Variational inference - Basics

Andrija Petrovic

August 7, 2025

Outline

The Goal: Posterior Inference

Bayes' Rule

The cornerstone of Bayesian machine learning is Bayes' rule, which tells us how to update our beliefs about parameters $\mathbf{z}$ after observing data $\mathbf{x}$:

$$p(\mathbf{z} \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})}{p(\mathbf{x})}$$

- $p(\mathbf{z} \mid \mathbf{x})$: The **Posterior**. What we believe about $\mathbf{z}$ after seeing data. This is what we want to compute.

The Goal: Posterior Inference

Bayes' Rule

The cornerstone of Bayesian machine learning is Bayes' rule, which tells us how to update our beliefs about parameters $\mathbf{z}$ after observing data $\mathbf{x}$:

$$p(\mathbf{z} \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \mathbf{z})p(\mathbf{z})}{p(\mathbf{x})}$$

- $p(\mathbf{z} \mid \mathbf{x})$: The **Posterior**. What we believe about $\mathbf{z}$ after seeing data. This is what we want to compute.
- $p(\mathbf{x} \mid \mathbf{z})$: The **Likelihood**. How we model the data generation process.

The Goal: Posterior Inference

Bayes' Rule

The cornerstone of Bayesian machine learning is Bayes' rule, which tells us how to update our beliefs about parameters $\mathbf{z}$ after observing data $\mathbf{x}$:

$$p(\mathbf{z} \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})}{p(\mathbf{x})}$$

- $p(\mathbf{z} \mid \mathbf{x})$: The **Posterior**. What we believe about $\mathbf{z}$ after seeing data. This is what we want to compute.
- $p(\mathbf{x} \mid \mathbf{z})$: The **Likelihood**. How we model the data generation process.
- $p(\mathbf{z})$: The **Prior**. Our beliefs about $\mathbf{z}$ before seeing any data.

The Goal: Posterior Inference

Bayes' Rule

The cornerstone of Bayesian machine learning is Bayes' rule, which tells us how to update our beliefs about parameters $\mathbf{z}$ after observing data $\mathbf{x}$:

$$p(\mathbf{z} \mid \mathbf{x}) = \frac{p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})}{p(\mathbf{x})}$$

- $p(\mathbf{z} \mid \mathbf{x})$: The **Posterior**. What we believe about $\mathbf{z}$ after seeing data. This is what we want to compute.
- $p(\mathbf{x} \mid \mathbf{z})$: The **Likelihood**. How we model the data generation process.
- $p(\mathbf{z})$: The **Prior**. Our beliefs about $\mathbf{z}$ before seeing any data.
- $p(\mathbf{x})$: The **Marginal Likelihood** or **Evidence**. The probability of the data, averaged over all possible parameters.

The Intractability of the Evidence

The denominator, the model evidence, is the main source of computational difficulty.

The Evidence Integral

To calculate the evidence, we must integrate (or marginalize) over all possible values of the latent variables or parameters $\mathbf{z}$:

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

- This is a high-dimensional integral for most interesting models (e.g., deep neural networks, topic models).

The Intractability of the Evidence

The denominator, the model evidence, is the main source of computational difficulty.

The Evidence Integral

To calculate the evidence, we must integrate (or marginalize) over all possible values of the latent variables or parameters $\mathbf{z}$:

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

- This is a high-dimensional integral for most interesting models (e.g., deep neural networks, topic models).
- The integral has no analytical (closed-form) solution except for very simple cases (e.g., conjugate priors).
- Beta prior $\rightarrow$ Binomial/Bernoulli likelihood $\rightarrow$ posterior is also Beta.
- Normal prior $\rightarrow$ Normal likelihood (known variance) $\rightarrow$ posterior is Normal.

The Intractability of the Evidence

The denominator, the model evidence, is the main source of computational difficulty.

The Evidence Integral

To calculate the evidence, we must integrate (or marginalize) over all possible values of the latent variables or parameters $\mathbf{z}$:

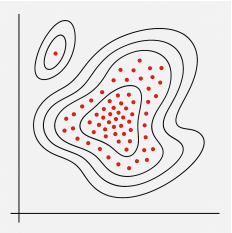
$$p(\mathbf{x}) = \int p(\mathbf{x}, \mathbf{z})\, d\mathbf{z} = \int p(\mathbf{x} \mid \mathbf{z}) p(\mathbf{z})\, d\mathbf{z}$$

- This is a high-dimensional integral for most interesting models (e.g., deep neural networks, topic models).
- The integral has no analytical (closed-form) solution except for very simple cases (e.g., conjugate priors).
- Beta prior $\rightarrow$ Binomial/Bernoulli likelihood $\rightarrow$ posterior is also Beta.
- Normal prior $\rightarrow$ Normal likelihood (known variance) $\rightarrow$ posterior is Normal.
- Without $p(\mathbf{x})$, we can't compute the true posterior $p(\mathbf{z} \mid \mathbf{x})$. We only know it up to a constant of proportionality: $p(\mathbf{z} \mid \mathbf{x}) \propto p(\mathbf{x}, \mathbf{z})$.

Two Paths Forward

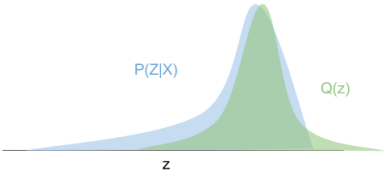
Since exact inference is intractable, we resort to approximation methods.

Stochastic Approximation



- Generate samples that approximate the posterior distribution.
- **Key Method:** Markov Chain Monte Carlo (MCMC).
- Asymptotically exact.

Deterministic Approximation



- Pose inference as an optimization problem.
- **Key Method:** Variational Inference (VI).
- Find the "closest" distribution to the true posterior from a simpler family.

Markov Chain Monte Carlo (MCMC)

The Core Idea

Construct a Markov chain whose states are values of the latent variables $\mathbf{z}$, and whose stationary distribution is the target posterior $p(\mathbf{z} \mid \mathbf{x})$.

- We don't need to know the normalizing constant $p(\mathbf{x})$ to run MCMC algorithms like Metropolis-Hastings or Gibbs Sampling. We only need to evaluate the unnormalized posterior $p(\mathbf{x}, \mathbf{z})$.

Markov Chain Monte Carlo (MCMC)

The Core Idea

Construct a Markov chain whose states are values of the latent variables $\mathbf{z}$, and whose stationary distribution is the target posterior $p(\mathbf{z} \mid \mathbf{x})$.

- We don't need to know the normalizing constant $p(\mathbf{x})$ to run MCMC algorithms like Metropolis-Hastings or Gibbs Sampling. We only need to evaluate the unnormalized posterior $p(\mathbf{x}, \mathbf{z})$.
- **How it works (high level):**
  1. Start at some initial state $\mathbf{z}^{(0)}$.
  2. Iteratively propose a new state $\mathbf{z}'$ from a proposal distribution $g(\mathbf{z}' \mid \mathbf{z}^{(t)})$.
  3. Accept or reject the new state based on a rule that ensures the chain converges to the target posterior. For Metropolis-Hastings, the acceptance probability is:

$$\alpha = \min\left(1, \frac{p(\mathbf{x}, \mathbf{z}')g(\mathbf{z}^{(t)} \mid \mathbf{z}')}{p(\mathbf{x}, \mathbf{z}^{(t)})g(\mathbf{z}' \mid \mathbf{z}^{(t)})}\right)$$

MCMC: Pros and Cons

Pros

- **Asymptotically Exact:** Given infinite time, the samples will be drawn from the true posterior.
- **Generality:** Can be applied to a wide range of models.
- **High-Quality Samples:** Often produces a very accurate representation of the posterior.

Cons

- **Slow:** Can take a very long time to "burn-in" (reach the stationary distribution) and to collect enough independent samples.
- **Hard to Scale:** Often difficult to parallelize and computationally expensive for large datasets.
- **Convergence Diagnostics:** It's hard to know for sure if the chain has converged.

MCMC's slowness motivates the need for a faster alternative like VI.

Derivation: The Goal is Stationarity

What properties must our chain have?

We want the distribution of our samples $\mathbf{z}^{(t)}$ to eventually stop changing and converge to our target posterior, $\pi(\mathbf{z}) \equiv p(\mathbf{z} \mid \mathbf{x})$. This is the **stationary distribution**.

Let $T(\mathbf{z}' \mid \mathbf{z})$ be the transition kernel, the probability of moving from state $\mathbf{z}$ to $\mathbf{z}'$.

Derivation: The Goal is Stationarity

What properties must our chain have?

We want the distribution of our samples $\mathbf{z}^{(t)}$ to eventually stop changing and converge to our target posterior, $\pi(\mathbf{z}) \equiv p(\mathbf{z} \mid \mathbf{x})$. This is the **stationary distribution**.

Let $T(\mathbf{z}' \mid \mathbf{z})$ be the transition kernel, the probability of moving from state $\mathbf{z}$ to $\mathbf{z}'$.

The Stationarity Condition

A distribution $\pi(\mathbf{z})$ is stationary for a transition kernel T if:

$$\pi(\mathbf{z}') = \int \pi(\mathbf{z})\, T(\mathbf{z}' \mid \mathbf{z}) d\mathbf{z}$$

In words: If the states are already distributed according to π, one step - unchanged.

Derivation: The Goal is Stationarity

What properties must our chain have?

