Latent Variable Models

Latent variables: a roadmap

We will look at four model families that use hidden structure:

1. **HMMs:** latent states over time (classical AI / ML)
2. **Autoencoders:** latent representations (deep learning)
3. **VAEs:** probabilistic latent representations (generative)
4. **Diffusion models:** latent *trajectories* (state-of-the-art generation)
5. **Multimodal models:** latent *shared* meaning (text-to-image generation, captioning, retrieval,. . .)

Theme: introduce hidden variables to make modeling easier.

Hidden Markov Models (HMMs): the idea

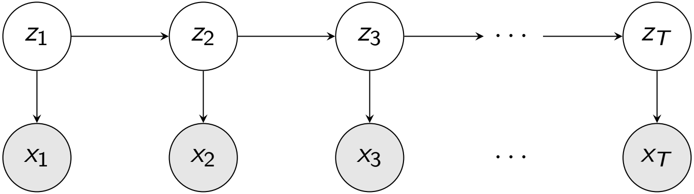
Goal: model a sequence $x_1, \ldots, x_T$ when the world has a hidden state.

- Hidden state: $z_t \in \{1, \ldots, K\}$ (not directly observed)
- Observation: x_t (what we measure)

Example: z_t = weather (hot/cold), x_t = ice cream sales (high/low).

Key assumption: the hidden state changes gradually over time.

HMM as a Bayesian network (generative model)



$$p(z_{1:T}, x_{1:T}) = p(z_1) \prod_{t=2}^{T} p(z_t \mid z_{t-1}) \prod_{t=1}^{T} p(x_t \mid z_t).$$

HMM: how generation works

To generate a sequence $(x_1, \ldots, x_T)$:

1. Sample an initial hidden state: $z_1 \sim p(z_1)$
2. Sample the first observation: $x_1 \sim p(x_1 \mid z_1)$
3. For $t = 2, \ldots, T$:

$$z_t \sim p(z_t \mid z_{t-1}), \qquad x_t \sim p(x_t \mid z_t).$$

Interpretation: the hidden state sequence $z_{1:T}$ controls the structure of the data.

Autoencoders (AEs)

Goal: learn a compact representation of data.

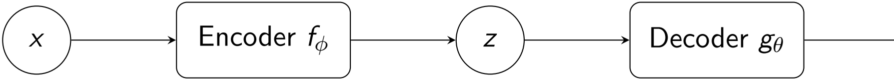
- Encoder: $z = f_\phi(x)$
- Decoder: $\hat{x} = g_\theta(z)$

Training objective:

$$\min_{\phi,\theta} \|x - \hat{x}\|^2.$$

Key point: a standard AE is **not probabilistic** (it is a neural network).

Autoencoder diagram (computational graph)



- z is a learned **representation** (a bottleneck).
- Good for compression, denoising, and feature learning.
- But: sampling new x is not well-defined.

Variational Autoencoders (VAEs): why they exist

Autoencoders learn a representation z, but they do not define a probability model.

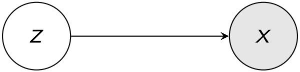
VAE idea: make the latent representation **random**.

$$z \sim p(z) \quad \text{(simple prior, e.g. } \mathcal{N}(0, I))$$

$$x \sim p_\theta(x \mid z) \quad \text{(decoder is probabilistic)}$$

Result: we can generate new samples by sampling z and decoding.

VAE as a Bayesian network (generative model)



$$p_\theta(x, z) = p(z)\, p_\theta(x \mid z), \qquad p_\theta(x) = \int p(z) p_\theta(x \mid z)\, dz.$$

Latent variable: z (not observed in the dataset).

VAE: generation vs inference

Generation (easy):

$$z \sim p(z), \qquad x \sim p_\theta(x \mid z).$$

Inference (hard):

$$p_\theta(z \mid x) = \frac{p(z)p_\theta(x \mid z)}{p_\theta(x)} \quad \text{is usually intractable.}$$

VAE solution: learn an approximate inference network

$$q_\phi(z \mid x)$$

(often called the "encoder").

