Course Materials for GEN-AI

Northeastern University

These materials have been prepared and sourced for the course **GEN-AI** at Northeastern University. Every effort has been made to provide proper citations and credit for all referenced works.

If you believe any material has been inadequately cited or requires correction, please contact me at:

Instructor: Ramin Mohammadi
r.mohammadi@northeastern.edu

Thank you for your understanding and collaboration.

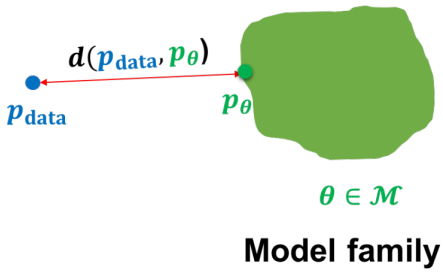
Content sourced from multiple references - see Reference section for details

Maximum Likelihood Learning

Learning a Generative Model

We are given a training set of examples, e.g., images of dogs.

$\mathbf{x}^{(j)} \sim p_{\text{data}}$
$j = 1, 2, \ldots, |\mathcal{D}|$



We want to learn a probability distribution $p(x)$ over images x such that:

- **Generation**: If we sample $x_{\text{new}} \sim p(x)$, x_{new} should look like a dog (sampling).
- **Density estimation**: $p(x)$ should be high if x looks like a dog, and low otherwise (anomaly detection).
- **Unsupervised representation learning**: We should be able to learn what these images have in common, e.g., ears, tail, etc. (features).

First Questions

1. How to represent $p_\theta(x)$?
2. How to learn it?

Setting

Let us assume that the domain is governed by some underlying distribution P_{data}.

We are given a dataset D of m samples from P_{data}.

- Each sample is an assignment of values to (a subset of) the variables, e.g., $(X_{\text{bank}} = 1, X_{\text{dollar}} = 0, \ldots, Y = 1)$ or pixel intensities.

The standard assumption is that the data instances are independent and identically distributed (IID).

We are also given a family of models $\mathcal{M}$, and our task is to learn some "good"model $\hat{M} \in \mathcal{M}$ (i.e., in this family) that defines a distribution $p_{\hat{M}}$.

- For example, all Bayes nets with a given graph structure, for all possible choices of the CPD tables.
- For example, a FVSBN for all possible choices of the logistic regression parameters. $\mathcal{M} = \{P_\theta, \theta \in \Theta\}$, θ = concatenation of all logistic regression coefficients.

The question here is how we can learn these unknown parameters when the only thing we have access to are some sampled data.

Goal of Learning

The goal of learning is to return a model P_θ that precisely captures the distribution P_{data} from which our data was sampled.

This is in general not achievable because of:

- Limited data only provides a rough approximation of the true underlying distribution — We don't have the actual P_{data}.
- Computational reasons.

Example Suppose we represent each image with a vector X of 784 binary variables (black vs. white pixel). How many possible states (= possible images) in the model? $2^{784} \approx 10^{236}$, which is way more than the number of atoms in the universe. Even 10^7 training examples provide extremely sparse coverage!

- We want to select P_θ to construct the "best"approximation to the underlying distribution P_{data}.
- What is "best"?

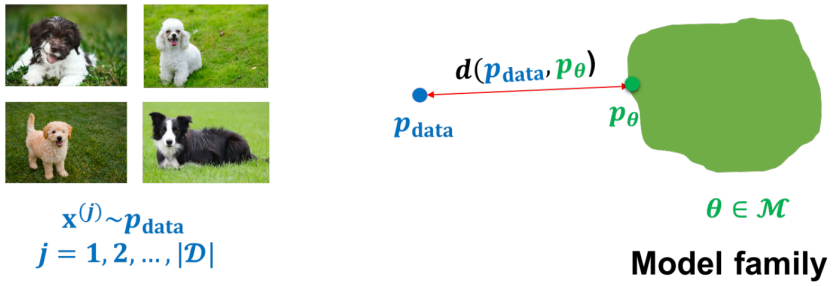
What is "best"?

This depends on what we want to do:

1. **Density estimation**: We are interested in the full distribution (so later we can compute whatever conditional probabilities we want).
2. **Specific prediction tasks**: We are using the distribution to make a prediction.
   - Is this email spam or not?
   - Predict next frame in a video.
3. **Structure or knowledge discovery**: We are interested in the model itself.
   - How do some genes interact with each other?
   - What causes cancer?
   - Take IE-7374 course?