We want the distribution of our samples $\mathbf{z}^{(t)}$ to eventually stop changing and converge to our target posterior, $\pi(\mathbf{z}) \equiv p(\mathbf{z} \mid \mathbf{x})$. This is the **stationary distribution**.

Let $T(\mathbf{z}' \mid \mathbf{z})$ be the transition kernel, the probability of moving from state $\mathbf{z}$ to $\mathbf{z}'$.

The Stationarity Condition

A distribution $\pi(\mathbf{z})$ is stationary for a transition kernel T if:

$$\pi(\mathbf{z}') = \int \pi(\mathbf{z})\, T(\mathbf{z}' \mid \mathbf{z}) d\mathbf{z}$$

In words: If the states are already distributed according to π, one step - unchanged.

- This condition ensures that once our chain reaches the target posterior, it will stay there.
- However, enforcing this condition directly is difficult.

Derivation: A Sufficient Condition – Detailed Balance

A simpler, stronger condition that guarantees stationarity is **detailed balance**.

The Detailed Balance Equation

The transition kernel T satisfies detailed balance with respect to π if:

$$\pi(\mathbf{z})\, T(\mathbf{z}' \mid \mathbf{z}) = \pi(\mathbf{z}')\, T(\mathbf{z} \mid \mathbf{z}')$$

In words: When the chain is in its stationary state, the rate of transitions from $\mathbf{z}$ *to* $\mathbf{z}'$ *is equal to the rate of transitions from* $\mathbf{z}'$ *to* $\mathbf{z}$.

Derivation: A Sufficient Condition – Detailed Balance

Proof: Detailed Balance $\implies$ Stationarity

We can show this by integrating the detailed balance equation over $\mathbf{z}$:

$$\begin{aligned}\int \pi(\mathbf{z})\, T(\mathbf{z}' \mid \mathbf{z}) d\mathbf{z} &= \int \pi(\mathbf{z}')\, T(\mathbf{z} \mid \mathbf{z}') d\mathbf{z} \\ &= \pi(\mathbf{z}') \int T(\mathbf{z} \mid \mathbf{z}') d\mathbf{z} \\ &= \pi(\mathbf{z}') \cdot 1 \\ &= \pi(\mathbf{z}') \qquad \square\end{aligned}$$

Since the result is the stationarity condition, our goal now is to construct a transition kernel T that satisfies detailed balance.

Derivation: Constructing the M-H Algorithm

Our transition $T(\mathbf{z}' \mid \mathbf{z})$ involves two steps: proposing a new state $\mathbf{z}'$ and then accepting it.

- **Proposal:** Use a proposal distribution $g(\mathbf{z}' \mid \mathbf{z})$.
- **Acceptance:** Accept the proposal with probability $A(\mathbf{z}', \mathbf{z})$.

So, for $\mathbf{z}' \neq \mathbf{z}$, the full transition is $T(\mathbf{z}' \mid \mathbf{z}) = g(\mathbf{z}' \mid \mathbf{z})A(\mathbf{z}', \mathbf{z})$.

Derivation: Constructing the M-H Algorithm

Our transition $T(\mathbf{z}' \mid \mathbf{z})$ involves two steps: proposing a new state $\mathbf{z}'$ and then accepting it.

- **Proposal:** Use a proposal distribution $g(\mathbf{z}' \mid \mathbf{z})$.
- **Acceptance:** Accept the proposal with probability $A(\mathbf{z}', \mathbf{z})$.

So, for $\mathbf{z}' \neq \mathbf{z}$, the full transition is $T(\mathbf{z}' \mid \mathbf{z}) = g(\mathbf{z}' \mid \mathbf{z})A(\mathbf{z}', \mathbf{z})$.

Let's plug this into the detailed balance equation:

$$\pi(\mathbf{z})g(\mathbf{z}' \mid \mathbf{z})A(\mathbf{z}', \mathbf{z}) = \pi(\mathbf{z}')g(\mathbf{z} \mid \mathbf{z}')A(\mathbf{z}, \mathbf{z}')$$

Derivation: Constructing the M-H Algorithm

Our transition $T(\mathbf{z}' \mid \mathbf{z})$ involves two steps: proposing a new state $\mathbf{z}'$ and then accepting it.

- **Proposal:** Use a proposal distribution $g(\mathbf{z}' \mid \mathbf{z})$.
- **Acceptance:** Accept the proposal with probability $A(\mathbf{z}', \mathbf{z})$.

So, for $\mathbf{z}' \neq \mathbf{z}$, the full transition is $T(\mathbf{z}' \mid \mathbf{z}) = g(\mathbf{z}' \mid \mathbf{z})A(\mathbf{z}', \mathbf{z})$.

Let's plug this into the detailed balance equation:

$$\pi(\mathbf{z})g(\mathbf{z}' \mid \mathbf{z})A(\mathbf{z}', \mathbf{z}) = \pi(\mathbf{z}')g(\mathbf{z} \mid \mathbf{z}')A(\mathbf{z}, \mathbf{z}')$$

Rearranging to find a constraint on the acceptance probabilities:

$$\frac{A(\mathbf{z}', \mathbf{z})}{A(\mathbf{z}, \mathbf{z}')} = \frac{\pi(\mathbf{z}')g(\mathbf{z} \mid \mathbf{z}')}{\pi(\mathbf{z})g(\mathbf{z}' \mid \mathbf{z})}$$

Derivation: Constructing the M-H Algorithm

Our transition $T(\mathbf{z}' \mid \mathbf{z})$ involves two steps: proposing a new state $\mathbf{z}'$ and then accepting it.

- **Proposal:** Use a proposal distribution $g(\mathbf{z}' \mid \mathbf{z})$.
- **Acceptance:** Accept the proposal with probability $A(\mathbf{z}', \mathbf{z})$.

So, for $\mathbf{z}' \neq \mathbf{z}$, the full transition is $T(\mathbf{z}' \mid \mathbf{z}) = g(\mathbf{z}' \mid \mathbf{z})A(\mathbf{z}', \mathbf{z})$.

Let's plug this into the detailed balance equation:

$$\pi(\mathbf{z})g(\mathbf{z}' \mid \mathbf{z})A(\mathbf{z}', \mathbf{z}) = \pi(\mathbf{z}')g(\mathbf{z} \mid \mathbf{z}')A(\mathbf{z}, \mathbf{z}')$$

Rearranging to find a constraint on the acceptance probabilities:

$$\frac{A(\mathbf{z}', \mathbf{z})}{A(\mathbf{z}, \mathbf{z}')} = \frac{\pi(\mathbf{z}')g(\mathbf{z} \mid \mathbf{z}')}{\pi(\mathbf{z})g(\mathbf{z}' \mid \mathbf{z})}$$

The **Metropolis choice** for A satisfies this ratio while maximizing the acceptance rate:

$$A(\mathbf{z}', \mathbf{z}) = \min\left(1, \frac{\pi(\mathbf{z}')g(\mathbf{z} \mid \mathbf{z}')}{\pi(\mathbf{z})g(\mathbf{z}' \mid \mathbf{z})}\right)$$

This is our acceptance probability α.

Derivation: Constructing the M-H Algorithm

The Final Step: Using the Unnormalized Posterior

Since $\pi(\mathbf{z}) = p(\mathbf{z} \mid \mathbf{x}) = \frac{p(\mathbf{x},\mathbf{z})}{p(\mathbf{x})}$, the normalization constant $p(\mathbf{x})$ cancels out in the ratio!

$$\alpha = \min\left(1, \frac{p(\mathbf{x}, \mathbf{z}')g(\mathbf{z} \mid \mathbf{z}')}{p(\mathbf{x}, \mathbf{z})g(\mathbf{z}' \mid \mathbf{z})}\right)$$

This is exactly the Metropolis-Hastings acceptance rule.

The Core Idea of Variational Inference

From Integration to Optimization

VI reframes the problem of computing the posterior $p(\mathbf{z} \mid \mathbf{x})$ as an optimization problem.

1. **Define a family of simple distributions**, $q(\mathbf{z}; \boldsymbol{\lambda})$, parameterized by $\boldsymbol{\lambda}$. This is called the *variational family*.
   - Example: A family of multivariate Gaussians with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. Here, $\boldsymbol{\lambda} = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$.

The Core Idea of Variational Inference

From Integration to Optimization

VI reframes the problem of computing the posterior $p(\mathbf{z} \mid \mathbf{x})$ as an optimization problem.

1. **Define a family of simple distributions**, $q(\mathbf{z}; \boldsymbol{\lambda})$, parameterized by $\boldsymbol{\lambda}$. This is called the *variational family*.
   - Example: A family of multivariate Gaussians with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$. Here, $\boldsymbol{\lambda} = \{\boldsymbol{\mu}, \boldsymbol{\Sigma}\}$.
2. **Find the member of this family** that is "closest" to the true posterior $p(\mathbf{z} \mid \mathbf{x})$.
   - We find the optimal parameters $\boldsymbol{\lambda}^*$ that minimize some notion of distance between $q(\mathbf{z}; \boldsymbol{\lambda})$ and $p(\mathbf{z} \mid \mathbf{x})$.

The resulting $q(\mathbf{z}; \boldsymbol{\lambda}^*)$ is our approximation to the posterior.

Measuring "Closeness": The KL Divergence

We measure the dissimilarity between two distributions using the **Kullback-Leibler (KL) divergence**.