Diffusion models

Goal: generate realistic images.

Diffusion uses a sequence:

$$x_0, x_1, \ldots, x_T$$

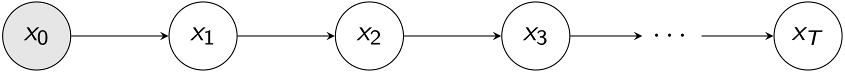
where:

- x_0 is a real image (data)
- x_T is almost pure noise

Key idea:

- Forward process: gradually add noise (easy, fixed).
- Reverse process: learn to remove noise step-by-step.

Diffusion: forward Bayes net (add noise)



- Each step adds a small amount of Gaussian noise.
- After many steps, x_T looks like random noise.

Diffusion: reverse Bayes net (learn to denoise)

$$x_T \rightarrow x_{T-1} \rightarrow x_{T-2} \rightarrow x_{T-3} \rightarrow \cdots \rightarrow x_0$$

- Start from noise: $x_T \sim \mathcal{N}(0, I)$
- Apply a learned denoiser repeatedly to obtain x_0

Diffusion: how it is trained (simplified version)

We do **not** start by learning a full likelihood.

Instead we train a neural network to solve a simpler task:

Given a noisy image x_t, predict the noise level / remove some noise.

Training loop (high level):

1. Take a real image x_0
2. Corrupt it to get a noisy x_t
3. Train the network to predict the noise that was added

Result: the network learns to reverse the forward noising chain.

Diffusion: how generation works (sampling)

Once trained, generation is simple:

1. Sample $x_T \sim \mathcal{N}(0, I)$ (pure noise)
2. For $t = T, T-1, \ldots, 1$:

$$x_{t-1} \leftarrow \text{Denoise}(x_t)$$

3. Output x_0

Key point: unlike VAEs, diffusion generates by **many small steps**.

Multimodal learning: why latent variables help

We often have multiple views of the same underlying content:

$$\text{image } x \qquad \text{and} \qquad \text{text } y.$$

Key challenge: x and y are very different data types.

Motivating idea: introduce a shared latent variable

$$z = \text{“meaning”} \ / \ \text{“concept”} \ / \ \text{“scene”}$$

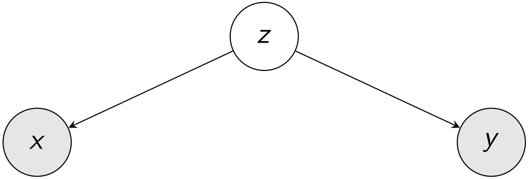
that explains both modalities.

- z captures what is common (semantics).
- x and y contain modality-specific details.

Shared-latent Bayes net for image–text pairs

Let:

$$x = \text{image}, \qquad y = \text{text}, \qquad z = \text{shared meaning}.$$



$$p(x, y, z) = p(z)\, p(x \mid z)\, p(y \mid z).$$

Conditional independence:

$$x \perp y \mid z.$$

How this model supports AI tasks (high level)

The shared-latent model can support multiple tasks:

- **Image captioning:** infer z from x, then generate y.
- **Text-to-image:** infer z from y, then generate x.
- **Cross-modal retrieval:** compare z inferred from x and y.

Takeaway: the latent variable z provides a common "semantic space".

Aligned latent variables

Some multimodal models do **not** explicitly generate x or y.

Instead, they learn two latent representations:

$$u = f(x) \quad \text{(image embedding)}, \qquad v = g(y) \quad \text{(text embedding)}.$$

Training goal: embeddings of matching pairs should be close:

$$u \approx v \quad \text{for paired } (x, y).$$

This enables:

- image–text matching,
- retrieval,
- conditioning (as in text-guided generation).

Bayes-net style view: aligned latent variables

- $u = f(x)$ and $v = g(y)$ are learned embeddings.
- Training encourages u and v to align for paired data.

Interpretation: aligned latent spaces provide a shared semantic geometry.