Learning as Density Estimation

We want to learn the full distribution so that later we can answer any probabilistic inference query. In this setting, we can view the learning problem as density estimation.
We want to construct P_θ as "close"as possible to P_{data} (recall we assume we are given a dataset D of samples from P_{data}).



How do we evaluate "closeness"? There are different kinds of methods we can use to measure similarities, and we will get a different kind of generative models by changing the way we measure these similarities.

Divergence Measures

(e.g., Comparing Probability Distributions)

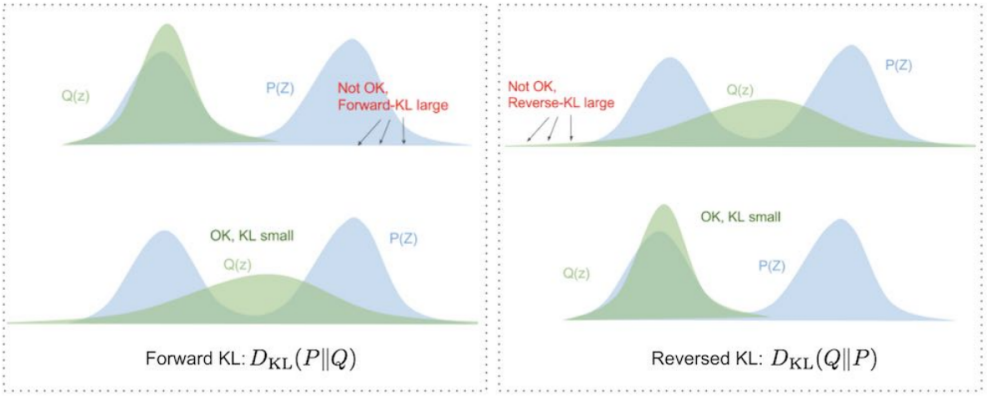
Kullback-Leibler (KL) Divergence

$$D_{KL}(P \parallel Q) = \sum_x P(x) \log \left(\frac{P(x)}{Q(x)} \right) \tag{1}$$

- $D(p \parallel q) \geq 0$ for all p, q, with equality if and only if $p = q$. Proof:

$$E_{x \sim p} \left[-\log \frac{q(x)}{p(x)} \right] \geq -\log \left(E_{x \sim p} \left[\frac{q(x)}{p(x)} \right] \right) = -\log \left(\sum_x p(x) \frac{q(x)}{p(x)} \right) = 0$$

- Notice that KL-divergence is asymmetric, i.e., $D(p \parallel q) \neq D(q \parallel p)$.
- Measures the expected number of extra bits required to describe samples from $p(x)$ using a compression code based on q instead of p.



Detour on KL-Divergence

To compress, it is useful to know the probability distribution the data is sampled from.

For example:

- Let $X_1, \cdots, X_{100}$ be samples of an unbiased coin. Roughly 50 heads and 50 tails. Optimal compression scheme is to record heads as 0 and tails as 1. In expectation, use 1 bit per sample, and cannot do better.
- Suppose the coin is biased, and $P[H] \gg P[T]$. Then it is more efficient to use fewer bits on average to represent heads and more bits to represent tails, e.g.,
  - Batch multiple samples together.
  - Use a short sequence of bits to encode HHHH (common) and a long sequence for TTTT (rare).
  - Like Morse code: E = •, A = •–, Q = –•–. In Morse code, we assign a shorter sequence to vowels to get a shorter representation as they are more likely to repeat.

KL-divergence: If your data comes from p, but you use a scheme optimized for q, the divergence $D_{KL}(p \parallel q)$ is the number of extra bits you will need on average.

So if you are trying to optimize for KL-divergence, you are equivalently trying to optimize for compression. In other terms, you are trying to build a model that can compress the data really well (learn it).

