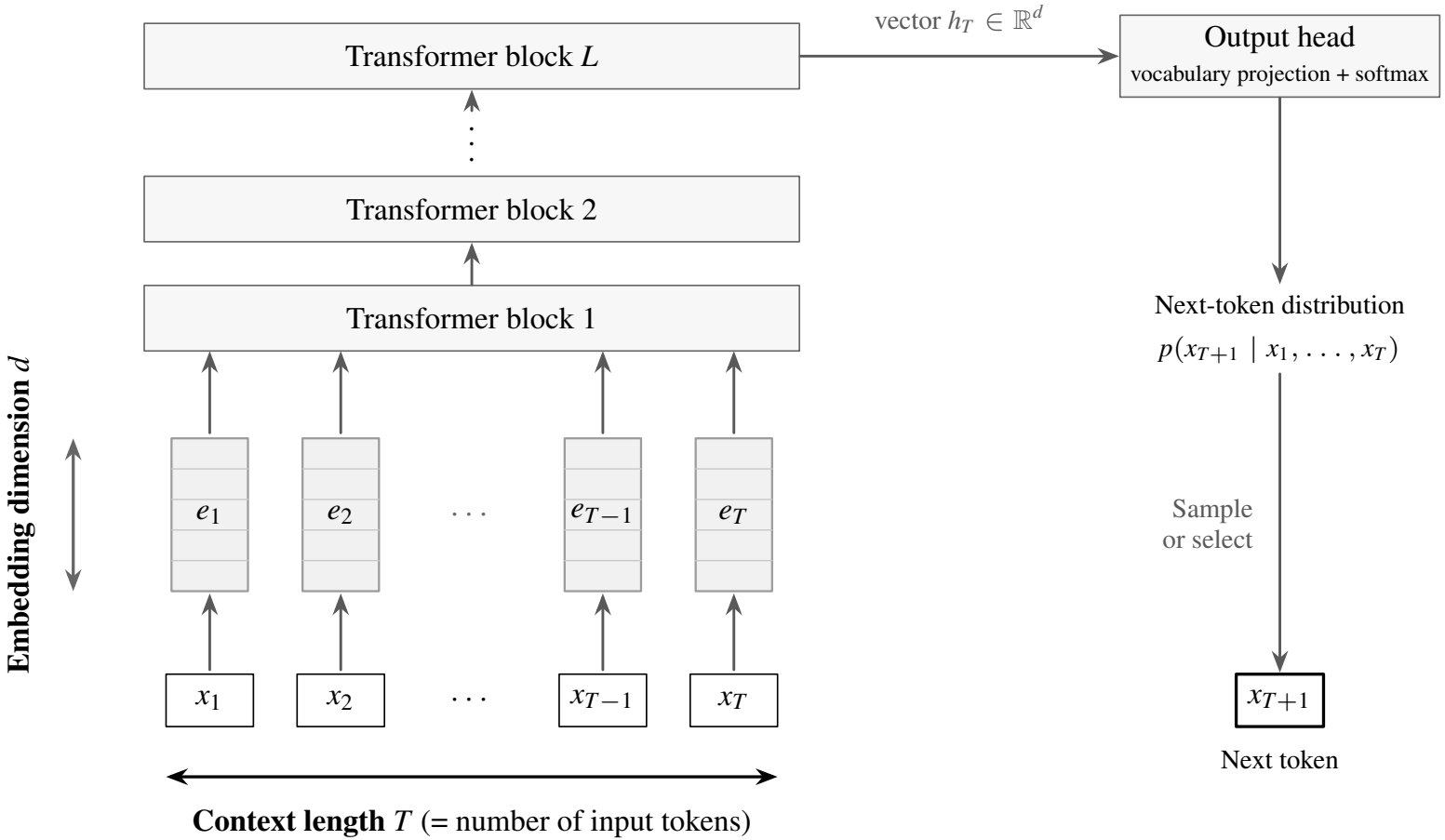
- Large language models represent probability distributions on sequences.
- They compute a distribution over the next word/token at each time step, as we can do with an HMM.
- Their hidden variables, however are deterministic, not stochastic.

How an LLM works, roughly

(The description here and on the next slides is one variant. There are many.)

- A sequence of words is translated into a sequence $x_1, \ldots, x_T$ of tokens.
- Each token is mapped to a vector (an "embedding") in $\mathbb{R}^d$.
- The sequence $\mathbf{e}_1, \ldots, \mathbf{e}_T$ of embedding vectors is passed to a function F called a transformer block, which produces output vectors $\mathbf{h}_1^{(1)}, \ldots, \mathbf{h}_T^{(1)} \in \mathbb{R}^d$.
- This is typically repeated some number L of times: A second tranformer block F_2 reads the vectors $\mathbf{h}_t^{(1)}$ and outputs vectors $\mathbf{h}_t^{(2)} \in \mathbb{R}^d$, and so forth.
- After L layers, the final (i.e. $t = T$) vector $\mathbf{h}_T = \mathbf{h}_T^{(L)}$ is transformed into a probability distribution $p(x_{T+1}|\mathbf{h}_T)$ on the set of tokens, which is used as a the distributions of x_{T+1}.
- We can then generate the next token x_{T+1} randomly, by sampling from $p(x_{T+1}|\mathbf{h}_T)$, or greedily, as the maximizer of $p(x_{T+1}|\mathbf{h}_T)$.

