

Maximum Likelihood Learning

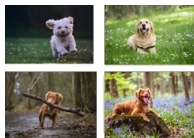
Stefano Ermon, Yang Song

Stanford University

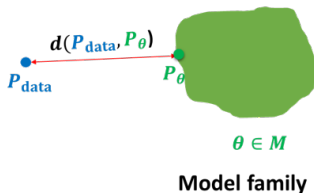
Lecture 4

Learning a generative model

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



$$x_i \sim P_{\text{data}} \\ i = 1, 2, \dots, n$$



- 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 question: how to represent $p_{\theta}(x)$. Second question: **how to learn it.**

- Lets assume that the domain is governed by some underlying distribution P_{data}
- We are given a dataset \mathcal{D} of m samples from P_{data}
 - Each sample is an assignment of values to (a subset of) the variables, e.g., $(X_{\text{bank}} = 1, X_{\text{dollar}} = 0, \dots, Y = 1)$ 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 $\hat{\mathcal{M}} \in \mathcal{M}$ (i.e., in this family) that defines a distribution $p_{\hat{\mathcal{M}}}$
 - 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

Goal of learning

- The goal of learning is to return a model $\hat{\mathcal{M}}$ 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
 - 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? $2^{784} \approx 10^{236}$. Even 10^7 training examples provide *extremely* sparse coverage!
- We want to select $\hat{\mathcal{M}}$ to construct the "best" approximation to the underlying distribution P_{data}
- What is "best"?

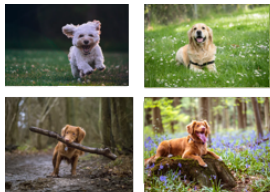
What is “best”?

This depends on what we want to do

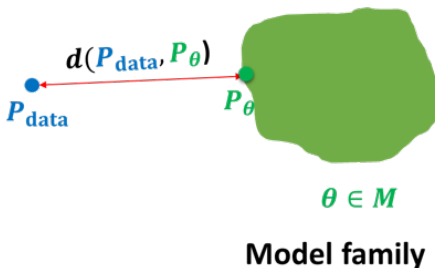
- 1 Density estimation: we are interested in the full distribution (so later we can compute whatever conditional probabilities we want)
- 2 Specific prediction tasks: we are using the distribution to make a prediction
 - Is this email spam or not?
 - Predict next frame in a video
- 3 Structure or knowledge discovery: we are interested in the model itself
 - How do some genes interact with each other?
 - What causes cancer?
 - Take CS 228

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 \mathcal{D} of samples from P_{data})



$$x_i \sim P_{\text{data}} \\ i = 1, 2, \dots, n$$



- How do we evaluate "closeness"?

KL-divergence

- How should we measure distance between distributions?
- The **Kullback-Leibler divergence** (KL-divergence) between two distributions p and q is defined as

$$D(p\|q) = \sum_x p(x) \log \frac{p(x)}{q(x)}.$$

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

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

- Notice that KL-divergence is **asymmetric**, i.e., $D(p\|q) \neq D(q\|p)$
- Measures the expected number of extra bits required to describe *samples from $p(x)$* using a 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, \dots, 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's 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: $E = \bullet$, $A = \bullet-$, $Q = - - \bullet-$
- KL-divergence: if your data comes from p , but you use a scheme optimized for q , the divergence $D_{KL}(p||q)$ is the number of *extra* bits you'll need on average

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 \mathcal{D} of samples from P_{data})
- How do we evaluate "closeness"?
- **KL-divergence** is one possibility:

$$D(P_{\text{data}}||P_\theta) = \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} \left[\log \left(\frac{P_{\text{data}}(\mathbf{x})}{P_\theta(\mathbf{x})} \right) \right] = \sum_{\mathbf{x}} P_{\text{data}}(\mathbf{x}) \log \frac{P_{\text{data}}(\mathbf{x})}{P_\theta(\mathbf{x})}$$

- $D(P_{\text{data}}||P_\theta) = 0$ iff the two distributions are the same.
- It measures the "compression loss" (in bits) of using P_θ instead of P_{data} .

