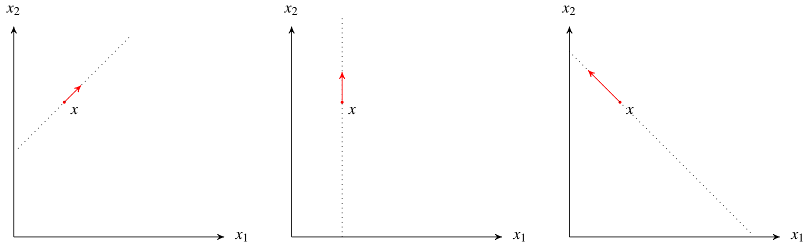
- The intersection of f with the plane is a 1-dimensional function f_H, and x corresponds to a point x_H in its domain.
- We can now compute the derivative f'_H of f_H at x_H. The idea is to use this as the derivative of f at x.

Back to Multiple Dimensions



- In the domain of f, we draw a vector from x *in direction of* H such that:
    1. The vector is oriented to point in the direction in which f_H increases.
    2. Its length is the value of the derivative $f'_H(x)$.
- That completely determines the vector (shown in red above).
- There is one problem still to be solved: f_H depends on H, that is, on the direction of the vector v. Which direction should we use?

The Gradient



- We now rotate the plane H around x. For each position of the plane, we get a new derivative $f'_H(x)$, and a new red vector.
- We choose the plane for which f'_H is largest:

$$H^* := \arg \max_{\text{all rotations of } H} f'_H(x)$$

  Provided that f_H is differentiable for all H, one can show that this is always unique (or $f'_H(x)$ is zero for all H).
- We then define the vector

$$\nabla f(x) := \text{ vector given by } H^* \text{ as above}$$

  The vector $\nabla f(x)$ is called the **gradient** of f at x.

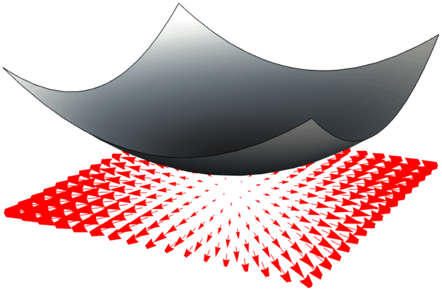
Properties of the Gradient

The gradient $\nabla f(x)$ of $f : \mathbb{R}^d \to \mathbb{R}$ at a point $x \in \mathbb{R}^d$ is a vector in the domain $\mathbb{R}^d$ in the direction in which f most rapidly increases at x.

- Recall that a contour line (or contour set) of f is a set of points along which f remains constant,
$$C[f,c] := \{x \in \mathbb{R}^d \,|\, f(x) = c\} \qquad \text{for some } c \in \mathbb{R}.$$
- One can show that if $C[f,c]$ contains x, the gradient at x is orthogonal to the contour:
$$\nabla f(x) \perp C[f,c] \qquad \text{if } x \in C[f,c]\;.$$
- Intuition: The gradient points in the direction of maximal change, whereas $C[f,c]$ is a direction in which there is no change. Locally, these two are orthogonal.

Gradients are orthogonal to contour lines.

Gradients and Contour Lines



- For this parabolic function, all contour lines are concentric circles around the minimum.
- The picture above shows the gradients plotted at various points in the plane.

Basic Gradient Descent

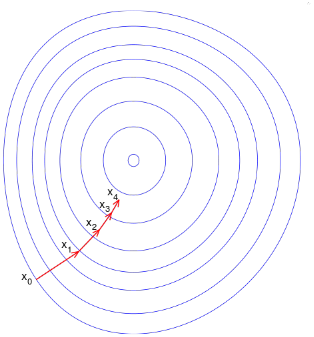
$$f : \mathbb{R}^d \to \mathbb{R}$$

Algorithm

Start with some point $x_0 \in \mathbb{R}^d$ and fix a precision $\varepsilon > 0$.

Repeat for $n = 1, 2, \dots$:

1. Check whether $\|\nabla f(x_n)\| < \varepsilon$. If so, report the solution $x^* := x_n$ and terminate.
2. Otherwise, set
$$x_{n+1} := x_n - \nabla f(x_n)$$



Gradient Descent

$$f : \mathbb{R}^d \to \mathbb{R}$$

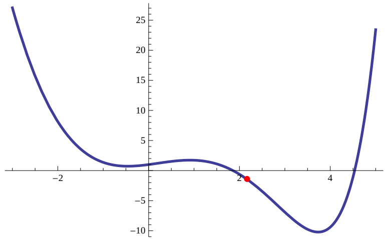
Algorithm

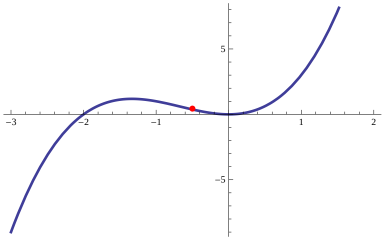
Start with some point $x_0 \in \mathbb{R}^d$ and fix a precision $\varepsilon > 0$.

Repeat for $n = 1, 2, \ldots$:

1. Check whether $\|\nabla f(x_n)\| < \varepsilon$. If so, report the solution $x^* := x_n$ and terminate.
2. Otherwise, set
$$x_{n+1} := x_n - \alpha(n)\nabla f(x_n)$$

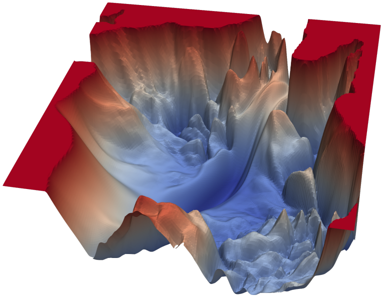
Here, $\alpha(n) > 0$ is a coefficient that may depend on n. It is called the **step size** in optimization, or the **learning rate** in machine learning.

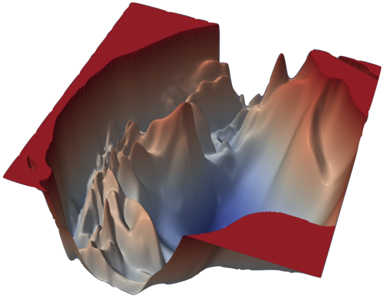




- Suppose for both functions above, gradient descent is started at the point marked red.
- It will "walk downhill" as far as possible, then terminate.
- For the function on the left, the minimum it finds is global. For the function on the right, it is only a local minimum.
- Since the derivative at both minima is 0, gradient descent cannot detect whether they are global or local.

For smooth functions, gradient descent finds *local* minima. If the function is complicated, there may be no way to tell whether the solution is also a global minimum.





Source: Li et al., arxiv 1712.09913

Mention saddle points.

Outlook

Summary so far

- The derivative/gradient provides local information about how a function changes around a point x.
- Optimization algorithms: If we know the gradient at our current location x, we can use this information to make a step in “downhill” direction, and move closer to a (local) minimum.

What we do not know yet

That assumes that we can compute the gradient. There are two possibilities:

- For some functions, we are able to derive $\nabla f(x)$ as a function of x. Gradient descent can evaluate the gradient by evaluating that function.
- Otherwise, we have to estimate $\nabla f(x)$ by evaluating the function f at points close to x.

For now, we will assume that we can compute the gradient as a function.

Next: Newton’s method

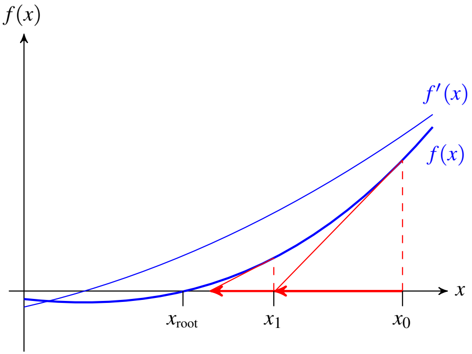
- Gradient descent is a **first-order method**. It uses only the first derivative.
- A similar method that uses two derivatives is called Newton’s method (or the Newton–Raphson algorithm).
- Roughly speaking, higher-order methods converge in fewer steps, at the price of more computation per step.

NEWTON'S METHOD: ROOTS

Algorithm

Newton's method searches for a **root** of f, i.e. it solves the equation $f(x) = 0$.

1. Start with some point $x \in \mathbb{R}$ and fix a precision $\varepsilon > 0$.
2. Repeat for $n = 1, 2, \ldots$
$$x_{n+1} := x_n - f(x_n)/f'(x_n)$$
3. Terminate when $|f(x_n)| < \varepsilon$.



Basic Applications

Function evaluation

Most numerical evaluations of functions ($\sqrt{a}$, $\sin(a)$, $\exp(a)$, etc) are implemented using Newton's method. To evaluate g at a, we have to transform $x = g(a)$ into an equivalent equation of the form

$$f(x, a) = 0 \ .$$

We then fix a and solve for x using Newton's method for roots.

Example: Square root

To eveluate $g(a) = \sqrt{a}$, we can solve

$$f(x, a) = x^2 - a = 0 \ .$$

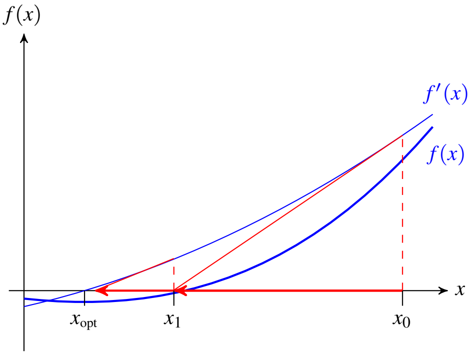
This is essentially how `sqrt()` is implemented in the standard C library.

Newton's Method: Minima

Algorithm

We can use Newton's method for minimization by applying it to solve $f'(x) = 0$.

1. Start with some point $x \in \mathbb{R}$ and fix a precision $\varepsilon > 0$.
2. Repeat for $n = 1, 2, \ldots$
$$x_{n+1} := x_n - f'(x_n)/f''(x_n)$$
3. Terminate when $|f'(x_n)| < \varepsilon$.



Multiple Dimensions

Recall: Gradient descent

$$x_{n+1} := x_n - \nabla f(x_n)$$

Newton's method for minima

$$x_{n+1} := x_n - H_f^{-1}(x_n) \cdot \nabla f(x_n)$$

That requires that the matrix $H_f(x)$ is invertible.

The Hessian measures the curvature of f.

Effect of the Hessian

Multiplication by H_f^{-1} in general changes the direction of $\nabla f(x_n)$. The correction takes into account how $\nabla f(x)$ changes away from x_n, as estimated using the Hessian at x_n.

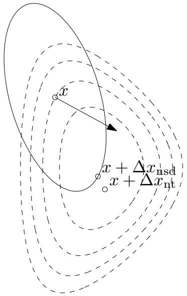


Figure: Arrow is ∇f, $x + \Delta x_{\text{nt}}$ is Newton step.

Newton: Properties

Convergence

- The algorithm always converges if $f'' > 0$ (or H_f positive definite).
- The speed of convergence separates into two phases:
    - In a (possibly small) region around the minimum, f can always be approximated by a quadratic function.
    - Once the algorithm reaches that region, the error decreases at quadratic rate. Roughly speaking, the number of correct digits in the solution doubles in each step.
    - Before it reaches that region, the convergence rate is linear.

High dimensions

- The required number of steps hardly depends on the dimension of $\mathbb{R}^d$. Even in $\mathbb{R}^{10000}$, you can usually expect the algorithm to reach high precision in half a dozen steps.
- Caveat I: The individual steps can become very expensive, since we have to invert H_f in each step, which is of size $d \times d$.
- Caveat II: High-dimensional functions tend to have many more local minima then low-dimensional ones. Even if Newton still converges quickly, we have to ask even more carefully what it is converging to.

NEXT: CONSTRAINED OPTIMIZATION

So far

- If f is differentiable, we can search for local minima using gradient descent.
- If f is sufficiently nice (twice continuously differentiable), we know how to speed up the search process using Newton's method.

Constrained problems

- The numerical minimizers use the criterion $\nabla f(x) = 0$ for the minimum.
- In a constrained problem, the minimum is *not* identified by this criterion.

Next steps

We will figure out how the constrained minimum can be identified. We have to distinguish two cases:

- Problems involving only equalities as constraints (easy).
- Problems also involving inequalities (a bit more complex).

CONSTRAINED OPTIMIZATION

- An **optimization problem** for a given function $f : \mathbb{R}^d \to \mathbb{R}$ is a problem of the form

$$\min_x f(x)$$

  which we read as "find $x^* = \arg\min_x f(x)$".
- A **constrained optimization problem** adds additional requirements on x,

$$\begin{aligned} &\min_x \; f(x) \\ \text{subject to} \quad & x \in G\,, \end{aligned}$$

  where $G \subset \mathbb{R}^d$ is called the **feasible set**.
- The set G is often defined by equations, e.g.

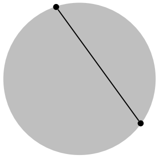
$$\begin{aligned} &\min_x \; f(x) \\ \text{subject to} \quad & g(x) = 0 \end{aligned}$$

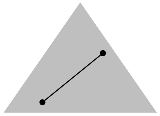
  The equation g is called a **constraint**. The constraint $g(x) = 0$ above is also called an **equality constraint**, whereas one of the form $g(x) \geq 0$ is an **inequality constraint**.

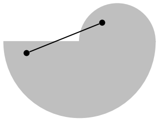
Convex sets

Definition

A set $A \subset \mathbb{R}^d$ is called **convex** if, for every two points $x, y \in A$, the straight line connecting x and y is completely contained in A.


convex


convex

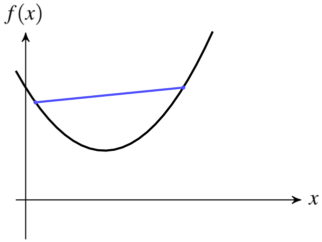

not convex

Quantitatively

$$A \text{ is convex if and only if} \qquad \lambda x + (1-\lambda) y \in A \qquad \text{for all } x, y \in A \text{ and } \lambda \in [0,1] \, .$$

Convex Functions

A function f is **convex** if every line segment between function values lies above the graph of f.



- Equivalently:
  If the area above (!) the curve is a convex set.
- Quantitatively: f is convex if

  $$\lambda f(x) + (1-\lambda)f(y) \geq f(\lambda x + (1-\lambda)y)$$

  for all x and y in the domain of f and all $\lambda \in [0, 1]$.
- A twice differentiable function is convex if $f''(x) \geq 0$ (or $H_f(x)$ positive semidefinite) for all x.

Implications for optimization

If f is convex, then:

- $f'(x) = 0$ is a sufficient criterion for a minimum.
- Local minima are global.
- If f is **strictly convex** ($f'' > 0$ or H_f positive definite), there is only one minimum (which is both gobal and local).

Optimization Under Constraints

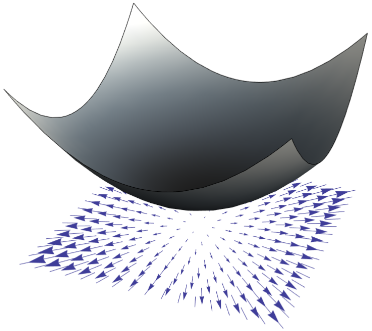
Objective

$$\min f(x)$$
$$\text{subject to } g(x) = 0$$

Idea

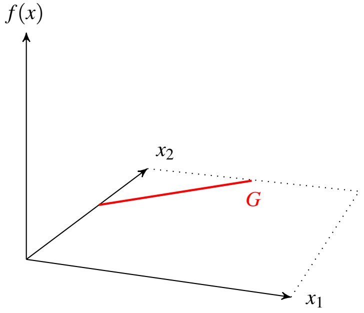
- The feasible set is the set of points x which satisfy $g(x) = 0$,
$$G := \{x \,|\, g(x) = 0\} \;.$$
If g is reasonably smooth, G is a smooth surface in $\mathbb{R}^d$.
- We restrict the function f to this surface and call the restricted function f_g.
- The constrained optimization problem says that we are looking for the minimum of f_g.

Lagrange Optimization



$$f(x) = x_1^2 + x_2^2$$

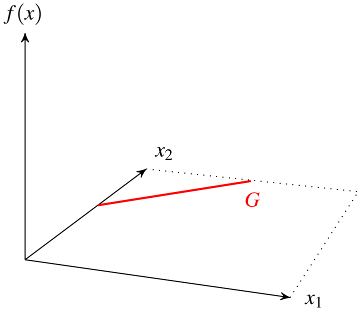
The blue arrows are the gradients $\nabla f(x)$ at various values of x.

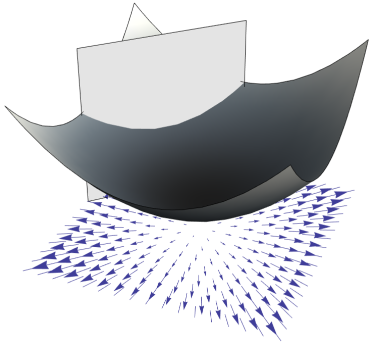


Constraint g.

Here, g is linear, so the graph of g is a (sloped) affine plane. The intersection of the plane with the x_1-x_2-plane is the set G of all points x with $g(x) = 0$.

LAGRANGE OPTIMIZATION





- We can make the function f_g given by the constraint $g(x) = 0$ visible by placing a plane vertically through G. The graph of f_g is the intersection of the graph of f with the plane.
- Here, f_g has parabolic shape.
- The gradient of f at the miniumum of f_g is *not* 0.

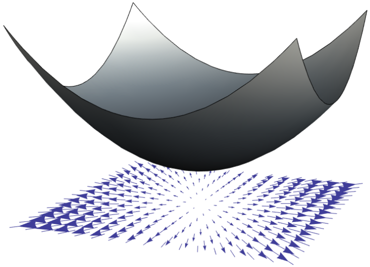
Gradients and Contours

Fact

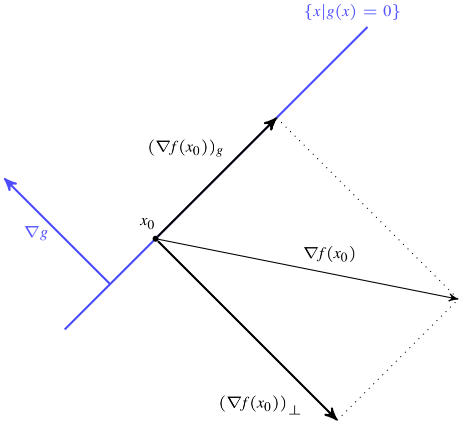
Gradients are orthogonal to contour lines.

Intuition

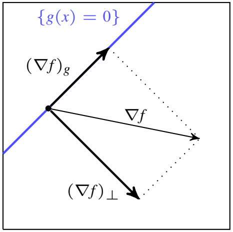
- The gradient points in the direction in which f grows most rapidly.
- Contour lines are sets along which f does not change.



The Crucial Bit



Again, in detail.



Idea

- Decompose ∇f into a component $(\nabla f)_g$ in the set $\{x \mid g(x) = 0\}$ and a remainder $(\nabla f)_\perp$.
- The two components are orthogonal.
- If f_g is minimal within $\{x \mid g(x) = 0\}$, the component within the set vanishes.
- The remainder need not vanish.

Consequence

- We need a criterion for $(\nabla f)_g = 0$.

Solution

- If $(\nabla f)_g = 0$, then ∇f is orthogonal to the set $g(x) = 0$.
- Since gradients are orthogonal to contours, and the set is a contour of g, ∇g is also orthogonal to the set.
- Hence: At a minimum of f_g, the two gradients point in the same direction: $\nabla f + \lambda \nabla g = 0$ for some scalar $\lambda \neq 0$.

SOLUTION: CONSTRAINED OPTIMIZATION

Solution
The constrained optimization problem

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & g(x) = 0 \end{aligned}$$

is solved by solving the equation system

$$\begin{aligned} \nabla f(x) + \lambda \nabla g(x) &= 0 \\ g(x) &= 0 \end{aligned}$$

The vectors ∇f and ∇g are d-dimensional, so the system contains $d + 1$ equations for the $d + 1$ variables $x_1, \dots, x_d, \lambda$.

INEQUALITY CONSTRAINTS

Objective

For a function f and a convex function g, solve

$$\min f(x)$$
$$\text{subject to } g(x) \leq 0$$

i.e. we replace $g(x) = 0$ as previously by $g(x) \leq 0$. This problem is called an optimization problem with **inequality constraint**.

Feasible set

We again write G for the set of all points which satisfy the constraint,

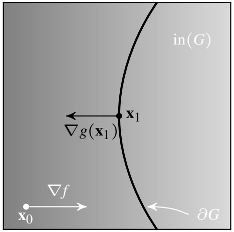
$$G := \{x \,|\, g(x) \leq 0\} \; .$$

G is often called the **feasible set** (the same name is used for equality constraints).

Two Cases

Case distinction

1. The location *x* of the minimum can be in the *interior* of *G*
2. *x* may be on the *boundary* of *G*.



lighter shade of blue = larger value of *f*

Decomposition of *G*

$$G = \text{in}(G) \cup \partial G = \text{ interior } \cup \text{ boundary}$$

The interior is given by $g(x) < 0$, the boundary by $g(x) = 0$.

Criteria for minimum

1. **In interior:** $f_g = f$ and hence $\nabla f_g = \nabla f$. We have to solve a standard optimization problem with criterion $\nabla f = 0$.
2. **On boundary:** Here, $\nabla f_g \neq \nabla f$. Since $g(x) = 0$, the geometry of the problem is the same as we have discussed for equality constraints, with criterion $\nabla f = \lambda \nabla g$. **However:** In this case, the sign of λ matters.

On the Boundary

Observation

- An extremum on the boundary is a minimum only if ∇f points *into* G.
- Otherwise, it is a maximum instead.

Criterion for minimum on boundary

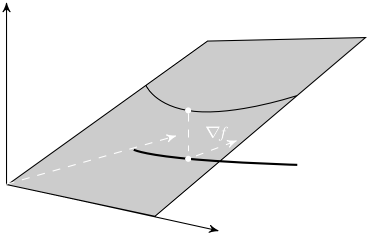
Since ∇g points *away* from G (since g increases away from G), ∇f and ∇g have to point in opposite directions:

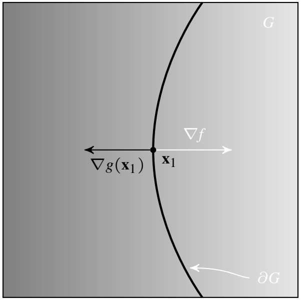
$$\nabla f = \lambda \nabla g \qquad \text{with } \lambda < 0$$

Convention

To make the sign of λ explicit, we constrain λ to positive values and instead write:

$$\nabla f = -\lambda \nabla g$$
$$\text{s.t. } \lambda > 0$$





Combining the Cases

Combined problem

$$\begin{aligned} \nabla f &= -\lambda \nabla g \\ \text{s.t.} \quad g(x) &\leq 0 \\ \lambda &= 0 \text{ if } x \in \text{in}(G) \\ \lambda &> 0 \text{ if } x \in \partial G \end{aligned}$$

Can we get rid of the "if $x \in \cdot$" distinction?

Yes: Note that $g(x) < 0$ holds if x is in the interior, and $g(x) = 0$ on the boundary.
Hence, we always have either $\lambda = 0$ or $g(x) = 0$ (and never both).

That means we can substitute

$$\begin{aligned} \lambda &= 0 \text{ if } x \in \text{in}(G) \\ \lambda &> 0 \text{ if } x \in \partial G \end{aligned}$$

by

$$\lambda \cdot g(x) = 0 \qquad \text{and} \qquad \lambda \geq 0 \; .$$

Solution: Inequality Constraints

Combined solution

The optimization problem with inequality constraints

$$\begin{gathered}\min f(x)\\ \text{subject to } g(x) \leq 0\end{gathered}$$

can be solved by solving

$$\begin{aligned} & \left.\begin{aligned} \nabla f(x) &= -\lambda \nabla g(x) \\ \text{s.t.} \qquad \lambda g(x) &= 0 \end{aligned}\right\} \longleftarrow \begin{array}{l}\text{system of } d+1 \text{ equations for } d+1\\ \text{variables } x_1, \dots, x_d, \lambda\end{array} \\ & \qquad\qquad g(x) \leq 0 \\ & \qquad\qquad \lambda \geq 0 \end{aligned}$$

These conditions are known as the **Karush-Kuhn-Tucker** (or **KKT**) conditions.

REMARKS

Haven't we made the problem more difficult?

- To simplify the minimization of f for $g(x) \leq 0$, we have made f more complicated and added a variable and two constraints. Well done.
- However: In the original problem, we *do not know how to minimize* f, since the usual criterion $\nabla f = 0$ does not work.
- By adding λ and additional constraints, we have reduced the problem to solving a system of equations.

Summary: Conditions

Condition	Ensures that...	Purpose
$\nabla f(x) = -\lambda \nabla g(x)$	If $\lambda = 0$: ∇f is 0	Opt. criterion inside G
	If $\lambda > 0$: ∇f is anti-parallel to ∇g	Opt. criterion on boundary
$\lambda g(x) = 0$	$\lambda = 0$ in interior of G	Distinguish cases $\text{in}(G)$ and ∂G
$\lambda \geq 0$	∇f cannot flip to orientation of ∇g	Optimum on ∂G is minimum

WHY SHOULD g BE CONVEX?

More precisely

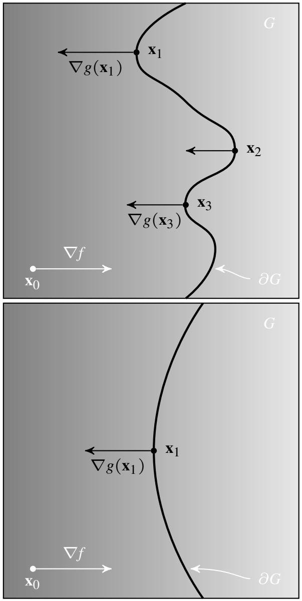
If g is a convex function, then $G = \{x \mid g(x) \leq 0\}$ is a convex set. Why do we require convexity of G?