SCHEMATIC OVERVIEW



- Parameters of all components shaded gray (the transformer blocks, embedding map, and output blocks) are learned.
- Computation and memory cost scale with (i) the context length T, (ii) the embedding dimension d, and (iii) the number of parameters in the transformer blocks and output head.

In (slightly) more detail

Define a function $H : \mathbb{R}^{d\times T} \times \{1,\dots,T\} \longrightarrow \mathbb{R}^d$ with parameters $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{d_k\times d}$ and $\mathbf{C} \in \mathbb{R}^{d\times d}$:

$$H(\mathbf{h}_1,\dots,\mathbf{h}_T,t) \;:=\; \sum_{s=1}^{t} \underbrace{\frac{\exp(\langle \mathbf{A}\cdot\mathbf{h}_s, \mathbf{B}\cdot\mathbf{h}_t\rangle / \sqrt{d_k})}{\sum_{r\le t}\exp(\langle \mathbf{A}\cdot\mathbf{h}_r, \mathbf{B}\cdot\mathbf{h}_t\rangle / \sqrt{d_k})}}_{\text{set to 0 for } s>t} \mathbf{C}\cdot\mathbf{h}_s$$

This function is often referred to as attention. The weights (the quotients) sum to one.

Transformer block

A transformer block is a function of the form

$$F(\mathbf{h}_1,\dots,\mathbf{h}_T,t) \;:=\; \mathrm{Norm}_2(\mathrm{Norm}_1(\mathbf{h}_t + H(\mathbf{h}_1,\dots,\mathbf{h}_T,t)) + W_2\phi(W_1\mathbf{u}_t + \mathbf{b}_1) + \mathbf{b}_2)\,.$$

- The matrix $W_1 \in \mathbb{R}^{D\times d}$ expands dimension to D, and $W_2 \in \mathbb{R}^{d\times D}$ contracts it back to d.
- $D > d$, often $D \approx 4d$.
- $\mathbf{b}_1 \in \mathbb{R}^D$ and $\mathbf{b}_2 \in \mathbb{R}^d$ are bias vectors. W_1, W_2, $\mathbf{b}_1 \in \mathbb{R}^D$ and $\mathbf{b}_2 \in \mathbb{R}^d$ are learned.
- ϕ is an elementwise non-linearity (e.g. a ReLU). Norm_1, Norm_2 are layer normalizations.

Multiple heads

One often combines multiple attention functions (or "heads") $H_1,\dots,H_m$, each with their own parameters $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$. Their output is concatenated to a vector of length $d\cdot m$, projected back to $\mathbb{R}^d$ by a learned $d \times dm$-matrix, and substituted into F for H.

Output head

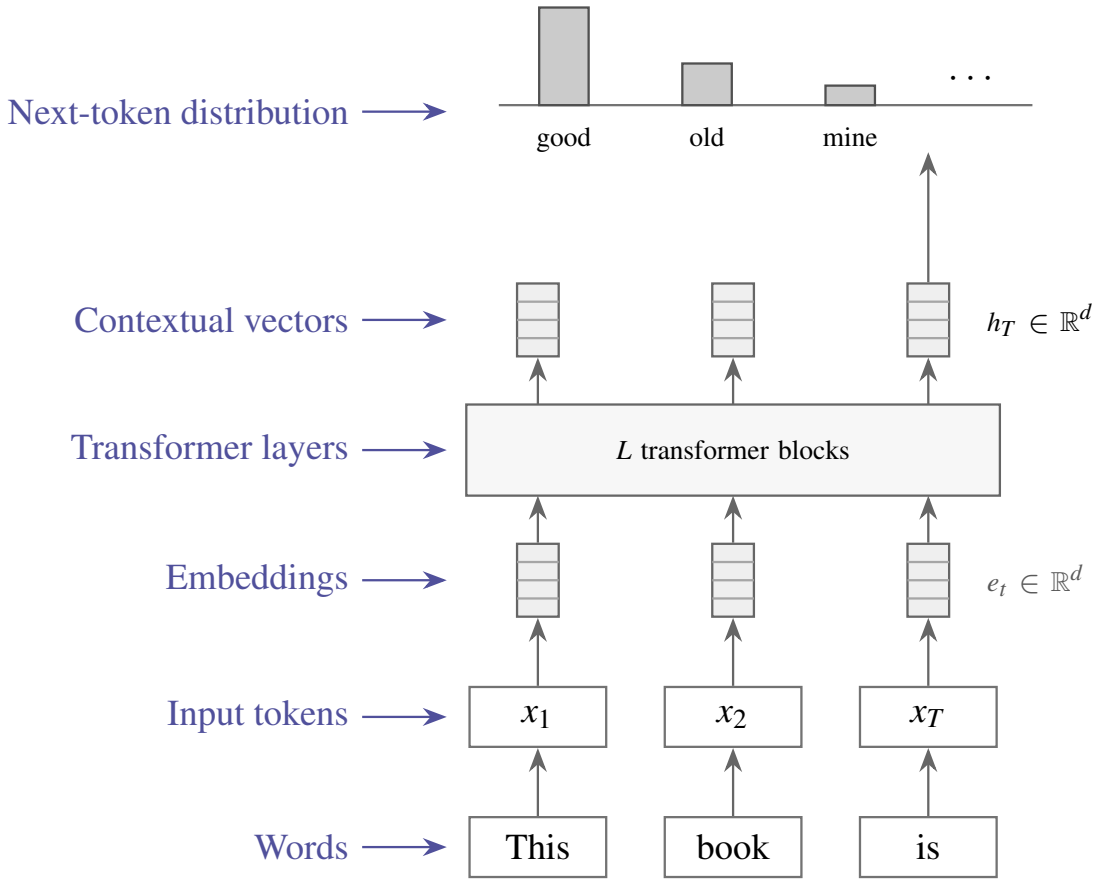
The final hidden vector $\mathbf{h}_T \in \mathbb{R}^d$ represents "the past" of token x_{T+1}. The predictive distribution of the next token is computed as