Summary: latent variables across AI model families

Model family	**Latent variable(s)**	**Typical use**
HMM	z_t (hidden state)	sequences (speec
Autoencoder (AE)	z (representation)	features, compre
VAE	z (random code)	generative mode
Diffusion model	$x_1, \ldots, x_T$ (noise trajectory)	high-quality gene
Multimodal (shared latent)	z (shared meaning)	captioning, text←
Multimodal (aligned latents)	$u = f(x),\ v = g(y)$	retrieval, matchi

Big idea: latent variables introduce hidden structure that makes learning and generation possible.

Learning in latent models: a unifying view

A Bayesian network (BN) gives two things:

1. A **graph** (who depends on whom)
2. A **factorization** of the joint distribution:

$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i \mid \mathrm{Pa}(x_i)).$$

Simple learning view:

Learning = learning the arrows.

Each arrow corresponds to a conditional model $p(\text{child} \mid \text{parents})$.

Two kinds of learning in Bayesian networks

(A) Parameter learning: the graph is fixed, learn the conditionals.

$$p(x_i \mid \mathrm{Pa}(x_i)) \quad \text{(numbers or neural nets)}$$

(B) Structure learning: learn the graph itself.

$$\mathrm{Pa}(x_i) \quad \text{(which arrows exist)}$$

In this module we focus mainly on parameter learning.

Extra note: structure learning is important in scientific discovery and causal modeling.

Observed vs latent nodes: why learning can be hard

A BN may contain:

- **Observed nodes:** given in the dataset.
- **Latent nodes:** not given; must be inferred.

Key point:

- If all nodes are observed: learning is often easy (counts / regression).
- If some nodes are latent: learning requires **inference** (or clever tricks).

This is the main reason latent-variable models are interesting.

HMM learning: learn two arrows (two conditionals)

HMM Bayes net structure:

$$
\begin{array}{ccc}
z_{t-1} & \longrightarrow & z_t \\
 & & \downarrow \\
 & & x_t
\end{array}
$$

So we learn:

$$p(z_t \mid z_{t-1}) \quad \text{(transition)} \qquad \text{and} \qquad p(x_t \mid z_t) \quad \text{(emission)}.$$

Special case: if z_t were observed, we could learn by counting.

HMM learning when states are observed: learning by counts

If the dataset contains $(z_{1:T}, x_{1:T})$:

Transition probabilities:

$$\hat{p}(j \mid i) = \frac{N(i \to j)}{\sum_{j'} N(i \to j')}.$$

Emission probabilities (discrete case):

$$\hat{p}(x = a \mid z = i) = \frac{N(z = i, x = a)}{\sum_{a'} N(z = i, x = a')}.$$

Interpretation: each conditional is just a normalized table of counts.

HMM learning when states are hidden: “soft counts” (EM idea)

Usually, we only observe $x_{1:T}$.

Then we cannot count transitions directly because z_t is unknown.

EM idea (high-level):

1. Infer probabilities of hidden states:

$$p(z_t = i \mid x_{1:T})$$

2. Infer probabilities of hidden transitions:

$$p(z_{t-1} = i, z_t = j \mid x_{1:T})$$

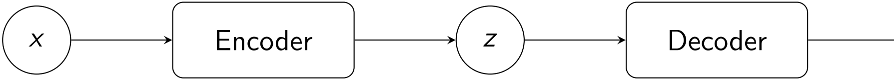
3. Update parameters using these as **expected counts**.

Same formulas as counting, but with probabilities instead of integers.

Autoencoders: the deterministic cousin of latent-variable BNs

Autoencoders are not usually presented as Bayesian networks.

But they still follow the same arrow-learning intuition:



$$z = f_\phi(x), \qquad \hat{x} = g_\theta(z).$$

Learning: adjust parameters so $\hat{x} \approx x$ (reconstruction).

Autoencoder learning = learn a good representation

Training objective:

$$\min_{\phi,\theta} \mathbb{E}_{x\sim\text{data}}[\|x - g_\theta(f_\phi(x))\|^2].$$

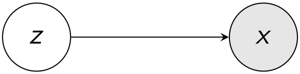
Interpretation in the BN spirit:

- The encoder learns a mapping $x \to z$.
- The decoder learns a mapping $z \to x$.