Problem

If G is not convex, the KKT conditions do not guarantee that x is a minimum. (The conditions still hold, i.e. if G is not convex, they are necessary conditions, but not sufficient.)

Example (Figure)

- f is a linear function (lighter color = larger value)
- ∇f is identical everywhere
- If G is not convex, there can be several points (here: $\mathbf{x}_1$, $\mathbf{x}_2$, $\mathbf{x}_3$) which satisfy the KKT conditions. Only $\mathbf{x}_1$ minimizes f on G.
- If G is convex, such problems cannot occur.



Interior Point Methods

Numerical methods for constrained problems

Once we have transformed our problem using Lagrange multipliers, we still have to solve a problem of the form

$$\begin{aligned} & \nabla f(x) = -\lambda \nabla g(x) \\ \text{s.t.} \qquad & \lambda g(x) = 0 \quad \text{and} \quad g(x) \leq 0 \quad \text{and} \quad \lambda \geq 0 \end{aligned}$$

numerically.

Barrier functions

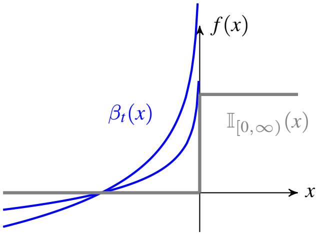
Idea
A constraint in the problem

$$\min f(x) \qquad \text{s.t.} \quad g(x) < 0$$

can be expressed as an indicator function:

$$\min f(x) + const. \cdot \mathbb{I}_{[0,\infty)}(g(x))$$

The constant must be chosen large enough to enforce the constraint.



Problem: The indicator function is piece-wise constant and not differentiable at 0. Newton or gradient descent are not applicable.

Barrier function
A **barrier function** approximates $\mathbb{I}_{[0,\infty)}$ by a smooth function, e.g.

$$\beta_t(x) := -\frac{1}{t}\log(-x) \; .$$

NEWTON FOR CONSTRAINED PROBLEMS

Interior point methods

We can (approximately) solve

$$\min f(x) \text{ s.t. } \quad g_i(x) < 0 \quad \text{for } i = 1, \ldots, m$$

by solving

$$\min f(x) + \sum_{i=1}^{m} \beta_{i,t}(x) \ .$$

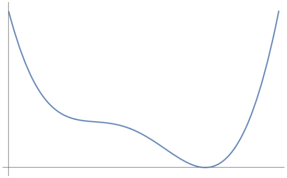
with one barrier function $\beta_{i,t}$ for each constraint g_i.
We do not have to adjust a multiplicative constant since $\beta_t(x) \to \infty$ as $x \nearrow 0$.

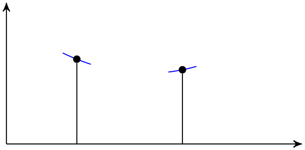
Constrained problems: General solution strategy

1. Convert constraints into solvable problem using Lagrange multipliers.
2. Convert constraints of transformed problem into barrier functions.
3. Apply numerical optimization (usually Newton's method).

Relevance of convexity

A common textbook claim is that convexity is important for optimization because convex functions have only one local and global minimum. That is correct, but not really the whole story.


Not convex, but unique minimum and well-suited for numerical optimization.


Recall that a numerical optimization algorithm can only query a function at points.

Convexity allows us to draw *global* conclusions from *local* properties.

Convexity as a generalization of linearity

- Constant functions: If we know $f(x)$ at a single point x, we know f.
- Linear functions: If we know $f(x)$ and $\nabla f(x)$ at a single point, we know f.
- Convex functions: If we know $f(x)$ and $\nabla f(x)$ at a single point, we know "on which side of x" the minimum occurs (that is: in which half-space).

There are various other global properties of convex functions that are determined by their behavior on the neighborhood of a single point, some of which are much deeper and more surprising than the one above.

STOCHASTIC GRADIENTS

If ε is a random variable with $\mathbb{E}[\varepsilon] = 0$, then

$$\widehat{\nabla} f(x) := \nabla f(x) + \varepsilon$$

is called a **stochastic gradient** of f at x.

Stochastic gradient descent

Substituting into gradient descent with step size $\alpha : \mathbb{N} \to \mathbb{R}_+$:

$$\hat{x}_{n+1} = \hat{x}_n - \alpha(n)\widehat{\nabla} f(\hat{x}_n)$$

CAN SGD CONVERGE?

Compared to gradient descent

- Fix x_0. Start a gradient descent sequence $x_1, x_2, \dots$ and a SGD sequence $\hat{x}_1, \hat{x}_2, \dots$. Then

$$x_{n+1} = x_0 - \sum_{i=1}^{n} \alpha(i)\nabla f(x_i) \quad \text{and} \quad \hat{x}_{n+1} = x_0 - \sum_{i=1}^{n} \alpha(i)\widehat{\nabla} f(\hat{x}_i)$$

- Each stochastic gradient is of the form $\widehat{\nabla} f(\hat{x}_i) = \nabla f(\hat{x}_i) + \varepsilon_i$. Define δ_i as

$$\widehat{\nabla} f(\hat{x}_i) = \nabla f(x_i) + \delta_i$$

  Note that $\delta_i \neq \varepsilon_i$ in general.

- We now have

$$\hat{x}_{n+1} = x_{n+1} - \sum_{i=1}^{n} \alpha(i)\delta_i$$

Convergence analysis

- We often ask convergence questions relative to gradient descent: If gradient descent converges for a given problem, would SGD also converge?
- The answer depends on the sequence $\alpha(1), \alpha(2), \dots$ and $\delta_1, \delta_2, \dots$. How dependent the random variables δ_n are is crucial.
- Even if $\varepsilon_1, \varepsilon_2, \dots$ are i.i.d., the same is typically *not* true for $\delta_1, \delta_2, \dots$.

Convergence Conditions

Robbins-Monro conditions

$$\underbrace{\sum_{n=1}^{\infty} \alpha(n) = \infty}_{\text{also relevant for for gradient descent}} \qquad\qquad \underbrace{\sum_{n=1}^{\infty} \alpha(n)^2 < \infty}_{\text{variance condition}}$$

- The first condition ensures points arbitrarily far from x_0 are reachable. This is typically also required for ordinary gradient descent in $\mathbb{R}^d$.
- The second condition ensures finite variance.
- You will often encounter these conditions in books and articles. They are only meaningful if we also make assumptions on the dependence between $\varepsilon_1, \varepsilon_2, \ldots$.

The variance condition

- For illustration, suppose $\delta_1, \delta_2, \ldots$ are i.i.d. with variance σ^2.
- Recall independent variables X, Y satisfy $\text{Var}[X + Y] = \text{Var}[X] + \text{Var}[Y]$.
- We hence have:

$$\text{Var}[\alpha(n)\delta_n] = \alpha(n)^2\sigma^2 \quad \text{and} \quad \text{Var}\Big[\sum_{n=1}^{\infty} \alpha(n)^2\delta_n\Big] = \Big(\sum_n \alpha(n)^2\Big)\sigma^2$$

The variance condition ensures the total variance is finite.

SGD in Machine Learning

- For an additive objective f, the cost of computing a gradient scales with sample size:

$$f(X_1,\dots,X_n,\theta) = \frac{1}{n}\sum_{i=1}^{n} f(X_i,\theta) \quad \text{hence} \quad \nabla_\theta f(X_1,\dots,X_n,\theta) = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta f(X_i,\theta)$$

- A **mini batch** is a random subset $\tilde{X}_1,\dots,\tilde{X}_k$ of a large data set $X_1,\dots,X_n$ (so $k < n$).
- A gradient is computed only for the cost on this subset.

Note X_i is now the ith data point, not the ith gradient step. The role of $\hat{x}_n$ on previous slides is now assumed by $\hat{\theta}_n$.

Stochastic gradient view

- Let $\widehat{\nabla}_\theta f(\tilde{X}_1,\dots,\tilde{X}_k)$ be gradient computed on the mini batch.
- It deviates from the actual gradient by an error term ε:

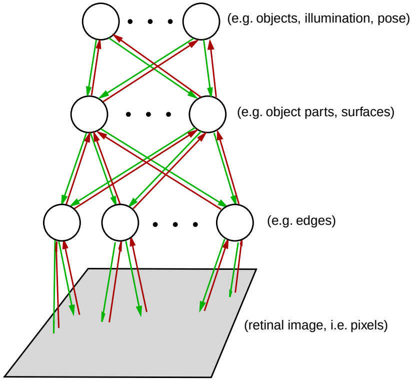
$$\widehat{\nabla}_\theta f(\tilde{X}_1,\dots,\tilde{X}_k,\theta) = \nabla_\theta f(X_1,\dots,X_n,\theta) + \varepsilon$$

  Since the mini batch is selected at random, ε is a random variable, and $\widehat{\nabla}_\theta f(\tilde{X}_1,\dots,\tilde{X}_k,\theta)$ is hence a stochastic gradient.

LATENT VARIABLE MODELS

Latent variable models

Explain correlations in $\mathbf{x}$ by assuming dependence on latent variables $\mathbf{z}$


(e.g. objects, illumination, pose)
(e.g. object parts, surfaces)
(e.g. edges)
(retinal image, i.e. pixels)

$$\mathbf{z} \sim \mathcal{P}[\theta_z]$$

$$\mathbf{x} \mid \mathbf{z} \sim \mathcal{P}[\theta_x]$$

$$p(\mathbf{x}, \mathbf{z}; \theta_x, \theta_z) = p(\mathbf{x} \mid \mathbf{z}; \theta_x) p(\mathbf{z}; \theta_z)$$

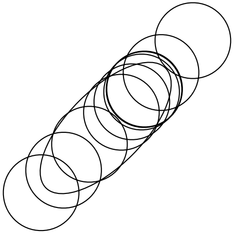
$$p(\mathbf{x}; \theta_x, \theta_z) = \int d\mathbf{z}\, p(\mathbf{x} \mid \mathbf{z}; \theta_x) p(\mathbf{z}; \theta_z)$$

Latent variable models

- Describe structured distributions.
    - Correlations in high-dimensional $\mathbf{x}$ may be captured by fewer parameters.
- Capture an underlying generative process.
    - $\mathbf{z}$ may describe *causes* of $\mathbf{x}$.
    - help to separate signal from noise.
- Combine exponential family distributions into richer, more flexible forms.
    - $P(\mathbf{z})$, $P(\mathbf{x}|\mathbf{z})$ and even $P(\mathbf{x}, \mathbf{z})$ may be in the exponential family
    - $P(\mathbf{x})$ rarely is. (Exception: Linear Gaussian models).

Latent variables and Gaussians

Gaussian correlation can be composed from latent components and uncorrelated noise.



$$\mathbf{x} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} 3 & 2 \\ 2 & 3 \end{bmatrix}\right) \qquad \Leftrightarrow \qquad z \sim \mathcal{N}(0,1) \qquad \mathbf{x} \sim \mathcal{N}\left(\sqrt{2}\begin{bmatrix} 1 \\ 1 \end{bmatrix} z, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right)$$

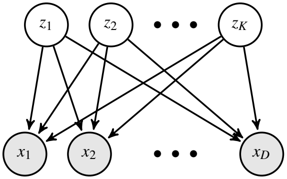
Probabilistic Principal Components Analysis (PPCA)

If the uncorrelated noise is assumed to be isotropic, this model is called PPCA.

Data: $\mathcal{D} = \mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}; \mathbf{x}_i \in \mathbb{R}^D$
Latents: $\mathcal{Z} = \{\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_N\}; \mathbf{z}_i \in \mathbb{R}^K$

Linear generative model: $x_d = \sum_{k=1}^{K} \Lambda_{dk}\, z_k + \epsilon_d$

- z_k are independent $\mathcal{N}(0,1)$ Gaussian factors
- ϵ_d are independent $\mathcal{N}(0,\psi)$ Gaussian noise
- $K < D$



Model for observations $\mathbf{x}$ is a correlated Gaussian:

$$p(\mathbf{z}) = \mathcal{N}(0, I)$$

$$p(\mathbf{x}|\mathbf{z}) = \mathcal{N}(\Lambda\mathbf{z}, \psi I)$$

Note: $\mathbb{E}_{\mathbf{x}}[f(x)] = \mathbb{E}_{\mathbf{z}}\left[\mathbb{E}_{\mathbf{x}|\mathbf{z}}[f(x)]\right]$

$\mathbb{V}_{\mathbf{x}}[x] = \mathbb{E}_{\mathbf{z}}[\mathbb{V}[\mathbf{x}|\mathbf{z}]] + \mathbb{V}_{\mathbf{z}}[\mathbb{E}[\mathbf{x}|\mathbf{z}]]$

$$p(\mathbf{x}) = \int p(\mathbf{z})p(\mathbf{x}|\mathbf{z})d\mathbf{z} = \mathcal{N}\left(\mathbb{E}_{\mathbf{z}}[\Lambda\mathbf{z}], \mathbb{E}_{\mathbf{z}}\left[\Lambda\mathbf{z}\mathbf{z}^{\mathsf{T}}\Lambda^{\mathsf{T}}\right] + \psi I\right) = \mathcal{N}\left(0, \Lambda\Lambda^{\mathsf{T}} + \psi I\right)$$

where Λ is a $D \times K$ matrix.

Multivariate Gaussians and latent variables

Two models:

$$p(\mathbf{x}) = \mathcal{N}(0, \Sigma)$$

$$\begin{aligned} p(\mathbf{z}) &= \mathcal{N}(0, I) \\ p(\mathbf{x}|\mathbf{z}) &= \mathcal{N}(\Lambda\mathbf{z}, \psi I) \\ \Rightarrow p(\mathbf{x}) &= \mathcal{N}\left(0, \Lambda\Lambda^{\mathsf{T}} + \psi I\right) \end{aligned}$$

- Descriptive density model: correlations are captured by off-diagonal elements of Σ.
- Σ has $\frac{D(D+1)}{2}$ free parameters.
- Only constrained to be positive definite.
- Simple ML estimate.

- Interpretable causal model: correlations captured by common influence of latent variable.
- $\Lambda\Lambda^{\mathsf{T}} + \psi I$ has $DK + 1$ free parameters.
- For $K < D$ covariance structure is constrained ("blurry pancake")
- ML estimation is more complex.

PPCA LIKELIHOOD

The marginal distribution on $\mathbf{x}$ gives us the PPCA likelihood:

$$\log p(\mathcal{X}|\Lambda, \psi) = -\frac{N}{2}\log\left|2\pi(\Lambda\Lambda^\mathsf{T} + \psi I)\right| - \frac{1}{2}\mathsf{Tr}\left[(\Lambda\Lambda^\mathsf{T} + \psi I)^{-1}\underbrace{\sum_n \mathbf{x}\mathbf{x}^\mathsf{T}}_{NS}\right]$$

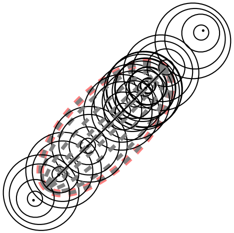
To find the ML values of (Λ, ψ) we could optimise numerically (gradient ascent / Newton's method), or we could use a different iterative algorithm called EM which we'll introduce soon.

In fact, however, ML for PPCA is more straightforward in principle, as we will see by first considering the limit $\psi \to 0$.

[Note: We may also add a constant mean $\boldsymbol{\mu}$ to the output, so as to model data that are not distributed around 0. In this case, the ML estimate $\widehat{\boldsymbol{\mu}} = \frac{1}{N}\sum_n \mathbf{x}_n$ and we can define $S = \frac{1}{N}\sum_n(\mathbf{x} - \widehat{\boldsymbol{\mu}})(\mathbf{x} - \widehat{\boldsymbol{\mu}})^\mathsf{T}$ in the likelihood above.]

THE $\psi \to 0$ LIMIT

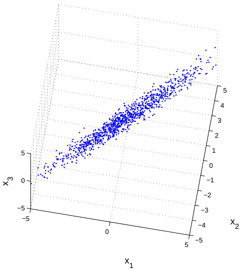
As $\psi \to 0$, the latent model can only capture *K* dimensions of variance.



In a Gaussian model, the ML parameters will find the *K*-dimensional space of most variance.

Principal Components Analysis

This leads us to an (old) algorithm called Principal Components Analysis (PCA).



Assume data $\mathcal{D} = \{\mathbf{x}_i\}$ have zero mean (if not, subtract it).

- Find direction of greatest variance – $\boldsymbol{\lambda}_{(1)}$.

$$\boldsymbol{\lambda}_{(1)} = \arg \max_{\|\mathbf{v}\|=1} \sum_n (\mathbf{x}_n^\mathsf{T}\mathbf{v})^2$$

- Find direction orthogonal to $\boldsymbol{\lambda}_{(1)}$ with greatest variance – $\boldsymbol{\lambda}_{(2)}$

$\vdots$

- Find direction orthogonal to $\{\boldsymbol{\lambda}_{(1)}, \boldsymbol{\lambda}_{(2)}, \dots, \boldsymbol{\lambda}_{(n-1)}\}$ with greatest variance – $\boldsymbol{\lambda}_{(n)}$.
- Terminate when remaining variance drops below a threshold.

Eigendecomposition of a covariance matrix

The eigendecomposition of a covariance matrix makes finding the PCs easy.

Recall that $\mathbf{u}$ is an eigenvector, with scalar eigenvalue ω, of a matrix S if

$$S\mathbf{u} = \omega\mathbf{u}$$

$\mathbf{u}$ can have any norm, but we will define it to be unity (i.e., $\mathbf{u}^\mathsf{T}\mathbf{u} = 1$).

For a covariance matrix $S = \langle \mathbf{x}\mathbf{x}^\mathsf{T} \rangle$ (which is $D \times D$, symmetric, positive semi-definite):

- In general there are D eigenvector-eigenvalue pairs $(\mathbf{u}_{(i)}, \omega_{(i)})$, except if two or more eigenvectors share the same eigenvalue (in which case the eigenvectors are degenerate — any linear combination is also an eigenvector).
- The D eigenvectors are orthogonal (or orthogonalisable, if $\omega_{(i)} = \omega_{(j)}$). Thus, they form an orthonormal basis. $\sum_i \mathbf{u}_{(i)}{\mathbf{u}_{(i)}}^\mathsf{T} = I$.
- Any vector $\mathbf{v}$ can be written as

$$\mathbf{v} = \Big(\sum_i \mathbf{u}_{(i)}{\mathbf{u}_{(i)}}^\mathsf{T}\Big)\mathbf{v} = \sum_i ({\mathbf{u}_{(i)}}^\mathsf{T}\mathbf{v})\mathbf{u}_{(i)} = \sum_i v_{(i)}\mathbf{u}_{(i)}$$

- The original matrix S can be written:

$$S = \sum_i \omega_{(i)}\mathbf{u}_{(i)}{\mathbf{u}_{(i)}}^\mathsf{T} = UWU^\mathsf{T}$$

where $U = [\mathbf{u}_{(1)}, \mathbf{u}_{(2)}, \ldots, \mathbf{u}_{(D)}]$ collects the eigenvectors and $W = \mathrm{diag}\big[(\omega_{(1)}, \omega_{(2)}, \ldots, \omega_{(D)})\big]$.

PCA AND EIGENVECTORS

- The variance in direction $\mathbf{u}_{(i)}$ is

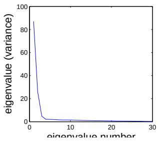
$$\left\langle (\mathbf{x}^\mathsf{T}\mathbf{u}_{(i)})^2 \right\rangle = \left\langle \mathbf{u}_{(i)}{}^\mathsf{T}\mathbf{x}\mathbf{x}^\mathsf{T}\mathbf{u}_{(i)} \right\rangle = \mathbf{u}_{(i)}{}^\mathsf{T} S \mathbf{u}_{(i)} = \mathbf{u}_{(i)}{}^\mathsf{T}\omega_{(i)}\mathbf{u}_{(i)} = \omega_{(i)}$$

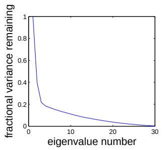
- The variance in an arbitrary direction $\mathbf{v}$ is

$$\left\langle (\mathbf{x}^\mathsf{T}\mathbf{v})^2 \right\rangle = \left\langle \left(\mathbf{x}^\mathsf{T}\left(\sum_i v_{(i)}\mathbf{u}_{(i)}\right)\right)^2 \right\rangle = \sum_{ij} v_{(i)}\mathbf{u}_{(i)}{}^\mathsf{T} S \mathbf{u}_{(j)} v_{(j)}$$

$$= \sum_{ij} v_{(i)}\omega_{(j)}v_{(j)}\mathbf{u}_{(i)}{}^\mathsf{T}\mathbf{u}_{(j)} = \sum_i v_{(i)}^2\omega_{(i)}$$

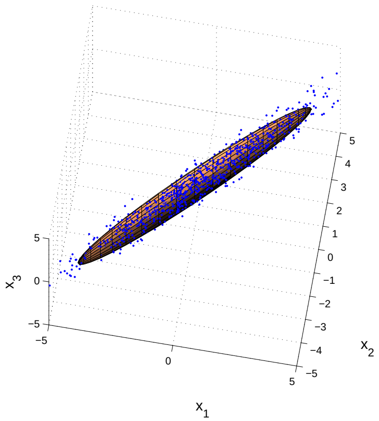
- If $\mathbf{v}^\mathsf{T}\mathbf{v} = 1$, then $\sum_i v_{(i)}^2 = 1$ and so $\arg\max_{\|\mathbf{v}\|=1} \left\langle (\mathbf{x}^\mathsf{T}\mathbf{v})^2 \right\rangle = \mathbf{u}_{(\max)}$
  The direction of greatest variance is the eigenvector the largest eigenvalue.
- In general, the PCs are exactly the eigenvectors of the empirical covariance matrix, ordered by decreasing eigenvalue.
- The eigenspectrum shows how the variance is distributed across dimensions; can identify transitions that might separate signal from noise, or the number of PCs that capture a pre-determined fraction of variance.





PCA SUBSPACE

The K principle components define the K-dimensional subspace of greatest variance.

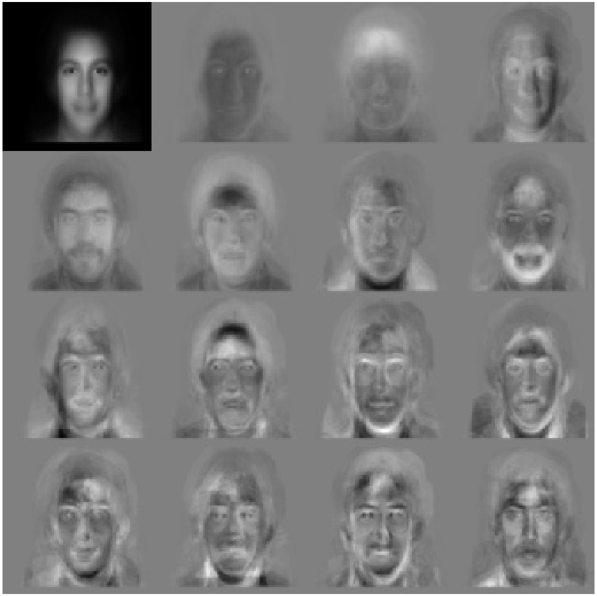


- Each data point $\mathbf{x}_n$ is associated with a projection $\hat{\mathbf{x}}_n$ into the principle subspace.

$$\hat{\mathbf{x}}_n = \sum_{k=1}^{K} x_{n(k)} \boldsymbol{\lambda}_{(k)}$$

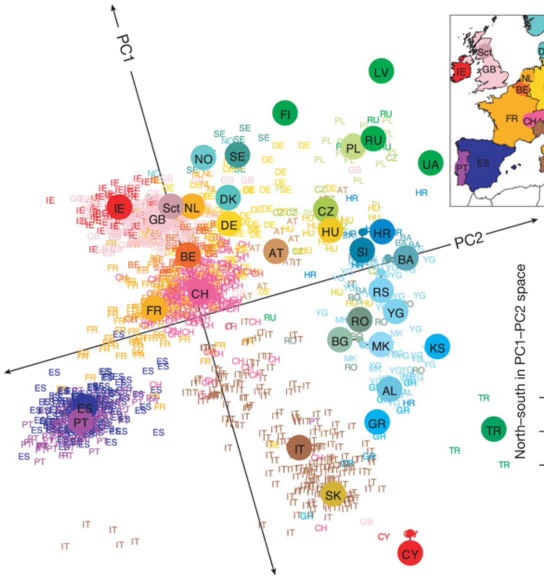
- This can be used for lossy compression, denoising, recognition, ...