$$P(x_{T+1}|\mathbf{h}_T) \;:=\; \frac{\exp(z_j)}{\sum_{k=1}^{|V|}\exp(z_k)} \qquad \text{for} \quad \mathbf{z} \;=\; W_{\text{out}}\mathbf{h}_T + \mathbf{b}$$

for a $|V| \times d$-matrix W_{out} and vectors $\mathbf{b}$ and $\mathbf{z}$ in $\mathbb{R}^{|V|}$. Many models use $\mathbf{b} = 0$.

Dimensions

Model	Released	Context limit	Embedding d	Vocabulary $\lvert V\rvert$	Weights N
GPT-2 XL	2019	1024	1600	50257	$\approx 1.5 \cdot 10^9$
GPT-4 (original)	2023	8192	(not disclosed)	$\approx$ 100000	(not disclosed)
Qwen3-8B	2025	32768	4096	151936	$\approx 8.2 \cdot 10^9$
Qwen3.8-27B	2026	262144	5120	248320	$\approx 27 \cdot 10^9$



Autoregressive language modelling

- Training: Learn model parameters. Training loss measures how well we predict x_{T+1}.
- Generation: Given context, select x_{T+1} from $p(x_{T+1}|\text{context})$ (randomly or greedily).

In language models, the term *autoregressive* refers to the fact that we condition only on the past.

MARKOV RANDOM FIELDS

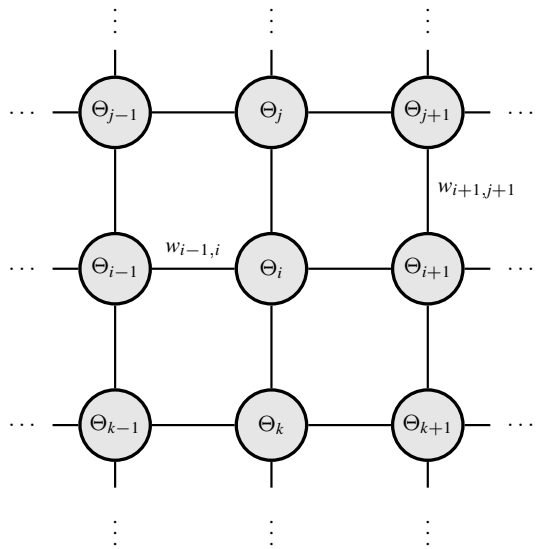
- We define a **neighbourhood graph**, which is a weighted, undirected graph:

vertex set

set of edge weights

$$\mathcal{N} = (V_{\mathcal{N}}, W_{\mathcal{N}})$$

- 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.