Expected log-likelihood

- We can simplify this somewhat:

$$\begin{aligned} D(P_{\text{data}}||P_{\theta}) &= \mathbb{E}_{x \sim P_{\text{data}}} \left[\log \left(\frac{P_{\text{data}}(x)}{P_{\theta}(x)} \right) \right] \\ &= \mathbb{E}_{x \sim P_{\text{data}}} [\log P_{\text{data}}(x)] - \mathbb{E}_{x \sim P_{\text{data}}} [\log P_{\theta}(x)] \end{aligned}$$

- The first term does not depend on P_{θ} .
- Then, *minimizing* KL divergence is equivalent to *maximizing* the **expected log-likelihood**

$$\arg \min_{P_{\theta}} D(P_{\text{data}}||P_{\theta}) = \arg \min_{P_{\theta}} -\mathbb{E}_{x \sim P_{\text{data}}} [\log P_{\theta}(x)] = \arg \max_{P_{\theta}} \mathbb{E}_{x \sim P_{\text{data}}} [\log P_{\theta}(x)]$$

- Asks that P_{θ} assign high probability to instances sampled from P_{data} , so as to reflect the true distribution
- Because of log, samples x where $P_{\theta}(x) \approx 0$ weigh heavily in objective
- Although we can now compare models, since we are ignoring $H(P_{\text{data}})$, we don't know how close we are to the optimum
- Problem: In general we do not know P_{data} .

Maximum likelihood

- Approximate the expected log-likelihood

$$E_{x \sim P_{\text{data}}} [\log P_{\theta}(x)]$$

with the *empirical log-likelihood*:

$$E_{\mathcal{D}} [\log P_{\theta}(x)] = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \log P_{\theta}(x)$$

- **Maximum likelihood learning** is then:

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

- Equivalently, maximize likelihood of the data
 $P_{\theta}(x^{(1)}, \dots, x^{(m)}) = \prod_{x \in \mathcal{D}} P_{\theta}(x)$

Main idea in Monte Carlo Estimation

- 1 Express the quantity of interest as the expected value of a random variable.

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

- 2 Generate T samples x^1, \dots, x^T from the distribution P with respect to which the expectation was taken.
- 3 Estimate the expected value from the samples using:

$$\hat{g}(x^1, \dots, x^T) \triangleq \frac{1}{T} \sum_{t=1}^T g(x^t)$$

where x^1, \dots, 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 law of large numbers

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

- **Variance:**

$$V_P[\hat{g}] = V_P \left[\frac{1}{T} \sum_{t=1}^T g(x^t) \right] = \frac{V_P[g(x)]}{T}$$

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

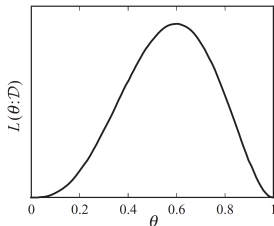
Single variable example: A biased coin

- Two outcomes: *heads* (H) and *tails* (T)
- Data set: Tosses of the biased coin, e.g., $\mathcal{D} = \{H, H, T, H, T\}$
- Assumption: the process is controlled by a probability distribution $P_{\text{data}}(x)$ where $x \in \{H, T\}$
- Class of models \mathcal{M} : all probability distributions over $x \in \{H, T\}$.
- Example learning task: How should we choose $P_{\theta}(x)$ from \mathcal{M} if 60 out of 100 tosses are heads in \mathcal{D} ?

MLE scoring for the coin example

We represent our model: $P_{\theta}(x = H) = \theta$ and $P_{\theta}(x = T) = 1 - \theta$

- Example data: $\mathcal{D} = \{H, H, T, H, T\}$
- Likelihood of data = $\prod_i P_{\theta}(x_i) = \theta \cdot \theta \cdot (1 - \theta) \cdot \theta \cdot (1 - \theta)$



- Optimize for θ which makes \mathcal{D} most likely. What is the solution in this case?