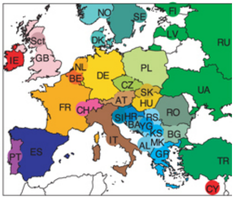
EXAMPLE OF PCA: EIGENFACES

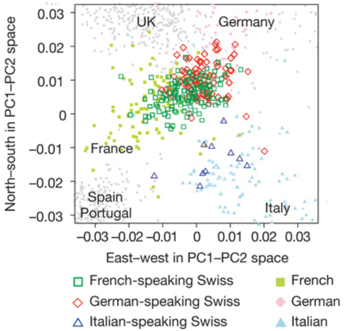


vismod.media.mit.edu/vismod/demos/facerec/basic.html

Example of PCA: Genetic variation within Europe







Novembre et al. (2008) Nature 456:98-101

Equivalent definitions of PCA

- Find K directions of greatest variance in data.
- Find K-dimensional orthogonal projection that *preserves* greatest variance.
- Find K-dimensional vectors $\mathbf{z}_i$ and matrix Λ so that $\hat{\mathbf{x}}_i = \Lambda \mathbf{z}_i$ is as close as possible (in squared distance) to $\mathbf{x}_i$.
- …(many others)

ML LEARNING FOR PPCA

$$\ell = -\frac{N}{2}\log|2\pi C| - \frac{N}{2}\mathsf{Tr}\left[C^{-1}S\right] \qquad \text{where } C = \Lambda\Lambda^{\mathsf{T}} + \psi I$$

$$\frac{\partial\ell}{\partial\Lambda} = \frac{N}{2}\left(-\frac{\partial}{\partial\Lambda}\log|C| - \frac{\partial}{\partial\Lambda}\mathsf{Tr}\left[C^{-1}S\right]\right) = N\left(-C^{-1}\Lambda + C^{-1}SC^{-1}\Lambda\right)$$

So at the stationary points we have $SC^{-1}\Lambda = \Lambda$. This implies either:

- $\Lambda = 0$, which turns out to be a minimum.
- $C = S \Rightarrow \Lambda\Lambda^{\mathsf{T}} = S - \psi I$. Now $\text{rank}(\Lambda\Lambda^{\mathsf{T}}) \leq K \Rightarrow \text{rank}(S - \psi I) \leq K$
  $\Rightarrow S$ has $D - K$ eigenvalues $= \psi$ and Λ aligns with space of remaining eigenvectors.
- or, taking the SVD: $\Lambda = ULV^{\mathsf{T}}$:

$$S(ULV^{\mathsf{T}}VLU^{\mathsf{T}} + \psi I)^{-1}ULV^{\mathsf{T}} = ULV^{\mathsf{T}} \qquad \times VL^{-1}$$

$$\Rightarrow \quad S(UL^2U^{\mathsf{T}} + \psi I)^{-1}U = U \qquad \begin{array}{l} U(L^2 + \psi I) = (UL^2U^{\mathsf{T}} + \psi I)U \\ \Rightarrow (UL^2U^{\mathsf{T}} + \psi I)^{-1}U = U(L^2 + \psi I)^{-1} \end{array}$$

$$\Rightarrow \quad SU(L^2 + \psi I)^{-1} = U \qquad \times (L^2 + \psi I)$$

$$\Rightarrow \quad SU = U\underbrace{(L^2 + \psi I)}_{\text{diagonal}}$$

$\Rightarrow$ columns of U are eigenvectors of S with eigenvalues given by $l_i^2 + \psi$.
Thus, $\Lambda = ULV^{\mathsf{T}}$ spans a space defined by K eigenvectors of S; and the lengths of the column vectors of L are given by the eigenvalues $-\psi$ (V selects an arbitrary basis in the latent space).

Remains to show (we won't, but it's intuitively reasonable) that the global ML solution is attained when Λ aligns with the *K leading* eigenvectors.

PPCA LATENTS

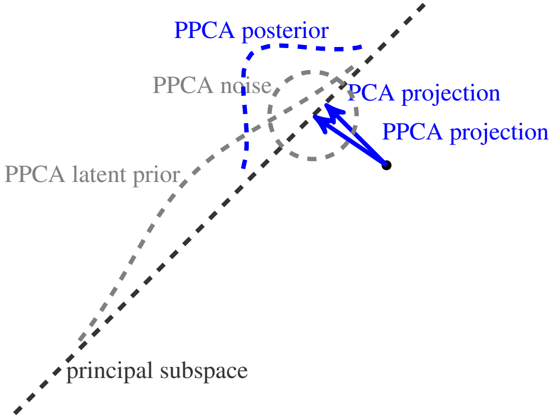
- In PCA the "noise" is orthogonal to the subspace, and we can project $\mathbf{x}_n \to \hat{\mathbf{x}}_n$ trivially.
- In PPCA, the noise is more sensible (equal in all directions). But what is the projection?
  Find the expected value $\bar{\mathbf{z}}_n = \mathbb{E}[\mathbf{z}_n | \mathbf{x}_n]$ and then take $\hat{\mathbf{x}}_n = \Lambda \bar{\mathbf{z}}_n$.
- **Tactic:** write $p(\mathbf{z}_n, \mathbf{x}_n | \theta)$, consider $\mathbf{x}_n$ to be fixed. What is this as a function of $\mathbf{z}_n$?

$$\begin{aligned}
p(\mathbf{z}_n, \mathbf{x}_n) &= p(\mathbf{z}_n) p(\mathbf{x}_n | \mathbf{z}_n) \\
&= (2\pi)^{-\frac{K}{2}} \exp\{-\frac{1}{2}\mathbf{z}_n^\top \mathbf{z}_n\} \, |2\pi\Psi|^{-\frac{1}{2}} \exp\{-\frac{1}{2}(\mathbf{x}_n - \Lambda\mathbf{z}_n)^\top \Psi^{-1} (\mathbf{x}_n - \Lambda\mathbf{z}_n)\} \\
&= \text{c} \times \exp\{-\frac{1}{2}[\mathbf{z}_n^\top \mathbf{z}_n + (\mathbf{x}_n - \Lambda\mathbf{z}_n)^\top \Psi^{-1} (\mathbf{x}_n - \Lambda\mathbf{z}_n)]\} \\
&= \text{c'} \times \exp\{-\frac{1}{2}[\mathbf{z}_n^\top (I + \Lambda^\top \Psi^{-1}\Lambda)\mathbf{z}_n - 2\mathbf{z}_n^\top \Lambda^\top \Psi^{-1}\mathbf{x}_n]\} \\
&= \text{c''} \times \exp\{-\frac{1}{2}[\mathbf{z}_n^\top \Sigma^{-1} \mathbf{z}_n - 2\mathbf{z}_n^\top \Sigma^{-1}\mu + \mu^\top \Sigma^{-1}\mu]\}
\end{aligned}$$

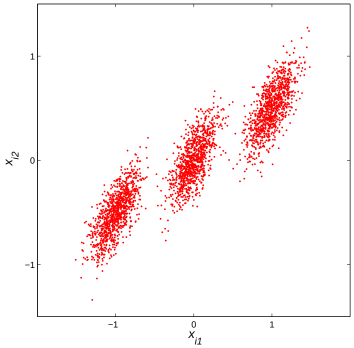
So $\Sigma = (I + \Lambda^\top \Psi^{-1}\Lambda)^{-1} = I - \beta\Lambda$ and $\mu = \Sigma\Lambda^\top \Psi^{-1}\mathbf{x}_n = \beta\mathbf{x}_n$. Where $\beta = \Sigma\Lambda^\top\Psi^{-1}$.

- Thus, $\hat{\mathbf{x}}_n = \Lambda(I + \Lambda^\top\Psi^{-1}\Lambda)^{-1}\Lambda^\top\Psi^{-1}\mathbf{x}_n = \mathbf{x}_n - \Psi(\Lambda\Lambda^\top + \Psi)^{-1}\mathbf{x}_n$
- This is not the same projection. PPCA takes into account noise in the principal subspace.
- As $\psi \to 0$, the PPCA estimate $\to$ the PCA value.

PPCA LATENTS



MIXTURE DISTRIBUTIONS



A mixture distribution has a single discrete latent variable:

$$s_i \overset{\text{iid}}{\sim} \textit{Discrete}[\boldsymbol{\pi}]$$
$$\mathbf{x}_i \mid s_i \sim \mathcal{P}_{s_i}[\theta_{s_i}]$$

Mixtures arise naturally when observations from different sources have been collated. They can also be used to *approximate* arbitrary distributions.

The Mixture Likelihood

The mixture model is

$$s_i \overset{\text{iid}}{\sim} \textit{Discrete}[\boldsymbol{\pi}]$$
$$\mathbf{x}_i \mid s_i \sim \mathcal{P}_{s_i}[\theta_{s_i}]$$

Under the discrete distribution

$$P(s_i = m) = \pi_m; \qquad \pi_m \geq 0,\ \sum_{m=1}^{k} \pi_m = 1$$

Thus, the probability (density) at a single data point $\mathbf{x}_i$ is

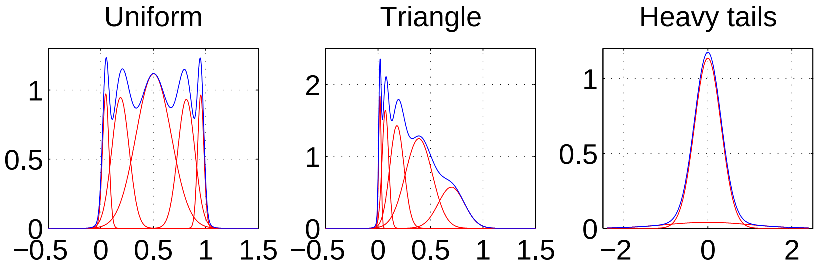
$$\begin{aligned} P(\mathbf{x}_i) &= \sum_{m=1}^{k} P(\mathbf{x}_i \mid s_i = m) P(s_i = m) \\ &= \sum_{m=1}^{k} \pi_m P_m(\mathbf{x}_i; \theta_m) \end{aligned}$$

The mixture distribution (density) is a convex combination (or *weighted average*) of the component distributions (densities).

APPROXIMATION WITH A MIXTURE OF GAUSSIANS (MOG)

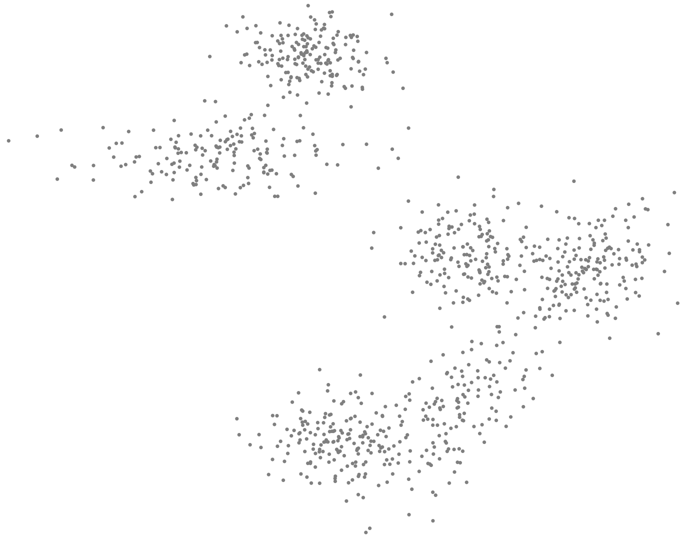
The component densities may be viewed as elements of a *basis* which can be combined to approximate arbitrary distributions.

Here are examples where non-Gaussian densities are modelled (aproximated) as a mixture of Gaussians. The red curves show the (weighted) Gaussians, and the blue curve the resulting density.

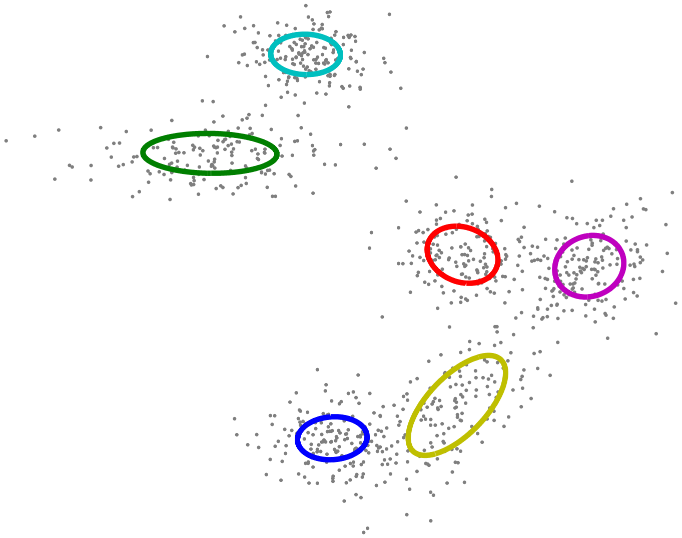


Given enough mixture components we can model (almost) any density (as accurately as desired), but still only need to work with the well-known Gaussian form.

Clustering with a MoG



CLUSTERING WITH A MOG



Clustering with a MoG

In clustering applications, the latent variable s_i represents the (unknown) identity of the cluster to which the ith observation belongs.

Thus, the latent distribution gives the *prior* probability of a data point coming from each cluster.

$$P(s_i = m \mid \pi) = \pi_m$$

Data from the mth cluster are distributed according to the mth component:

$$P(\mathbf{x}_i \mid s_i = m) = P_m(\mathbf{x}_i)$$

Once we observe a data point, the *posterior* probability distribution for the cluster it belongs to is

$$P(s_i = m \mid \mathbf{x}_i) = \frac{P_m(\mathbf{x}_i)\pi_m}{\sum_m P_m(\mathbf{x}_i)\pi_m}$$

This is often called the responsibility of the mth cluster for the ith data point.

The MoG likelihood

Each component of a MoG is a Gaussian, with mean $\boldsymbol{\mu}_m$ and covariance matrix Σ_m. Thus, the probability density evaluated at a set of n iid observations, $\mathcal{D} = \{\mathbf{x}_1 \dots \mathbf{x}_n\}$ (i.e. the likelihood) is

$$\begin{aligned} p(\mathcal{D} \mid \{\boldsymbol{\mu}_m\}, \{\Sigma_m\}, \boldsymbol{\pi}) &= \prod_{i=1}^{n} \sum_{m=1}^{k} \pi_m \mathcal{N}\left(\mathbf{x}_i \mid \boldsymbol{\mu}_m, \Sigma_m\right) \\ &= \prod_{i=1}^{n} \sum_{m=1}^{k} \pi_m \frac{1}{\sqrt{|2\pi\Sigma_m|}} e^{-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_m)^{\mathsf{T}} \Sigma_m^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_m)} \end{aligned}$$

The log of the likelihood is

$$\log p(\mathcal{D} \mid \{\boldsymbol{\mu}_m\}, \{\Sigma_m\}, \boldsymbol{\pi}) = \sum_{i=1}^{n} \log \sum_{m=1}^{k} \pi_m \frac{1}{\sqrt{|2\pi\Sigma_m|}} e^{-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_m)^{\mathsf{T}} \Sigma_m^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_m)}$$

Note that the logarithm fails to simplify the component density terms. A mixture distribution does not lie in the exponential family. Direct optimisation is not easy.

Maximum Likelihood for a Mixture Model

The log likelihood is:

$$\mathcal{L} = \sum_{i=1}^{n} \log \sum_{m=1}^{k} \pi_m P_m(\mathbf{x}_i; \theta_m)$$

Its partial derivative wrt θ_m is

$$\frac{\partial \mathcal{L}}{\partial \theta_m} = \sum_{i=1}^{n} \frac{\pi_m}{\sum_{m=1}^{k} \pi_m P_m(\mathbf{x}_i; \theta_m)} \frac{\partial P_m(\mathbf{x}_i; \theta_m)}{\partial \theta_m}$$

or, using $\partial P / \partial \theta = P \times \partial \log P / \partial \theta$,

$$= \sum_{i=1}^{n} \underbrace{\frac{\pi_m P_m(\mathbf{x}_i; \theta_m)}{\sum_{m=1}^{k} \pi_m P_m(\mathbf{x}_i; \theta_m)}} \frac{\partial \log P_m(\mathbf{x}_i; \theta_m)}{\partial \theta_m}$$

$$= \sum_{i=1}^{n} r_{im} \frac{\partial \log P_m(\mathbf{x}_i; \theta_m)}{\partial \theta_m}$$

And its partial derivative wrt π_m is

$$\frac{\partial \mathcal{L}}{\partial \pi_m} = \sum_{i=1}^{n} \frac{P_m(\mathbf{x}_i; \theta_m)}{\sum_{m=1}^{k} \pi_m P_m(\mathbf{x}_i; \theta_m)} = \sum_{i=1}^{n} \frac{r_{im}}{\pi_m}$$

MoG Derivatives

For a MoG, with $\theta_m = \{\boldsymbol{\mu}_m, \Sigma_m\}$ we get

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\mu}_m} = \sum_{i=1}^{n} r_{im} \Sigma_m^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_m)$$

$$\frac{\partial \mathcal{L}}{\partial \Sigma_m^{-1}} = \frac{1}{2} \sum_{i=1}^{n} r_{im} \left(\Sigma_m - (\mathbf{x}_i - \boldsymbol{\mu}_m)(\mathbf{x}_i - \boldsymbol{\mu}_m)^{\mathsf{T}} \right)$$

These equations can be used (along with Lagrangian derivatives wrt π_m that enforce normalisation) for gradient based learning; e.g., taking small steps in the direction of the gradient (or using conjugate gradients).

The K-means Algorithm

The K-means algorithm is a limiting case of the mixture of Gaussians (c.f. PCA and Factor Analysis).

Take $\pi_m = 1/k$ and $\Sigma_m = \sigma^2 I$, with $\sigma^2 \to 0$. Then the responsibilities become binary

$$r_{im} \to \delta(m, \arg\min_l \|\mathbf{x}_i - \boldsymbol{\mu}_l\|^2)$$

with 1 for the component with the closest mean and 0 for all other components. We can then solve directly for the means by setting the gradient to 0.

The k-means algorithm iterates these two steps:

- assign each point to its closest mean $\left(\text{set } r_{im} = \delta(m, \arg\min_l \|\mathbf{x}_i - \boldsymbol{\mu}_l\|^2)\right)$
- update the means to the average of their assigned points $\left(\text{set } \boldsymbol{\mu}_m = \frac{\sum_i r_{im}\mathbf{x}_i}{\sum_i r_{im}}\right)$

This usually converges within a few iterations, although the fixed point depends on the initial values chosen for $\boldsymbol{\mu}_m$. The algorithm has no learning rate, but the assumptions are quite limiting.

ILLUSTRATION: k-MEANS

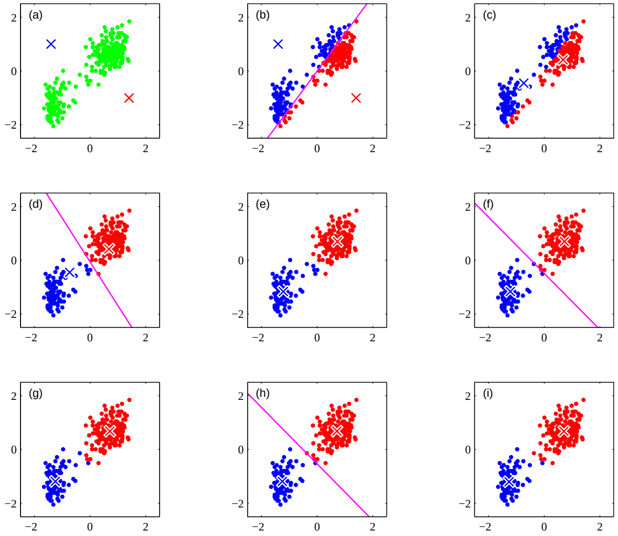


Illustration: Bishop, “Pattern recognition and machine learning”

A PREVIEW OF THE EM ALGORITHM

We wrote the k-means algorithm in terms of binary responsibilities. Suppose, instead, we used

the fractional responsibilities from the full (non-limiting) MoG, but still neglected the dependence of the responsibilities on the parameters. We could then solve for both $\boldsymbol{\mu}_m$ and Σ_m.

The EM algorithm for MoGs iterates these two steps:

- Evaluate the responsibilities for each point given the current parameters.
- Optimise the parameters assuming the responsibilities stay fixed:

$$\boldsymbol{\mu}_m = \frac{\sum_i r_{im}\mathbf{x}_i}{\sum_i r_{im}} \quad \text{and} \quad \Sigma_m = \frac{\sum_i r_{im}(\mathbf{x}_i - \boldsymbol{\mu}_m)(\mathbf{x}_i - \boldsymbol{\mu}_m)^\mathsf{T}}{\sum_i r_{im}}$$

Although this appears *ad hoc*, we will see (later) that it is a special case of a general algorithm, and is actually guaranteed to increase the likelihood at each iteration.

ISSUES

There are several problems with these algorithms:

- slow convergence for the gradient based method
- gradient based method may develop invalid covariance matrices
- local minima; the end configuration may depend on the starting state
- how do you adjust k? Using the likelihood alone is no good.
- singularities; components with a single data point will have their covariance going to zero and the likelihood will tend to infinity.

We will attempt to address many of these as the course goes on.

Expectation Maximisation

LOG-LIKELIHOODS

- Exponential family models: $p(\mathbf{x}|\boldsymbol{\theta}) = f(\mathbf{x})e^{\boldsymbol{\theta}^\mathsf{T}\mathbf{T}(\mathbf{x})}/Z(\boldsymbol{\theta})$

$$\ell(\theta) = \theta^\mathsf{T} \sum_n T(\mathbf{x}_n) - N \log Z(\theta) \quad (+\text{ constants})$$

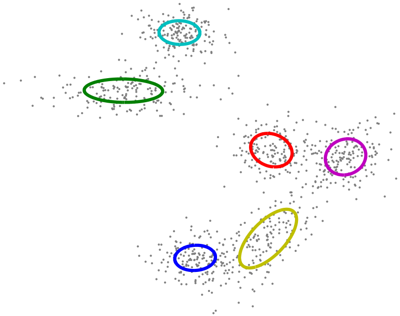
  - Concave function.
  - Maximum may be closed-form.
  - If not, numerical optimisation is still generally straightforward.

- Latent variable models: $p(\mathbf{x}|\boldsymbol{\theta}_x, \boldsymbol{\theta}_z) = \int d\mathbf{z}\, \underbrace{f_x(\mathbf{x}) \frac{e^{\boldsymbol{\phi}(\boldsymbol{\theta}_x,\mathbf{z})^\mathsf{T}\mathbf{T}_x(\mathbf{x})}}{Z_x(\boldsymbol{\phi}(\boldsymbol{\theta}_x,\mathbf{z}))}}_{p(\mathbf{x}|\mathbf{z},\boldsymbol{\theta}_x)} \underbrace{f_z(\mathbf{z}) \frac{e^{\boldsymbol{\theta}_z^\mathsf{T}\mathbf{T}_z(\mathbf{z})}}{Z_z(\boldsymbol{\theta}_z)}}_{p(\mathbf{z}|\boldsymbol{\theta}_z)}$

$$\ell(\theta_x, \theta_z) = \sum_n \log \int d\mathbf{z} f_x(\mathbf{x}) \frac{e^{\boldsymbol{\phi}(\boldsymbol{\theta}_x,\mathbf{z})^\mathsf{T}\mathbf{T}_x(\mathbf{x})}}{Z_x(\boldsymbol{\phi}(\boldsymbol{\theta}_x,\mathbf{z}))} f_z(\mathbf{z}) \frac{e^{\boldsymbol{\theta}_z^\mathsf{T}\mathbf{T}_z(\mathbf{z})}}{Z_z(\boldsymbol{\theta}_z)}$$

  - Usually no closed form optimum.
  - Often multiple local maxima.
  - Direct numerical optimisation may be possible but infrequently easy.

EXAMPLE: MIXTURE OF GAUSSIANS



Data: $\mathcal{X} = \{\mathbf{x}_1 \dots \mathbf{x}_N\}$

Latent process:

$$s_i \overset{\text{iid}}{\sim} \text{Disc}[\boldsymbol{\pi}]$$

Component distributions:

$$\mathbf{x}_i \mid (s_i = m) \sim \mathcal{P}_m[\theta_m] = \mathcal{N}(\boldsymbol{\mu}_m, \Sigma_m)$$

Marginal distribution:

$$P(\mathbf{x}_i) = \sum_{m=1}^{k} \pi_m P_m(\mathbf{x}; \theta_m)$$

Log-likelihood:

$$\ell(\{\boldsymbol{\mu}_m\}, \{\Sigma_m\}, \boldsymbol{\pi}) = \sum_{i=1}^{n} \log \sum_{m=1}^{k} \frac{\pi_m}{\sqrt{|2\pi\Sigma_m|}} e^{-\frac{1}{2}(\mathbf{x}_i - \boldsymbol{\mu}_m)^\mathsf{T} \Sigma_m^{-1} (\mathbf{x}_i - \boldsymbol{\mu}_m)}$$

THE JOINT-DATA LIKELIHOOD AND EM

- For many models, maximisation might be straightforward if $\mathbf{z}$ were not latent, and we could just maximise the joint-data likelihood:

$$\ell(\theta_x, \theta_z) = \sum_n \phi(\boldsymbol{\theta}_x, \mathbf{z}_n)^{\mathsf{T}} \mathbf{T}_x(\mathbf{x}_n) + \boldsymbol{\theta}_z^{\mathsf{T}} \sum_n \mathbf{T}_z(\mathbf{z}_n) - \sum_n \log Z_x(\phi(\boldsymbol{\theta}_x, \mathbf{z}_n)) - N \log Z_z(\boldsymbol{\theta}_z)$$

- Conversely, if we knew θ, we might easily compute (the posterior over) the values of $\mathbf{z}$.

- Idea: update $\boldsymbol{\theta}$ and (the distribution on) $\mathbf{z}$ in alternation, to reach a self-consistent answer. Will this yield the right answer?

- Typically, it will (as we shall see). This is the Expectation Maximisation (EM) algorithm.

THE EXPECTATION MAXIMISATION (EM) ALGORITHM

The EM algorithm (Dempster, Laird & Rubin, 1977; but significant earlier precedents) finds a (local) maximum of a latent variable model likelihood.