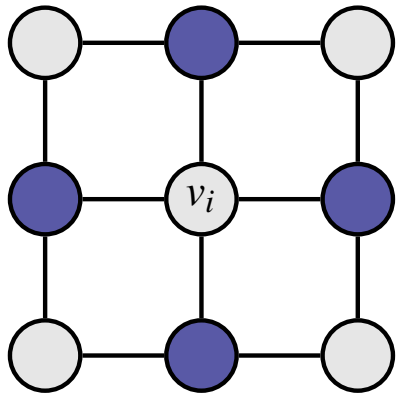


Neighbourhoods

- The **neighbourhood** of vertex v_i is the set

$$\partial(i) := \{j \,|\, w_{ij} \neq 0\} .$$

- The set $\{\Theta_j, j \in \partial(i)\}$ of random variables associated with the neighbourhood is the **Markov blanket** of Θ_i.



purple = $\partial(i)$

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

$$p(x) = \frac{1}{Z}\exp(-H(x)) \qquad \text{where} \qquad H : \mathcal{X} \to \mathbb{R} \ \text{ and } \ Z := \int e^{-H(x)}dx$$

  and Z is a normalization constant. In the discrete case, $Z = \sum e^{-H(x)}$.
- 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 neighbourhood graph with weights w_{ij}, and $\beta > 0$. The Markov random field

$$p(\theta_1, \ldots, \theta_n) := \frac{1}{Z(\beta)} \exp\Big(\beta \sum_{i,j \text{ with } i<j} w_{ij} \mathbb{I}\{\theta_i = \theta_j\}\Big) \qquad \text{for } \theta_i \in \{1, \ldots, K\}$$

is called a **Potts model**.

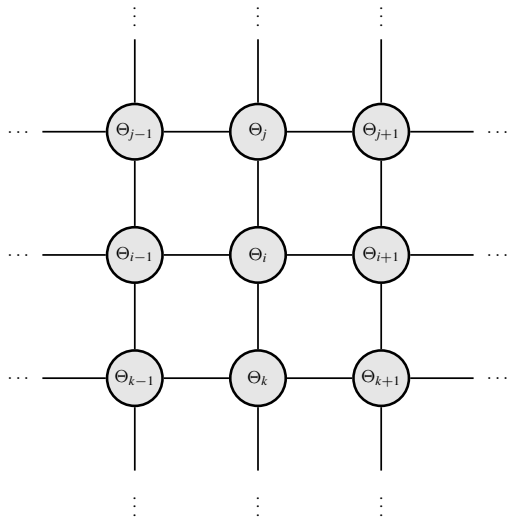
- Note the energy is additive over pairs.
- Positive weights encourage *smoothness*:

$w_{ij} > 0$ $\quad$ $\theta_i = \theta_j$ *increases* probability
$w_{ij} < 0$ $\quad$ $\theta_i = \theta_j$ *decreases* probability
$w_{ij} = 0$ $\quad$ no interaction between θ_i and θ_j

- If $\theta_i \in \{-1, +1\}$ and $w_{ij} \in \{0, 1\}$, we obtain

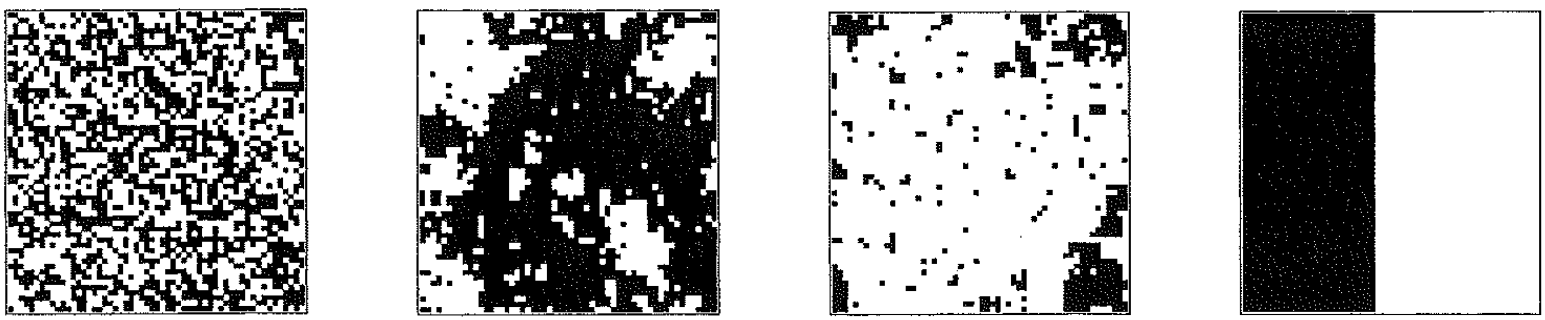
$$p(\theta_1, \ldots, \theta_n) = \frac{1}{Z(\beta)} \exp\Big(\sum_{(i,j) \text{ is an edge}} \beta \mathbb{I}\{\theta_i = \theta_j\} \Big)$$

- 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 $\beta \longrightarrow$

Colour coding: Black = -1, white = $+1$.

Illustration: Winkler, "Image analysis, random fields, and MCMC methods"

MRFs pose two main computational problems.

Problem 1: Sampling

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

Problem 2: Inference

MRF can be used as priors in Bayesian models (for example to encourage smoothness in image processing problems). In this case, we have to compute or approximate the posterior distribution.

Solution

- MRF distributions on grids are *not* analytically tractable.

  Exceptions: The 1-dimensional case (can be solved using forward-backward), and the Ising model on a 2-dimensional grid with no magnetic field (much harder).
- Both sampling and inference are based on Markov chain sampling algorithms.

