CS 6501: Constrained-Aware Generative AI

Lecture Notes 2
Principles of Maximum Likelihood and Generative Learning

Prof. Ferdinando Fioretto
Department of Computer Science, University of Virginia

Thursday, January 15, 2026

Abstract

These notes formalize likelihood-based learning as the backbone of modern generative modeling, and develop the latent-variable viewpoint that makes *approximate inference* unavoidable. We begin by casting learning as density estimation and showing that maximum likelihood is equivalent to minimizing $\mathrm{KL}(P_{\text{data}} \| P_{\boldsymbol{\theta}})$, together with an information-theoretic interpretation of this KL as compression loss. We then review empirical maximum likelihood, Monte Carlo estimation, and the decomposition of log-likelihood for autoregressive and conditional models, together with stochastic gradient training and generalization considerations.

1 Introduction

Lecture 1 framed constrained-aware generation through a target distribution that combines a base model with soft potentials and hard feasible sets. In that framework, one does not only *train* a model but also *conditions* and *steers* it at inference time. This lecture provides the probabilistic substrate behind that view.

The main claim is that likelihood-based learning naturally leads to (i) KL-based notions of closeness between the learned model and the data distribution, and (ii) posterior inference problems once latent variables are introduced. These two points are inseparable from constraint injection: constraints typically appear as additional factors that make posteriors intractable, forcing us to approximate inference with gradients and stochastic estimators.

2 Learning as density estimation and KL divergence

We begin with the density estimation viewpoint: learning is posed as approximating an unknown data-generating distribution with a tractable parametric model. Let $\mathcal{X}$ denote the sample space, for instance $\mathcal{X} = \{0, 1\}$ for a binary outcome, $\mathcal{X} = \mathbb{R}^d$ for continuous vectors, or $\mathcal{X} = \mathcal{V}^T$ for sequences of tokens from a vocabulary $\mathcal{V}$.

We assume data is generated from an unknown distribution P_{data} over $\mathcal{X}$. When $\mathcal{X}$ is continuous, we implicitly assume P_{data} admits a density p_{data} with respect to a reference measure (typically Lebesgue), and we write $\log P_{\text{data}}(\mathbf{x})$ as shorthand for $\log p_{\text{data}}(\mathbf{x})$. Similarly, a model $P_{\boldsymbol{\theta}}$ has density $p_{\boldsymbol{\theta}}$ and we use $P_{\boldsymbol{\theta}}(\mathbf{x})$ and $p_{\boldsymbol{\theta}}(\mathbf{x})$ interchangeably when the meaning is clear. The parameter $\boldsymbol{\theta} \in \Theta$ indexes the model class.

We observe a dataset $\mathcal{D} = \{\mathbf{x}^{(i)}\}_{i=1}^m$ of i.i.d. samples from P_{data}. The i.i.d. assumption is an idealization; in sequential control or time series, observations are correlated, and one typically works with conditional models or trajectory distributions. For the present section, the i.i.d. abstraction is convenient because it cleanly connects expected objectives to empirical ones.

We choose a model family $\mathcal{M} = \{P_{\boldsymbol{\theta}} : \boldsymbol{\theta} \in \Theta\}$ and seek a "good" approximation to P_{data}. The operational reasons for learning a distribution are threefold. First, a learned model supports generation by sampling $\mathbf{x} \sim P_{\boldsymbol{\theta}}$, which is the primitive required by most generative pipelines. Second, it supports density evaluation, which enables tasks such as anomaly detection, rejection sampling, or risk scoring through the magnitude of $P_{\boldsymbol{\theta}}(\mathbf{x})$. Third, it enables representation learning because internal features used to parameterize $P_{\boldsymbol{\theta}}$ can serve as task-relevant summaries of $\mathbf{x}$. To formalize what it means to be "good," we need a notion of mismatch between distributions.