Start from arbitrary values of the parameters, and iterate two steps:

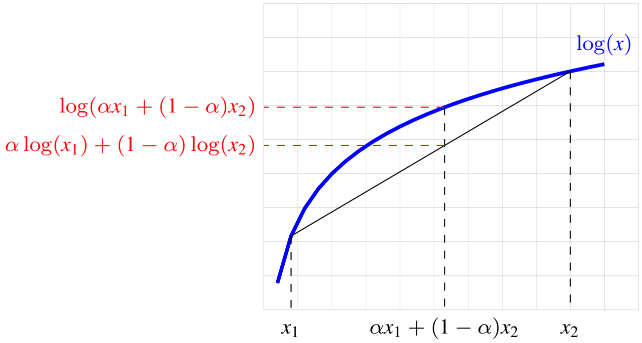
E step: Fill in values of latent variables according to posterior given data.

M step: Maximise likelihood as if latent variables were not hidden.

- Decomposes difficult problems into series of tractable steps.
- An alternative to gradient-based iterative methods.
- No learning rate.
- In ML, the E step is called inference, and the M step learning. In stats, these are often imputation and inference or estimation.
- Not essential for simple models (like MoGs/FA), though often more efficient than alternatives. Crucial for learning in complex settings.
- Provides a framework for principled approximations.

JENSEN'S INEQUALITY

One view: EM iteratively refines a lower bound on the log-likelihood.



$\log(x)$
$\log(\alpha x_1 + (1-\alpha)x_2)$
$\alpha \log(x_1) + (1-\alpha)\log(x_2)$
x_1
$\alpha x_1 + (1-\alpha)x_2$
x_2

In general:

For $\alpha_i \geq 0$, $\sum \alpha_i = 1$ (and $\{x_i > 0\}$):

$$\log\left(\sum_i \alpha_i x_i\right) \geq \sum_i \alpha_i \log(x_i)$$

For probability measure α and concave f

$$f\left(\mathbb{E}_\alpha[x]\right) \geq \mathbb{E}_\alpha[f(x)]$$

Equality (if and) only if $f(x)$ is almost surely constant or linear on (convex) support of α.

THE LOWER BOUND FOR EM – “FREE ENERGY”

Observed data $\mathcal{X} = \{\mathbf{x}_i\}$; Latent variables $\mathcal{Z} = \{\mathbf{z}_i\}$; Parameters $\theta = \{\theta_x, \theta_z\}$.

Log-likelihood:

$$\ell(\theta) = \log P(\mathcal{X}|\theta) = \log \int d\mathcal{Z}\ P(\mathcal{Z}, \mathcal{X}|\theta)$$

By Jensen, any distribution, $q(\mathcal{Z})$, over the latent variables generates a lower bound:

$$\ell(\theta) = \log \int d\mathcal{Z}\ q(\mathcal{Z}) \frac{P(\mathcal{Z}, \mathcal{X}|\theta)}{q(\mathcal{Z})} \geq \int d\mathcal{Z}\ q(\mathcal{Z}) \log \frac{P(\mathcal{Z}, \mathcal{X}|\theta)}{q(\mathcal{Z})} \overset{\text{def}}{=} \mathcal{F}(q, \theta).$$

Now,

$$\int d\mathcal{Z}\ q(\mathcal{Z}) \log \frac{P(\mathcal{Z}, \mathcal{X}|\theta)}{q(\mathcal{Z})} = \int d\mathcal{Z}\ q(\mathcal{Z}) \log P(\mathcal{Z}, \mathcal{X}|\theta) - \int d\mathcal{Z}\ q(\mathcal{Z}) \log q(\mathcal{Z})$$

$$= \int d\mathcal{Z}\ q(\mathcal{Z}) \log P(\mathcal{Z}, \mathcal{X}|\theta) + \mathbf{H}[q],$$

where $\mathbf{H}[q]$ is the entropy of $q(\mathcal{Z})$.

So:

$$\mathcal{F}(q, \theta) = \langle \log P(\mathcal{Z}, \mathcal{X}|\theta) \rangle_{q(\mathcal{Z})} + \mathbf{H}[q]$$

THE E AND M STEPS OF EM

The free-energy lower bound on $\ell(\theta)$ is a function of θ and a distribution q:

$$\mathcal{F}(q,\theta) = \langle \log P(\mathcal{Z},\mathcal{X}|\theta)\rangle_{q(\mathcal{Z})} + \mathbf{H}[q],$$

The EM steps can be re-written:

- **E step:** optimize $\mathcal{F}(q,\theta)$ wrt distribution over hidden variables holding parameters fixed:

$$q^{(k)}(\mathcal{Z}) := \underset{q(\mathcal{Z})}{\arg\max}\ \mathcal{F}\big(q(\mathcal{Z}), \theta^{(k-1)}\big).$$

- **M step:** maximize $\mathcal{F}(q,\theta)$ wrt parameters holding hidden distribution fixed:

$$\theta^{(k)} := \underset{\theta}{\arg\max}\ \mathcal{F}\big(q^{(k)}(\mathcal{Z}), \theta\big) = \underset{\theta}{\arg\max}\ \langle \log P(\mathcal{Z},\mathcal{X}|\theta)\rangle_{q^{(k)}(\mathcal{Z})}$$

The second equality comes from the fact that $\mathbf{H}\big[q^{(k)}(\mathcal{Z})\big]$ does not depend directly on θ.

The E Step

The free energy can be re-written

$$\begin{aligned}\mathcal{F}(q,\theta) &= \int q(\mathcal{Z}) \log \frac{P(\mathcal{Z},\mathcal{X}|\theta)}{q(\mathcal{Z})}\, d\mathcal{Z} \\ &= \int q(\mathcal{Z}) \log \frac{P(\mathcal{Z}|\mathcal{X},\theta)P(\mathcal{X}|\theta)}{q(\mathcal{Z})}\, d\mathcal{Z} \\ &= \int q(\mathcal{Z}) \log P(\mathcal{X}|\theta)\, d\mathcal{Z} + \int q(\mathcal{Z}) \log \frac{P(\mathcal{Z}|\mathcal{X},\theta)}{q(\mathcal{Z})}\, d\mathcal{Z} \\ &= \ell(\theta) - \mathbf{KL}[q(\mathcal{Z})\|P(\mathcal{Z}|\mathcal{X},\theta)]\end{aligned}$$

The second term is the Kullback-Leibler divergence.

This means that, for fixed θ, $\mathcal{F}$ is bounded above by ℓ, and achieves that bound when $\mathbf{KL}[q(\mathcal{Z})\|P(\mathcal{Z}|\mathcal{X},\theta)] = 0$.

But $\mathbf{KL}[q\|p]$ is zero if and only if $q = p$ (see appendix.)

So, the E step sets

$$q^{(k)}(\mathcal{Z}) = P(\mathcal{Z}|\mathcal{X},\theta^{(k-1)}) \qquad \text{[inference / imputation]}$$

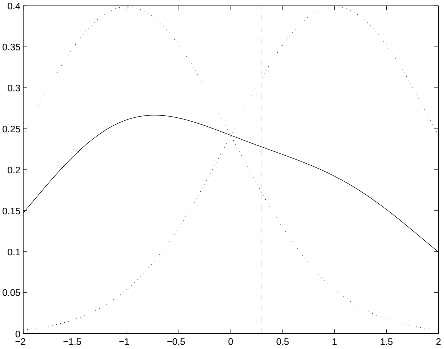
and, after an E step, the free energy equals the likelihood.

COORDINATE ASCENT IN $\mathcal{F}$ (DEMO)

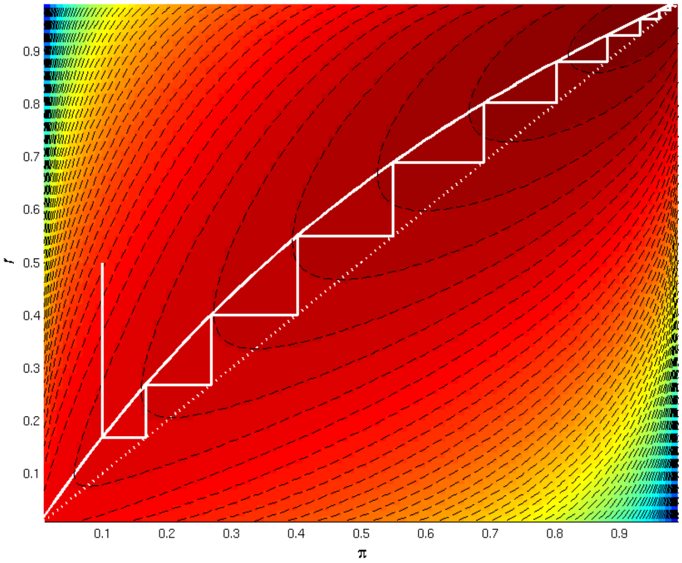
To visualise, we consider a one parameter / one latent mixture:

$$s \sim \text{Bernoulli}[\pi]$$
$$x|s = 0 \sim \mathcal{N}[-1, 1] \qquad x|s = 1 \sim \mathcal{N}[1, 1] \, .$$

Single data point $x_1 = .3$.
$q(s)$ is a distribution on a single binary latent, and so is represented by $r_1 \in [0, 1]$.



Coordinate Ascent in $\mathcal{F}$ (Demo)



EM Never Decreases the Likelihood

The E and M steps together never decrease the log likelihood:

$$\ell(\theta^{(k-1)}) \underset{\text{E step}}{=} \mathcal{F}(q^{(k)}, \theta^{(k-1)}) \underset{\text{M step}}{\leq} \mathcal{F}(q^{(k)}, \theta^{(k)}) \underset{\text{Jensen}}{\leq} \ell(\theta^{(k)}),$$

- The E step brings the free energy to the likelihood.
- The M-step maximises the free energy wrt θ.
- $\mathcal{F} \leq \ell$ by Jensen – or, equivalently, from the non-negativity of KL

If the M-step is executed so that $\theta^{(k)} \neq \theta^{(k-1)}$ iff $\mathcal{F}$ increases, then the overall EM iteration will step to a new value of θ iff the likelihood increases.

Can also show that fixed points of EM (generally) correspond to maxima of the likelihood (see appendices).

EM Summary

- An iterative algorithm that finds (local) maxima of the likelihood of a latent variable model.

$$\ell(\theta) = \log P(\mathcal{X}|\theta) = \log \int d\mathcal{Z}\; P(\mathcal{X}|\mathcal{Z},\theta)P(\mathcal{Z}|\theta)$$

- Increases a variational lower bound on the likelihood by coordinate ascent.

$$\mathcal{F}(q,\theta) = \langle \log P(\mathcal{Z},\mathcal{X}|\theta)\rangle_{q(\mathcal{Z})} + \mathbf{H}[q] = \ell(\theta) - \mathbf{KL}[q(\mathcal{Z})\|P(\mathcal{Z}|\mathcal{X})] \leq \ell(\theta)$$

- **E step:**

$$q^{(k)}(\mathcal{Z}) := \operatorname*{arg\,max}_{q(\mathcal{Z})}\; \mathcal{F}\big(q(\mathcal{Z}), \theta^{(k-1)}\big) = P(\mathcal{Z}|\mathcal{X},\theta^{(k-1)})$$

- **M step:**

$$\theta^{(k)} := \operatorname*{arg\,max}_{\theta}\; \mathcal{F}\big(q^{(k)}(\mathcal{Z}), \theta\big) = \operatorname*{arg\,max}_{\theta}\; \langle \log P(\mathcal{Z},\mathcal{X}|\theta)\rangle_{q^{(k)}(\mathcal{Z})}$$

- After E-step $\mathcal{F}(q,\theta) = \ell(\theta) \Rightarrow$ maximum of free-energy is maximum of likelihood.

Partial M steps and Partial E steps

Partial M steps: The proof holds even if we just *increase* $\mathcal{F}$ wrt θ rather than maximize. (Dempster, Laird and Rubin (1977) call this the generalized EM, or GEM, algorithm).

In fact, immediately after an E step

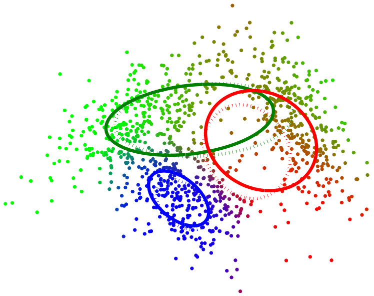
$$\frac{\partial}{\partial\theta}\bigg|_{\theta^{(k-1)}} \langle \log P(\mathcal{X}, \mathcal{Z}|\theta)\rangle_{q^{(k)}(\mathcal{Z})[=P(\mathcal{Z}|\mathcal{X},\theta^{(k-1)})]} = \frac{\partial}{\partial\theta}\bigg|_{\theta^{(k-1)}} \log P(\mathcal{X}|\theta)$$

[cf. mixture gradients from last lecture.] So E-step (inference) can be used to construct other gradient-based optimisation schemes (e.g. "Expectation Conjugate Gradient", Salakhutdinov et al. *ICML* 2003).

Partial E steps: We can also just *increase* $\mathcal{F}$ wrt to some of the qs.

For example, sparse or online versions of the EM algorithm would compute the posterior for a subset of the data points or as the data arrives, respectively. One might also update the posterior over a subset of the hidden variables, while holding others fixed...

EM for MoGs



- Evaluate responsibilities

$$r_{im} = \frac{P_m(\mathbf{x}_i)\pi_m}{\sum_{m'} P_{m'}(\mathbf{x}_i)\pi_{m'}}$$

- Update parameters

$$\boldsymbol{\mu}_m \leftarrow \frac{\sum_i r_{im}\mathbf{x}_i}{\sum_i r_{im}}$$

$$\Sigma_m \leftarrow \frac{\sum_i r_{im}(\mathbf{x}_i - \boldsymbol{\mu}_m)(\mathbf{x}_i - \boldsymbol{\mu}_m)^\mathsf{T}}{\sum_i r_{im}}$$

$$\pi_m \leftarrow \frac{\sum_i r_{im}}{N}$$

The Gaussian mixture model (E-step)

In a univariate Gaussian mixture model, the density of a data point x is:

$$p(x|\theta) = \sum_{m=1}^{k} p(s=m|\theta)p(x|s=m,\theta) \propto \sum_{m=1}^{k} \frac{\pi_m}{\sigma_m} \exp\Big\{-\frac{1}{2\sigma_m^2}(x-\mu_m)^2\Big\},$$

where θ is the collection of parameters: means μ_m, variances σ_m^2 and mixing proportions $\pi_m = p(s=m|\theta)$.

The hidden variable s_i indicates which component generated observation x_i.

The E-step computes the posterior for s_i given the current parameters:

$$q(s_i) = p(s_i|x_i,\theta) \propto p(x_i|s_i,\theta)p(s_i|\theta)$$

$$r_{im} \overset{\text{def}}{=} q(s_i=m) \propto \frac{\pi_m}{\sigma_m} \exp\Big\{-\frac{1}{2\sigma_m^2}(x_i-\mu_m)^2\Big\} \quad \text{(responsibilities)} \quad \leftarrow \langle\delta_{s_i=m}\rangle_q$$

with the normalization such that $\sum_m r_{im} = 1$.

The Gaussian mixture model (M-step)

In the M-step we optimize the sum (since s is discrete):

$$E = \langle \log p(x,s|\theta) \rangle_{q(s)} = \sum q(s) \log[p(s|\theta)\, p(x|s,\theta)]$$
$$= \sum_{i,m} r_{im} \Big[\log \pi_m - \log \sigma_m - \frac{1}{2\sigma_m^2}(x_i - \mu_m)^2 \Big].$$

Optimum is found by setting the partial derivatives of E to zero:

$$\frac{\partial}{\partial \mu_m} E = \sum_i r_{im} \frac{(x_i - \mu_m)}{2\sigma_m^2} = 0 \quad \Rightarrow \quad \mu_m = \frac{\sum_i r_{im} x_i}{\sum_i r_{im}},$$

$$\frac{\partial}{\partial \sigma_m} E = \sum_i r_{im} \Big[-\frac{1}{\sigma_m} + \frac{(x_i - \mu_m)^2}{\sigma_m^3} \Big] = 0 \quad \Rightarrow \quad \sigma_m^2 = \frac{\sum_i r_{im} (x_i - \mu_m)^2}{\sum_i r_{im}},$$

$$\frac{\partial}{\partial \pi_m} E = \sum_i r_{im} \frac{1}{\pi_m}, \qquad \frac{\partial E}{\partial \pi_m} + \lambda = 0 \quad \Rightarrow \quad \pi_m = \frac{1}{n} \sum_i r_{im},$$

where λ is a Lagrange multiplier ensuring that the mixing proportions sum to unity.

EM FOR EXPONENTIAL FAMILIES

EM is often applied to models whose **joint** over $\boldsymbol{\xi} = (\mathbf{z}, \mathbf{x})$ has exponential-family form:

$$p(\boldsymbol{\xi}|\theta) = f(\boldsymbol{\xi}) \exp\{\theta^{\mathsf{T}} \mathsf{T}(\boldsymbol{\xi})\}/Z(\theta)$$

$\big($with $Z(\theta) = \int f(\boldsymbol{\xi}) \exp\{\theta^{\mathsf{T}} \mathsf{T}(\boldsymbol{\xi})\} d\boldsymbol{\xi}\big)$ but whose marginal $p(\mathbf{x}) \notin$ *ExpFam*.
The free energy dependence on θ is given by:

$$\begin{aligned}
\mathcal{F}(q, \theta) &= \int q(\mathbf{z}) \log p(\mathbf{z}, \mathbf{x}|\theta) d\mathbf{z} + \mathbf{H}[q] \\
&= \int q(\mathbf{z}) \big[\theta^{\mathsf{T}} \mathsf{T}(\mathbf{z}, \mathbf{x}) - \log Z(\theta)\big] d\mathbf{z} + \text{const wrt } \theta \\
&= \theta^{\mathsf{T}} \langle \mathsf{T}(\mathbf{z}, \mathbf{x}) \rangle_{q(\mathbf{z})} - \log Z(\theta) + \text{const wrt } \theta
\end{aligned}$$

So, in the **E step** all we need to compute are the expected sufficient statistics under q.
We also have:

$$\begin{aligned}
\frac{\partial}{\partial \theta} \log Z(\theta) &= \frac{1}{Z(\theta)} \frac{\partial}{\partial \theta} Z(\theta) = \frac{1}{Z(\theta)} \frac{\partial}{\partial \theta} \int f(\boldsymbol{\xi}) \exp\{\theta^{\mathsf{T}} \mathsf{T}(\boldsymbol{\xi})\} \\
&= \int \tfrac{1}{Z(\theta)} f(\boldsymbol{\xi}) \exp\{\theta^{\mathsf{T}} \mathsf{T}(\boldsymbol{\xi})\} \cdot \mathsf{T}(\boldsymbol{\xi}) = \langle \mathsf{T}(\boldsymbol{\xi}) \rangle_{p(\boldsymbol{\xi}|\theta)}
\end{aligned}$$

Thus, the **M step** solves:
$$\frac{\partial \mathcal{F}}{\partial \theta} = \langle \mathsf{T}(\mathbf{z}, \mathbf{x}) \rangle_{q(\mathbf{z})} - \langle \mathsf{T}(\mathbf{z}, \mathbf{x}) \rangle_{p(\boldsymbol{\xi}|\theta)} = 0$$

EM FOR EXPONENTIAL FAMILY MIXTURES

To derive EM formally for models with discrete latents (including mixtures) it is useful to introduce an indicator vector $\mathbf{s}$ in place of the discrete s.

$$s_i = m \quad \Leftrightarrow \quad \mathbf{s}_i = [0, 0, \ldots, \underbrace{1}_{m\text{th position}}, \ldots 0]$$

Collecting the M component distributions' natural params into a matrix $\Theta = [\boldsymbol{\theta}_m]$:

$$\log P(\mathcal{X}, \mathcal{S}) = \sum_i \left[(\log \boldsymbol{\pi})^\mathsf{T} \mathbf{s}_i + \mathbf{s}_i^\mathsf{T} \Theta^\mathsf{T} T(\mathbf{x}_i) - \mathbf{s}_i^\mathsf{T} \log \mathbf{Z}(\Theta) \right] + const$$

where $\log \mathbf{Z}(\Theta)$ collects the log-normalisers for all components into an M-element vector. Then, the expected sufficient statistics (E-step) are:

$$\sum_i \langle \mathbf{s}_i \rangle_q \qquad \text{(responsibilities } r_{im}\text{)}$$

$$\sum_i T(\mathbf{x}_i) \left\langle \mathbf{s}_i^\mathsf{T} \right\rangle_q \qquad \text{(responsibility-weighted sufficient stats)}$$

And maximisation of the expected log-joint (M-step) gives:

$$\boldsymbol{\pi}^{(k+1)} \propto \sum_i \langle \mathbf{s}_i \rangle_q$$

$$\left\langle T(\mathbf{x}) | \boldsymbol{\theta}_m^{(k+1)} \right\rangle = \left(\sum_i T(\mathbf{x}_i) \langle [\mathbf{s}_i]_m \rangle_q \right) \Big/ \left(\sum_i \langle [\mathbf{s}_i]_m \rangle_q \right)$$

EM FOR MAP

What if we have a prior?

$$p(\boldsymbol{\xi}|\theta) = f(\boldsymbol{\xi}) \exp\{\theta^{\mathsf{T}} \mathsf{T}(\boldsymbol{\xi})\}/Z(\theta) \qquad p(\theta) = F(\nu, \boldsymbol{\tau}) \exp\{\theta^{\mathsf{T}} \boldsymbol{\tau}\}/Z(\theta)^{\nu}$$

Augment the free energy by adding the log prior:

$$\begin{aligned} \mathcal{F}_{\text{MAP}}(q, \theta) &= \int q(\mathcal{Z}) \log p(\mathcal{Z}, \mathcal{X}, \theta) d\mathcal{Z} + \mathbf{H}[q] \quad \leq \log P(\mathcal{X}|\theta) + \log P(\theta) \\ &= \int q(\mathcal{Z}) \big[\theta^{\mathsf{T}} (\sum_i \mathsf{T}(\boldsymbol{\xi}_i) + \boldsymbol{\tau}) - (N+\nu) \log Z(\theta)\big] d\mathcal{Z} + \text{const wrt } \theta \\ &= \theta^{\mathsf{T}} (\langle \mathsf{T}(\boldsymbol{\xi}) \rangle_{q(\mathbf{z})} + \boldsymbol{\tau}) - (N+\nu) \log Z(\theta) + \text{const wrt } \theta \end{aligned}$$

So, the expected sufficient statistics in the E step are unchanged.

Thus, after an E-step the augmented free-energy equals the log-joint, and so free-energy maxima are log-joint maxima (i.e. MAP values).

Can we find posteriors? Only approximately – we'll return to this later as "Variational Bayes".

REFERENCES

- A. P. Dempster, N. M. Laird and D. B. Rubin (1977).
  Maximum Likelihood from Incomplete Data via the EM Algorithm. Journal of the Royal Statistical Society. Series B (Methodological), Vol. 39, No. 1 (1977), pp. 1-38.

  `http://www.jstor.org/stable/2984875`

- R. M. Neal and G. E. Hinton (1998).
  A view of the EM algorithm that justifies incremental, sparse, and other variants.
  In M. I. Jordan (editor) Learning in Graphical Models, pp. 355-368, Dordrecht: Kluwer Academic Publishers.

  `http://www.cs.utoronto.ca/∼radford/ftp/emk.pdf`

- R. Salakhutdinov, S. Roweis and Z. Ghahramani, (2003).
  Optimization with EM and expectation-conjugate-gradient.
  In ICML (pp. 672-679).

  `http://www.cs.utoronto.ca/∼rsalakhu/papers/emecg.pdf`

- Z. Ghahramani and G. E. Hinton (1996).
  The EM Algorithm for Mixtures of Factor Analyzers.
  University of Toronto Technical Report CRG-TR-96-1.

  `http://learning.eng.cam.ac.uk/zoubin/papers/tr-96-1.pdf`

$\mathbf{KL}[q(x)\|p(x)] \geq 0$, WITH EQUALITY IFF $\forall x: \; p(x) = q(x)$

First consider discrete distributions; the Kullback-Liebler divergence is:

$$\mathbf{KL}[q\|p] = \sum_i q_i \log \frac{q_i}{p_i}.$$

To minimize wrt distribution q we need a Lagrange multiplier to enforce normalisation:

$$E \overset{\text{def}}{=} \mathbf{KL}[q\|p] + \lambda\big(1 - \sum_i q_i\big) = \sum_i q_i \log \frac{q_i}{p_i} + \lambda\big(1 - \sum_i q_i\big)$$

Find conditions for stationarity

$$\left.\begin{aligned} \frac{\partial E}{\partial q_i} &= \log q_i - \log p_i + 1 - \lambda = 0 \Rightarrow q_i = p_i \exp(\lambda - 1) \\ \frac{\partial E}{\partial \lambda} &= 1 - \sum_i q_i = 0 \Rightarrow \sum_i q_i = 1 \end{aligned}\right\} \Rightarrow q_i = p_i.$$

Check sign of curvature (Hessian):

$$\frac{\partial^2 E}{\partial q_i \partial q_i} = \frac{1}{q_i} > 0, \qquad \frac{\partial^2 E}{\partial q_i \partial q_j} = 0,$$

so unique stationary point $q_i = p_i$ is indeed a minimum. Easily verified that at that minimum, $\mathbf{KL}[q\|p] = \mathbf{KL}[p\|p] = 0$.
A similar proof holds for continuous densities, using functional derivatives.