KL Divergence

The KL divergence from q to p is defined as:

$$\mathrm{KL}(q(\mathbf{z}) \parallel p(\mathbf{z} \mid \mathbf{x})) = \int q(\mathbf{z}) \log \frac{q(\mathbf{z})}{p(\mathbf{z} \mid \mathbf{x})} \, d\mathbf{z} = \mathbb{E}_{q(\mathbf{z})} \left[\log \frac{q(\mathbf{z})}{p(\mathbf{z} \mid \mathbf{x})} \right]$$

- $\mathrm{KL}(q \parallel p) \geq 0$.
- $\mathrm{KL}(q \parallel p) = 0$ if and only if $q = p$.
- It is not symmetric: $\mathrm{KL}(q \parallel p) \neq \mathrm{KL}(p \parallel q)$.

Measuring "Closeness": The KL Divergence

We measure the dissimilarity between two distributions using the **Kullback-Leibler (KL) divergence**.

KL Divergence

The KL divergence from q to p is defined as:

$$\mathrm{KL}(q(\mathbf{z}) \parallel p(\mathbf{z} \mid \mathbf{x})) = \int q(\mathbf{z}) \log \frac{q(\mathbf{z})}{p(\mathbf{z} \mid \mathbf{x})} \, d\mathbf{z} = \mathbb{E}_{q(\mathbf{z})} \left[\log \frac{q(\mathbf{z})}{p(\mathbf{z} \mid \mathbf{x})} \right]$$

- $\mathrm{KL}(q \parallel p) \geq 0$.
- $\mathrm{KL}(q \parallel p) = 0$ if and only if $q = p$.
- It is not symmetric: $\mathrm{KL}(q \parallel p) \neq \mathrm{KL}(p \parallel q)$.

The Goal: Find the variational parameters $\boldsymbol{\lambda}$ that minimize this KL divergence.

$$\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda}} \mathrm{KL}(q(\mathbf{z}; \boldsymbol{\lambda}) \parallel p(\mathbf{z} \mid \mathbf{x}))$$

The Problem with Direct KL Minimization

Let's expand the KL divergence definition:

$$\begin{aligned}
\mathrm{KL}(q \parallel p) &= \mathbb{E}_q \left[\log q(\mathbf{z}; \boldsymbol{\lambda}) - \log p(\mathbf{z} \mid \mathbf{x})\right] \\
&= \mathbb{E}_q \left[\log q(\mathbf{z}; \boldsymbol{\lambda})\right] - \mathbb{E}_q \left[\log \frac{p(\mathbf{x}, \mathbf{z})}{p(\mathbf{x})}\right] \\
&= \mathbb{E}_q \left[\log q(\mathbf{z}; \boldsymbol{\lambda})\right] - \mathbb{E}_q \left[\log p(\mathbf{x}, \mathbf{z})\right] + \mathbb{E}_q \left[\log p(\mathbf{x})\right] \\
&= \mathbb{E}_q \left[\log q(\mathbf{z}; \boldsymbol{\lambda})\right] - \mathbb{E}_q \left[\log p(\mathbf{x}, \mathbf{z})\right] + \log p(\mathbf{x})
\end{aligned}$$

The last term, $\log p(\mathbf{x})$, is the log model evidence!

The Problem with Direct KL Minimization

Let's expand the KL divergence definition:

$$\begin{aligned}
\mathrm{KL}(q \parallel p) &= \mathbb{E}_q \left[\log q(\mathbf{z}; \boldsymbol{\lambda}) - \log p(\mathbf{z} \mid \mathbf{x}) \right] \\
&= \mathbb{E}_q \left[\log q(\mathbf{z}; \boldsymbol{\lambda}) \right] - \mathbb{E}_q \left[\log \frac{p(\mathbf{x}, \mathbf{z})}{p(\mathbf{x})} \right] \\
&= \mathbb{E}_q \left[\log q(\mathbf{z}; \boldsymbol{\lambda}) \right] - \mathbb{E}_q \left[\log p(\mathbf{x}, \mathbf{z}) \right] + \mathbb{E}_q \left[\log p(\mathbf{x}) \right] \\
&= \mathbb{E}_q \left[\log q(\mathbf{z}; \boldsymbol{\lambda}) \right] - \mathbb{E}_q \left[\log p(\mathbf{x}, \mathbf{z}) \right] + \log p(\mathbf{x})
\end{aligned}$$

The last term, $\log p(\mathbf{x})$, is the log model evidence!

The Catch-22

To minimize the KL divergence, we need to compute the very quantity, $p(\mathbf{x})$, that made the problem intractable in the first place. We need another way.

The Magic: Evidence Lower Bound (ELBO)

Let's rearrange the previous equation:

$$\log p(\mathbf{x}) = \mathrm{KL}(q(\mathbf{z};\boldsymbol{\lambda}) \parallel p(\mathbf{z} \mid \mathbf{x})) + (\mathbb{E}_q[\log p(\mathbf{x},\mathbf{z})] - \mathbb{E}_q[\log q(\mathbf{z};\boldsymbol{\lambda})])$$

The Magic: Evidence Lower Bound (ELBO)

Let's rearrange the previous equation:

$$\log p(\mathbf{x}) = \mathrm{KL}(q(\mathbf{z};\boldsymbol{\lambda}) \parallel p(\mathbf{z} \mid \mathbf{x})) + (\mathbb{E}_q[\log p(\mathbf{x},\mathbf{z})] - \mathbb{E}_q[\log q(\mathbf{z};\boldsymbol{\lambda})])$$

Since $\mathrm{KL}(\cdot) \geq 0$, we have:

$$\log p(\mathbf{x}) \geq \underbrace{\mathbb{E}_{q(\mathbf{z};\boldsymbol{\lambda})}[\log p(\mathbf{x},\mathbf{z}) - \log q(\mathbf{z};\boldsymbol{\lambda})]}_{\text{This is the ELBO}}$$

The Magic: Evidence Lower Bound (ELBO)

Let's rearrange the previous equation:

$$\log p(\mathbf{x}) = \mathrm{KL}(q(\mathbf{z};\boldsymbol{\lambda}) \parallel p(\mathbf{z} \mid \mathbf{x})) + (\mathbb{E}_q[\log p(\mathbf{x},\mathbf{z})] - \mathbb{E}_q[\log q(\mathbf{z};\boldsymbol{\lambda})])$$

Since $\mathrm{KL}(\cdot) \geq 0$, we have:

$$\log p(\mathbf{x}) \geq \underbrace{\mathbb{E}_{q(\mathbf{z};\boldsymbol{\lambda})}[\log p(\mathbf{x},\mathbf{z}) - \log q(\mathbf{z};\boldsymbol{\lambda})]}_{\text{This is the ELBO}}$$

This quantity is called the **Evidence Lower Bound (ELBO)**, often denoted $\mathcal{L}(\boldsymbol{\lambda})$.

The Key Insight

$$\log p(\mathbf{x}) = \mathcal{L}(\boldsymbol{\lambda}) + \mathrm{KL}(q(\mathbf{z};\boldsymbol{\lambda}) \parallel p(\mathbf{z} \mid \mathbf{x}))$$

Since the log evidence $\log p(\mathbf{x})$ is a constant with respect to our choice of q, **maximizing the ELBO is equivalent to minimizing the KL divergence**.

The Magic: Evidence Lower Bound (ELBO)

Let's rearrange the previous equation:

$$\log p(\mathbf{x}) = \mathrm{KL}(q(\mathbf{z};\boldsymbol{\lambda}) \parallel p(\mathbf{z}\,|\,\mathbf{x})) + (\mathbb{E}_q\left[\log p(\mathbf{x},\mathbf{z})\right] - \mathbb{E}_q\left[\log q(\mathbf{z};\boldsymbol{\lambda})\right])$$

Since $\mathrm{KL}(\cdot) \geq 0$, we have:

$$\log p(\mathbf{x}) \geq \underbrace{\mathbb{E}_{q(\mathbf{z};\boldsymbol{\lambda})}\left[\log p(\mathbf{x},\mathbf{z}) - \log q(\mathbf{z};\boldsymbol{\lambda})\right]}_{\text{This is the ELBO}}$$

This quantity is called the **Evidence Lower Bound (ELBO)**, often denoted $\mathcal{L}(\boldsymbol{\lambda})$.

The Key Insight

$$\log p(\mathbf{x}) = \mathcal{L}(\boldsymbol{\lambda}) + \mathrm{KL}(q(\mathbf{z};\boldsymbol{\lambda}) \parallel p(\mathbf{z}\,|\,\mathbf{x}))$$

Since the log evidence $\log p(\mathbf{x})$ is a constant with respect to our choice of q, **maximizing the ELBO is equivalent to minimizing the KL divergence**.

Crucially, the ELBO **does not** depend on $p(\mathbf{x})$! We can compute it.

Deconstructing the ELBO

The ELBO can be written in two common, insightful forms.

Form 1: Expected Log Likelihood + Entropy

$$\mathcal{L}(\boldsymbol{\lambda}) = \underbrace{\mathbb{E}_q[\log p(\mathbf{x}, \mathbf{z})]}_{\text{Energy}} - \underbrace{\mathbb{E}_q[\log q(\mathbf{z}; \boldsymbol{\lambda})]}_{\text{Negative Entropy}}$$

- The first term encourages q to place its mass on configurations of $\mathbf{z}$ that explain the data well (high $\log p(\mathbf{x}, \mathbf{z})$).
- The second term, the entropy of q, encourages q to be as spread out as possible, preventing it from collapsing to a single point.