Expected Log-Likelihood

We can simplify this somewhat:

$$D(P_{\text{data}} \parallel P_\theta) = E_{x\sim P_{\text{data}}}\left[\log\left(\frac{P_{\text{data}}(x)}{P_\theta(x)}\right)\right] = E_{x\sim P_{\text{data}}}[\log P_{\text{data}}(x)] - E_{x\sim P_{\text{data}}}[\log P_\theta(x)].$$

- The first term does not depend on P_θ (so for the purpose of learning, we can ignore it as it is a constant).
- Then, minimizing KL divergence is equivalent to maximizing the expected log-likelihood:

$$\arg\min_{P_\theta} D(P_{\text{data}} \parallel P_\theta) = \arg\min_{P_\theta} -E_{x\sim P_{\text{data}}}[\log P_\theta(x)] = \arg\max_{P_\theta} E_{x\sim P_{\text{data}}}[\log P_\theta(x)].$$

- This asks that P_θ assign high probability to instances sampled from P_{data}, so as to reflect the true distribution.
- Because of the log, samples x where $P_\theta(x) \approx 0$ (wrong answers) weigh heavily in the objective.
- Although we can now compare models, since we are ignoring $H(P_{\text{data}}) = -E_{x\sim P_{\text{data}}}[\log P_{\text{data}}(x)]$, we don't know how close we are to the optimum.

Problems:

In general, we do not know P_{data}. So we can approximate the expected log-likelihood:

$$E_{x\sim P_{\text{data}}}[\log P_\theta(x)] \approx \frac{1}{|D|}\sum_{x\in D}\log P_\theta(x). \tag{2}$$

Maximum likelihood learning is then:

$$\max_{P_\theta}\frac{1}{|D|}\sum_{x\in D}\log P_\theta(x). \tag{3}$$

Equivalently, maximize likelihood of the data:

$$P_\theta(x^{(1)},\ldots,x^{(m)}) = \prod_{x\in D} P_\theta(x). \tag{4}$$

Why does this work? This is basically a Monte Carlo estimate.

Main Idea in Monte Carlo Estimation

The idea is that if you have an expectation of some function over a random variable x, you can approximate it by all the true/real things that could happen (x) weighted by a probability $P(x)$ using:

$$E_{x\sim P}[g(x)] = \sum_x g(x)P(x). \tag{5}$$

Alternatively, you can solve it by sampling as:

1. Generate T samples $x_1,\ldots,x_T$ from the distribution P with respect to which the expectation was taken.
2. Estimate the expected value from the samples using:

$$\hat{g}(x_1,\ldots,x_T) := \frac{1}{T}\sum_{t=1}^{T} g(x_t). \tag{6}$$

3. Where $x_1,\ldots,x_T$ are independent samples from P. Note: $\hat{g}$ is a random variable. Why?

Properties of the Monte Carlo Estimate

Unbiased:

$$E_P[\hat{g}] = E_P[g(x)].$$

Convergence: By the law of large numbers

$$\hat{g} = \frac{1}{T}\sum_{t=1}^{T} g(x_t) \to E_P[g(x)] \quad \text{for } T \to \infty.$$

Variance:

$$\mathrm{Var}_P[\hat{g}] = \mathrm{Var}_P\left[\frac{1}{T}\sum_{t=1}^{T} g(x_t)\right] = \frac{\mathrm{Var}_P[g(x)]}{T}.$$

Thus, the variance of the estimator can be reduced by increasing the number of samples.

Extending the MLE principle to Autoregressive models

Given an autoregressive model with n variables and factorization

$$P_\theta(x) = \prod_{i=1}^{n} p_{\text{neural}}(x_i \mid x_{<i}; \theta_i),$$

where $\theta = (\theta_1, \cdots, \theta_n)$ are the parameters of all the conditionals.

Training data $D = \{x^{(1)}, \cdots, x^{(m)}\}$. Maximum likelihood estimate of the parameters?

- **Decomposition of Likelihood function:**

$$L(\theta, D) = \prod_{j=1}^{m} P_\theta(x^{(j)}) = \prod_{j=1}^{m}\prod_{i=1}^{n} p_{\text{neural}}(x_i^{(j)} \mid x_{<i}^{(j)}; \theta_i).$$

- **Goal:**

$$\arg\max_\theta L(\theta, D) = \arg\max_\theta \log L(\theta, D).$$

We no longer have a closed-form solution. So we need to solve it iteratively.