2.1 KL divergence as a measure of mismatch

Definition 1 (Kullback–Leibler divergence)**.** *Let P and Q be distributions over $\mathcal{X}$ such that P is absolutely continuous with respect to Q (equivalently, $Q(x) = 0 \Rightarrow P(x) = 0$ in the discrete case). The KL divergence from P to Q is*

$$\mathrm{KL}(P\|Q) \;\coloneqq\; \mathbb{E}_{\mathbf{x}\sim P}\left[\log \frac{P(\mathbf{x})}{Q(\mathbf{x})}\right]. \tag{1}$$

In the discrete case this is $\sum_{x\in\mathcal{X}} P(x) \log \frac{P(x)}{Q(x)}$; in the continuous case it is the corresponding integral over densities.

The absolute continuity condition is not a technicality. If there exists an event that occurs under P but has zero probability under Q, then $\log \frac{P(\mathbf{x})}{Q(\mathbf{x})}$ becomes infinite on that event and $\mathrm{KL}(P\|Q) = +\infty$. In learning, this captures a concrete failure mode: if a model assigns zero (or numerically negligible) probability to observations that do occur, the mismatch is catastrophic.

KL divergence satisfies two key properties that will matter throughout the course.

1. First, it is nonnegative, and $\mathrm{KL}(P\|Q) = 0$ if and only if $P = Q$ almost surely. A standard proof uses Gibbs' inequality: since log is concave:

$$\mathbb{E}_P[\log(Q(\mathbf{x})/P(\mathbf{x}))] \le \log(\mathbb{E}_P[Q(\mathbf{x})/P(\mathbf{x})]) = \log(1) = 0,$$

   hence (1) is always ≥ 0.

2. Second, it is asymmetric in general: $\mathrm{KL}(P\|Q) \neq \mathrm{KL}(Q\|P)$. This asymmetry implies that the direction of KL matters for which errors are penalized more heavily (mode covering versus mode seeking), which later becomes crucial when constraints shape the target distribution.

This provides an intuitive bridge from "fit a model" to "compress the data" and explains why log-likelihood, rather than raw likelihood, is the natural objective.

It is also helpful to relate KL to cross-entropy. Define the (Shannon) entropy of P as $H(P) \coloneqq -\mathbb{E}_P[\log P(\mathbf{x})]$ and the cross-entropy as $H(P, Q) \coloneqq -\mathbb{E}_P[\log Q(\mathbf{x})]$. Then

$$\mathrm{KL}(P\|Q) \;=\; H(P, Q) \;-\; H(P). \tag{2}$$

Since $H(P)$ does not depend on Q, minimizing KL over Q is equivalent to minimizing the cross-entropy $H(P, Q)$, or equivalently maximizing $\mathbb{E}_P[\log Q(\mathbf{x})]$. *This identity is the mathematical reason maximum likelihood appears as the canonical training objective for normalized models.*

2.2 Expected log-likelihood and maximum likelihood

We now connect the mismatch criterion to a learnable objective. Fix a model $P_{\boldsymbol{\theta}}$ and consider the mismatch $\mathrm{KL}(P_{\text{data}}\|P_{\boldsymbol{\theta}})$. Expanding Definition 1 yields

$$\mathrm{KL}(P_{\text{data}}\|P_{\boldsymbol{\theta}}) = \mathbb{E}_{\mathbf{x}\sim P_{\text{data}}}[\log P_{\text{data}}(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim P_{\text{data}}}[\log P_{\boldsymbol{\theta}}(\mathbf{x})]. \tag{3}$$

The first term is $-H(P_{\text{data}})$ and does not depend on $\boldsymbol{\theta}$. Therefore, minimizing KL divergence is equivalent to maximizing the expected log-likelihood of the model under the data distribution:

$$\arg\min_{\boldsymbol{\theta}} \mathrm{KL}(P_{\text{data}} \| P_{\boldsymbol{\theta}}) = \arg\max_{\boldsymbol{\theta}} \ \mathbb{E}_{\mathbf{x}\sim P_{\text{data}}}[\log P_{\boldsymbol{\theta}}(\mathbf{x})]. \tag{4}$$

This equivalence is the conceptual link between the density estimation goal and maximum likelihood estimation. The information-theoretic interpretation is that minimizing KL corresponds to finding a model $P_{\boldsymbol{\theta}}$ that compresses samples from P_{data} as much as possible.