Sampling Algorithms

A **sampling algorithm** takes a distribution P as input and outputs random values $X_1, \ldots, X_n$ 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 $X_1, \ldots, X_n$ 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 $\Pi := \mathcal{L}(\Theta|X_{1:n})$ cannot be computed analytically.

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

Posterior expectations

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

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

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:

$$p(x_{n+1}|x_{1:n}) := \int_{\mathcal{T}} p(x_{n+1}|\theta)\Pi(d\theta|X_{1:n} = x_{1:n}) \ .$$

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

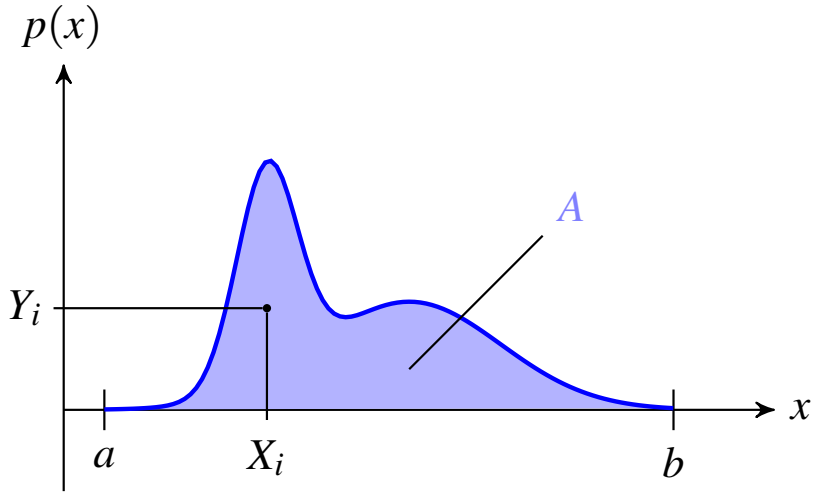
Computation from samples

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

$$p(x_{n+1}|x_{1:n}) = \mathbb{E}_\Pi[\,p(x_{n+1}|\Theta)] \approx \frac{1}{m}\sum_{i=1}^{m} p(x_{n+1}|\Theta_i)$$

The Key Idea

Consider a probability density p on the interval $[a, b]$.



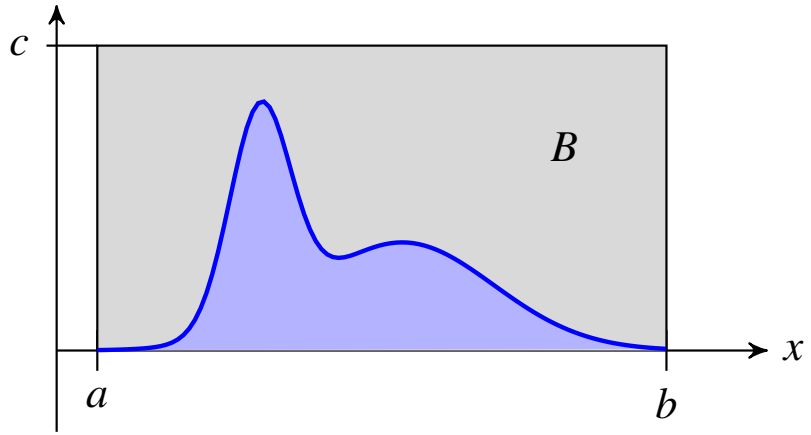
Suppose we can define a uniform distribution U_A on the blue area A under the curve.

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

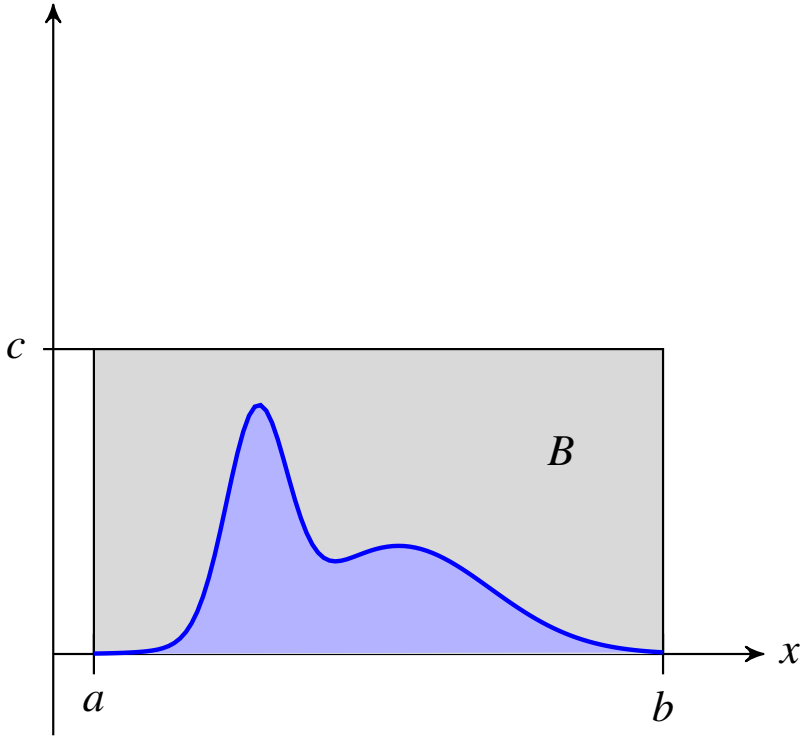
- Generate (X_i, Y_i) uniformly on B, by independently sampling