Limitation: without probabilities, sampling new x is not principled.

VAE: a true latent-variable Bayesian network

The generative model is a simple BN:



- Prior: $z \sim p(z)$ (e.g. $\mathcal{N}(0, I)$)
- Decoder: $x \sim p_\theta(x \mid z)$
- Encoder: $q_\phi(z \mid x)$ approximates $p_\theta(z \mid x)$

Main difference: the encoder outputs a **distribution** over z, not a single point.

VAE learning: why a trick is needed

We want to maximize the likelihood:

$$\log p_\theta(x) = \log \int p(z) p_\theta(x \mid z)\, dz.$$

Problem: the integral over z is usually intractable.

VAE introduces an **inference network** (encoder):

$$q_\phi(z \mid x)$$

which approximates the posterior $p_\theta(z \mid x)$.

High-level view: VAE learns two arrows:

$$x \to z \quad \text{(encoder)} \qquad \text{and} \qquad z \to x \quad \text{(decoder)}.$$

Diffusion models: two Bayes nets on the same variables

Diffusion uses variables:

$$x_0, x_1, \ldots, x_T.$$

There are two chains:

Forward (fixed): add noise

$$x_0 \to x_1 \to \cdots \to x_T$$

Reverse (learned): remove noise

$$x_T \to x_{T-1} \to \cdots \to x_0$$

Learning the arrows: learn each reverse conditional

$$p_\theta(x_{t-1} \mid x_t).$$

Diffusion learning: learn the reverse arrow by denoising

High-level training story:

1. Take a real image x_0.
2. Add noise to obtain x_t (easy to simulate).
3. Train a neural network to predict the noise.

Key supervised learning view:

$$\text{Input: } (x_t, t) \qquad \Rightarrow \qquad \text{Target: noise } \epsilon.$$

Result: the network learns the reverse conditional steps.

Diffusion learning objective (simple form)

A common diffusion loss is:

$$\min_{\theta} \mathbb{E}\left[\|\epsilon - \epsilon_\theta(x_t, t)\|^2\right]$$

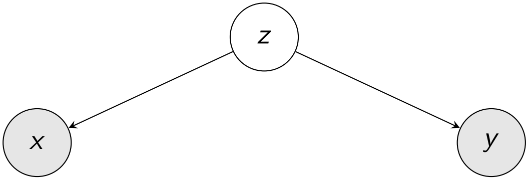
where:

- ϵ is the noise that was added to create x_t,
- $\epsilon_\theta(x_t, t)$ is the network's prediction.

Interpretation: diffusion turns generative modeling into denoising.

Multimodal models: learning with shared latent meaning

A simple generative BN for paired data (x, y):



$$p(x, y, z) = p(z)\, p(x \mid z)\, p(y \mid z).$$

Learning the arrows: learn $p(x \mid z)$ and $p(y \mid z)$ so z captures shared meaning.

Multimodal learning: aligned latent variables (CLIP-style)

Many modern multimodal models learn **aligned embeddings**:

$$u = f(x) \quad \text{(image embedding)}, \qquad v = g(y) \quad \text{(text embedding)}.$$

$$\begin{tikzcd} u \arrow[rrrr, leftrightarrow, dashed] & & & & v \\ x \arrow[u] & & & & y \arrow[u] \end{tikzcd}$$

Learning idea: bring (u_i, v_i) close for matching pairs, push apart non-matching pairs.

Final takeaway: learning = fitting conditional models on a graph

Across all these models, learning can be summarized as:

- Choose a graph (dependencies).
- Learn each conditional model on the arrows:

$$p(\text{child} \mid \text{parents}).$$

What changes between models:

- Are latent nodes present?
- Are the conditionals tables or neural nets?
- Do we need inference (EM / variational / denoising tricks)?

Same core idea: learn the arrows.