Note that, in practice, P_{data} is unknown, thus we replace the expectation in (4) with its empirical approximation over $\mathcal{D}$:

$$\max_{\boldsymbol{\theta}} \ \frac{1}{|\mathcal{D}|} \sum_{\mathbf{x}\in\mathcal{D}} \log P_{\boldsymbol{\theta}}(\mathbf{x}). \tag{5}$$

This is the maximum likelihood estimator (MLE). Under mild regularity conditions and for sufficiently expressive model families, MLE is statistically consistent: as $m \to \infty$, the empirical objective concentrates around the population objective in (4). In finite data regimes, the gap between the empirical and population objectives is the source of overfitting, and it motivates regularization, early stopping, and architectural inductive bias.

The logarithm plays two roles that become important later in constrained inference. First, it turns products into sums, which yields additive objectives and stable gradients. Second, it heavily penalizes assigning very small probability to observed samples, because $\log P_{\boldsymbol{\theta}}(\mathbf{x}) \to -\infty$ as $P_{\boldsymbol{\theta}}(\mathbf{x}) \to 0$. In density estimation language, MLE discourages "holes" in the learned distribution at observed locations. In constraint-aware generation language, it discourages base models whose support is misaligned with feasible regions, because such misalignment makes posterior reweighting or projection much more difficult.

Finally, note how the objective will change once we begin *injecting constraints*. In the density-estimation setting, we learn $\boldsymbol{\theta}$ by bringing $P_{\boldsymbol{\theta}}$ close to P_{data} in the forward direction $\mathrm{KL}(P_{\text{data}} \| P_{\boldsymbol{\theta}})$, which yields maximum likelihood. In the constrained-generation setting from Lecture 1, the goal is instead to generate from a *constraint-tilted* target distribution $\pi(\mathbf{x} \mid \mathbf{c}) \propto P_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{c}) \exp(-\lambda\phi(\mathbf{x}, \mathbf{c}))\, \mathbf{1}\{\mathbf{x} \in \mathcal{C}(\mathbf{c})\}$, whose normalization constant is typically intractable. Thus the computational problem becomes approximate inference under π, most commonly posed as selecting an auxiliary distribution q (over $\mathbf{x}$ alone or over $(\mathbf{x}, \mathbf{z})$ when latent variables are present) by minimizing $\mathrm{KL}(q\|\pi)$ within a tractable family, as we will see later throughout the course.

Example 1 (MLE for a Bernoulli model)**.** *Consider the following example. Let $\mathcal{X} = \{H, T\}$ and consider a Bernoulli model $P_{\boldsymbol{\theta}}(H) = \theta$ and $P_{\boldsymbol{\theta}}(T) = 1 - \theta$. Given a dataset $\mathcal{D}$ of m coin flips with n_H heads, the log-likelihood is*

$$\sum_{i=1}^{m} \log P_{\boldsymbol{\theta}}(x^{(i)}) = n_H \log\theta + (m - n_H)\log(1-\theta).$$

The maximizer is $\hat{\theta} = n_H/m$. In the running example in **?**, *$n_H = 3$ and $m = 5$, so $\hat{\theta} = 0.6$.*

The coin example is intentionally simple: it isolates the idea that MLE selects parameters that make the observed dataset probable under the model. Once $P_{\boldsymbol{\theta}}$ is parameterized by deep networks, the optimization is no longer closed-form, but the principle and the KL interpretation remain unchanged. Indeed, in most deep generative models, the MLE objective (5) is nonconvex in $\boldsymbol{\theta}$. Nevertheless, training proceeds by gradient-based optimization:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha_t \nabla_{\boldsymbol{\theta}} \left(\sum_{\mathbf{x}\in\mathcal{B}_t} \log P_{\boldsymbol{\theta}}(\mathbf{x}) \right), \tag{6}$$

where $\mathcal{B}_t \subseteq \mathcal{D}$ is a minibatch. The computational question becomes whether $\log P_{\boldsymbol{\theta}}(\mathbf{x})$ and its gradient can be computed efficiently. This computational tractability, more than the abstract MLE principle itself, is what differentiates model classes (e.g., autoregressive flows versus GANs versus Energy based models, etc.).