Deconstructing the ELBO

Form 2: Reconstruction + Regularization (common in VAEs)

$$\mathcal{L}(\boldsymbol{\lambda}) = \underbrace{\mathbb{E}_q[\log p(\mathbf{x} \mid \mathbf{z})]}_{\text{Reconstruction Term}} - \underbrace{\mathrm{KL}(q(\mathbf{z}; \boldsymbol{\lambda}) \parallel p(\mathbf{z}))}_{\text{Regularizer}}$$

- The first term measures how well we can reconstruct the data $\mathbf{x}$ from latent codes $\mathbf{z}$ sampled from our approximation q.
- The second term is a regularizer that forces our approximate posterior q to stay close to the prior $p(\mathbf{z})$.

The VI Recipe

1. **Define a model:** Specify the prior $p(\mathbf{z})$ and the likelihood $p(\mathbf{x} \mid \mathbf{z})$. This defines the joint $p(\mathbf{x}, \mathbf{z})$.

The VI Recipe

1. **Define a model:** Specify the prior $p(\mathbf{z})$ and the likelihood $p(\mathbf{x} \mid \mathbf{z})$. This defines the joint $p(\mathbf{x}, \mathbf{z})$.
2. **Define a variational family:** Choose a tractable family of distributions $q(\mathbf{z}; \boldsymbol{\lambda})$.

The VI Recipe

1. **Define a model:** Specify the prior $p(\mathbf{z})$ and the likelihood $p(\mathbf{x} \mid \mathbf{z})$. This defines the joint $p(\mathbf{x}, \mathbf{z})$.
2. **Define a variational family:** Choose a tractable family of distributions $q(\mathbf{z}; \boldsymbol{\lambda})$.
3. **Write down the ELBO:** Using your model and variational family, write the expression for $\mathcal{L}(\boldsymbol{\lambda})$.

The VI Recipe

1. **Define a model:** Specify the prior $p(\mathbf{z})$ and the likelihood $p(\mathbf{x} \mid \mathbf{z})$. This defines the joint $p(\mathbf{x}, \mathbf{z})$.
2. **Define a variational family:** Choose a tractable family of distributions $q(\mathbf{z}; \boldsymbol{\lambda})$.
3. **Write down the ELBO:** Using your model and variational family, write the expression for $\mathcal{L}(\boldsymbol{\lambda})$.
4. **Optimize:** Find the parameters $\boldsymbol{\lambda}^*$ that maximize the ELBO.

$$\boldsymbol{\lambda}^* = \arg\max_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda})$$

The VI Recipe

1. **Define a model:** Specify the prior $p(\mathbf{z})$ and the likelihood $p(\mathbf{x} \mid \mathbf{z})$. This defines the joint $p(\mathbf{x}, \mathbf{z})$.
2. **Define a variational family:** Choose a tractable family of distributions $q(\mathbf{z}; \boldsymbol{\lambda})$.
3. **Write down the ELBO:** Using your model and variational family, write the expression for $\mathcal{L}(\boldsymbol{\lambda})$.
4. **Optimize:** Find the parameters $\boldsymbol{\lambda}^*$ that maximize the ELBO.

$$\boldsymbol{\lambda}^* = \arg\max_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda})$$

5. **Use the result:** Use $q(\mathbf{z}; \boldsymbol{\lambda}^*)$ as your approximation to the posterior for downstream tasks (prediction, analysis, etc.).

Choosing the Variational Family q

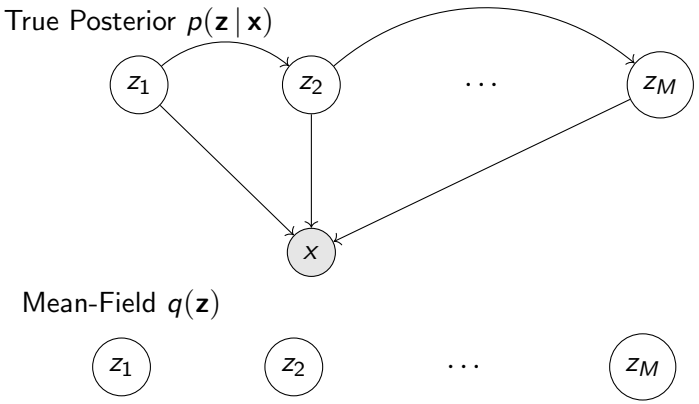
The choice of $q(\mathbf{z}; \boldsymbol{\lambda})$ determines the trade-off between the quality of the approximation and the difficulty of the optimization.
A common and powerful choice is the **mean-field variational family**.

The Mean-Field Assumption

We assume the latent variables are mutually independent in the variational distribution. The joint $q(\mathbf{z})$ fully factorizes over its components z_j:

$$q(\mathbf{z}; \boldsymbol{\lambda}) = \prod_{j=1}^{M} q_j(z_j; \boldsymbol{\lambda}_j)$$

Choosing the Variational Family q



This assumption simplifies the ELBO optimization significantly, but it cannot capture correlations in the true posterior.

Coordinate Ascent Variational Inference (CAVI)

With the mean-field assumption, we can optimize the ELBO using an iterative algorithm called **Coordinate Ascent Variational Inference (CAVI)**.
We optimize the ELBO with respect to each factor q_j one at a time, holding the others fixed.

The CAVI Update Rule

The optimal solution for a single factor $q_j(z_j)$ is given by:

$$q_j^*(z_j) \propto \exp\left(\mathbb{E}_{i \neq j}\left[\log p(\mathbf{x}, \mathbf{z})\right]\right)$$

where the expectation is taken with respect to all other factors $q_i(z_i)$ for $i \neq j$.

The algorithm iterates through $j = 1, \ldots, M$, updating each q_j until the ELBO converges.

Derivation of the CAVI Update (Sketch)

Let's find the optimal q_j. We expand the ELBO and isolate terms involving q_j.

$$\begin{aligned}\mathcal{L}(q_j) &= \mathbb{E}_q[\log p(\mathbf{x}, \mathbf{z})] - \mathbb{E}_q[\log q(\mathbf{z})] \\ &= \int q_j(z_j) \left(\int \prod_{i \neq j} q_i(z_i) \log p(\mathbf{x}, \mathbf{z}) \, d\mathbf{z}_{-j} \right) dz_j - \int q_j(z_j) \log q_j(z_j) dz_j + \text{const} \\ &= \int q_j(z_j) \, \mathbb{E}_{i \neq j}[\log p(\mathbf{x}, \mathbf{z})] dz_j + H(q_j) + \text{const}\end{aligned}$$

This is the negative KL divergence between $q_j(z_j)$ and another distribution:

$$\mathcal{L}(q_j) = -\,\mathrm{KL}(q_j(z_j) \parallel \tilde{p}(z_j)) + \text{const}$$

where $\log \tilde{p}(z_j) = \mathbb{E}_{i \neq j}[\log p(\mathbf{x}, \mathbf{z})] + \text{const}$.
To maximize $\mathcal{L}$ (i.e., minimize the KL), we must set $q_j(z_j) = \tilde{p}(z_j)$.

$$\implies q_j^*(z_j) \propto \exp\left(\mathbb{E}_{i \neq j}\left[\log p(\mathbf{x}, \mathbf{z})\right]\right)$$

Mean-Field VI: Pros and Cons

Pros

- **Fast and Scalable:** CAVI often converges much faster than MCMC.
- **Deterministic:** No randomness in the optimization process. Convergence is easy to assess (just monitor the ELBO).
- **Simple Updates:** If the model has conjugate priors, the expectation in the update rule is often analytic.

Cons

- **Restrictive Assumption:** The factorization assumption is often wrong. It cannot capture posterior correlations.
- **Underestimates Variance:** Tends to find a q that is more compact than the true posterior.
- **Model-Specific Derivations:** The CAVI update equations must be re-derived for every new model.

The Need for Generality

The biggest drawback of CAVI is the need for model-specific derivations. This is a major bottleneck for research and practice.

Question: Can we develop a "black box" VI algorithm that only requires us to be able to *evaluate* the log-joint probability $\log p(\mathbf{x}, \mathbf{z})$ and its gradients?

The Need for Generality

The biggest drawback of CAVI is the need for model-specific derivations. This is a major bottleneck for research and practice.

Question: Can we develop a "black box" VI algorithm that only requires us to be able to *evaluate* the log-joint probability $\log p(\mathbf{x}, \mathbf{z})$ and its gradients?

The Answer is Yes! We can use **stochastic gradient ascent** on the ELBO.

$$\boldsymbol{\lambda}_{t+1} = \boldsymbol{\lambda}_t + \rho_t \nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}_t)$$

The challenge is computing the gradient $\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda})$.