Fixed Points of EM are Stationary Points in ℓ

Let a fixed point of EM occur with parameter θ^*. Then:

$$\frac{\partial}{\partial\theta}\langle \log P(\mathcal{Z},\mathcal{X} \mid \theta)\rangle_{P(\mathcal{Z}|\mathcal{X},\theta^*)}\Big|_{\theta^*} = 0$$

Now,

$$\begin{aligned}\ell(\theta) &= \log P(\mathcal{X}|\theta) = \langle \log P(\mathcal{X}|\theta)\rangle_{P(\mathcal{Z}|\mathcal{X},\theta^*)}\\ &= \left\langle \log \frac{P(\mathcal{Z},\mathcal{X}|\theta)}{P(\mathcal{Z}|\mathcal{X},\theta)}\right\rangle_{P(\mathcal{Z}|\mathcal{X},\theta^*)}\\ &= \langle \log P(\mathcal{Z},\mathcal{X}|\theta)\rangle_{P(\mathcal{Z}|\mathcal{X},\theta^*)} - \langle \log P(\mathcal{Z}|\mathcal{X},\theta)\rangle_{P(\mathcal{Z}|\mathcal{X},\theta^*)}\end{aligned}$$

so,

$$\frac{d}{d\theta}\ell(\theta) = \frac{d}{d\theta}\langle \log P(\mathcal{Z},\mathcal{X}|\theta)\rangle_{P(\mathcal{Z}|\mathcal{X},\theta^*)} - \frac{d}{d\theta}\langle \log P(\mathcal{Z}|\mathcal{X},\theta)\rangle_{P(\mathcal{Z}|\mathcal{X},\theta^*)}$$

The second term is 0 at θ^* if the derivative exists (minimum of **KL**$[\cdot\|\cdot]$), and thus:

$$\frac{d}{d\theta}\ell(\theta)\Big|_{\theta^*} = \frac{d}{d\theta}\langle \log P(\mathcal{Z},\mathcal{X}|\theta)\rangle_{P(\mathcal{Z}|\mathcal{X},\theta^*)}\Big|_{\theta^*} = 0$$

So, EM converges to a stationary point of $\ell(\theta)$.

MAXIMA IN $\mathcal{F}$ CORRESPOND TO MAXIMA IN ℓ

Let θ^* now be the parameter value at a local maximum of $\mathcal{F}$ (and thus at a fixed point)

Differentiating the previous expression wrt θ again we find

$$\frac{d^2}{d\theta^2}\ell(\theta) = \frac{d^2}{d\theta^2}\langle \log P(\mathcal{Z},\mathcal{X}|\theta)\rangle_{P(\mathcal{Z}|\mathcal{X},\theta^*)} - \frac{d^2}{d\theta^2}\langle \log P(\mathcal{Z}|\mathcal{X},\theta)\rangle_{P(\mathcal{Z}|\mathcal{X},\theta^*)}$$

The first term on the right is negative (a maximum) and the second term is positive (a minimum). Thus the curvature of the likelihood is negative and

θ^* is a maximum of ℓ.

[...as long as the derivatives exist. They sometimes don't (zero-noise ICA)].

LATENT VARIABLE MODELS FOR TIME SERIES

Modeling time series

Thus far, our data have been (marginally) iid. Now consider a sequence of observations:

$$\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_t$$

that are *not* independent.

Examples:

- Sequence of images
- Stock prices
- Recorded speech or music, English sentences
- Kinematic variables in a robot
- Sensor readings from an industrial process
- Amino acids, DNA, etc...

Goal: To build a joint probabilistic model of the data $p(\mathbf{x}_1, \ldots, \mathbf{x}_t)$.

- Predict $p(\mathbf{x}_t | \mathbf{x}_1, \ldots, \mathbf{x}_{t-1})$
- Detect abnormal/changed behaviour (if $p(\mathbf{x}_t, \mathbf{x}_{t+1}, \ldots | \mathbf{x}_1, \ldots, \mathbf{x}_{t-1})$ small)
- Recover underlying/latent/hidden causes linking entire sequence

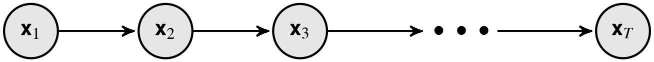
Markov models

In general:

$$P(\mathbf{x}_1, \ldots, \mathbf{x}_t) = P(\mathbf{x}_1)P(\mathbf{x}_2|\mathbf{x}_1)P(\mathbf{x}_3|\mathbf{x}_1, \mathbf{x}_2) \cdots P(\mathbf{x}_t|\mathbf{x}_1, \mathbf{x}_2 \ldots \mathbf{x}_{t-1})$$

First-order Markov model:

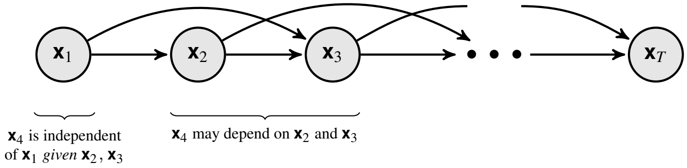
$$P(\mathbf{x}_1, \ldots, \mathbf{x}_t) = P(\mathbf{x}_1)P(\mathbf{x}_2|\mathbf{x}_1) \cdots P(\mathbf{x}_t|\mathbf{x}_{t-1})$$



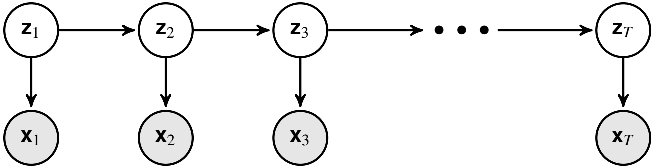
The term *Markov* refers to a conditional independence relationship. In this case, the Markov property is that, given the present observation ($\mathbf{x}_t$), the future ($\mathbf{x}_{t+1}, \ldots$) is independent of the past ($\mathbf{x}_1, \ldots, \mathbf{x}_{t-1}$).

Second-order Markov model:

$$P(\mathbf{x}_1, \ldots, \mathbf{x}_t) = P(\mathbf{x}_1)P(\mathbf{x}_2|\mathbf{x}_1) \cdots P(\mathbf{x}_{t-1}|\mathbf{x}_{t-3}, \mathbf{x}_{t-2})P(\mathbf{x}_t|\mathbf{x}_{t-2}, \mathbf{x}_{t-1})$$


$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{x}_T$
$\mathbf{x}_4$ is independent of $\mathbf{x}_1$ *given* $\mathbf{x}_2, \mathbf{x}_3$
$\mathbf{x}_4$ may depend on $\mathbf{x}_2$ and $\mathbf{x}_3$

Causal structure and latent variables


$$\begin{tikzcd}
\mathbf{z}_1 \arrow[r] \arrow[d] & \mathbf{z}_2 \arrow[r] \arrow[d] & \mathbf{z}_3 \arrow[r] \arrow[d] & \bullet\bullet\bullet \arrow[r] & \mathbf{z}_T \arrow[d] \\
\mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & & \mathbf{x}_T
\end{tikzcd}$$

Temporal dependence captured by latents, with observations conditionally independent.
Speech recognition:

- $\mathbf{z}$ - underlying phonemes or words
- $\mathbf{x}$ - acoustic waveform

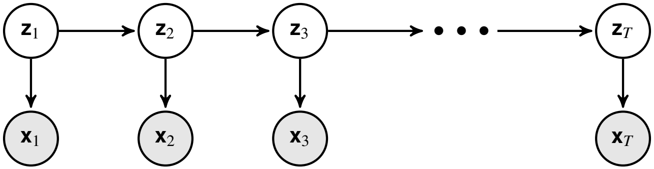
Vision:

- $\mathbf{z}$ - object identities, poses, illumination
- $\mathbf{x}$ - image pixel values

Industrial Monitoring:

- $\mathbf{z}$ - current state of molten steel in caster
- $\mathbf{x}$ - temperature and pressure sensor readings

LATENT-CHAIN MODELS



Joint probability factorizes:

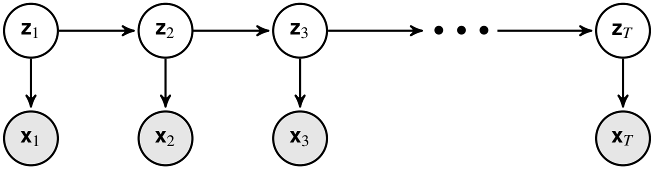
$$P(\mathbf{z}_{1:T}, \mathbf{x}_{1:T}) = P(\mathbf{z}_1)P(\mathbf{x}_1|\mathbf{z}_1)\prod_{t=2}^{T} P(\mathbf{z}_t|\mathbf{z}_{t-1})P(\mathbf{x}_t|\mathbf{z}_t)$$

where $\mathbf{z}_t$ and $\mathbf{x}_t$ are both real-valued vectors, and $\square_{1:T} \equiv \square_1, \ldots, \square_T$.

Two frequently-used tractable models:

- Linear-Gaussian state-space models
- Hidden Markov models

Linear-Gaussian state-space models (SSMs)



In a linear Gaussian SSM all conditional distributions are linear and Gaussian:

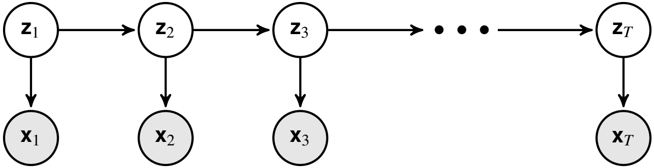
Output equation:	$\mathbf{x}_t = C\mathbf{z}_t + \mathbf{v}_t$
State dynamics equation:	$\mathbf{z}_t = A\mathbf{z}_{t-1} + \mathbf{w}_t$

where $\mathbf{v}_t$ and $\mathbf{w}_t$ are uncorrelated zero-mean multivariate Gaussian noise vectors.

Also assume $\mathbf{z}_1$ is multivariate Gaussian. The joint distribution over all variables $\mathbf{x}_{1:T}, \mathbf{z}_{1:T}$ is (one big) multivariate Gaussian.

These models are also known as stochastic linear dynamical systems, Kalman filter models.

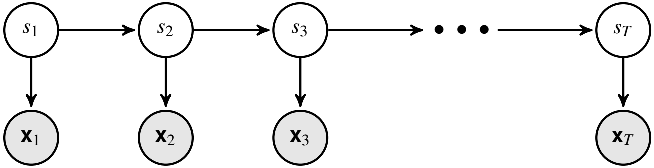
LINEAR DYNAMICAL SYSTEMS


$$\begin{tikzcd}
\mathbf{z}_1 \arrow[r] \arrow[d] & \mathbf{z}_2 \arrow[r] \arrow[d] & \mathbf{z}_3 \arrow[r] \arrow[d] & \bullet\bullet\bullet \arrow[r] & \mathbf{z}_T \arrow[d] \\
\mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 & & \mathbf{x}_T
\end{tikzcd}$$

Interpretation 2:

- Markov chain with linear dynamics $\mathbf{z}_t = A\mathbf{z}_{t-1} \ldots$
- …perturbed by Gaussian innovations noise – may describe stochasticity, unknown control, or model mismatch.
- Observations are a linear projection of the dynamical state, with additive iid Gaussian noise.
- Note:
  - Dynamical process ($\mathbf{z}_t$) may be higher dimensional than the observations ($\mathbf{x}_t$).
  - Observations do not form a Markov chain – longer-scale dependence reflects/reveals latent dynamics.

HIDDEN MARKOV MODELS



Discrete hidden states $s_t \in \{1\dots,K\}$; outputs $\mathbf{x}_t$ can be discrete or continuous.
Joint probability factorizes:

$$P(s_{1:T},\mathbf{x}_{1:T}) = P(s_1)P(\mathbf{x}_1|s_1)\prod_{t=2}^{T} P(s_t|s_{t-1})P(\mathbf{x}_t|s_t)$$

Generative process:

- A first-order Markov chain generates the hidden state sequence (path):

  initial state probs: $\pi_j = P(s_1 = j)$ transition matrix: $\Phi_{ij} = P(s_{t+1} = j|s_t = i)$

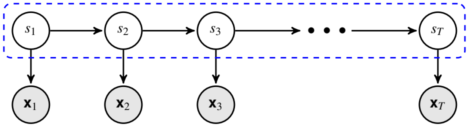
- A set of emission (output) distributions $A_j(\cdot)$ (one per state) converts state path to a sequence of observations $\mathbf{x}_t$.

$$A_j(\mathbf{x}) = P(\mathbf{x}_t = \mathbf{x}|s_t = j) \qquad \text{(for continuous } \mathbf{x}_t\text{)}$$
$$A_{jk} = P(\mathbf{x}_t = k|s_t = j) \qquad \text{(for discrete } \mathbf{x}_t\text{)}$$

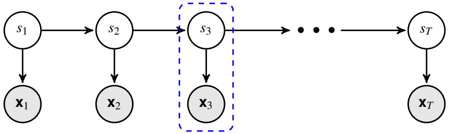
Hidden Markov models

Two interpretations:

- a Markov chain with stochastic measurements:



- or a mixture model with states coupled across time:



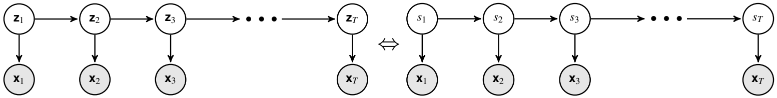
Even though hidden state sequence is first-order Markov, the output process may not be Markov of any order (for example: 1111121111311121111131 . . .).

Discrete state, discrete output models can approximate any continuous dynamics and observation mapping even if nonlinear; however this is usually not practical.

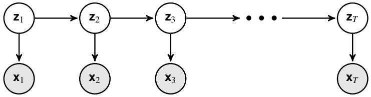
HMMs are related to stochastic finite state machines/automata.

HMMs and SSMs

(Linear Gaussian) State space models are the continuous state analogue of hidden Markov models.

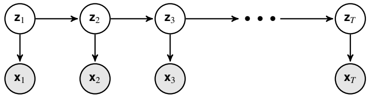


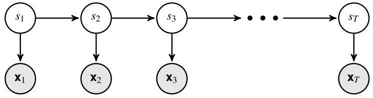
- A continuous vector state is a very powerful representation.
  For an HMM to communicate N bits of information about the past, it needs 2^N states! But a real-valued state vector can store an arbitrary number of bits in principle.



- Linear-Gaussian output/dynamics are very weak.
  The types of dynamics linear SSMs can capture is very limited. HMMs can in principle represent arbitrary stochastic dynamics and output mappings.

Chain models: ML Learning with EM





$$\begin{aligned} \mathbf{z}_1 &\sim \mathcal{N}(\boldsymbol{\mu}_0, Q_0) \\ \mathbf{z}_t|\mathbf{z}_{t-1} &\sim \mathcal{N}(A\mathbf{z}_{t-1}, Q) \\ \mathbf{x}_t|\mathbf{z}_t &\sim \mathcal{N}(C\mathbf{z}_t, R) \end{aligned}$$

$$\begin{aligned} s_1 &\sim \boldsymbol{\pi} \\ s_t|s_{t-1} &\sim \Phi_{s_{t-1},\cdot} \\ \mathbf{x}_t|s_t &\sim \mathsf{A}_{s_t} \end{aligned}$$

The structure of learning and inference for both models is dictated by the factored structure.

$$P(\mathbf{x}_1, \ldots, \mathbf{x}_T, \mathbf{z}_1, \ldots, \mathbf{z}_T) = P(\mathbf{z}_1) \prod_{t=2}^{T} P(\mathbf{z}_t|\mathbf{z}_{t-1}) \prod_{t=1}^{T} P(\mathbf{x}_t|\mathbf{z}_t)$$

Learning (M-step):

$$\arg\max \langle \log P(\mathbf{x}_1, \ldots, \mathbf{x}_T, \mathbf{z}_1, \ldots, \mathbf{z}_T) \rangle_{q(\mathbf{z}_1, \ldots, \mathbf{z}_T)} =$$

$$\arg\max \left[\langle \log P(\mathbf{z}_1) \rangle_{q(\mathbf{z}_1)} + \sum_{t=2}^{T} \langle \log P(\mathbf{z}_t|\mathbf{z}_{t-1}) \rangle_{q(\mathbf{z}_t, \mathbf{z}_{t-1})} + \sum_{t=1}^{T} \langle \log P(\mathbf{x}_t|\mathbf{z}_t) \rangle_{q(\mathbf{z}_t)} \right]$$

So the expectations needed (in E-step) are derived from singleton and pairwise marginals.

Chain models: Inference

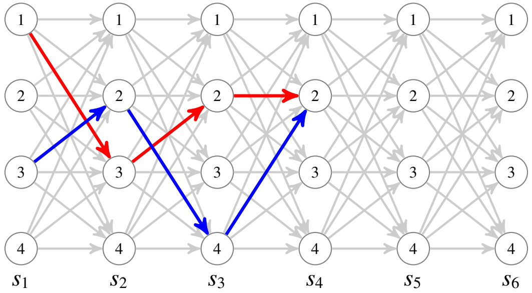
Three general inference problems:

Filtering:	$P(\mathbf{z}_t \vert \mathbf{x}_1, \ldots, \mathbf{x}_t)$	
Smoothing:	$P(\mathbf{z}_t \vert \mathbf{x}_1, \ldots, \mathbf{x}_T)$	(also $P(\mathbf{z}_t, \mathbf{z}_{t-1} \vert \mathbf{x}_1, \ldots, \mathbf{x}_T)$ for learning)
Prediction:	$P(\mathbf{z}_t \vert \mathbf{x}_1, \ldots, \mathbf{x}_{t-\Delta t})$	

Naively, these marginal posteriors seem to require very large integrals (or sums)

$$P(\mathbf{z}_t|\mathbf{x}_1, \ldots, \mathbf{x}_t) = \int \cdots \int d\mathbf{z}_1 \ldots d\mathbf{z}_{t-1} \; P(\mathbf{z}_1, \ldots, \mathbf{z}_t|\mathbf{x}_1, \ldots, \mathbf{x}_t)$$

but again the factored structure of the distributions will help us. The algorithms rely on a form of temporal updating or message passing.



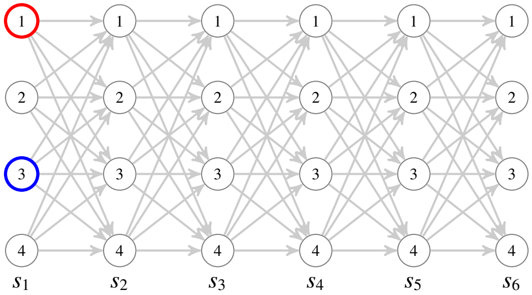
Consider an HMM, where we want to find $P(s_t = k|\mathbf{x}_1 \dots \mathbf{x}_t) =$

$$\sum_{k_1,\dots,k_{t-1}} P(s_1 = k_1, \dots, s_t = k|\mathbf{x}_1 \dots \mathbf{x}_t) \propto$$

$$\sum_{k_1,\dots,k_{t-1}} \pi_{k_1} A_{k_1}(\mathbf{x}_1) \Phi_{k_1,k_2} A_{k_2}(\mathbf{x}_2) \dots \Phi_{k_{t-1},k} A_k(\mathbf{x}_t)$$

Naïve algorithm:

- start a “bug” at each of the $k_1 = 1 \dots K$ states at $t = 1$ holding value $\pi_{k_1} A_{k_1}(\mathbf{x}_1)$
- move each bug forward in time: make copies of each bug to each subsequent state and multiply the value of each copy by transition prob. $\times$ output emission prob.
- repeat
- sum up values on all K^{t-1} bugs that reach state $s_t = k$ (one bug per state path)



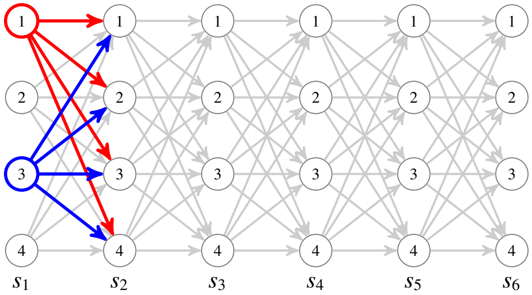
Consider an HMM, where we want to find $P(s_t = k|\mathbf{x}_1 \dots \mathbf{x}_t) =$

$$\sum_{k_1,\dots,k_{t-1}} P(s_1 = k_1, \dots, s_t = k|\mathbf{x}_1 \dots \mathbf{x}_t) \propto$$

$$\sum_{k_1,\dots,k_{t-1}} \pi_{k_1} A_{k_1}(\mathbf{x}_1) \Phi_{k_1,k_2} A_{k_2}(\mathbf{x}_2) \dots \Phi_{k_{t-1},k} A_k(\mathbf{x}_t)$$

Naïve algorithm:

- start a "bug" at each of the $k_1 = 1 \dots K$ states at $t = 1$ holding value $\pi_{k_1} A_{k_1}(\mathbf{x}_1)$
- move each bug forward in time: make copies of each bug to each subsequent state and multiply the value of each copy by transition prob. $\times$ output emission prob.
- repeat
- sum up values on all K^{t-1} bugs that reach state $s_t = k$ (one bug per state path)



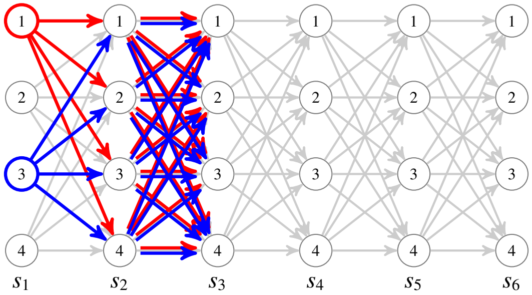
Consider an HMM, where we want to find $P(s_t = k | \mathbf{x}_1 \dots \mathbf{x}_t) =$

$$\sum_{k_1,\dots,k_{t-1}} P(s_1 = k_1, \dots, s_t = k | \mathbf{x}_1 \dots \mathbf{x}_t) \propto$$

$$\sum_{k_1,\dots,k_{t-1}} \pi_{k_1} A_{k_1}(\mathbf{x}_1) \Phi_{k_1,k_2} A_{k_2}(\mathbf{x}_2) \dots \Phi_{k_{t-1},k} A_k(\mathbf{x}_t)$$

Naïve algorithm:

- start a "bug" at each of the $k_1 = 1 \dots K$ states at $t = 1$ holding value $\pi_{k_1} A_{k_1}(\mathbf{x}_1)$
- move each bug forward in time: make copies of each bug to each subsequent state and multiply the value of each copy by transition prob. $\times$ output emission prob.
- repeat
- sum up values on all K^{t-1} bugs that reach state $s_t = k$ (one bug per state path)



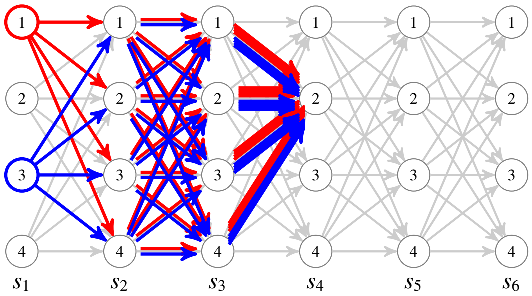
Consider an HMM, where we want to find $P(s_t = k|\mathbf{x}_1 \dots \mathbf{x}_t) =$

$$\sum_{k_1,\dots,k_{t-1}} P(s_1 = k_1, \dots, s_t = k|\mathbf{x}_1 \dots \mathbf{x}_t) \propto$$

$$\sum_{k_1,\dots,k_{t-1}} \pi_{k_1} A_{k_1}(\mathbf{x}_1) \Phi_{k_1,k_2} A_{k_2}(\mathbf{x}_2) \dots \Phi_{k_{t-1},k} A_k(\mathbf{x}_t)$$

Naïve algorithm:

- start a “bug” at each of the $k_1 = 1 \dots K$ states at $t = 1$ holding value $\pi_{k_1} A_{k_1}(\mathbf{x}_1)$
- move each bug forward in time: make copies of each bug to each subsequent state and multiply the value of each copy by transition prob. $\times$ output emission prob.
- repeat
- sum up values on all K^{t-1} bugs that reach state $s_t = k$ (one bug per state path)



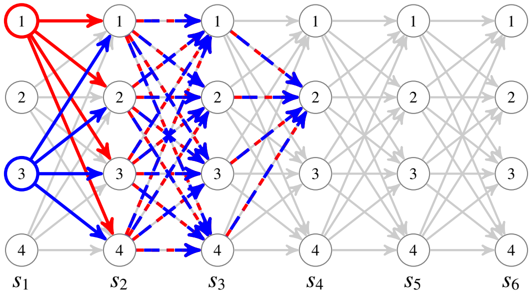
Consider an HMM, where we want to find $P(s_t = k|\mathbf{x}_1 \dots \mathbf{x}_t) =$

$$\sum_{k_1,\dots,k_{t-1}} P(s_1 = k_1, \dots, s_t = k|\mathbf{x}_1 \dots \mathbf{x}_t) \propto$$

$$\sum_{k_1,\dots,k_{t-1}} \pi_{k_1} A_{k_1}(\mathbf{x}_1) \Phi_{k_1,k_2} A_{k_2}(\mathbf{x}_2) \dots \Phi_{k_{t-1},k} A_k(\mathbf{x}_t)$$

Naïve algorithm:

- start a "bug" at each of the $k_1 = 1 \dots K$ states at $t = 1$ holding value $\pi_{k_1} A_{k_1}(\mathbf{x}_1)$
- move each bug forward in time: make copies of each bug to each subsequent state and multiply the value of each copy by transition prob. $\times$ output emission prob.
- repeat until all bugs have reached time t
- sum up values on all K^{t-1} bugs that reach state $s_t = k$ (one bug per state path)



Consider an HMM, where we want to find $P(s_t = k|\mathbf{x}_1 \dots \mathbf{x}_t) =$

$$\sum_{k_1,\dots,k_{t-1}} P(s_1 = k_1, \dots, s_t = k|\mathbf{x}_1 \dots \mathbf{x}_t) \propto$$

$$\sum_{k_1,\dots,k_{t-1}} \pi_{k_1} A_{k_1}(\mathbf{x}_1) \Phi_{k_1,k_2} A_{k_2}(\mathbf{x}_2) \dots \Phi_{k_{t-1},k} A_k(\mathbf{x}_t)$$

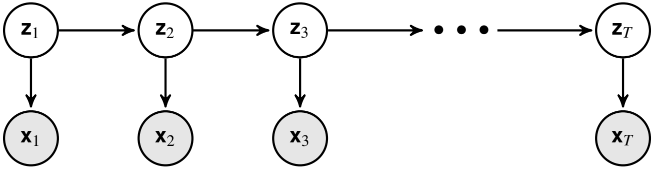
Naïve algorithm:

- start a “bug” at each of the $k_1 = 1 \dots K$ states at $t = 1$ holding value $\pi_{k_1} A_{k_1}(\mathbf{x}_1)$
- move each bug forward in time: make copies of each bug to each subsequent state and multiply the value of each copy by transition prob. $\times$ output emission prob.
- repeat
- sum up values on all K^{t-1} bugs that reach state $s_t = k$ (one bug per state path)

Clever recursion:

- at every step, replace bugs at each node with a single bug carrying sum of values

Probability updating: "Bayesian filtering"



$$\begin{aligned} P(\mathbf{z}_t|\mathbf{x}_{1:t}) &= \int P(\mathbf{z}_t, \mathbf{z}_{t-1}|\mathbf{x}_t, \mathbf{x}_{1:t-1})\; d\mathbf{z}_{t-1} \\ &= \int \frac{P(\mathbf{x}_t, \mathbf{z}_t, \mathbf{z}_{t-1}|\mathbf{x}_{1:t-1})}{P(\mathbf{x}_t|\mathbf{x}_{1:t-1})}\; d\mathbf{z}_{t-1} \\ &\propto \int P(\mathbf{x}_t|\mathbf{z}_t, \mathbf{z}_{t-1}, \mathbf{x}_{1:t-1}) P(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{x}_{1:t-1}) P(\mathbf{z}_{t-1}|\mathbf{x}_{1:t-1})\; d\mathbf{z}_{t-1} \\ &= \int P(\mathbf{x}_t|\mathbf{z}_t) P(\mathbf{z}_t|\mathbf{z}_{t-1}) P(\mathbf{z}_{t-1}|\mathbf{x}_{1:t-1})\; d\mathbf{z}_{t-1} \end{aligned}$$

Markov property

This is a **forward recursion** based on Bayes rule.