3 Monte Carlo estimation

Many quantities of interest in learning and inference can be written as expectations, yet closed-form evaluation is often impossible. Monte Carlo estimation is the default computational primitive for approximating such expectations. The generic recipe is: given $\mathbb{E}_{x\sim P}[g(x)]$, sample $x_1, \ldots, x_T \sim P$, and estimate with $\hat{g} = \frac{1}{T}\sum_{t=1}^{T} g(x_t)$. Interestingly, this approach has 3 key properties:

1. First, the resulting estimator $\hat{g}$ is unbiased: $\mathbb{E}[\hat{g}] = \mathbb{E}_{x\sim P}[g(x)]$.
2. Second, it converges by the law of large numbers: $\hat{g} \to \mathbb{E}_{x\sim P}[g(x)]$ almost surely as $T \to \infty$.
3. Third, its variance decreases as $1/T$ under mild assumptions. That is, if $\sigma^2 := \mathrm{Var}_{x\sim P}(g(x)) < \infty$, then $\mathrm{Var}(\hat{g}) = \sigma^2/T$ and (by the central limit theorem) $\sqrt{T}(\hat{g} - \mathbb{E}[g])$ is approximately Gaussian for large T.

This scaling is the reason Monte Carlo is the default computational primitive in modern generative modeling: it is dimension-agnostic, but its accuracy is controlled by sample size and variance. A recurring practical theme is therefore *variance management*, either by choosing g carefully, by reparameterizing the expectation (pathwise gradients), or by introducing control variates.

It is also important to separate two roles of Monte Carlo that will appear throughout the course. First, Monte Carlo approximates *objectives* that are expectations, for example $\mathbb{E}[\log P_{\boldsymbol{\theta}}(x)]$ or $\mathbb{E}_{q_{\boldsymbol{\phi}}}[\log p_{\boldsymbol{\theta}}(x \mid z)]$. Second, Monte Carlo approximates *gradients* of such objectives, producing stochastic gradient estimators that enable SGD and its variants. In both cases, randomness enters as a computational device rather than as part of the modeling assumptions.

The course will reuse this primitive repeatedly. In diffusion models, Monte Carlo appears in score matching objectives and in the discretization of SDE samplers. In latent-variable models, Monte Carlo appears in ELBO gradients and importance sampling bounds. Finally, in constrained inference, Monte Carlo appears when constraints are enforced through stochastic projection, rejection, or randomized proximal operators.

4 Maximum likelihood for AR and conditional models

4.1 AR factorization and log-likelihood decomposition

Autoregressive (AR) models are the most widely used *tractable* likelihood family because they reduce a high-dimensional density into a product of one-step conditional distributions via the chain rule of probability. Let $\mathbf{x} = (x_1, \ldots, x_n)$ be a vector or sequence, where the ordering $1, \ldots, n$ is chosen by the modeler. In language modeling, x_i is the i-th token; in images, x_i might be a pixel (under a raster scan order); in trajectories, x_i might be a state-action pair at time i after an appropriate flattening. The chain rule states that for any joint distribution over $\mathbf{x}$,

$$P(\mathbf{x}) = \prod_{i=1}^{n} P(x_i \mid x_{<i}), \qquad x_{<i} := (x_1, \ldots, x_{i-1}). \tag{7}$$

Autoregressive *modeling* corresponds to choosing a parameterization $P_{\boldsymbol{\theta}}(x_i \mid x_{<i})$ for each conditional, yielding $P_{\boldsymbol{\theta}}(\mathbf{x}) = \prod_{i=1}^{n} P_{\boldsymbol{\theta}}(x_i \mid x_{<i})$. Tractability here follows because evaluating $P_{\boldsymbol{\theta}}(\mathbf{x})$

only requires evaluating n conditionals, each of which is normalized in its own output space (for instance, a softmax over a vocabulary, or a Gaussian density in $\mathbb{R}^d$).