The Gradient of the ELBO

Let's compute the gradient of the ELBO: $\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_{q(\mathbf{z};\boldsymbol{\lambda})}\left[\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z}; \boldsymbol{\lambda})\right]$.

$$\begin{aligned}\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) &= \nabla_{\boldsymbol{\lambda}} \int q(\mathbf{z}; \boldsymbol{\lambda}) \left[\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z}; \boldsymbol{\lambda})\right] d\mathbf{z} \\ &= \int \nabla_{\boldsymbol{\lambda}} q(\mathbf{z}; \boldsymbol{\lambda}) \left[\dots\right] d\mathbf{z} - \int q(\mathbf{z}; \boldsymbol{\lambda}) \nabla_{\boldsymbol{\lambda}} \log q(\mathbf{z}; \boldsymbol{\lambda}) d\mathbf{z}\end{aligned}$$

The gradient is inside the integral, acting on q. This prevents us from using simple Monte Carlo estimation.
We need a way to rewrite this as an expectation that we can approximate with samples. There are two main approaches.

The Gradient of the ELBO

Let's compute the gradient of the ELBO: $\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_{q(\mathbf{z};\boldsymbol{\lambda})}\left[\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z}; \boldsymbol{\lambda})\right]$.

$$\begin{aligned}\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) &= \nabla_{\boldsymbol{\lambda}} \int q(\mathbf{z}; \boldsymbol{\lambda}) \left[\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z}; \boldsymbol{\lambda})\right] d\mathbf{z} \\ &= \int \nabla_{\boldsymbol{\lambda}} q(\mathbf{z}; \boldsymbol{\lambda}) \left[\dots\right] d\mathbf{z} - \int q(\mathbf{z}; \boldsymbol{\lambda}) \nabla_{\boldsymbol{\lambda}} \log q(\mathbf{z}; \boldsymbol{\lambda}) d\mathbf{z}\end{aligned}$$

The gradient is inside the integral, acting on q. This prevents us from using simple Monte Carlo estimation.

The Gradient of the ELBO

Let's compute the gradient of the ELBO: $\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_{q(\mathbf{z};\boldsymbol{\lambda})}\left[\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z}; \boldsymbol{\lambda})\right]$.

$$\begin{aligned}\nabla_{\boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\lambda}) &= \nabla_{\boldsymbol{\lambda}} \int q(\mathbf{z}; \boldsymbol{\lambda}) \left[\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z}; \boldsymbol{\lambda})\right] d\mathbf{z} \\ &= \int \nabla_{\boldsymbol{\lambda}} q(\mathbf{z}; \boldsymbol{\lambda}) \left[\dots\right] d\mathbf{z} - \int q(\mathbf{z}; \boldsymbol{\lambda}) \nabla_{\boldsymbol{\lambda}} \log q(\mathbf{z}; \boldsymbol{\lambda}) d\mathbf{z}\end{aligned}$$

The gradient is inside the integral, acting on q. This prevents us from using simple Monte Carlo estimation.
We need a way to rewrite this as an expectation that we can approximate with samples. There are two main approaches.

Estimator 1: The Score Function (REINFORCE)

We can use the "log-derivative trick": $\nabla_{\lambda} q(\mathbf{z}) = q(\mathbf{z}) \nabla_{\lambda} \log q(\mathbf{z})$.

The Score Function Estimator

Applying this trick gives:

$$\begin{aligned} \nabla_{\lambda} \mathcal{L}(\boldsymbol{\lambda}) &= \int q(\mathbf{z}; \boldsymbol{\lambda}) \nabla_{\lambda} \log q(\mathbf{z}; \boldsymbol{\lambda}) \left[\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z}; \boldsymbol{\lambda})\right] d\mathbf{z} \\ &= \mathbb{E}_{q(\mathbf{z};\boldsymbol{\lambda})} \left[\nabla_{\lambda} \log q(\mathbf{z}; \boldsymbol{\lambda}) \left(\log p(\mathbf{x}, \mathbf{z}) - \log q(\mathbf{z}; \boldsymbol{\lambda})\right)\right] \end{aligned}$$

- This is now an expectation! We can approximate it with Monte Carlo samples.
- For one sample $\mathbf{z}^{(s)} \sim q(\mathbf{z}; \boldsymbol{\lambda})$:

$$\widehat{\nabla_{\lambda} \mathcal{L}} \approx \nabla_{\lambda} \log q(\mathbf{z}^{(s)}; \boldsymbol{\lambda}) \left(\log p(\mathbf{x}, \mathbf{z}^{(s)}) - \log q(\mathbf{z}^{(s)}; \boldsymbol{\lambda})\right)$$

- **Problem:** This estimator is known to have very high variance, making optimization slow and unstable.

Estimator 2: The Reparameterization Trick

This is the "variational trick" that unlocked modern deep generative models like VAEs.

The Reparameterization Trick

Instead of sampling $\mathbf{z}$ directly from $q(\mathbf{z}; \boldsymbol{\lambda})$, we express $\mathbf{z}$ as a deterministic and differentiable transformation of a "base" random variable $\boldsymbol{\epsilon}$ whose distribution does not depend on $\boldsymbol{\lambda}$.

$$\mathbf{z} = g(\boldsymbol{\epsilon}, \boldsymbol{\lambda}) \quad \text{where} \quad \boldsymbol{\epsilon} \sim p(\boldsymbol{\epsilon})$$

Example: Gaussian Case

- Let $q(z; \mu, \sigma) = \mathcal{N}(z \mid \mu, \sigma^2)$.
- We can reparameterize this by first sampling $\epsilon \sim \mathcal{N}(0, 1)$.
- Then, we deterministically compute $z = \mu + \sigma \cdot \epsilon$.
- Here, $\boldsymbol{\lambda} = \{\mu, \sigma\}$, $p(\epsilon)$ is a standard normal, and $g(\epsilon, \boldsymbol{\lambda}) = \mu + \sigma\epsilon$.

Reparameterization Gradient

The trick allows us to move the gradient operator *inside* the expectation.

$$\begin{aligned}\nabla_{\lambda}\mathcal{L}(\boldsymbol{\lambda}) &= \nabla_{\lambda}\,\mathbb{E}_{q(\mathbf{z};\boldsymbol{\lambda})}[f(\mathbf{z})] \quad \text{where } f(\mathbf{z}) = \log p(\mathbf{x},\mathbf{z}) - \log q(\mathbf{z};\boldsymbol{\lambda}) \\ &= \nabla_{\lambda}\,\mathbb{E}_{p(\boldsymbol{\epsilon})}[f(g(\boldsymbol{\epsilon},\boldsymbol{\lambda}))] \quad \text{(Change of variables)} \\ &= \mathbb{E}_{p(\boldsymbol{\epsilon})}[\nabla_{\lambda} f(g(\boldsymbol{\epsilon},\boldsymbol{\lambda}))] \quad \text{(Leibniz rule)}\end{aligned}$$

The Reparameterization Gradient Estimator

We can now get a low-variance Monte Carlo estimate by sampling $\boldsymbol{\epsilon}$ and using auto-differentiation:

$$\widehat{\nabla_{\lambda}\mathcal{L}} \approx \frac{1}{S}\sum_{s=1}^{S} \nabla_{\lambda} f(g(\boldsymbol{\epsilon}^{(s)},\boldsymbol{\lambda})) \quad \text{where } \boldsymbol{\epsilon}^{(s)} \sim p(\boldsymbol{\epsilon})$$

This works because the source of randomness, $\boldsymbol{\epsilon}$, is now independent of the parameters $\boldsymbol{\lambda}$ we are differentiating with respect to.

BBVI Summary

Black Box Variational Inference (BBVI) refers to using stochastic gradient ascent on the ELBO with a generic gradient estimator.

Estimator	**Requirement**	**Variance**
Score Function (REINFORCE)	Differentiable $\log q(\mathbf{z}; \boldsymbol{\lambda})$	High
Reparameterization Trick	Differentiable mapping $g(\cdot)$	Low

- The reparameterization trick is preferred whenever applicable due to its much lower variance.
- It enabled the training of complex models like Variational Autoencoders (VAEs).
- We can now perform VI on any model where we can evaluate $\log p(\mathbf{x}, \mathbf{z})$ and its gradients, without needing model-specific update derivations.

Normalizing Flows: The Core Idea

From Simple to Complex via Transformation

A Normalizing Flow transforms a simple probability distribution into a more complex one by applying a sequence of invertible and differentiable transformations.

1. Start with a random variable $\mathbf{z}_0$ with a simple, known density $q_0(\mathbf{z}_0)$.
   - E.g., $\mathbf{z}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.

Normalizing Flows: The Core Idea

From Simple to Complex via Transformation

A Normalizing Flow transforms a simple probability distribution into a more complex one by applying a sequence of invertible and differentiable transformations.

1. Start with a random variable $\mathbf{z}_0$ with a simple, known density $q_0(\mathbf{z}_0)$.
   - E.g., $\mathbf{z}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
2. Apply a sequence of K invertible mappings f_k:

$$\mathbf{z}_k = f_k(\mathbf{z}_{k-1})$$

Normalizing Flows: The Core Idea

From Simple to Complex via Transformation

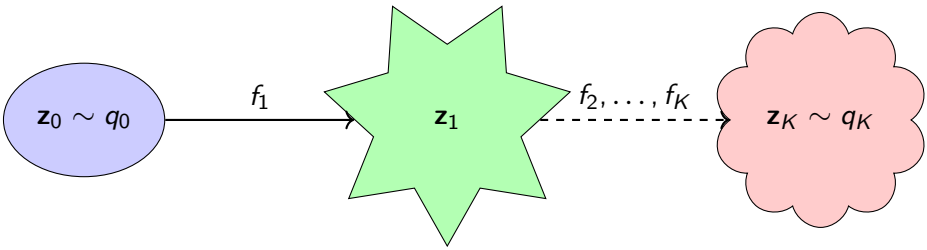
A Normalizing Flow transforms a simple probability distribution into a more complex one by applying a sequence of invertible and differentiable transformations.