The HMM: Forward pass

The **forward recursion** for the HMM is a form of **dynamic programming**. Define:

$$\alpha_t(i) = P(\mathbf{x}_1, \dots, \mathbf{x}_t, s_t = i|\theta)$$

Then much like the Bayesian filtering updates, we have:

$$\alpha_1(i) = \pi_i A_i(\mathbf{x}_1) \qquad\qquad \alpha_{t+1}(i) = \left(\sum_{j=1}^{K} \alpha_t(j)\Phi_{ji} \right) A_i(\mathbf{x}_{t+1})$$

We've defined $\alpha_t(i)$ to be a joint rather than a posterior. It's easy to obtain the posterior by normalisation:

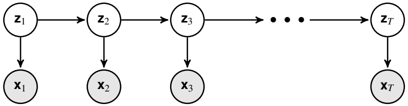
$$P(s_t = i|\mathbf{x}_1, \dots, \mathbf{x}_t, \theta) = \frac{\alpha_t(i)}{\sum_k \alpha_t(k)}$$

This form enables us to compute the likelihood for $\theta = \{\mathsf{A}, \Phi, \boldsymbol{\pi}\}$ efficiently in $\mathcal{O}(TK^2)$ time:

$$P(\mathbf{x}_1 \dots \mathbf{x}_T|\theta) = \sum_{s_1,\dots,s_T} P(\mathbf{x}_1, \dots, \mathbf{x}_T, s_1, \dots, s_T, \theta) = \sum_{k=1}^{K} \alpha_T(k)$$

avoiding the exponential number of paths in the naïve sum (number of paths = K^T).

The LGSSM: Kalman Filtering



$$\mathbf{z}_1 \sim \mathcal{N}(\boldsymbol{\mu}_0, Q_0)$$
$$\mathbf{z}_t|\mathbf{z}_{t-1} \sim \mathcal{N}(A\mathbf{z}_{t-1}, Q)$$
$$\mathbf{x}_t|\mathbf{z}_t \sim \mathcal{N}(C\mathbf{z}_t, R)$$

For the SSM, the sums become integrals. Let $\hat{\mathbf{z}}_1^0 = \boldsymbol{\mu}_0$ and $\hat{V}_1^0 = Q_0$; then

$$P(\mathbf{z}_1|\mathbf{x}_1) = \mathcal{N}(\underbrace{\hat{\mathbf{z}}_1^0 + K_1(\mathbf{x}_1 - C\hat{\mathbf{z}}_i^0)}_{\hat{\mathbf{z}}_1^1}, \underbrace{\hat{V}_1^0 - K_1 C \hat{V}_1^0}_{\hat{V}_1^1}) \qquad K_1 = \hat{V}_1^0 C^\mathsf{T}(C\hat{V}_1^0 C^\mathsf{T} + R)^{-1}$$

In general, we define $\hat{\mathbf{z}}_t^\tau \equiv E[\mathbf{z}_t|\mathbf{x}_1, \ldots, \mathbf{x}_\tau]$ and $\hat{V}_t^\tau \equiv V[\mathbf{z}_t|\mathbf{x}_1, \ldots, \mathbf{x}_\tau]$. Then,

$$P(\mathbf{z}_t|\mathbf{x}_{1:t-1}) = \int d\mathbf{z}_{t-1} P(\mathbf{z}_t|\mathbf{z}_{t-1}) P(\mathbf{z}_{t-1}|\mathbf{x}_{1:t-1}) = \mathcal{N}(\underbrace{A\hat{\mathbf{z}}_{t-1}^{t-1}}_{\hat{\mathbf{z}}_t^{t-1}}, \underbrace{A\hat{V}_{t-1}^{t-1}A^\mathsf{T} + Q}_{\hat{V}_t^{t-1}})$$

$$P(\mathbf{z}_t|\mathbf{x}_{1:t}) = \mathcal{N}(\underbrace{\hat{\mathbf{z}}_t^{t-1} + K_t(\mathbf{x}_t - C\hat{\mathbf{z}}_t^{t-1})}_{\hat{\mathbf{z}}_t^t}, \underbrace{\hat{V}_t^{t-1} - K_t C \hat{V}_t^{t-1}}_{\hat{V}_t^t})$$

$$\underbrace{K_t = \overbrace{\hat{V}_t^{t-1} C^\mathsf{T}}^{\langle \mathbf{z}\mathbf{x}^\mathsf{T} \rangle} \overbrace{(C\hat{V}_t^{t-1} C^\mathsf{T} + R)}^{\langle \mathbf{x}\mathbf{x}^\mathsf{T} \rangle}{}^{-1}}_{\text{Kalman gain}}$$

BACKGROUND I

Conditioning within a multivariate Gaussian

Let (A, B) be a partition of the set $\{1, \ldots, d\}$ and let $\mathbf{X} = (\mathbf{X}_{\mathrm{A}}, \mathbf{X}_{\mathrm{B}})$ be a Gaussian random vector in $\mathbb{R}^d = \mathbb{R}^A \times \mathbb{R}^B$, with

$$\mathbb{E}[\mathbf{X}] = \begin{pmatrix} \mu_{\mathrm{A}} \\ \mu_{\mathrm{B}} \end{pmatrix} \qquad \text{and} \qquad \mathrm{Cov}[\mathbf{X}] = \begin{pmatrix} \Sigma_{\mathrm{A}} & \Sigma_{\mathrm{AB}} \\ \Sigma_{\mathrm{AB}}^t & \Sigma_{\mathrm{B}} \end{pmatrix} .$$

Then the conditional distribution of $\mathbf{X}_{\mathrm{A}} | (\mathbf{X}_{\mathrm{B}} = x_{\mathrm{B}})$ is again Gaussian, with mean

$$\mathbb{E}[\mathbf{X}_{\mathrm{A}} | \mathbf{X}_{\mathrm{B}} = x_{\mathrm{B}}] = \mu_{\mathrm{A}} - \Sigma_{\mathrm{AB}} \Sigma_{\mathrm{B}}^{-1} (x_{\mathrm{B}} - \mu_{\mathrm{B}})$$

and covariance

$$\mathrm{Cov}[\mathbf{X}_{\mathrm{A}} | \mathbf{X}_{\mathrm{B}} = x_{\mathrm{B}}] = \Sigma_{\mathrm{A}} - \Sigma_{\mathrm{AB}}^t \Sigma_{\mathrm{B}}^{-1} \Sigma_{\mathrm{AB}} .$$

Background II

Matrix inversion lemma (Woodbury identity)

For $k \leq n$, invertible matrices $A \in \mathbb{R}^{n\times n}$ and $C \in \mathbb{R}^{k\times k}$, and any $B \in \mathbb{R}^{n\times k}$ and $D \in \mathbb{R}^{k\times n}$:

$$(\overset{n\times n}{A} - \underset{n\times k}{B}\overset{k\times k}{C}\underset{k\times n}{D})^{-1} \;=\; A^{-1} - A^{-1}B(C^{-1} + DA^{-1}B)^{-1}DA^{-1}$$

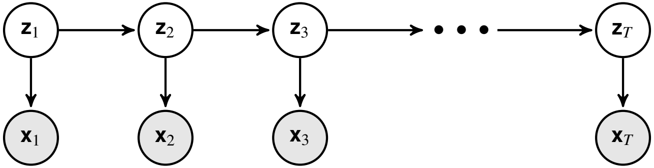
provided A and C are invertible.

Note: A and C refers to generic matrices, not to A and C defined on the Kalman filter slide.

Intepretation

- In essence: $(A - BCD)^{-1} = f(A^{-1}, C^{-1}, B, D)$
- Since C is $k\times k$, the $n\times n$-matrix BCD has rank $\leq k$.
- Substracting BCD modifies A on a k-dimensional subspace.
- Typical use case: We already know A^{-1}, and k is small. We can then invert the $n\times n$ matrix $(A - BCD)$ by inverting only a $k\times k$ matrix.

The marginal posterior: “Bayesian smoothing”



$$P(\mathbf{z}_t|\mathbf{x}_{1:T}) = \frac{P(\mathbf{z}_t, \mathbf{x}_{t+1:T}|\mathbf{x}_{1:t})}{P(\mathbf{x}_{t+1:T}|\mathbf{x}_{1:t})}$$

$$= \frac{P(\mathbf{x}_{t+1:T}|\mathbf{z}_t)P(\mathbf{z}_t|\mathbf{x}_{1:t})}{P(\mathbf{x}_{t+1:T}|\mathbf{x}_{1:t})}$$

The marginal combines a backward message with the forward message found by filtering.

THE HMM: FORWARD–BACKWARD ALGORITHM

State estimation: compute marginal posterior distribution over state at time t:

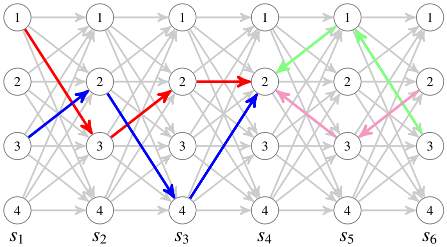
$$\gamma_t(i) \equiv P(s_t = i|\mathbf{x}_{1:T}) = \frac{P(s_t = i, \mathbf{x}_{1:t})P(\mathbf{x}_{t+1:T}|s_t = i)}{P(\mathbf{x}_{1:T})} = \frac{\alpha_t(i)\beta_t(i)}{\sum_j \alpha_t(j)\beta_t(j)}$$

where there is a simple backward recursion for

$$\beta_t(i) \equiv P(\mathbf{x}_{t+1:T}|s_t = i) = \sum_{j=1}^{K} P(s_{t+1} = j, \mathbf{x}_{t+1}, \mathbf{x}_{t+2:T}|s_t = i)$$

$$= \sum_{j=1}^{K} P(s_{t+1} = j|s_t = i)P(\mathbf{x}_{t+1}|s_{t+1} = j)P(\mathbf{x}_{t+2:T}|s_{t+1} = j) = \sum_{j=1}^{K} \Phi_{ij}A_j(\mathbf{x}_{t+1})\beta_{t+1}(j)$$

$\alpha_t(i)$ gives total *inflow* of probabilities to node (t,i); $\beta_t(i)$ gives total *outflow* of probabiilties.

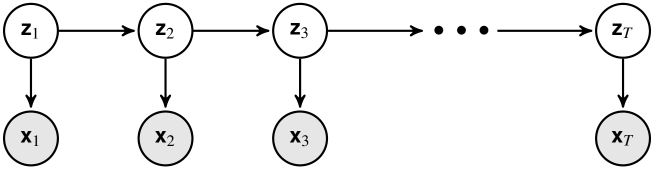


Bugs again: the bugs run forward from time 0 to t and backward from time T to t.

VITERBI DECODING

- The numbers $\gamma_t(i)$ computed by forward-backward give the marginal posterior distribution over states at each time.
- By choosing the state i_t^* with the largest $\gamma_t(i)$ at each time, we can make a "best" state path. This is the path with the maximum expected number of correct states.
- But it is not the single path with the highest probability of generating the data. In fact it may be a path of probability zero!
- To find the single best path, we use the *Viterbi decoding algorithm* which is just Bellman's dynamic programming algorithm applied to this problem. This is an inference algorithm which computes the most probable state sequences: $\arg\max_{s_{1:T}} P(s_{1:T}|\mathbf{x}_{1:T}, \theta)$
- The recursions look the same as forward-backward, except with max instead of $\sum$.
- Bugs once more: same trick except at each step kill all bugs but the one with the highest value at the node.
- There is also a modified EM training based on the Viterbi decoder (assignment).

The LGSSM: Kalman smoothing



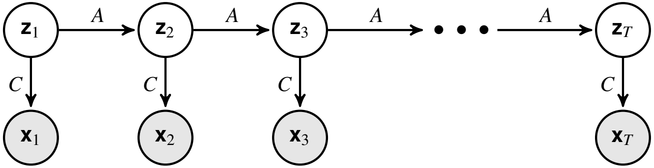
We use a slightly different decomposition:

$$\begin{aligned} P(\mathbf{z}_t|\mathbf{x}_{1:T}) &= \int P(\mathbf{z}_t, \mathbf{z}_{t+1}|\mathbf{x}_{1:T})\, d\mathbf{z}_{t+1} \\ &= \int P(\mathbf{z}_t|\mathbf{z}_{t+1}, \mathbf{x}_{1:T}) P(\mathbf{z}_{t+1}|\mathbf{x}_{1:T})\, d\mathbf{z}_{t+1} \\ &\underset{\text{Markov property}}{=} \int P(\mathbf{z}_t|\mathbf{z}_{t+1}, \mathbf{x}_{1:t}) P(\mathbf{z}_{t+1}|\mathbf{x}_{1:T})\, d\mathbf{z}_{t+1} \end{aligned}$$

This gives the additional **backward recursion**:

$$\begin{aligned} J_t &= \hat{V}_t^t A^\mathsf{T} (\hat{V}_{t+1}^t)^{-1} \\ \hat{\mathbf{z}}_t^T &= \hat{\mathbf{z}}_t^t + J_t(\hat{\mathbf{z}}_{t+1}^T - A\hat{\mathbf{z}}_t^t) \\ \hat{V}_t^T &= \hat{V}_t^t + J_t(\hat{V}_{t+1}^T - \hat{V}_{t+1}^t) J_t^\mathsf{T} \end{aligned}$$

ML Learning for SSMs using batch EM



Parameters: $\theta = \{\boldsymbol{\mu}_0, Q_0, A, Q, C, R\}$
Free energy:

$$\mathcal{F}(q, \theta) = \int d\mathbf{z}_{1:T}\ q(\mathbf{z}_{1:T})(\log P(x_{1:T}, \mathbf{z}_{1:T}|\theta) - \log q(\mathbf{z}_{1:T}))$$

E-step: Maximise $\mathcal{F}$ w.r.t. q with θ fixed: $q^*(\mathbf{z}) = p(\mathbf{z}|\mathbf{x}, \theta)$
This can be achieved with a two-state extension of the Kalman smoother.

M-step: Maximize $\mathcal{F}$ w.r.t. θ with q fixed.
This boils down to solving a few weighted least squares problems, since all the variables in:

$$p(\mathbf{z}, \mathbf{x}|\theta) = p(\mathbf{z}_1)p(\mathbf{x}_1|\mathbf{z}_1)\prod_{t=2}^{T} p(\mathbf{z}_t|\mathbf{z}_{t-1})p(\mathbf{x}_t|\mathbf{z}_t)$$

form a multivariate Gaussian.

THE M STEP FOR C

$$p(\mathbf{x}_t|\mathbf{z}_t) \propto \exp\left[-\tfrac{1}{2}(\mathbf{x}_t - C\mathbf{z}_t)^\mathsf{T} R^{-1}(\mathbf{x}_t - C\mathbf{z}_t)\right] \quad \Rightarrow$$

$$\begin{aligned}
C_{\text{new}} &= \arg\max_C \left\langle \sum_t \ln p(\mathbf{x}_t|\mathbf{z}_t) \right\rangle_q \\
&= \arg\max_C \left\langle -\frac{1}{2}\sum_t (\mathbf{x}_t - C\mathbf{z}_t)^\mathsf{T} R^{-1}(\mathbf{x}_t - C\mathbf{z}_t) \right\rangle_q + \text{const} \\
&= \arg\max_C \left\{ -\frac{1}{2}\sum_t \mathbf{x}_t^\mathsf{T} R^{-1}\mathbf{x}_t - 2\mathbf{x}_t^\mathsf{T} R^{-1} C\langle \mathbf{z}_t\rangle + \langle \mathbf{z}_t^\mathsf{T} C^\mathsf{T} R^{-1} C\mathbf{z}_t\rangle \right\} \\
&= \arg\max_C \left\{ \mathsf{Tr}\left[C\sum_t \langle \mathbf{z}_t\rangle \mathbf{x}_t^\mathsf{T} R^{-1}\right] - \frac{1}{2}\mathsf{Tr}\left[C^\mathsf{T} R^{-1} C \left\langle \sum_t \mathbf{z}_t\mathbf{z}_t^\mathsf{T} \right\rangle \right] \right\}
\end{aligned}$$

using $\frac{\partial \mathsf{Tr}[AB]}{\partial A} = B^\mathsf{T}$, we have $\dfrac{\partial\{\cdot\}}{\partial C} = R^{-1}\sum_t \mathbf{x}_t\langle \mathbf{z}_t\rangle^\mathsf{T} - R^{-1}C\left\langle \sum_t \mathbf{z}_t\mathbf{z}_t^\mathsf{T}\right\rangle$

$$\Rightarrow C_{\text{new}} = \left(\sum_t \mathbf{x}_t\langle \mathbf{z}_t\rangle^\mathsf{T}\right)\left(\sum_t \left\langle \mathbf{z}_t\mathbf{z}_t^\mathsf{T}\right\rangle\right)^{-1}$$

Note the connection to linear regression.

THE M STEP FOR A

$$p(\mathbf{z}_{t+1}|\mathbf{z}_t) \propto \exp\left\{-\tfrac{1}{2}(\mathbf{z}_{t+1}-A\mathbf{z}_t)^{\mathsf{T}}Q^{-1}(\mathbf{z}_{t+1}-A\mathbf{z}_t)\right\} \quad \Rightarrow$$

$$\begin{aligned}
A_{\text{new}} &= \arg\max_A \left\langle \sum_t \ln p(\mathbf{z}_{t+1}|\mathbf{z}_t)\right\rangle_q \\
&= \arg\max_A \left\langle -\frac{1}{2}\sum_t (\mathbf{z}_{t+1}-A\mathbf{z}_t)^{\mathsf{T}}Q^{-1}(\mathbf{z}_{t+1}-A\mathbf{z}_t)\right\rangle_q + \text{const} \\
&= \arg\max_A \left\{-\frac{1}{2}\sum_t \mathbf{z}_{t+1}^{\mathsf{T}}Q^{-1}\mathbf{z}_{t+1} - 2\left\langle \mathbf{z}_{t+1}^{\mathsf{T}}Q^{-1}A\mathbf{z}_t\right\rangle + \left\langle \mathbf{z}_t^{\mathsf{T}}A^{\mathsf{T}}Q^{-1}A\mathbf{z}_t\right\rangle\right\} \\
&= \arg\max_A \left\{\mathsf{Tr}\left[A\sum_t\left\langle \mathbf{z}_t\mathbf{z}_{t+1}^{\mathsf{T}}\right\rangle Q^{-1}\right] - \frac{1}{2}\mathsf{Tr}\left[A^{\mathsf{T}}Q^{-1}A\sum_t\left\langle \mathbf{z}_t\mathbf{z}_t^{\mathsf{T}}\right\rangle\right]\right\}
\end{aligned}$$

using $\frac{\partial \mathsf{Tr}[AB]}{\partial A} = B^{\mathsf{T}}$, we have $\dfrac{\partial\{\cdot\}}{\partial A} = Q^{-1}\sum_t\left\langle \mathbf{z}_{t+1}\mathbf{z}_t^{\mathsf{T}}\right\rangle - Q^{-1}A\sum_t\left\langle \mathbf{z}_t\mathbf{z}_t^{\mathsf{T}}\right\rangle$

$$\Rightarrow A_{\text{new}} = \left(\sum_t\left\langle \mathbf{z}_{t+1}\mathbf{z}_t^{\mathsf{T}}\right\rangle\right)\left(\sum_t\left\langle \mathbf{z}_t\mathbf{z}_t^{\mathsf{T}}\right\rangle\right)^{-1}$$

This is still analagous to linear regression, with an extra expectation.

LEARNING (ONLINE GRADIENT)

Time series data must often be processed in real-time, and we may want to update parameters online as observations arrive. We can do so by updating a local version of the likelihood based on the Kalman filter estimates.
Consider the log likelihood contributed by each data point (ℓ_t):

$$\ell = \sum_{t=1}^{T} \ln p(\mathbf{x}_t | \mathbf{x}_1, \dots, \mathbf{x}_{t-1}) = \sum_{t=1}^{T} \ell_t$$

Then,

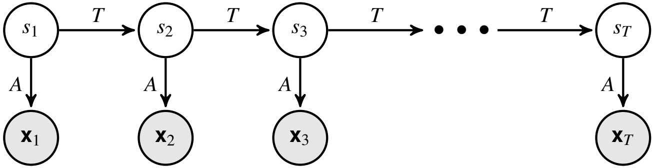
$$\ell_t = -\frac{D}{2} \ln 2\pi - \frac{1}{2} \ln |\Sigma| - \frac{1}{2} (\mathbf{x}_t - C\hat{\mathbf{z}}_t^{t-1})^\top \Sigma^{-1} (\mathbf{x}_t - C\hat{\mathbf{z}}_t^{t-1})$$

where D is dimension of $\mathbf{x}$, and:

$$\begin{aligned} \hat{\mathbf{z}}_t^{t-1} &= A\hat{\mathbf{z}}_{t-1}^{t-1} \\ \Sigma &= C\hat{V}_t^{t-1}C^\top + R \\ \hat{V}_t^{t-1} &= A\hat{V}_{t-1}^{t-1}A^\top + Q \end{aligned}$$

We differentiate ℓ_t to obtain gradient rules for A, C, Q, R. The size of the gradient step (learning rate) reflects our expectation about nonstationarity.

LEARNING HMMS USING EM



Parameters: $\theta = \{\boldsymbol{\pi}, \Phi, \mathsf{A}\}$

Free energy:

$$\mathcal{F}(q,\theta) = \sum_{s_{1:T}} q(s_{1:T})(\log P(x_{1:T}, s_{1:T}|\theta) - \log q(s_{1:T}))$$

E-step: Maximise $\mathcal{F}$ w.r.t. q with θ fixed: $\quad q^*(s_{1:T}) = P(s_{1:T}|\mathbf{x}_{1:T}, \theta)$

We will only need the marginal probabilities $q(s_t, s_{t+1})$, which can also be obtained from the forward–backward algorithm.

M-step: Maximize $\mathcal{F}$ w.r.t. θ with q fixed.

We can re-estimate the parameters by computing the expected number of times the HMM was in state i, emitted symbol k and transitioned to state j.

This is the **Baum-Welch algorithm** and it predates the (more general) EM algorithm.

M step: Parameter updates are given by ratios of expected counts

We can derive the following updates by taking derivatives of $\mathcal{F}$ w.r.t. θ.

- The initial state distribution is the expected number of times in state i at $t = 1$:

$$\hat{\pi}_i = \gamma_1(i)$$

- The expected number of transitions from state i to j which begin at time t is:

$$\xi_t(i \to j) \equiv P(s_t = i, s_{t+1} = j | \mathbf{x}_{1:T}) = \alpha_t(i)\Phi_{ij}A_j(\mathbf{x}_{t+1})\beta_{t+1}(j)/P(x_{1:T})$$

  so the estimated transition probabilities are:

$$\widehat{\Phi}_{ij} = \sum_{t=1}^{T-1} \xi_t(i \to j) \Bigg/ \sum_{t=1}^{T-1} \gamma_t(i)$$

- The output distributions are the expected number of times we observe a particular symbol in a particular state:

$$\widehat{A}_{ik} = \sum_{t:\mathbf{x}_t=k} \gamma_t(i) \Bigg/ \sum_{t=1}^{T} \gamma_t(i)$$

  (or the state-probability-weighted mean and variance for a Gaussian output model).

HMM PRACTICALITIES

- Numerical scaling: the conventional message definition is in terms of a large joint:

  $\alpha_t(i) = P(\mathbf{x}_{1:t}, s_t = i) \;\to 0$ as t grows, and so can easily underflow.