Taking logs converts the product into a sum,

$$\log P_{\boldsymbol{\theta}}(\mathbf{x}) = \sum_{i=1}^{n} \log P_{\boldsymbol{\theta}}(x_i \mid x_{<i}), \tag{8}$$

and this decomposition explains why AR models are natural for MLE: *maximizing the joint log-likelihood becomes maximizing a sum of local predictive log-likelihoods.* In a discrete setting, each term is a log-softmax score; in a continuous setting, each term is typically a Gaussian (or mixture) log-density. For language models, (8) is the token-level cross-entropy objective used in pretraining.

Example 2 (Language modeling)**.** *Let $\mathbf{x} = (x_1, \ldots, x_n)$ be a sentence with tokens $x_i \in \mathcal{V}$. A Transformer parameterizes $P_{\boldsymbol{\theta}}(x_i \mid x_{<i})$ by mapping the prefix $x_{<i}$ to logits $\mathbf{h}_i \in \mathbb{R}^{|\mathcal{V}|}$ and then applying a softmax. The contribution of position i to the log-likelihood is $\log P_{\boldsymbol{\theta}}(x_i \mid x_{<i}) = \mathbf{h}_i[x_i] - \log \sum_{v \in \mathcal{V}} \exp(\mathbf{h}_i[v])$, so the joint likelihood is the sum of these per-token terms.*

Example 3 (Trajectory as an AR model)**.** *Let $\mathbf{x}$ represent a trajectory $\mathbf{x} = (s_0, a_0, s_1, a_1, \ldots, s_T)$, flattened into a length-n sequence. An autoregressive policy model can represent $P_{\boldsymbol{\theta}}(\mathbf{x})$ by predicting each next element conditioned on the past, for example $P_{\boldsymbol{\theta}}(a_t \mid s_{\le t}, a_{<t})$ and $P_{\boldsymbol{\theta}}(s_{t+1} \mid s_{\le t}, a_{\le t})$. Even when one ultimately cares about $P_{\boldsymbol{\theta}}(a_{0:T-1} \mid s_0, \mathit{goal})$, this AR factorization provides a likelihood that is easy to train and evaluate, which later enables constraint-aware inference through reweighting, guidance, or projection.*

On a dataset $\mathcal{D} = \{\mathbf{x}^{(j)}\}_{j=1}^{m}$, the empirical log-likelihood is

$$\ell(\boldsymbol{\theta}) \;\coloneqq\; \sum_{j=1}^{m} \sum_{i=1}^{n} \log P_{\boldsymbol{\theta}}(x_i^{(j)} \mid x_{<i}^{(j)}). \tag{9}$$

If each conditional $P_{\boldsymbol{\theta}}(x_i \mid x_{<i})$ had its own independent parameter block, then (9) would decouple across i and could be optimized as n separate problems. This is rarely desirable: it would prevent generalization across positions, and it would scale parameters linearly with n. Modern AR models therefore rely on heavy parameter sharing, for example a single Transformer with causal masking for all positions, or a convolutional architecture. From the optimization standpoint, this means $\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})$ aggregates gradient contributions from all positions and all examples through the shared network, which is exactly what backpropagation computes efficiently.

Remark 1. *A subtle but important point is that AR tractability is asymmetric with respect to* evaluation *versus* sampling*. Evaluation of $P_{\boldsymbol{\theta}}(\mathbf{x})$ is parallelizable across positions once the model has computed hidden states, but sampling is inherently sequential because generating x_i requires having already generated $x_{<i}$. This asymmetry foreshadows later lectures: constraints that refer to global properties of $\mathbf{x}$ are difficult to enforce when sampling proceeds token-by-token, which is one reason alternative generative families (diffusion, flow matching) can be advantageous for global control.*