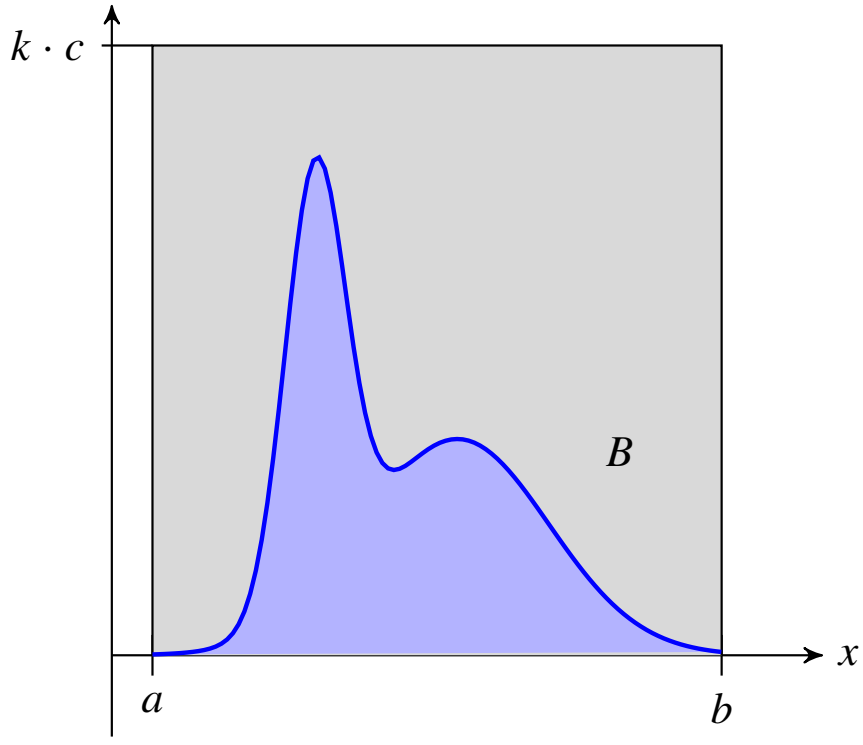
$$X_i \sim \text{Uniform}[a, b] \qquad \text{and} \qquad Y_i \sim \text{Uniform}[0, c] \ .$$

- 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.

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

$$\Pi(\theta|x_{1:n}) = \frac{\prod_{i=1}^n p(x_i|\theta)q(\theta)}{\tilde{Z}} \qquad \text{with} \qquad \tilde{Z} = \int_{\mathcal{T}} \prod_{i=1}^n p(x_i|\theta)q(\theta)d\theta \ .$$

Provided that we can compute the numerator, we can sample without computing the normalization integral $\tilde{Z}$.

Example 2: 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

$$Z(\beta) = \sum_{\theta_{1:n} \in \{-1,1\}^n} \exp\Big(\sum_{(i,j) \text{ is an edge}} \beta \mathbb{I}\{\theta_i = \theta_j\} \Big)$$

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

On the other hand, evaluating

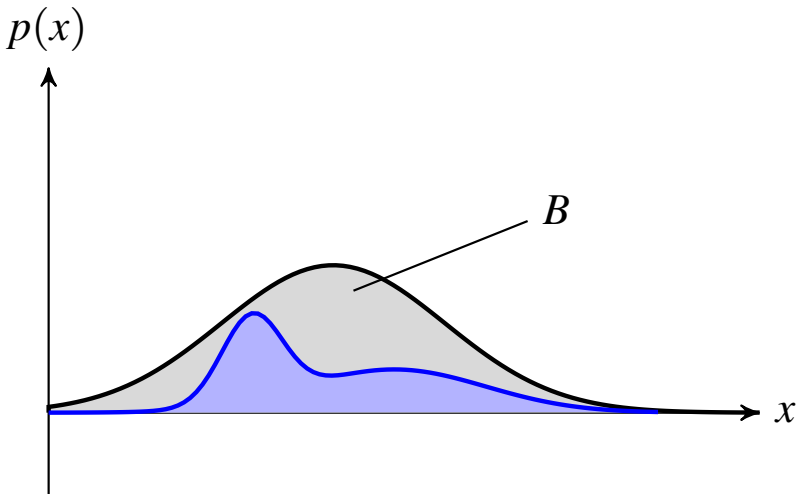
$$\tilde{p}(\theta_{1:n}) = \exp\Big(\sum_{(i,j) \text{ is an edge}} \beta \mathbb{I}\{\theta_i = \theta_j\} \Big)$$

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.