1. Start with a random variable $\mathbf{z}_0$ with a simple, known density $q_0(\mathbf{z}_0)$.
   - E.g., $\mathbf{z}_0 \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
2. Apply a sequence of K invertible mappings f_k:

$$\mathbf{z}_k = f_k(\mathbf{z}_{k-1})$$

3. The final variable $\mathbf{z}_K = f_K \circ \cdots \circ f_1(\mathbf{z}_0)$ has a complex distribution, which we will use as our variational approximation $q_K(\mathbf{z}_K)$.

Normalizing Flows: The Core Idea



The Change of Variables Formula

To use a flow as a variational distribution, we must be able to compute its log-density $\log q_K(\mathbf{z}_K)$ for the ELBO. This is possible via the change of variables formula.
For one transformation $\mathbf{z}_1 = f_1(\mathbf{z}_0)$, the density is:

$$q_1(\mathbf{z}_1) = q_0(\mathbf{z}_0) \left| \det \left(\frac{\partial \mathbf{z}_0}{\partial \mathbf{z}_1} \right) \right| = q_0(f_1^{-1}(\mathbf{z}_1)) \left| \det \left(\frac{\partial f_1^{-1}}{\partial \mathbf{z}_1} \right) \right|$$

The Change of Variables Formula

To use a flow as a variational distribution, we must be able to compute its log-density $\log q_K(\mathbf{z}_K)$ for the ELBO. This is possible via the change of variables formula.
For one transformation $\mathbf{z}_1 = f_1(\mathbf{z}_0)$, the density is:

$$q_1(\mathbf{z}_1) = q_0(\mathbf{z}_0) \left| \det \left(\frac{\partial \mathbf{z}_0}{\partial \mathbf{z}_1} \right) \right| = q_0(f_1^{-1}(\mathbf{z}_1)) \left| \det \left(\frac{\partial f_1^{-1}}{\partial \mathbf{z}_1} \right) \right|$$

Taking the log:

$$\log q_1(\mathbf{z}_1) = \log q_0(f_1^{-1}(\mathbf{z}_1)) - \log \left| \det \left(\frac{\partial f_1}{\partial \mathbf{z}_0} \right) \right|$$

where the second equality uses the inverse function theorem for Jacobians.

The Change of Variables Formula

To use a flow as a variational distribution, we must be able to compute its log-density $\log q_K(\mathbf{z}_K)$ for the ELBO. This is possible via the change of variables formula.
For one transformation $\mathbf{z}_1 = f_1(\mathbf{z}_0)$, the density is:

$$q_1(\mathbf{z}_1) = q_0(\mathbf{z}_0) \left| \det \left(\frac{\partial \mathbf{z}_0}{\partial \mathbf{z}_1} \right) \right| = q_0(f_1^{-1}(\mathbf{z}_1)) \left| \det \left(\frac{\partial f_1^{-1}}{\partial \mathbf{z}_1} \right) \right|$$

Taking the log:

$$\log q_1(\mathbf{z}_1) = \log q_0(f_1^{-1}(\mathbf{z}_1)) - \log \left| \det \left(\frac{\partial f_1}{\partial \mathbf{z}_0} \right) \right|$$

where the second equality uses the inverse function theorem for Jacobians.

Log-Density of a Flow

For a chain of K transformations, the log-density is a sum:

$$\log q_K(\mathbf{z}_K) = \log q_0(\mathbf{z}_0) - \sum_{k=1}^{K} \log \left| \det \left(\frac{\partial f_k}{\partial \mathbf{z}_{k-1}} \right) \right|$$

Using Flows for Variational Inference

We can now define a highly flexible variational family q_K and optimize it.

1. Let the variational parameters $\boldsymbol{\lambda}$ be the parameters of the transformations $\{f_k\}_{k=1}^K$.

Using Flows for Variational Inference

We can now define a highly flexible variational family q_K and optimize it.

1. Let the variational parameters $\boldsymbol{\lambda}$ be the parameters of the transformations $\{f_k\}_{k=1}^K$.
2. The ELBO is:

$$\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_{q_K(\mathbf{z}_K)}[\log p(\mathbf{x}, \mathbf{z}_K) - \log q_K(\mathbf{z}_K)]$$

Using Flows for Variational Inference

We can now define a highly flexible variational family q_K and optimize it.

1. Let the variational parameters $\boldsymbol{\lambda}$ be the parameters of the transformations $\{f_k\}_{k=1}^K$.
2. The ELBO is:

$$\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_{q_K(\mathbf{z}_K)}[\log p(\mathbf{x}, \mathbf{z}_K) - \log q_K(\mathbf{z}_K)]$$

3. We can estimate this using Monte Carlo sampling and the reparameterization trick:
   1. Sample $\mathbf{z}_0 \sim q_0(\mathbf{z}_0)$.
   2. Compute $\mathbf{z}_K = f_K(\dots f_1(\mathbf{z}_0))$.
   3. Compute $\log q_K(\mathbf{z}_K)$ using the change of variables formula.
   4. The ELBO sample is $\log p(\mathbf{x}, \mathbf{z}_K) - \log q_K(\mathbf{z}_K)$.

Using Flows for Variational Inference

We can now define a highly flexible variational family q_K and optimize it.

1. Let the variational parameters $\boldsymbol{\lambda}$ be the parameters of the transformations $\{f_k\}_{k=1}^K$.
2. The ELBO is:

$$\mathcal{L}(\boldsymbol{\lambda}) = \mathbb{E}_{q_K(\mathbf{z}_K)}[\log p(\mathbf{x}, \mathbf{z}_K) - \log q_K(\mathbf{z}_K)]$$

3. We can estimate this using Monte Carlo sampling and the reparameterization trick:
   1. Sample $\mathbf{z}_0 \sim q_0(\mathbf{z}_0)$.
   2. Compute $\mathbf{z}_K = f_K(\ldots f_1(\mathbf{z}_0))$.
   3. Compute $\log q_K(\mathbf{z}_K)$ using the change of variables formula.
   4. The ELBO sample is $\log p(\mathbf{x}, \mathbf{z}_K) - \log q_K(\mathbf{z}_K)$.
4. We can then compute gradients w.r.t. $\boldsymbol{\lambda}$ and perform stochastic gradient ascent.

Example Flow Architectures

The key is to design bijections f_k that are expressive but have efficient Jacobian determinant calculations.

- **Planar Flows:** Stretch and contract the space along a specific direction.

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^T\mathbf{z} + b)$$

Jacobian determinant is cheap to compute via the matrix determinant lemma.

Example Flow Architectures

The key is to design bijections f_k that are expressive but have efficient Jacobian determinant calculations.

- **Planar Flows:** Stretch and contract the space along a specific direction.

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^T\mathbf{z} + b)$$

  Jacobian determinant is cheap to compute via the matrix determinant lemma.

- **Radial Flows:** Modify the space around a reference point.

$$f(\mathbf{z}) = \mathbf{z} + \beta h(\alpha, r)(\mathbf{z} - \mathbf{z}_{\text{ref}})$$

  Also has an efficient Jacobian determinant.

Example Flow Architectures

The key is to design bijections f_k that are expressive but have efficient Jacobian determinant calculations.

- **Planar Flows:** Stretch and contract the space along a specific direction.

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^T\mathbf{z} + b)$$

  Jacobian determinant is cheap to compute via the matrix determinant lemma.

- **Radial Flows:** Modify the space around a reference point.

$$f(\mathbf{z}) = \mathbf{z} + \beta h(\alpha, r)(\mathbf{z} - \mathbf{z}_{\text{ref}})$$

  Also has an efficient Jacobian determinant.

- **Autoregressive Flows (e.g., MAF, IAF):** Model each dimension conditioned on the previous ones.

$$z_i = \mu_i + \sigma_i \epsilon_i \quad \text{where } (\mu_i, \sigma_i) = \text{NN}(z_{1:i-1})$$

  The Jacobian matrix is triangular, so the determinant is just the product of the diagonal elements (very fast).

Example Flow Architectures

The key is to design bijections f_k that are expressive but have efficient Jacobian determinant calculations.

- **Planar Flows:** Stretch and contract the space along a specific direction.

$$f(\mathbf{z}) = \mathbf{z} + \mathbf{u}h(\mathbf{w}^T\mathbf{z} + b)$$

  Jacobian determinant is cheap to compute via the matrix determinant lemma.

- **Radial Flows:** Modify the space around a reference point.

$$f(\mathbf{z}) = \mathbf{z} + \beta h(\alpha, r)(\mathbf{z} - \mathbf{z}_{\text{ref}})$$

  Also has an efficient Jacobian determinant.

- **Autoregressive Flows (e.g., MAF, IAF):** Model each dimension conditioned on the previous ones.

$$z_i = \mu_i + \sigma_i \epsilon_i \quad \text{where } (\mu_i, \sigma_i) = \text{NN}(z_{1:i-1})$$

  The Jacobian matrix is triangular, so the determinant is just the product of the diagonal elements (very fast).

Beyond Explicit Densities: The Score Function

VAEs and Normalizing Flows model an explicit, normalized probability density $q(\mathbf{z})$. This requires careful design (e.g., tractable Jacobians for flows) and can be restrictive.