4.2 Stochastic gradients for large datasets

When m is large, evaluating the full gradient of (9) is expensive because it requires a full pass over the dataset. Stochastic gradient methods approximate the population objective by sampling a minibatch. Formally, define the per-example log-likelihood $\ell_j(\boldsymbol{\theta}) \coloneqq \sum_{i=1}^{n} \log P_{\boldsymbol{\theta}}(x_i^{(j)} \mid x_{<i}^{(j)})$, so that $\ell(\boldsymbol{\theta}) = \sum_{j=1}^{m} \ell_j(\boldsymbol{\theta})$. If $B \subset \{1, \ldots, m\}$ is a minibatch sampled uniformly without replacement, then

$$\widehat{\nabla \ell}(\boldsymbol{\theta}) \;\coloneqq\; \frac{m}{|B|} \sum_{j \in B} \nabla_{\boldsymbol{\theta}} \ell_j(\boldsymbol{\theta}) \tag{10}$$

is an unbiased estimator of the full gradient $\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta})$. Note that this is exactly Monte Carlo estimation with the dataset-induced empirical distribution: the random variable is the index j, and $g(j) = \nabla_{\boldsymbol{\theta}} \ell_j(\boldsymbol{\theta})$.

4.3 Conditional modeling as a proxy for control

In constrained-aware generation, we nearly always model conditionals. Let $\mathbf{c}$ represent context, a goal, a prompt, or an observation history, and let $\mathbf{y}$ represent the object we wish to generate (a completion, an action sequence, a design, or a plan). A conditional model specifies $P_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{c})$ and is trained by minimizing the conditional negative log-likelihood

$$\min_{\boldsymbol{\theta}} \ \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{c},\mathbf{y}) \in \mathcal{D}} - \log P_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{c}). \tag{11}$$

This objective is the conditional analogue of the KL minimization in the previous section. In particular, minimizing (11) corresponds (in the population limit) to minimizing $\mathrm{KL}(P_{\mathrm{data}}(\mathbf{y} \mid \mathbf{c}) \,\|\, P_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{c}))$ averaged over $\mathbf{c}$. If the task only requires predicting $\mathbf{y}$ from $\mathbf{c}$, then learning the conditional suffices and one need not model the full joint distribution $P(\mathbf{c}, \mathbf{y})$.

The connection to control is that many control objectives can be cast as conditional generation. If $\mathbf{c}$ encodes the available information at decision time (state, observations, goal), then sampling $\mathbf{y}$ from $P_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{c})$ produces candidate decisions. This is why conditional density estimation is often described as a probabilistic formulation of behavioral cloning or imitation learning.

Example 4 (Trajectory generation as conditional density estimation)**.** *Let $\mathbf{y} = (u_1, \ldots, u_T)$ be an action sequence and $\mathbf{c}$ include an initial observation o_0, a goal description g, and possibly a coarse plan or map information. A conditional model $P_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{c})$ is a* generative policy*: sampling from it yields candidate control sequences. If the model is autoregressive, it factorizes as $P_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{c}) = \prod_{t=1}^{T} P_{\boldsymbol{\theta}}(u_t \mid u_{<t}, \mathbf{c})$, so training decomposes into a sum of per-time-step prediction losses, while generation proceeds sequentially. In later lectures, feasibility constraints (collision avoidance, terminal sets, resource budgets) will be represented as additional factors that reshape this conditional distribution at inference time, which raises the central tension: constraints are often global in $\mathbf{y}$, whereas AR sampling is local in t.*

Example 5 (Conditional generation in language as "control" by prompting)**.** *In an LLM, $\mathbf{c}$ is the prompt and $\mathbf{y}$ is the continuation. The conditional AR factorization is $P_{\boldsymbol{\theta}}(\mathbf{y} \mid \mathbf{c}) = \prod_{i=1}^{|\mathbf{y}|} P_{\boldsymbol{\theta}}(y_i \mid \mathbf{c}, y_{<i})$. Prompting, system instructions, and in-context examples are forms of specifying $\mathbf{c}$. Hard constraints, however, such as requiring that certain facts be correct or that a generated plan be feasible under external rules, generally cannot be guaranteed by conditioning alone, which motivates the constraint-injection mechanisms developed later in the course.*

5 Generalization, overfitting, and the role of restrictions

The objective studied so far is maximum likelihood, equivalently the minimization of the forward $\mathrm{KL}(P_{\mathrm{data}} \| P_{\boldsymbol{\theta}})$. This objective is defined at the population level through an expectation under P_{data}, but in practice we optimize its empirical approximation over a finite dataset $\mathcal{D}$.