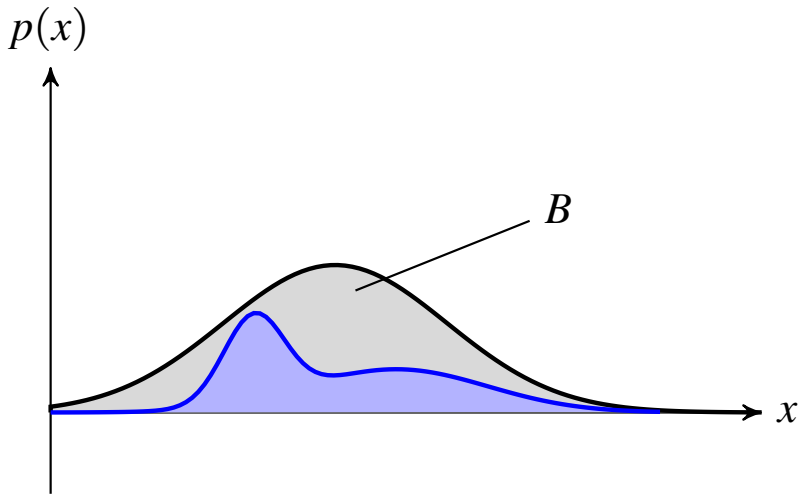


To sample B uniformly from the region under r, we apply the logic of the previous slide backwards:

- Sample $\mathbf{X}_i \sim r$.
- Sample $Y_i|\mathbf{X}_i \sim \text{Uniform}[0, r(\mathbf{X}_i)]$.

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 $\tilde{p}(\mathbf{x}) < r(\mathbf{x})$ everywhere.

Algorithm: Rejection sampling For $i = 1, 2, \ldots$:

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

The retained samples are distributed according to p.

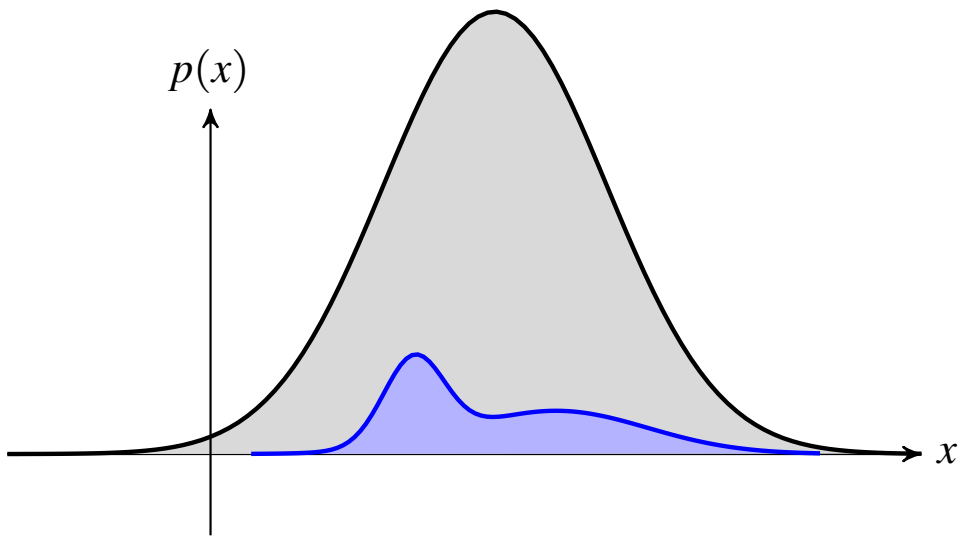
Independence

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 $X_1, X_2, \ldots$ are generated by rejection sampling, $\frac{1}{n}\sum_{i\leq n} f(X_i)$ is an unbiased estimate of $\mathbb{E}_p[f(X)]$.

Efficiency

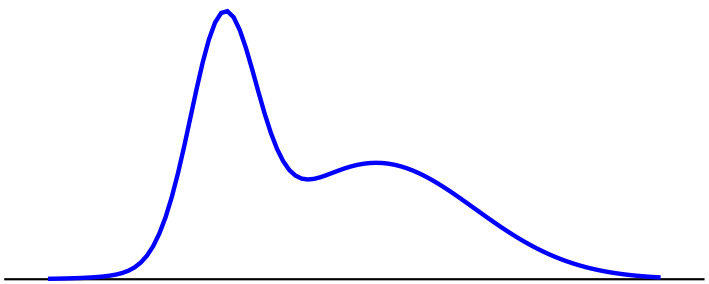
The fraction of accepted samples is the ratio $\frac{|A|}{|B|}$ of the areas under the curves $\tilde{p}$ and r.