Beyond Explicit Densities: The Score Function

VAEs and Normalizing Flows model an explicit, normalized probability density $q(\mathbf{z})$. This requires careful design (e.g., tractable Jacobians for flows) and can be restrictive.

A Different Paradigm

What if we only model the **gradient** of the log-density, without ever needing the density itself (or its normalizing constant)?

Beyond Explicit Densities: The Score Function

The Score Function

The score of a distribution $p(\mathbf{x})$ is the gradient of its log-probability with respect to the data $\mathbf{x}$:

$$\nabla_{\mathbf{x}} \log p(\mathbf{x})$$

- The score is a vector field. At any point $\mathbf{x}$, it points in the direction of the steepest ascent of the log-density.
- Crucially, if $p(\mathbf{x}) = \frac{\tilde{p}(\mathbf{x})}{Z}$, the score does not depend on the intractable normalizing constant Z:

$$\nabla_{\mathbf{x}} \log p(\mathbf{x}) = \nabla_{\mathbf{x}} (\log \tilde{p}(\mathbf{x}) - \log Z) = \nabla_{\mathbf{x}} \log \tilde{p}(\mathbf{x})$$

The Goal: Score Matching

Our goal is to train a neural network, the **score model** $s_\theta(\mathbf{x})$, to approximate the true data score $\nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x})$.

The Score Matching Objective

The most natural objective is the squared L2 distance between the model's score and the true data score, averaged over the data distribution:

$$J(\boldsymbol{\theta}) = \frac{1}{2} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[\| s_\theta(\mathbf{x}) - \nabla_{\mathbf{x}} \log p_{\text{data}}(\mathbf{x}) \|_2^2 \right]$$

The Goal: Score Matching

Our goal is to train a neural network, the **score model** $s_\theta(\mathbf{x})$, to approximate the true data score $\nabla_\mathbf{x} \log p_{\text{data}}(\mathbf{x})$.

The Score Matching Objective

The most natural objective is the squared L2 distance between the model's score and the true data score, averaged over the data distribution:

$$J(\boldsymbol{\theta}) = \frac{1}{2} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[\| s_\theta(\mathbf{x}) - \nabla_\mathbf{x} \log p_{\text{data}}(\mathbf{x}) \|_2^2 \right]$$

The Problem

This objective is intractable because we don't know the true data score $\nabla_\mathbf{x} \log p_{\text{data}}(\mathbf{x})$. We only have samples from $p_{\text{data}}(\mathbf{x})$.

The Goal: Score Matching

Our goal is to train a neural network, the **score model** $s_\theta(\mathbf{x})$, to approximate the true data score $\nabla_\mathbf{x} \log p_{\text{data}}(\mathbf{x})$.

The Score Matching Objective

The most natural objective is the squared L2 distance between the model's score and the true data score, averaged over the data distribution:

$$J(\boldsymbol{\theta}) = \frac{1}{2} \mathbb{E}_{p_{\text{data}}(\mathbf{x})} \left[\| s_\theta(\mathbf{x}) - \nabla_\mathbf{x} \log p_{\text{data}}(\mathbf{x}) \|_2^2 \right]$$

The Problem

This objective is intractable because we don't know the true data score $\nabla_\mathbf{x} \log p_{\text{data}}(\mathbf{x})$. We only have samples from $p_{\text{data}}(\mathbf{x})$.

Solution (Vincent, 2011): Denoising Score Matching.

The Breakthrough: Denoising Score Matching (DSM)

Instead of matching the score of clean data, we match the score of **noise-perturbed data**.

1. Define a noise process, typically Gaussian: $q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2 \mathbf{I})$.

The Breakthrough: Denoising Score Matching (DSM)

Instead of matching the score of clean data, we match the score of **noise-perturbed data**.

1. Define a noise process, typically Gaussian: $q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2 \mathbf{I})$.
2. Define the perturbed data distribution: $q_\sigma(\tilde{\mathbf{x}}) = \int p_{\text{data}}(\mathbf{x}) q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) d\mathbf{x}$.

The Breakthrough: Denoising Score Matching (DSM)

Instead of matching the score of clean data, we match the score of **noise-perturbed data**.

1. Define a noise process, typically Gaussian: $q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2 \mathbf{I})$.
2. Define the perturbed data distribution: $q_\sigma(\tilde{\mathbf{x}}) = \int p_{\text{data}}(\mathbf{x}) q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) d\mathbf{x}$.
3. The new goal is to match the score of this perturbed distribution: $\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}})$.

The Breakthrough: Denoising Score Matching (DSM)

Instead of matching the score of clean data, we match the score of **noise-perturbed data**.

1. Define a noise process, typically Gaussian: $q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2 \mathbf{I})$.
2. Define the perturbed data distribution: $q_\sigma(\tilde{\mathbf{x}}) = \int p_{\text{data}}(\mathbf{x}) q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) d\mathbf{x}$.
3. The new goal is to match the score of this perturbed distribution: $\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}})$.

The Key Insight

It can be shown that minimizing the score matching objective for the perturbed data is equivalent to minimizing a much simpler objective:

$$J_{DSM}(\boldsymbol{\theta}) = \frac{1}{2} \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}}, \mathbf{x})} \left[\| s_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) \|_2^2 \right]$$

The Breakthrough: Denoising Score Matching (DSM)

Instead of matching the score of clean data, we match the score of **noise-perturbed data**.

1. Define a noise process, typically Gaussian: $q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) = \mathcal{N}(\tilde{\mathbf{x}}; \mathbf{x}, \sigma^2 \mathbf{I})$.
2. Define the perturbed data distribution: $q_\sigma(\tilde{\mathbf{x}}) = \int p_{\text{data}}(\mathbf{x}) q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) d\mathbf{x}$.
3. The new goal is to match the score of this perturbed distribution: $\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}})$.

The Key Insight

It can be shown that minimizing the score matching objective for the perturbed data is equivalent to minimizing a much simpler objective:

$$J_{DSM}(\boldsymbol{\theta}) = \frac{1}{2} \mathbb{E}_{q_\sigma(\tilde{\mathbf{x}}, \mathbf{x})} \left[\| s_{\boldsymbol{\theta}}(\tilde{\mathbf{x}}) - \nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) \|_2^2 \right]$$

For Gaussian noise, the target score is simply: $\nabla_{\tilde{\mathbf{x}}} \log q_\sigma(\tilde{\mathbf{x}} \mid \mathbf{x}) = -\frac{\tilde{\mathbf{x}} - \mathbf{x}}{\sigma^2}$. **This is tractable!** We just need to predict the noise from the noisy data.

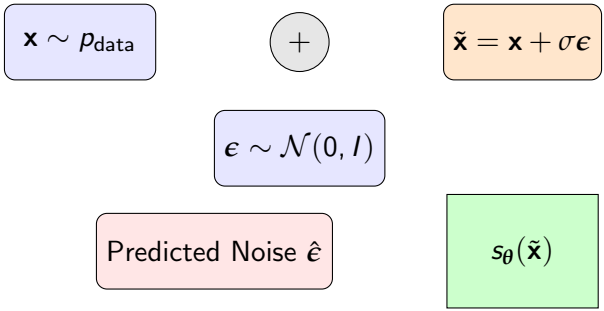
DSM as Denoising

The Denoising Score Matching objective is remarkably simple:

The DSM Loss

$$J_{DSM}(\boldsymbol{\theta}) \propto \mathbb{E}_{\mathbf{x}\sim p_{\text{data}},\boldsymbol{\epsilon}\sim\mathcal{N}(0,I)}\left[\|s_{\boldsymbol{\theta}}(\mathbf{x}+\sigma\boldsymbol{\epsilon}) - (-\boldsymbol{\epsilon}/\sigma)\|_2^2\right]$$

Letting the network predict the noise $\boldsymbol{\epsilon}$ directly, we get the standard denoising objective.



The score model is trained as a simple denoising autoencoder.

Noise Conditional Score Networks (NCSN)

A single noise level σ is not enough.

- Small σ: Captures fine details of the data distribution but fails in low-density regions.
- Large σ: "Washes out" details but provides a stable, smooth score field globally.

Noise Conditional Score Networks (NCSN)

A single noise level σ is not enough.

- Small σ: Captures fine details of the data distribution but fails in low-density regions.
- Large σ: "Washes out" details but provides a stable, smooth score field globally.

The Solution: Condition on Noise Level

Use a geometric progression of L noise levels: $\sigma_1 > \sigma_2 > \cdots > \sigma_L$. Train a single neural network that is conditioned on the noise level: $s_\theta(\mathbf{x}, \sigma_i)$.

The training objective becomes a weighted sum of the DSM losses for each noise level:

$$J(\boldsymbol{\theta}) = \sum_{i=1}^{L} w_i \cdot \mathbb{E}\left[\|s_\theta(\tilde{\mathbf{x}}_i, \sigma_i) - \nabla_{\tilde{\mathbf{x}}_i} \log q_{\sigma_i}(\tilde{\mathbf{x}}_i \mid \mathbf{x})\|_2^2\right]$$

- This single network $s_\theta(\mathbf{x}, \sigma)$ learns the score field across all scales.
- In practice, we randomly sample a noise level σ_i for each data point in a minibatch.