Generalization refers to the gap between these two quantities: a model generalizes if its empirical log-likelihood is a good proxy for its expected log-likelihood under fresh draws from P_{data}. **Overfitting** occurs when the model achieves high likelihood on $\mathcal{D}$ while assigning substantially lower likelihood to unseen samples from the same source.

Maximum likelihood can overfit when the model class is too expressive relative to the dataset size. A useful way to state this is that the hypothesis space $\mathcal{M} = \{P_{\boldsymbol{\theta}}\}$ can have enough degrees

of freedom that many distributions fit the finite sample well. In the extreme limit, the model can memorize: for discrete domains one can assign essentially all probability mass to the training set and negligible mass elsewhere. Such a model obtains near-optimal empirical likelihood while being useless for generation or inference because it fails to place probability on valid but unseen outcomes. Even in continuous domains, expressive models can behave similarly by concentrating density in narrow regions around training points, yielding high training likelihood but poor robustness.

The reason **restrictions** help is that they reduce the space of distributions compatible with the training data. From the statistical viewpoint, restricting the hypothesis class reduces variance and improves concentration of empirical estimates around their expectations. From the modeling viewpoint, restrictions encode inductive bias: they express assumptions about smoothness, invariances, compositional structure, or locality that are not explicit in the likelihood objective but are essential for learning from finite data.

The classical bias-variance tradeoff summarizes the tension. If the hypothesis space is too small, even infinite data may not allow $P_{\boldsymbol{\theta}}$ to approximate P_{data} well; this manifests as *approximation error* or bias. If the hypothesis space is too large, finite datasets admit many parameter values with similar training likelihood but different population likelihood; this manifests as variance. While bias-variance is most familiar in supervised regression, the same idea applies here if we interpret each conditional term in an autoregressive model as a prediction problem and the overall negative log-likelihood as an average prediction loss.

Example 6 (Parameter sharing as a restriction in autoregressive models)**.** *In an AR model, if each conditional $P_{\boldsymbol{\theta}}(x_i \mid x_{<i})$ had independent parameters, one could fit each position nearly independently, which increases variance and encourages memorization of position-specific traits. A Transformer imposes strong sharing across positions and across examples, forcing the model to reuse features and patterns. This restriction is an inductive bias that often improves generalization even though it does not change the formal MLE objective.*

In practice, restrictions are implemented through architectural choices (weight sharing, limited depth or width, convolutional locality, attention patterns), explicit regularization (weight decay, dropout, spectral norms), and implicit regularization induced by optimization (early stopping, learning-rate schedules).

In constrained-aware modeling, these considerations have an additional interpretation that will recur in later lectures. Constraints typically reshape the distribution at inference time, yielding a target $\pi(\mathbf{x} \mid \mathbf{c})$ that can differ substantially from the training distribution. If $P_{\boldsymbol{\theta}}$ has overfit to features of $\mathcal{D}$, then small changes induced by constraints can push inference into regions where the model has not learned a stable likelihood geometry, causing brittle behavior and unreliable gradients. Regularization and architectural restrictions therefore do not only prevent overfitting in the classical sense; they also improve the smoothness and robustness of the learned generator, which is essential when constraint injection induces distribution shift, reweighting, or projection during inference.

6 Looking ahead

Lecture 3 will build on this foundation by comparing latent-variable models (VAEs) to implicit models (GANs) and by highlighting which training objectives do and do not yield tractable likelihoods. From the course perspective, the enduring message from Lecture 2 is that constraint-aware generation lives inside approximate inference, and approximate inference is built from KL identities, Monte Carlo estimators, and differentiable computations.

References

D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning (ICML)*, 2014.

D. P. Kingma and M. Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations (ICLR)*, 2014.

R. Ranganath, S. Gerrish, and D. M. Blei. Black box variational inference. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2014.

Y. Burda, R. Grosse, and R. Salakhutdinov. Importance weighted autoencoders. In *International Conference on Learning Representations (ICLR)*, 2015.

D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference: a review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877, 2017.