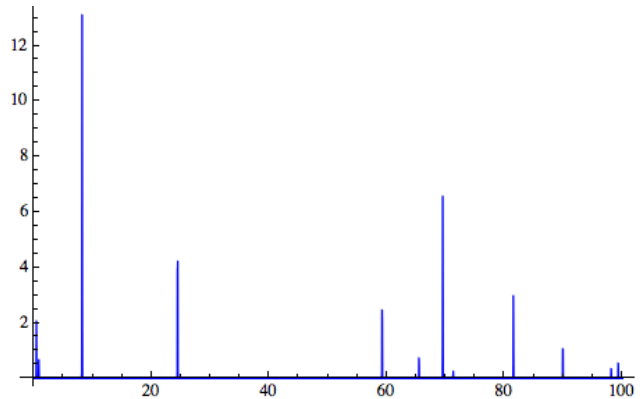
If r is not a reasonably close approximation of p, most proposals are rejected.

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.



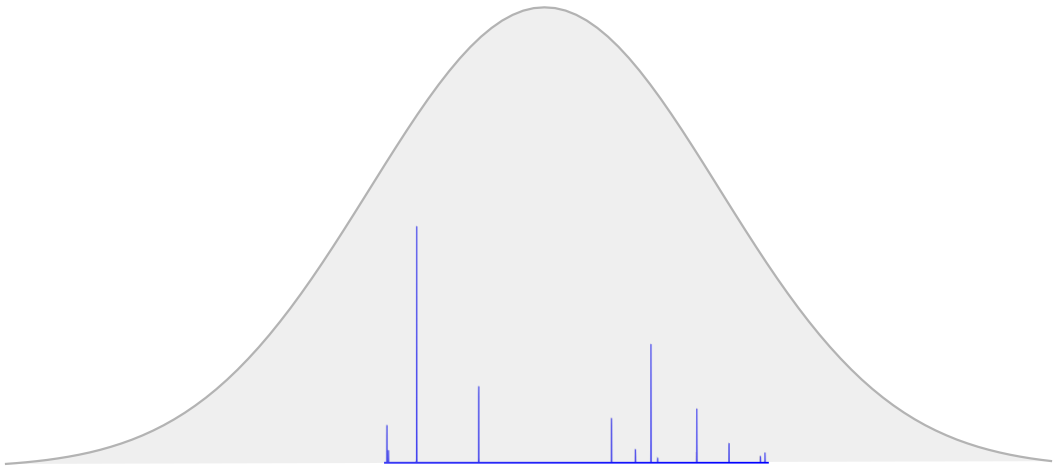
Textbook illustrations of target distributions tend to look like this.



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 grey 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.

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

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

(We require $q(x) > 0$ whenever $p(x) > 0$.)

Algorithm: Importance sampling

- Draw $X_1, X_2, \ldots$ i.i.d. from proposal q. Do not discard any samples.
- Approximate the expectation of f as

$$\mathbb{E}_p[f(X)] \;\approx\; \tfrac{1}{n}\textstyle\sum_{i\leq n} f(X_i)\frac{p(X_i)}{q(X_i)}$$

- 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 $X_1, \ldots, X_n \sim_{\text{iid}} q$:

$$\frac{Z_p}{Z_q} = \frac{\int \tilde{p}(x)dx}{Z_q} = \frac{\int \tilde{p}(x)\frac{q(x)}{q(x)}dx}{Z_q} = \int \tilde{p}(x)\frac{q(x)}{Z_q \cdot q(x)}dx = \mathbb{E}_q\left[\frac{\tilde{p}(X)}{\tilde{q}(X)}\right] \quad \approx \quad \frac{1}{n}\sum_{i\leq n}\frac{\tilde{p}(X_i)}{\tilde{q}(X_i)}$$

- The estimator of the expectation of f is then:

$$\mathbb{E}_p[f(X)] \;\approx\; \frac{1}{n}\sum_{i=1}^n f(X_i)\frac{p(X_i)}{q(X_i)} \;=\; \frac{1}{n}\sum_{i=1}^n f(X_i)\frac{Z_q}{Z_p}\frac{\tilde{p}(X_i)}{\tilde{q}(X_i)} \;\approx\; \sum_{i=1}^n \frac{f(X_i)\frac{\tilde{p}(X_i)}{\tilde{q}(X_i)}}{\sum_{j=1}^n \frac{\tilde{p}(X_j)}{\tilde{q}(X_j)}}$$

Algorithm: Importance sampling (p known up to scaling)

Draw $X_1, X_2, \ldots$ i.i.d. from proposal q, and approximate the expectation as

$$\mathbb{E}_p[f(X)] \;\approx\; \sum_{i=1}^n \frac{f(X_i)\frac{\tilde{p}(X_i)}{\tilde{q}(X_i)}}{\sum_{j=1}^n \frac{\tilde{p}(X_j)}{\tilde{q}(X_j)}}$$