Generating Samples: Langevin Dynamics

Once we have a trained score model $s_\theta(\mathbf{x})$, how do we get samples from $p_{\text{data}}(\mathbf{x})$?

Langevin Dynamics

An MCMC method that uses the score function to generate samples. The update rule is:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \frac{\alpha}{2}\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t) + \sqrt{\alpha}\mathbf{z}_t, \quad \text{where } \mathbf{z}_t \sim \mathcal{N}(0, \mathbf{I})$$

- **First Term (Gradient Step):** A step in the direction of increasing log probability. It pushes samples towards high-density regions (modes).
- **Second Term (Noise Step):** Injected noise prevents the sampler from collapsing to a single mode and ensures it explores the distribution.

Generating Samples: Langevin Dynamics

Once we have a trained score model $s_\theta(\mathbf{x})$, how do we get samples from $p_{\text{data}}(\mathbf{x})$?

Langevin Dynamics

An MCMC method that uses the score function to generate samples. The update rule is:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \frac{\alpha}{2}\nabla_{\mathbf{x}_t} \log p(\mathbf{x}_t) + \sqrt{\alpha}\mathbf{z}_t, \quad \text{where } \mathbf{z}_t \sim \mathcal{N}(0, \mathbf{I})$$

- **First Term (Gradient Step):** A step in the direction of increasing log probability. It pushes samples towards high-density regions (modes).
- **Second Term (Noise Step):** Injected noise prevents the sampler from collapsing to a single mode and ensures it explores the distribution.
- **With our score model:** We replace the true score with our learned one:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \frac{\alpha}{2}s_\theta(\mathbf{x}_t) + \sqrt{\alpha}\mathbf{z}_t$$

Putting It All Together: Annealed Langevin Dynamics

This algorithm combines the multi-scale score network (NCSN) with Langevin Dynamics for high-fidelity sample generation.

Algorithm: Annealed Langevin Dynamics

1. **Initialize:** Start with a sample from a simple noise distribution, $\mathbf{x}_0 \sim \mathcal{N}(0, \sigma_1^2 \mathbf{I})$.

Putting It All Together: Annealed Langevin Dynamics

This algorithm combines the multi-scale score network (NCSN) with Langevin Dynamics for high-fidelity sample generation.

Algorithm: Annealed Langevin Dynamics

1. **Initialize:** Start with a sample from a simple noise distribution, $\mathbf{x}_0 \sim \mathcal{N}(0, \sigma_1^2 \mathbf{I})$.
2. **Anneal over noise levels:** For $i = 1, \ldots, L$:
   - Let α_i be the step size for level i.
   - **Run Langevin steps:** For $t = 1, \ldots, T$:

$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} + \frac{\alpha_i}{2} s_\theta(\mathbf{x}_{t-1}, \sigma_i) + \sqrt{\alpha_i} \mathbf{z}_t$$

Putting It All Together: Annealed Langevin Dynamics

This algorithm combines the multi-scale score network (NCSN) with Langevin Dynamics for high-fidelity sample generation.

Algorithm: Annealed Langevin Dynamics

1. **Initialize:** Start with a sample from a simple noise distribution, $\mathbf{x}_0 \sim \mathcal{N}(0, \sigma_1^2 \mathbf{I})$.
2. **Anneal over noise levels:** For $i = 1, \ldots, L$:
   - Let α_i be the step size for level i.
   - **Run Langevin steps:** For $t = 1, \ldots, T$:

$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} + \frac{\alpha_i}{2} s_\theta(\mathbf{x}_{t-1}, \sigma_i) + \sqrt{\alpha_i} \mathbf{z}_t$$

3. **Return** the final sample $\mathbf{x}_T$ from the last (lowest noise) level.

This process starts sampling from a very blurry version of the data distribution (σ_1 is large) and gradually refines the samples by moving to lower noise levels, "annealing" them into sharp, realistic samples.

Connection to Diffusion Models

Score-based generative models are deeply connected to another class of models: **Denoising Diffusion Probabilistic Models (DDPMs)**.

Score-Based Models

- Define score via continuous-time Stochastic Differential Equation (SDE).
- Training: Learn the score function $\nabla \log p_t(\mathbf{x}_t)$.
- Sampling: Solve a reverse-time SDE using numerical solvers (like Langevin).

Diffusion Models (DDPMs)

- Define a discrete-time Markov chain that adds noise (forward process).
- Training: Learn to reverse this process one step at a time (denoise).
- Sampling: Simulate the learned reverse Markov chain from pure noise.

Connection to Diffusion Models

Score-based generative models are deeply connected to another class of models: **Denoising Diffusion Probabilistic Models (DDPMs)**.

Score-Based Models

- Define score via continuous-time Stochastic Differential Equation (SDE).
- Training: Learn the score function $\nabla \log p_t(\mathbf{x}_t)$.
- Sampling: Solve a reverse-time SDE using numerical solvers (like Langevin).

Diffusion Models (DDPMs)

- Define a discrete-time Markov chain that adds noise (forward process).
- Training: Learn to reverse this process one step at a time (denoise).
- Sampling: Simulate the learned reverse Markov chain from pure noise.

The Unification

It has been shown that the denoising network in DDPMs is implicitly learning the score. Both frameworks can be unified under the lens of Stochastic Differential Equations. For practical purposes, they are two powerful perspectives on the same core idea, and both achieve state-of-the-art results.

Score-Based Models: Pros and Cons

Pros

- **SOTA Sample Quality:** Currently produce the highest-quality samples for images, audio, etc.
- **Flexible Architectures:** No restrictions like invertible transforms or tractable Jacobians. Any standard architecture (e.g., U-Net) works.
- **No Partition Function:** Avoids the need to compute intractable normalizing constants.

Cons

- **Slow Sampling:** Generation is an iterative process (hundreds or thousands of steps), making it slower than one-shot methods like VAEs or Flows. (This is improving rapidly).
- **No Explicit Likelihood:** Cannot directly compute $p(\mathbf{x})$. This requires solving another SDE and is computationally expensive.
- **No Free Encoder:** Unlike VAEs, there is no built-in encoder to get latent representations of data.

Generative Model Families: A Comparison

Property	VAEs	Normalizing Flows	Score/Diffusion
Primary Goal	Approx. Inference	Density Estimation	Sample Generation
Sample Quality	Good, but often blurry	Good, but can be rigid	State-of-the-Art
Sampling Speed	Fast (one-shot)	Fast (one-shot)	Slow (iterative)
Latent Encoder	Yes (built-in)	No (but invertible)	No (requires separate training)
Likelihood Eval.	No (ELBO is a lower bound)	Yes (exact and fast)	No (slow and approximate)
Key Idea	Maximize ELBO	Change of Variables	Learn Score / Denoise

Each model class has a unique set of trade-offs, making them suitable for different tasks. The field is rapidly evolving, with hybrid models combining the best of each approach.

Summary: VI vs. MCMC

Variational Inference	**Markov Chain Monte Carlo**
Turns inference into an optimization problem.	Turns inference into a sampling problem.
Fast, often scales well to large data.	Slow, can be prohibitive for large data.
Provides an approximation to the posterior. Accuracy depends on the variational family.	Asymptotically exact, provides samples from the true posterior.
Deterministic. Convergence is easy to check (monitor ELBO).	Stochastic. Convergence can be hard to diagnose.
Tends to underestimate posterior variance.	Captures the full posterior structure, including variance and correlations.
Provides a lower bound on the model evidence.	Does not provide the model evidence.

The Journey of Approximate Inference

- **The Core Idea:** Turn intractable integration into a tractable problem, either via **optimization (VI)** or **sampling (MCMC)**.

The Journey of Approximate Inference

- **The Core Idea:** Turn intractable integration into a tractable problem, either via **optimization (VI)** or **sampling (MCMC)**.
- **Classic VI: Mean-Field VI** is fast but restrictive and model-specific.

The Journey of Approximate Inference

- **The Core Idea:** Turn intractable integration into a tractable problem, either via **optimization (VI)** or **sampling (MCMC)**.
- **Classic VI: Mean-Field VI** is fast but restrictive and model-specific.
- **Modern VI: Black Box VI** with the **Reparameterization Trick** makes the method general and scalable, enabling VAEs.

The Journey of Approximate Inference

- **The Core Idea:** Turn intractable integration into a tractable problem, either via **optimization (VI)** or **sampling (MCMC)**.
- **Classic VI: Mean-Field VI** is fast but restrictive and model-specific.
- **Modern VI: Black Box VI** with the **Reparameterization Trick** makes the method general and scalable, enabling VAEs.
- **Expressive VI: Normalizing Flows** build highly flexible, exact-likelihood models.

The Journey of Approximate Inference

- **The Core Idea:** Turn intractable integration into a tractable problem, either via **optimization (VI)** or **sampling (MCMC)**.
- **Classic VI: Mean-Field VI** is fast but restrictive and model-specific.
- **Modern VI: Black Box VI** with the **Reparameterization Trick** makes the method general and scalable, enabling VAEs.
- **Expressive VI: Normalizing Flows** build highly flexible, exact-likelihood models.
- **Implicit Models: Score Matching** and **Diffusion Models** achieve SOTA sample quality by learning the gradient field of the data distribution instead of the density itself.

Probabilistic machine learning offers a rich toolbox for tackling uncertainty, with each method providing a different trade-off between speed, accuracy, and flexibility.

Thank You

Questions?
andrijaster@gmail.com