  Rescale:

  $$\overline{\alpha}_t(i) = A_i(\mathbf{x}_t) \sum_j \tilde{\alpha}_{t-1}(j)\Phi_{ji} \qquad \rho_t = \sum_{i=1}^{K} \overline{\alpha}_t(i) \qquad \tilde{\alpha}_t(i) = \overline{\alpha}_t(i)/\rho_t$$

  Exercise: show that:

  $$\rho_t = P(\mathbf{x}_t | \mathbf{x}_{1:t-1}, \theta) \qquad \prod_{t=1}^{T} \rho_t = P(\mathbf{x}_{1:T}|\theta)$$

  What does this make $\tilde{\alpha}_t(i)$?

- Multiple observed sequences: average numerators and denominators in the ratios of updates.

HMM PSEUDOCODE: INFERENCE (E STEP)

Forward-backward including scaling tricks.
[$\circ$ is the element-by-element (Hadamard/Schur) product: '$.*$' in matlab.]

```
for $t = 1:T,\ i = 1:K$      $p_t(i) = A_i(\mathbf{x}_t)$
                             $\alpha_1 = \boldsymbol{\pi} \circ p_1$      $\rho_1 = \sum_{i=1}^K \alpha_1(i)$      $\alpha_1 = \alpha_1/\rho_1$
for $t = 2:T$                $\alpha_t = (\Phi^\mathsf{T} * \alpha_{t-1}) \circ p_t$      $\rho_t = \sum_{i=1}^K \alpha_t(i)$      $\alpha_t = \alpha_t/\rho_t$
                             $\beta_T = 1$
for $t = T-1:1$              $\beta_t = \Phi * (\beta_{t+1} \circ p_{t+1})/\rho_{t+1}$
                             $\log P(\mathbf{x}_{1:T}) = \sum_{t=1}^T \log(\rho_t)$
for $t = 1:T$                $\gamma_t = \alpha_t \circ \beta_t$
for $t = 1:T-1$              $\xi_t = \Phi \circ (\alpha_t * (\beta_{t+1} \circ p_{t+1})^\mathsf{T})/\rho_{t+1}$
```

HMM PSEUDOCODE: PARAMETER RE-ESTIMATION (M STEP)

Baum-Welch parameter updates:

For each sequence $l = 1 : L$, run forward–backward to get $\gamma^{(l)}$ and $\xi^{(l)}$, then

$$\pi_i = \frac{1}{L} \sum_{l=1}^{L} \gamma_1^{(l)}(i)$$

$$\Phi_{ij} = \frac{\sum_{l=1}^{L} \sum_{t=1}^{T^{(l)}-1} \xi_t^{(l)}(ij)}{\sum_{l=1}^{L} \sum_{t=1}^{T^{(l)}-1} \gamma_t^{(l)}(i)}$$

$$A_{ik} = \frac{\sum_{l=1}^{L} \sum_{t=1}^{T^{(l)}} \delta(\mathbf{x}_t = k)\gamma_t^{(l)}(i)}{\sum_{l=1}^{L} \sum_{t=1}^{T^{(l)}} \gamma_t^{(l)}(i)}$$

APPLICATION: SPEECH RECOGNITION

Problem
Given speech in form of a sound signal, determine the words that have been spoken.

Method

- Words are broken down into small sound units (called *phonemes*). The states in the HMM represent phonemes.
- The incoming sound signal is transformed into a sequence of vectors (feature extraction). Each vector x_n is indexed by a time step n.
- The sequence $x_{1:N}$ of feature vectors is the observed data in the HMM.

Phoneme Models

Phoneme

A **phoneme** is defined as the smallest unit of sound in a language that distinguishes between distinct meanings. English uses about 50 phonemes.

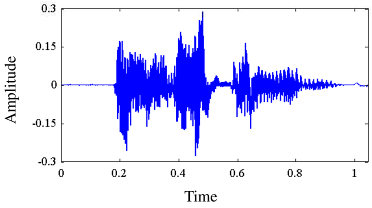
Example

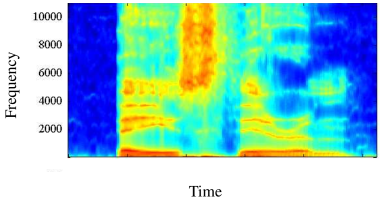
Zero	Z IH R OW	Six	S IH K S
One	W AH N	Seven	S EH V AX N
Two	T UW	Eight	EY T
Three	TH R IY	Nine	N AY N
Four	F OW R	Oh	OW
Five	F AY V		

Subphonemes

Phonemes can be further broken down into subphonemes. The standard in speech processing is to represent a phoneme by three subphonemes ("triphons").

PREPROCESSING SPEECH

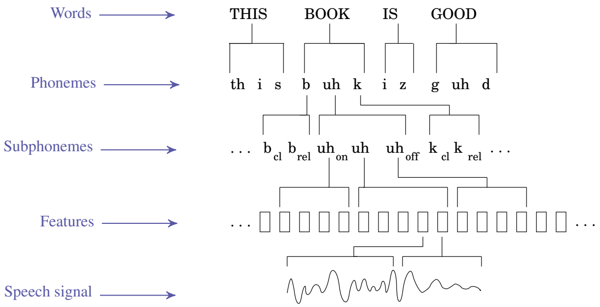




Feature extraction

- A speech signal is measured as amplitude over time.
- The signal is typically transformed into various types of features, including (windowed) Fourier- or cosine-transforms and so-called "cepstral features".
- Each of these transforms is a scalar function of time. All function values for the different transforms at time t are collected in a vector, which is the feature vector (at time t).

Layers in Phoneme Models



HMM speech recognition

- Training: The HMM parameters (emission parameters and transition probabilities) are estimated from data, often using both supervised and unsupervised techniques.
- Recognition: Given a language signal (= observation sequence $x_{1:N}$, estimate the corresponding sequence of subphonemes (= states $z_{1:N}$). This is a decoding problem.

Speaker Adaptation

Factory model

Training requires a lot of data; software is typically shipped with a model trained on a large corpus (i.e. the HMM parameters are set to "factory settings").

The adaptation problem

- The factory model represents an average speaker. Recognition rates can be improved drastically by adapting to the specific speaker using the software.
- Before using the software, the user is presented with a few sentences and asked to read them out, which provides labelled training data.

Speaker adaptation

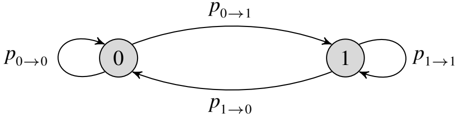
- Transition probabilities are properties of the language. Differences between speakers (pronounciation) are reflected by the emission parameters θ_k.
- Emission probabilities in speech are typically multi-dimensional Gaussians, so we have to adapt means and covariance matrices.
- The arguably most widely used method is **maximum likelihood linear regression** (**MLLR**), which uses a regression technique to make small changes to the covariance matrices.

MORE ON MARKOV CHAINS

GRAPHICAL REPRESENTATION

A simple binary chain

We consider a Markov chain $X_1, X_2, \ldots$ with state space $\mathcal{X}$. Suppose $\mathcal{X} = \{0, 1\}$.



- We regard 0 and 1 as possible "states" of X, represented as vertices in a graph.
- Each pair $X_{n-1} = s, X_n = t$ in the sequence is regarded as a "transition" from s to t and represented as an edge in the graph.
- Each edge $s \to t$ is weighted by the probability

$$p_{s\to t} := P(X_n = t | X_{n-1} = s) .$$

Caution: The graph is *not* a graphical model. (For computer scientists: It is a probabilistic finite automaton.)

Stationarity

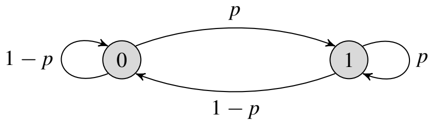
The graph representation is only possible if $p_{s\to t}$ is independent of n. Otherwise we would have to draw a different graph for each n.

If $p_{s\to t}$ does not depend on n, the Markov chain is called **stationary**.

Graphical Representation

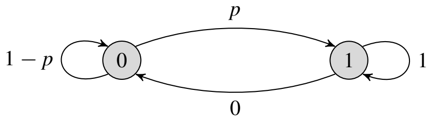
First example: Independent coin flips

Suppose X is a biased coin with $P(X_n = 1) = p$ independently of X_{n-1}. In other words, the sequence (X_n) is iid Bernoulli with parameter p.



Breaking independence

Here is a simple modification to the chain above; only $p_{1\to 0}$ and $p_{1\to 1}$ have changed:



This is still a valid Markov chain, but the elements of the sequence are no longer independent.

GRAPHICAL REPRESENTATION

Transition matrix

The probabilities $p_{s \to t}$ are called the **transition probabilities** of the Markov chain. If $|\mathcal{X}| = d$, the $d \times d$-matrix

$$\mathbf{p} := (p_{i \to j})_{j,i \leq d} = \begin{pmatrix} p_{1\to 1} & \cdots & p_{d \to 1} \\ \vdots & & \vdots \\ p_{1 \to d} & \cdots & p_{d \to d} \end{pmatrix}$$

is called the **transition matrix** of the chain. This is precisely the adjacency matrix of the graph representing the chain. Each column is a probability distribution on d events.

Initial distribution

The distribution of the first state, i.e. the vector

$$P_{\text{init}} := (P(X_0 = 1), \ldots, P(X_0 = d)) ,$$

is called the **initial distribution** of the Markov chain.

The distribution of a stationary Markov chain with finite state space is completely determined by the pair $(\mathbf{p}, P_{\text{init}})$.

STATE PROBABILITIES

Distribution after the first step

- If we *know* the initial state, then
$$P(X_1 = s_1 \,|\, X_0 = s_0) = p_{s_0 \to s_1} \;.$$
- Now consider the marginal distribution P_1 of X_1. This is the distribution X_1 if we do *not* know the initial state (e.g. before we start the chain):
$$P_1(s_1) = P(X_1 = s_1) = \sum_{s_0 \in \mathcal{X}} P(X_1 = s_1 \,|\, X_0 = s_0) P_{\text{init}}(s_0) = \sum_{s_0 \in \mathcal{X}} p_{s_0 \to s_1} P_{\text{init}}(s_0) \;.$$
- Since $\mathbf{p}$ is a $d \times d$-matrix and P_{init} a vector of length d, we can write that as
$$P_1 = \mathbf{p} \cdot P_{\text{init}} \;.$$
- The same argument shows that P_2 is given by $P_2 = \mathbf{p} \cdot P_1 = \mathbf{p} \cdot \mathbf{p} \cdot P_{\text{init}}$ et cetera.

The marginal distribution of chain's state after n steps is
$$P_n = \mathbf{p}^n P_{\text{init}}$$

Limits and Equilibria

Limiting distribution

Instead of considering P_n for a specific, large n, we take the limit

$$P_\infty := \lim_{n\to\infty} P_n = \lim_{n\to\infty} \mathbf{p}^n P_{\text{init}} \ ,$$

provided that the limit exists.

Observation

If the limit P_∞ exists, then

$$\mathbf{p} \cdot P_\infty = \mathbf{p} \cdot \lim_{n\to\infty} \mathbf{p}^n P_{\text{init}} = \lim_{n\to\infty} \mathbf{p}^n P_{\text{init}} = P_\infty \ ,$$

which motivates the next definition.

Let $\mathbf{p}$ be the transition matrix of a Markov chain. A distribution P on $\mathcal{X}$ which is invariant under $\mathbf{p}$ in the sense that

$$\mathbf{p} \cdot P = P$$

is called an **equilibrium distribution** or **invariant distribution** of the chain.

Does a chain have an equilibrium?

- A stationary Markov chain is: **aperiodic** if
$$P(X_n = s \mid X_{n-1} = s) = p_{s \to s} > 0 \qquad \text{for all } s \in \mathcal{X} \; .$$
That is: The transition matrix has non-zero diagonal.
- It is **irreducible** if there is a path (with non-zero probability) from each state to every other state in the transition graph.

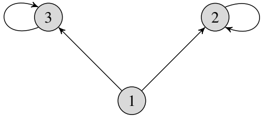
Theorem

If a first-order, stationary Markov chain with finite state space is aperiodic and irreducible, then:

- The limit distribution P_∞ exists.
- The limit distribution is also the equlibrium distribution.
- The equilibrium distribution is unique.
- The equilibrium does not depend on the initial distribution.

MEANING OF THE CONDITIONS

Why an equilibrium distribution may not be unique



- For this chain, both $P = (0, 1, 0)$ and $P' = (0, 0, 1)$ are valid equilibria.
- Which one emerges depends on the initial state and (if we start in state 1) on the first transition.

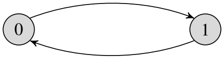
Remedy: Irreducibility. An irreducible chain could move between states 2 and 3.

Why the limit may not exist

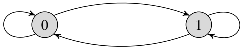
Recall that a sequence in $\mathbb{R}$ does not have a limit if it "oscillates". For example, $\lim_n 1^n = 1$, but $\lim_n (-1)^n$ does not exist.

- The chain on the right has no limit distribution.
- If we start e.g. in state 0, then 0 can only be reached in even steps, 1 only in odd steps.
- The distribution P_n oscillates between

$$P_{\text{even}} = (1, 0) \qquad \text{and} \qquad P_{\text{odd}} = (0, 1) \ .$$



Remedy: Require the chain to be aperiodic. That would add two edges to the graph:



The theorem shows that these simple fixes to the two most obvious problems suffice to ensure existence of a unique equilibrium.

COMPUTING THE EQUILIBRIUM

Power method

If the the transition matrix $\mathbf{p}$ makes the chain irreducible and aperiodic, we know that

$$\text{equilibrium distribution} = \text{limit distribution} \quad .$$

This means we can approximate the equilibrium P_∞ by P_n:

- Initialize with any distribution P_{init} (e.g. uniform).
- Repeat $P_{n+1} = \mathbf{p} \cdot P_n$.
- Terminate once $\|P_{n+1} - P_n\| < \tau$ for some small τ.

Eigenstructure

- The definition $P = \mathbf{p} \cdot P$ of the equilibrium means that $P = P_\infty$ is an eigenvector of $\mathbf{p}$ with eigenvalue 1.
- If $\mathbf{p}$ is irreducible and aperiodic, it can be shown that 1 is the largest eigenvalue.
- How quickly the power method converges depends on the ratio between the largest and second-largest eigenvalue (the spectral gap).

APPLICATION: WEB SEARCH

The link structure of the web is represented by the **web graph**, defined by the adjacency matrix

$$A = (A_{ij})_{i,j \leq \#\text{pages}} \quad \text{where} \quad A_{ij} = \begin{cases} 1 & \text{page } i \text{ links to page } j \\ 0 & \text{otherwise} \end{cases}$$

Vertices represent pages, edges represent links. The graph is directed.

Simple random walk

Let G be a directed graph with d vertices. Generate a sequence $X_0, X_1, \ldots$ of vertices:

- Select a vertex X_0 in G uniformly at random.
- For $n = 1, 2, \ldots$, select X_n uniformly at random from the children of X_{n-1} in the graph.

That defines a Markov chain whose state space is the vertex set of the graph, with

$$P_{\text{init}} = \left(\tfrac{1}{d}, \ldots, \tfrac{1}{d}\right) \qquad \text{and} \qquad p_{i \to j} = \begin{cases} \frac{1}{\#\text{ edges out of } i} & \text{if } i \text{ links to } j \\ 0 & \text{otherwise} \end{cases}$$

This Markov chain is called **simple random walk** on G.

The PageRank Algorithm

(Approximately) compute P_∞ for simple random walk on the web graph.

Interpretation of PageRank

Problem: Ranking queries

The first step in internet search is query matching:

- The user enters a search query (a string of words).
- The search engine determines all web pages in its database which match the query.
- This is a large set (typically tens or hundreds of millions). The results are only useful if the "best" links can be filtered out by the engine.

Available data

- Using a web crawler, we can (approximately) determine the link structure of the internet.
- We can determine which pages there are, and which pages they link to.
- We cannot determine how often a link is followed, or how often a page is visited.

PageRank solution

- PageRank uses $P_\infty(v)$ as the score of web page v. It ranks by decreasing score.
- This uses (i) popularity as a proxy for quality/usefulness, and (ii) the amount of incoming links as a proxy for popularity.
- P_1 would measure how often a page is linked. P_2 weights this by how often the linking page is linked, etc.
- The PageRank paper interpreted $P_\infty(v)$ as the probability that "random web surfer" would end up on page v after randomly following a large number of links. Note the start page does not matter for $n \to \infty$.

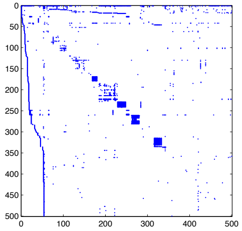
PageRank

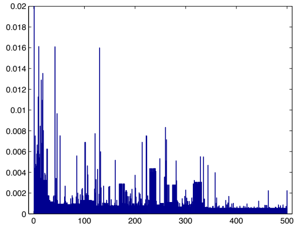
- Simple random walk on the web graph has transition matrix T with

$$T_{ij} := \begin{cases} \frac{1}{\#\text{ edges out of } i} & \text{if } i \text{ links to } j \\ 0 & \text{otherwise} \end{cases}$$

  This is typically not irreducible (think of web pages which do not link anywhere).
- PageRank therefore uses the regularized transition matrix $\mathbf{p}_{ij} := (1-\alpha)T_{ij} + \frac{\alpha}{d}$, for some small $\alpha \in (0,1)$. Clearly, this makes $\mathbf{p}$ both irreducible and aperiodic.
- The equilibrium can be approximated by the power method. Since the web changes, it can be re-run every few days with the previous equilibrium as initial distribution.

Example


Adjacence matrix of the web graph of 500 web pages. The root (index 0) is www.harvard.edu.


Equilibrium distribution computed by PageRank.

SAMPLING ALGORITHMS

Sampling Algorithms

A **sampling algorithm** takes a distribution P as input and outputs random values $X_1, \dots, X_n$ whose marginal distribution is P. Ideally, these draws are independent.

We will see that there are distributions that are hard to work with analytically, but relatively easy to sample. If so, we can use the sampler output for example to:

- Compute expectations: If $X_1, \dots, X_n$ are independent,
$$\mathbb{E}_P[f(X)] \approx \frac{1}{n}\sum_{i=1}^{n} f(X_i)$$
by the law of large numbers (for a given function f).
- Approximate the distribution by using the samples as input for density estimation.

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 $\Theta_1, \Theta_2, \dots$ distributed according to $\hat{Q}_n$.
- This reduces posterior estimation to a density estimation problem (i.e. estimate $\hat{Q}_n$ from $\Theta_1, \Theta_2, \dots$).

PREDICTIVE DISTRIBUTIONS

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:

$$p(x_{n+1}|x_{1:n}) := \int_{\mathbf{T}} p(x_{n+1}|\theta)\hat{Q}_n(d\theta|X_{1:n} = x_{1:n}) \; .$$

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

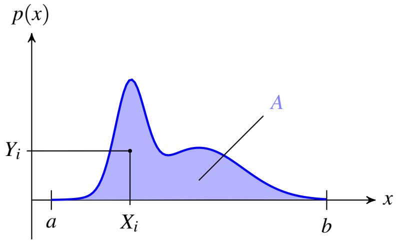
Computation from samples

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

$$p(x_{n+1}|x_{1:n}) = \mathbb{E}_{\hat{Q}_n}[p(x_{n+1}|\Theta)] \approx \frac{1}{m}\sum_{i=1}^{m} p(x_{n+1}|\Theta_i)$$

The Key Idea

Consider a probability density p on the interval $[a, b]$.



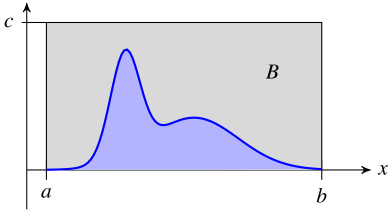
Suppose we can define a uniform distribution U_A on the blue area A under the curve.

If we generate $(X_1, Y_1), (X_2, Y_2), \ldots \sim_{\text{iid}} U_A$ then $X_1, X_2, \ldots \sim_{\text{iid}} p$.

Rejection Sampling on the Interval

Problem: We do not know how to define a uniform distribution on an arbitrarily shaped area.

Solution: We enclose p in a box B, sample uniformly from the box, and discard all draws not in the blue area.



Algorithm: Rejection sampling

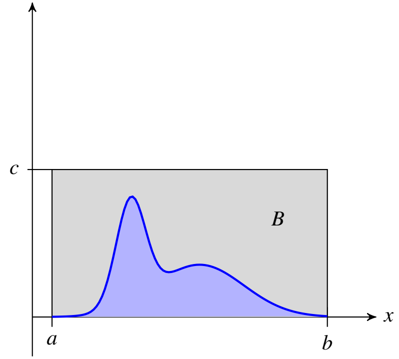
- Generate (X_i, Y_i) uniformly on B, by independently sampling

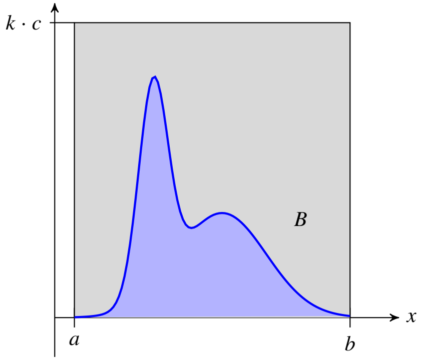
$$X_i \sim \text{Uniform}[a,b] \qquad \text{and} \qquad Y_i \sim \text{Uniform}[0,c] \; .$$

- If $Y_i \leq p(X_i)$, keep the sample.
- Otherwise: Discard ("reject") it.

Result: The remaining (non-rejected) samples are uniformly distributed on A.

SCALING





- This strategy still works if we scale vertically by some constant $k > 0$.
- We simply draw $Y_i \sim \text{Uniform}[0, kc]$ instead of $Y_i \sim \text{Uniform}[0, c]$.

For sampling, it suffices to know a distribution only up to normalization. That is, only the shape of p needs to be known.

Distributions Known up to Scaling

Sampling methods usually assume that we can evaluate the target distribution p up to a constant. That is:

$$p(x) \quad = \quad \frac{1}{\tilde{Z}}\,\tilde{p}(x)\;,$$

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

$$\Pi(\theta|x_{1:n}) = \frac{\prod_{i=1}^{n} p(x_i|\theta)q(\theta)}{\tilde{Z}} \qquad \text{with} \qquad \tilde{Z} = \int_{\mathbf{T}} \prod_{i=1}^{n} p(x_i|\theta)q(\theta)d\theta\;.$$

Provided that we can compute the numerator, we can sample without computing the normalization integral $\tilde{Z}$.

DISTRIBUTIONS KNOWN UP TO SCALING

Example 2: Bayesian Mixture Model

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

$$\hat{q}_n(c_{1:K}, \theta_{1:K} | x_{1:n}) \propto \prod_{i=1}^{n} \Big(\sum_{k=1}^{K} c_k p(x_i | \theta_k) \Big) \Big(\prod_{k=1}^{K} q(\theta_k) \Big) q_{\text{Dirichlet}}(c_{1:K})$$

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 $\prod_{i=1}^{n} \Big(\sum_{k=1}^{K} \dots \Big)$ blows up into K^n individual terms.
- If we *evaluate* $\tilde{q}_n$ for specific values of c, x and θ, $\sum_{k=1}^{K} c_k p(x_i | \theta_k)$ 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.

Distributions Known up to Scaling

Example 3: Markov random field

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

For example, recall the Ising model:

$$p(\theta_{1:n}) = \frac{1}{Z(\beta)} \exp\Big(\sum_{(i,j) \text{ is an edge}} \beta \mathbb{I}\{\theta_i = \theta_j\} \Big)$$

The normalization function is

$$Z(\beta) = \sum_{\theta_{1:n} \in \{0,1\}^n} \exp\Big(\sum_{(i,j) \text{ is an edge}} \beta \mathbb{I}\{\theta_i = \theta_j\} \Big)$$

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

On the other hand, evaluating

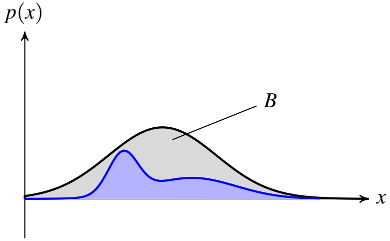
$$\tilde{p}(\theta_{1:n}) = \exp\Big(\sum_{(i,j) \text{ is an edge}} \beta \mathbb{I}\{\theta_i = \theta_j\} \Big)$$

for a given configuration $\theta_{1:n}$ is straightforward.

REJECTION SAMPLING ON $\mathbb{R}^d$

Problem: 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: Instead of a box, we use *another distribution r*, called a **proposal density**, to enclose p.

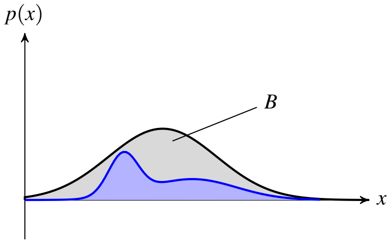


To generate B under r, we apply similar logic as before backwards:

- Sample $X_i \sim r$.
- Sample $Y_i | X_i \sim \text{Uniform}[0, r(X_i)]$.

We always choose r as a simple distribution that we know how to sample and evaluate.

REJECTION SAMPLING ON $\mathbb{R}^d$



- Choose a distribution r from which we know how to sample.
- Scale $\tilde{p}$ such that $\tilde{p}(x) < r(x)$ everywhere.

Algorithm: Rejection sampling For $i = 1, 2, \ldots$:

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

The retained samples are distributed according to p.