MLE Learning: Gradient Descent

$$L(\theta, D) = \prod_{j=1}^{m} P_\theta(x^{(j)}) = \prod_{j=1}^{m}\prod_{i=1}^{n} p_{\text{neural}}(x_i^{(j)} \mid x_{<i}^{(j)}; \theta_i)$$

Goal:

$$\arg\max_\theta L(\theta, D) = \arg\max_\theta \log L(\theta, D)$$

$$\ell(\theta) = \log L(\theta, D) = \sum_{j=1}^{m}\sum_{i=1}^{n} \log p_{\text{neural}}(x_i^{(j)} \mid x_{<i}^{(j)}; \theta_i)$$

1. Initialize θ_0 at random
2. Compute $\nabla_\theta \ell(\theta)$ (by backpropagation)
3. Update $\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta \ell(\theta)$

Note: Non-convex optimization problem, but often works well in practice.

MLE Learning: Stochastic Gradient Descent

$$\ell(\theta) = \log L(\theta, D) = \sum_{j=1}^{m} \sum_{i=1}^{n} \log p_{\text{neural}}(x_i^{(j)} \mid \text{pa}(x_i^{(j)}); \theta_i)$$

1. Initialize θ_0 at random
2. Compute $\nabla_\theta \ell(\theta)$ (by backpropagation)
3. Update $\theta_{t+1} = \theta_t + \alpha_t \nabla_\theta \ell(\theta)$

$$\nabla_\theta \ell(\theta) = \sum_{j=1}^{m} \sum_{i=1}^{n} \nabla_\theta \log p_{\text{neural}}(x_i^{(j)} \mid \text{pa}(x_i^{(j)}); \theta_i)$$

What if $m = |D|$ is huge?

$$\nabla_\theta \ell(\theta) = m E_{x^{(j)} \sim D} \left[\sum_{i=1}^{n} \nabla_\theta \log p_{\text{neural}}(x_i^{(j)} \mid \text{pa}(x_i^{(j)}); \theta_i) \right]$$

Monte Carlo: Sample $x^{(j)} \sim D$; approximate:

$$\nabla_\theta \ell(\theta) \approx m \sum_{i=1}^{n} \nabla_\theta \log p_{\text{neural}}(x_i^{(j)} \mid \text{pa}(x_i^{(j)}); \theta_i)$$

Empirical Risk and Overfitting

Empirical risk minimization can easily overfit the data.

- **Extreme example:** The data is the model (remember all training data).
- **Generalization:** The data is a sample, usually there is a vast amount of samples that you have never seen. Your model should generalize well to these "never-seen" samples.

Thus, we typically restrict the hypothesis space of distributions that we search over.

Bias-Variance Trade Off

If the hypothesis space is very limited, it might not be able to represent P_{data}, even with unlimited data. This type of limitation is called **bias**, as the learning is limited on how close it can approximate the target distribution.

If we select a highly expressive hypothesis class, we might represent the data better. However:

- When we have a small amount of data, multiple models can fit well, or even better than the true model.
- Small perturbations on D will result in very different estimates. This limitation is called **variance**.

Bias-Variance Decomposition

The total error for a model can be decomposed into three terms:

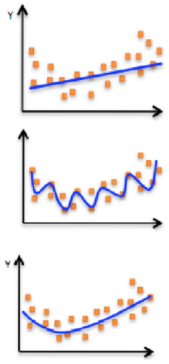
$$E[(\hat{f}(x) - f(x))^2] = \underbrace{(E[\hat{f}(x)] - f(x))^2}_{\text{Bias}^2} + \underbrace{E[(\hat{f}(x) - E[\hat{f}(x)])^2]}_{\text{Variance}} + \underbrace{\sigma^2}_{\text{Irreducible Error}}$$

Where:

- $\hat{f}(x)$: The prediction of the model.
- $f(x)$: The true function generating the data.
- σ^2: The variance of the noise in the data (irreducible error).

Trade-Off

- **High bias:** Simplistic models (e.g., linear models) underfit the data.
- **High variance:** Complex models (e.g., high-degree polynomials) overfit the data.
- **Optimal trade-off:** Choose a model complexity that balances bias and variance to minimize total error.



How to avoid overfitting?

- **Hard constraints:** For example, by selecting a less expressive hypothesis class:
  - Bayesian networks with at most d parents.
  - Smaller neural networks with fewer parameters.
  - Weight sharing.
- **Soft preference for "simpler" models:** Occam's Razor.
  - Augment the objective function with regularization:

$$\text{objective}(x, M) = \text{loss}(x, M) + R(M)$$

- Evaluate generalization performance on a held-out validation set.

References

[1] Stefano Ermon. *CS236*.