PROPERTIES

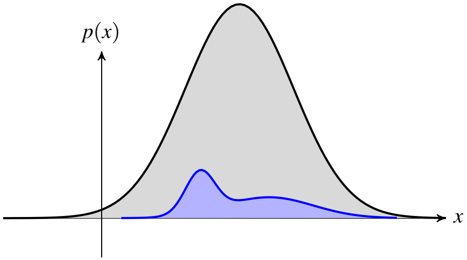
Independence

If we draw proposal samples i.i.d. from r, the accepted samples are i.i.d.

Rejection samplers produce i.i.d. sequences of samples. If $X_1, X_2, \dots$ are generated by rejection sampling, $\frac{1}{m}\sum_{i \leq m} f(X_i)$ is an unbiased estimate of $\mathbb{E}_p[f(X)]$.

Efficiency

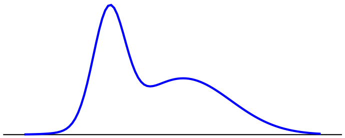
The fraction of accepted samples is the ratio $\frac{|A|}{|B|}$ of the areas under the curves $\tilde{p}$ and r.

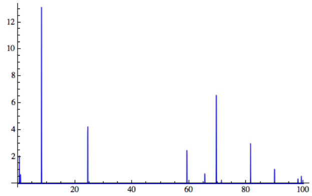


If r is not a reasonably close approximation of p, most proposals are rejected.

An important bit of imprecise intuition

- Sampling is typically used for distributions of multiple, dependent random variables.
- Reason: One-dimensional distributions can usually be handled without sampling. Multiple independent variables factorize into one-dimensional distributions.
- High-dimensional distributions with dependence often concentrate on many small areas strewn out over the sample space, with regions of effectively zero probability in between.

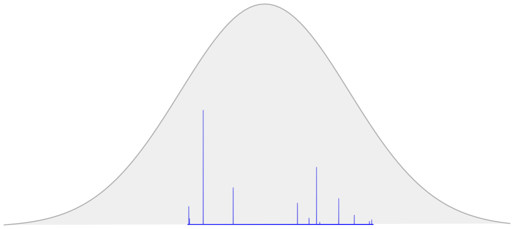

Textbook illustrations of target distributions tend to look like this.



This is a (purely qualitative) attempt to visualize in one dimension what a high-dimensional distribution might look like.

WHY IS NOT EVERY SAMPLER A REJECTION SAMPLER?

- Try to picture the illustration above overlayed with a “simple” proposal distribution.



- Recall that the efficiency is the ratio of the blue and gray areas.
- We can easily end up in situations where we accept only one in 10^6 (or 10^{10}, or 10^{20},...) proposal samples.
- Even in moderate dimensions, we have to expect this to be not the exception but the rule.

Importance Sampling

There is a simple way to improve on rejection sampling if we are specifically interested in approximating an expectation $\mathbb{E}_p[f(X)]$.

Simple case: We can evaluate p

Let p be the target and q a proposal density. Rewrite the expecation under p as

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

Algorithm

Algorithm: Importance sampling

- Draw $X_1, X_2, \dots$ i.i.d. from proposal q. Do not discard any samples.
- Approximate the expectation of f as

$$\mathbb{E}_p[f(X)] \;\approx\; \tfrac{1}{m}\textstyle\sum_{i\leq m} f(X_i)\tfrac{p(X_i)}{q(X_i)}$$

- The coefficients $\frac{p(X_i)}{q(X_i)}$ are called **importance weights**.
- There are no rejections (but there are samples with small weights).

IMPORTANCE SAMPLING: GENERAL CASE

- Now assume we can only evaluate p up to scaling:

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

where Z_p (and possibly Z_q) are unknown constants.

- Observe that we can estimate the fraction Z_p/Z_q using samples $X_1, \dots, X_m \sim_{\text{iid}} q$:

$$\frac{Z_p}{Z_q} = \frac{\int \tilde{p}(x)dx}{Z_q} = \frac{\int \tilde{p}(x)\frac{q(x)}{q(x)}dx}{Z_q} = \int \tilde{p}(x)\frac{q(x)}{Z_q \cdot q(x)}dx = \mathbb{E}_q\left[\frac{\tilde{p}(X)}{\tilde{q}(X)}\right] \quad \approx \quad \frac{1}{m}\sum_{i\leq m}\frac{\tilde{p}(X_i)}{\tilde{q}(X_i)}$$

- The estimator of the expectation of f is then:

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

Algorithm: Importance sampling (*p* known up to scaling)

Draw $X_1, X_2, \dots$ i.i.d. from proposal q, and approximate the expectation as

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

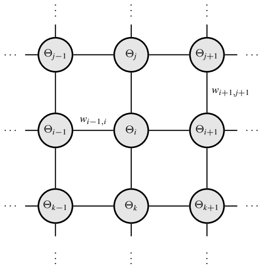
MARKOV RANDOM FIELDS

RANDOM FIELDS

- We define a **neighborhood graph**, which is a weighted, undirected graph:

$$\mathcal{N} = (\overset{\text{vertex set}}{\overbrace{V_{\mathcal{N}}}}, \overset{\text{set of edge weights}}{\overbrace{W_{\mathcal{N}}}})$$

- The edge weights are scalars $w_{ij} \in \mathbb{R}$. An edge weight $w_{ij} = 0$ means "no edge between v_i and v_j".
- Since $\mathcal{N}$ is undirected, $w_{ij} = w_{ji}$.
- With each vertex v_i, we associate a random variable Θ_i.
- With each vertex v_i in the graph, we associate a random variable Θ_i. The joint distribution of these variables is called a random field.

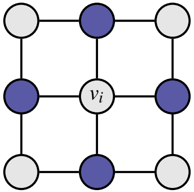


Neighborhoods

- The **neighborhood** of vertex v_i is the set

$$\partial(i) := \{ j \mid w_{ij} \neq 0 \} .$$

- The set $\{\Theta_j, j \in \partial(i)\}$ of random variables associated with the neighborhood is the **Markov blanket** of Θ_i.



purple = $\partial(i)$

Markov Random Fields

The Markov property

- A random field has the **Markov property** if
$$P(\theta_i|\theta_j,\ j \neq i) = P(\theta_i|\theta_j,\ j \in \partial(i))\ .$$
That is: Each Θ_i is conditionally independent of the remaining field given its Markov blanket.
- A random field with the Markov property is a **Markov random field (MRF)**.

Energy functions

- Any (strictly positive) probability or density p can be written in the form
$$p(x) = \frac{1}{Z}\exp(-H(x)) \qquad \text{where} \qquad H : \mathcal{X} \to \mathbb{R}_+ \;\; \text{and} \;\; Z := \int e^{-H(x)}p(x)dx$$
and Z is a normalization constant.
- The function H is called an **energy function** or **potential**, or sometimes a **cost function**.
- In particular, we can write a MRF density for RVs $\Theta_{1:n}$ as
$$p(\theta_1, \ldots, \theta_n) = \frac{1}{Z}\exp(-H(\theta_1, \ldots, \theta_n))$$

The Potts Model

Let $\mathcal{N}$ be a neighborhood graph with weights w_{ij}, and $\beta > 0$. The Markov random field

$$p(\theta_1, \dots, \theta_n) := \frac{1}{Z(\beta)} \exp\Big(\beta \sum_{i,j} w_{ij} \mathbb{I}\{\theta_i = \theta_j\}\Big)$$

is called a **Potts model**.

- Note the energy is additive over pairs.
- Positive weights encourage *smoothness*:

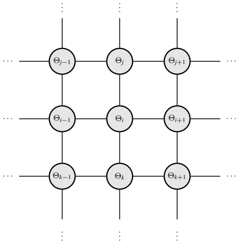
$w_{ij} > 0$	$\theta_i = \theta_j$ *increases* probability
$w_{ij} < 0$	$\theta_i = \theta_j$ *decreases* probability
$w_{ij} = 0$	no interaction between θ_i and θ_j

THE ISING MODEL

- If $\theta_i \in \{-1, +1\}$ and $w_{ij} \in \{0, 1\}$, we obtain

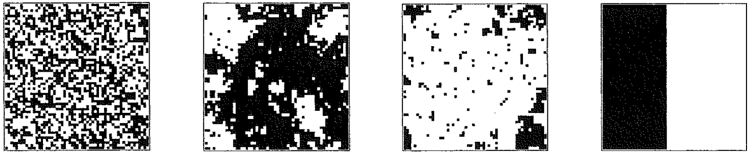
$$p(\theta_1, \dots, \theta_n) = \frac{1}{Z(\beta)} \exp\Big(\sum_{(i,j) \text{ is an edge}} \beta \mathbb{I}\{\theta_i = \theta_j\} \Big)$$

- If $\mathcal{N}$ is a d-dim. grid, this model is called the **Ising model**.
- This is the simplest non-trivial Potts model, but many of its mathematical properties remain unsolved.



Example

Samples from an Ising model on a 56 × 56 grid graph.



Increasing $\beta \quad \longrightarrow$

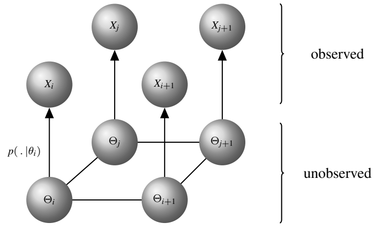
Color coding: Black = -1, white = $+1$.

MRFs as Smoothness Priors

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 Θ_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.

BAYESIAN MIXTURE MODELS

Definition
A model of the form

$$\pi(x) = \sum_{k=1}^{K} C_k p(x|\Theta_k)$$

is called a **Bayesian mixture model** if $p(x|\theta)$ is an exponential family model and

- $\Theta_1, \ldots, \Theta_K \sim_{\text{iid}} q$, where q is a prior we have chosen for Θ.
- $(C_1, \ldots, C_K)$ is sampled from a K-dimensional Dirichlet distribution.

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

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

EXAMPLE: SEGMENTATION OF NOISY IMAGES

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 θ_i. For K mixture components, $\theta_{1:n}$ contains only K different values.
- The joint BMM prior on $\theta_{1:n}$ is

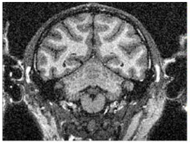
$$q_{\text{BMM}}(\theta_{1:n}) = \prod_{i=1}^{n} q(\theta_i) \ .$$

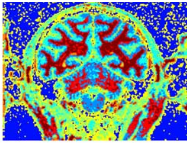
Smoothing term

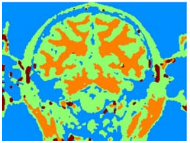
We multiply the BMM prior $q_{\text{BMM}}(\theta)$ with an MRF prior

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

This encourages spatial smoothnes of the segmentation.


Input image.


Segmentation w/o smoothing.


Segmentation with MRF smoothing.

Sampling and Inference

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.

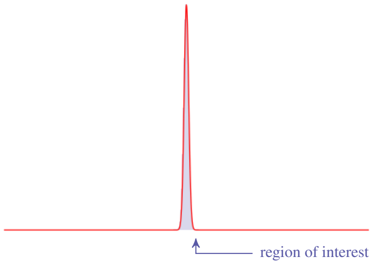
Solution

- 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.

MARKOV CHAIN MONTE CARLO

MOTIVATION

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.

MCMC: Idea

Recall: Markov chain

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

CONTINUOUS MARKOV CHAIN

The Markov chains we discussed so far had a finite state space $\mathcal{X}$. 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_{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 $\mathbf{p}_{ij}$ of the transition matrix $\mathbf{p}$.

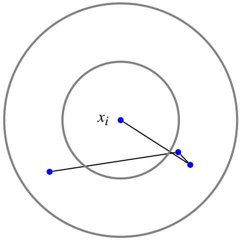
Example: A Markov chain on $\mathbb{R}^2$

We can define a very simple Markov chain by sampling

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

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

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



A Gaussian (gray contours) is placed around the current point x_i to sample X_{i+1}.

Invariant Distribution

Recall: Finite case

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

$$\mathbf{p} \cdot P_{\text{inv}} = P_{\text{inv}}$$

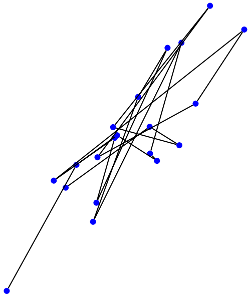
Continuous case

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

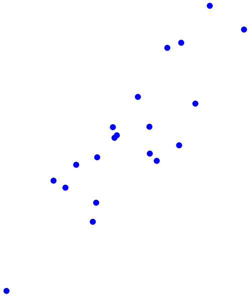
$$\int_{\mathcal{X}} t(y|x) p_{\text{inv}}(x) dx = p_{\text{inv}}(y)$$

  This is simply the continuous analogue of the equation $\sum_i \mathbf{p}_{ij}(P_{\text{inv}})_i = (P_{\text{inv}})_j$.

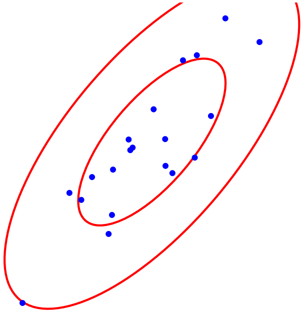
Markov Chain Sampling



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.

Constructing the Markov Chain

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 $\mathcal{X}$.
   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
   $$A(x_{i+1}|x_i) := \min\left\{1, \frac{q(x_i|x_{i+1})p(x_{i+1})}{q(x_{i+1}|x_i)p(x_i)}\right\}$$
   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
   $$t(x_{i+1}|x_i) := q(x_{i+1}|x_i)A(x_{i+1}|x_i)+\delta_{x_i}(x_{i+1})c(x_i) \quad \text{where} \quad c(x_i) := \overbrace{\int q(y|x_i)(1-A(y|x_i))dy}^{\text{total probability that a proposal is sampled and then rejected}}$$

Sampling from the MH chain

At each step $i+1$, generate a proposal $X^* \sim q(\,.\,|x_i)$ 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 $x_{i+1} := x_i$.

Stochastic Hill-Climbing

The Metropolis-Hastings rejection kernel was defined as:

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

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.

Problem 1: Initial distribution

Recall: Fundamental theorem on Markov chains

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

$$P_i \to P_{\text{inv}} \qquad \text{for} \qquad i \to \infty \,.$$

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_{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:

$$\underbrace{X_1, \dots, X_{m-1}}_{\text{Burn-in; discard.}}, \underbrace{X_m, X_{m+1}, \dots}_{\text{Samples from (approximately) } p\text{; keep.}}$$

Convergence diagnostics

In practice, we do not know how large m 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

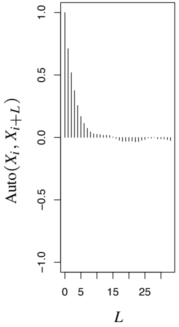
- Estimate empirically how many steps L are needed for x_i and x_{i+L} to be approximately independent. The number L is called the **lag**.
- After burn-in, keep only every Lth sample; discard samples in between.

Estimating the lag

The most commen method uses the **autocorrelation function**:

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

where μ_i is the mean and σ_i the standard deviation of X_i. 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.

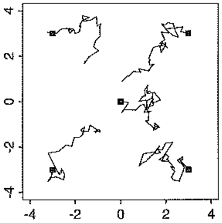


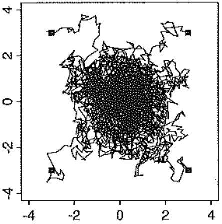
CONVERGENCE DIAGNOSTICS

There are about half a dozen popular convergence crieteria; the one below is an example.

Gelman-Rubin criterion

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

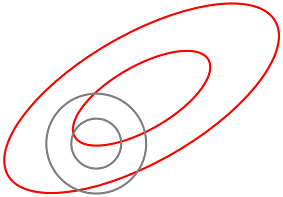




Reference: A. Gelman and D. B. Rubin: "Inference from Iterative Simulation Using Multiple Sequences", *Statistical Science*, Vol. 7 (1992) 457-511.

Selecting a Proposal Distribution

Everyone's favorite example: Two Gaussians



red = target distribution p
gray = proposal distribution q

- Var$[q]$ too large:
  Will overstep p; many rejections.
- 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, 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 Lth sample (where $L =$ lag) to make sure the x_i are (approximately) independent.

$$\underbrace{X_1,\ldots,X_{m-1}}_{\text{Burn-in; discard.}}, \overset{\text{Keep.}}{\overset{\downarrow}{X_m}}, \underbrace{X_{m+1},\ldots,X_{m+L-1}}_{\text{Samples correlated with } X_j\text{; discard.}}, \overset{\text{Keep.}}{\overset{\downarrow}{X_{m+L}}}, \underbrace{X_{m+L+1},\ldots X_{m+2L-1}}_{\text{Samples correlated with } X_{j+L}\text{; discard.}}, \overset{\text{Keep.}}{\overset{\downarrow}{X_{m+2L}}},\ldots$$

THE GIBBS SAMPLER

GIBBS SAMPLING

By far the most widely used MCMC algorithm is the Gibbs sampler.

Full conditionals

Suppose $\mathcal{L}(X)$ is a distribution on $\mathbb{R}^{\mathrm{D}}$, so $X = (X_1, \dots, X_{\mathrm{D}})$. The conditional probability of the entry X_d given all other entries,

$$\mathcal{L}(X_d | X_1, \dots, X_{d-1}, X_{d+1}, \dots, X_{\mathrm{D}})$$

is called the **full conditional** distribution of X_d.

If X has density p, that means we are interested in a density

$$p(x_d | x_1, \dots, x_{d-1}, x_{d+1}, \dots, x_{\mathrm{D}})$$

Gibbs sampling

- The Gibbs sampler is Markov chain Monte Carlo algorithm that generates each step dimension by dimension by sampling from the full conditionals.
- 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.

The Gibbs Sampler

Proposal distribution

Suppose p is a density or mass function of a random vector with D entries. Given a random vector X_i, we generate X_{i+1} coordinate-by-coordinate as follows:

$$
\begin{aligned}
X_{i+1,1} &\sim p(\,.\,|X_{i,2},\ldots,X_{i,\mathrm{D}}) \\
&\vdots \\
X_{i+1,d} &\sim p(\,.\,|X_{i+1,1},\ldots,X_{i+1,d-1},X_{i,d+1},\ldots,X_{i,\mathrm{D}}) \\
&\vdots \\
X_{i+1,\mathrm{D}} &\sim p(\,.\,|X_{i+1,1},\ldots,X_{i+1,\mathrm{D}-1})
\end{aligned}
$$

Note well: Each new draw $X_{i+1,d}$ is used *immediately* to update the next dimension $d+1$.

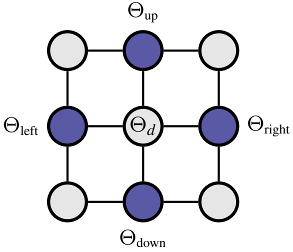
The MCMC algorithm that generates vectord $X_1, X_2, \ldots$ as above is called a **Gibbs sampler**.

Relationship to Metropolis-Hastings

- For each dimension $d \in \{1,\ldots,D\}$, the Markov process above defines a process $X_{1,d}, X_{2,d}, \ldots$, which is again a Markov process. One can show that this is a MH sampler, so a Gibbs sampler with D full conditionals is a family of coupled MH samplers.
- These MH samplers all have acceptance probability 1, so *proposals in Gibbs sampling are always accepted*.

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

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

Specifically: Potts model with binary weights

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

$$\tilde{p}(\theta_d|\theta_1,\dots,\theta_{d-1},\theta_{d+1},\dots,\theta_{\text{D}}) =$$
$$\exp\Big(\beta(\mathbb{I}\{\theta_d=\theta_{\text{left}}\}+\mathbb{I}\{\theta_d=\theta_{\text{right}}\}+\mathbb{I}\{\theta_d=\theta_{\text{up}}\}+\mathbb{I}\{\theta_d=\theta_{\text{down}}\}\Big)$$

This is clearly very efficiently computable.

EXAMPLE: MRF

Sampling the Potts model

Each step of the sampler generates a sample

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

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

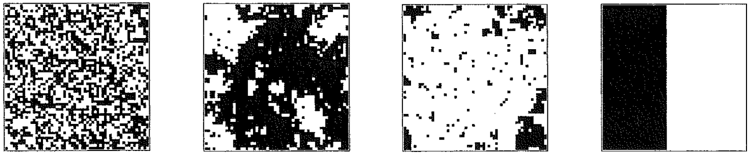
Gibbs sampler

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

$$\theta_{i+1,d} \sim p(\,.\, | \theta_{i+1,1}, \dots, \theta_{i+1,d-1}, \theta_{i,d+1}, \dots, \theta_{i,\mathrm{D}}) \\ \propto \exp\Big(\beta(\mathbb{I}\{\theta_{i+1,d} = \theta_{\text{left}}\} + \mathbb{I}\{\theta_{i+1,d} = \theta_{\text{right}}\} + \mathbb{I}\{\theta_{i+1,d} = \theta_{\text{up}}\} + \mathbb{I}\{\theta_{i+1,d} = \theta_{\text{down}}\}) \Big)$$

Recall the Ising model example

Samples from an Ising model on a 56×56 grid graph.

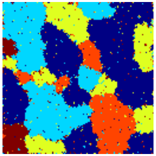


Increasing $\beta \quad \longrightarrow$

BURN-IN MATTERS

This example is due to Erik Sudderth (UC Irvine).

MRFs as "segmentation" priors

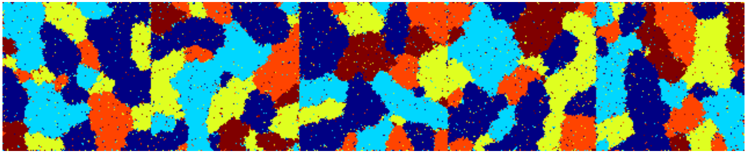


- MRFs where introduced as tools for image smoothing and segmentation by D. and S. Geman in 1984.
- They sampled from a Potts model with a Gibbs sampler, discarding 200 iterations as burn-in.
- Such a sample (after 200 steps) is shown above, for a Potts model in which each variable can take one out of 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.

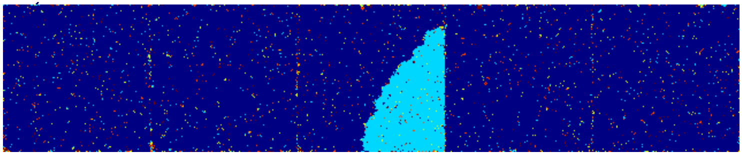
EXAMPLE: BURN-IN MATTERS

E. Sudderth and M. I. Jordan ran a Gibbs sampler on the same model, and recorded the state of the chain once after 200 iterations (as Geman & Geman had done), and again after 10000 iterations:

200 iterations



10000 iterations



Chain 1 Chain 5

- The "segmentation" patterns visible after 200 iterations are not sampled from the MRF distribution $p \equiv P_{\text{inv}}$, but rather from $P_{200} \neq P_{\text{inv}}$.
- The patterns do not show that MRFs are "natural" priors for segmentation problems, but simply *that the sampler's Markov chain is still in burn-in.*
- MRFs are smoothness priors, not segmentation priors.

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

$$[x]_2 = 2_j \cdot 10^j + b_{j-1} \cdot 2^{j-1} + \ldots + b_1 \cdot 2^0$$

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:

$$[0.1]_{10} = [0.0\overline{0011}]_2 = [0.000110011001100110011\ldots]_2$$

FLOATING POINT ARITHMETIC

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)

$$\begin{array}{cccccccc} s & e_1 & \cdots & e_n & m_1 & \cdots & m_k \\ 0 & 1 & & 8 & 9 & & 31 \end{array}$$

- 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)

$$\begin{array}{cccccccc} s & e_1 & \cdots & e_n & m_1 & \cdots & m_k \\ 0 & 1 & & 11 & 12 & & 63 \end{array}$$

- A double 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 Errors

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 two 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** $\frac{\hat{z}-z}{z}$.

The premise of FP arithmetic is that the relative rounding error is more important.

Rounding Errors

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

$$2 = (-1)^0 \cdot 2^1 \cdot 1 = (-1)^0 \cdot 2^0 \cdot 2 = \dots$$

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

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

IEEE 754 only permits normalized numbers. That means every number has a unique representation.

Machine Precision

Error guarantees

IEEE 754 guarantees that

$$x \hat{\oplus} y = (x \oplus y)(1+r) \qquad \text{satisfies} \qquad |r| < \text{eps}$$

where

- r is the relative rounding error.
- eps is machine precision.
- $\oplus$ is one of the operations $+$, $-$, $\times$ or $/$, and $\hat{\oplus}$ its machine version.

Machine precision

The **machine precision**, denoted eps, is the smallest machine number for which $\hat{1}\widehat{+}\text{eps}$ is distinguishable from 1:

$$\text{eps} = \min_{\hat{1}\widehat{+}e > \hat{1}} \{e \in M\}$$

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 this may cause serious problems are linear equation systems.
- If a matrix is inverted by Gaussian elimination, many terms are 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 use 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 (at the software level).
- 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.

RANDOM NUMBER GENERATION

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 $\mod$ operation outputs the "remainder after division":

$$m \mod n \quad = \quad m - kn \qquad \text{for largest } k \in \mathbb{N}_0 \text{ such that } kn < m$$

For example: $13 \mod 5 = 3$

Linear Congruence Generator

An example are generators of the form

$$x_{n+1} = C \cdot x_n \mod D \qquad \text{for fixed constants } C, D \in \mathbb{N} \,.$$

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

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

PERIODICITY

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 $(x_1, \dots, x_{n-1})$ repeats:
$$(x_1, \dots, x_{n-1}) = (x_n, \dots, x_{2n-1}) = \dots$$
- 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 seeds

Random seed of a generator

- A recursive generator $x_{n+1} = f(x_n)$ can be started at any intial 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.
- A proper experimental protocol for evaluating e.g. a machine learning algorithm:
  (i) Re-runs experiments several times with different seeds.
  (ii) Records the seeds used along with the code and other parameters for reproducibility.