MLE scoring for the coin example: Analytical derivation

Distribution: $P_{\theta}(x = H) = \theta$ and $P_{\theta}(x = T) = 1 - \theta$

- More generally, log-likelihood function

$$\begin{aligned}L(\theta) &= \theta^{\#heads} \cdot (1 - \theta)^{\#tails} \\ \log L(\theta) &= \log(\theta^{\#heads} \cdot (1 - \theta)^{\#tails}) \\ &= \#heads \cdot \log(\theta) + \#tails \cdot \log(1 - \theta)\end{aligned}$$

- MLE Goal: Find $\theta^* \in [0, 1]$ such that $\log L(\theta^*)$ is maximum.
- Differentiate the log-likelihood function with respect to θ and set the derivative to zero. We get:

$$\theta^* = \frac{\#heads}{\#heads + \#tails}$$

Extending the MLE principle to autoregressive models

Given an autoregressive model with n variables and factorization

$$P_{\theta}(\mathbf{x}) = \prod_{i=1}^n p_{\text{neural}}(x_i | \mathbf{x}_{<i}; \theta_i)$$

$\theta = (\theta_1, \dots, \theta_n)$ are the parameters of all the conditionals. Training data $\mathcal{D} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$. Maximum likelihood estimate of the parameters θ ?

- Decomposition of Likelihood function

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

- Goal : maximize $\arg \max_{\theta} L(\theta, \mathcal{D}) = \arg \max_{\theta} \log L(\theta, \mathcal{D})$
- We no longer have a closed form solution

MLE Learning: Gradient Descent

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

Goal : maximize $\arg \max_{\theta} L(\theta, \mathcal{D}) = \arg \max_{\theta} \log L(\theta, \mathcal{D})$

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

- 1 Initialize $\theta^0 = (\theta_1, \dots, \theta_n)$ at random
- 2 Compute $\nabla_{\theta} \ell(\theta)$ (by back propagation)
- 3 $\theta^{t+1} = \theta^t + \alpha_t \nabla_{\theta} \ell(\theta)$

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

MLE Learning: Stochastic Gradient Descent

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

- 1 Initialize θ^0 at random
- 2 Compute $\nabla_{\theta} \ell(\theta)$ (by back propagation)
- 3 $\theta^{t+1} = \theta^t + \alpha_t \nabla_{\theta} \ell(\theta)$

What is the gradient with respect to θ_i ?

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

Each conditional $p_{\text{neural}}(x_i | x_{<i}; \theta_i)$ can be optimized separately if there is no parameter sharing. In practice, parameters θ_i are shared (e.g., NADE, PixelRNN, PixelCNN, etc.)

MLE Learning: Stochastic Gradient Descent

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

- 1 Initialize θ^0 at random
- 2 Compute $\nabla_{\theta} \ell(\theta)$ (by back propagation)
- 3 $\theta^{t+1} = \theta^t + \alpha_t \nabla_{\theta} \ell(\theta)$

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

What if $m = |\mathcal{D}|$ is huge?

$$\begin{aligned} \nabla_{\theta} \ell(\theta) &= m \sum_{j=1}^m \frac{1}{m} \sum_{i=1}^n \nabla_{\theta} \log p_{\text{neural}}(x_i^{(j)} | x_{<i}^{(j)}; \theta_i) \\ &= m E_{x^{(j)} \sim \mathcal{D}} \left[\sum_{i=1}^n \nabla_{\theta} \log p_{\text{neural}}(x_i^{(j)} | x_{<i}^{(j)}; \theta_i) \right] \end{aligned}$$

Monte Carlo: Sample $x^{(j)} \sim \mathcal{D}$; $\nabla_{\theta} \ell(\theta) \approx m \sum_{i=1}^n \nabla_{\theta} \log p_{\text{neural}}(x_i^{(j)} | x_{<i}^{(j)}; \theta_i)$

Empirical Risk and Overfitting

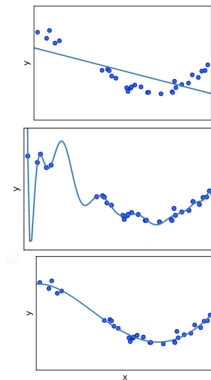
- Empirical risk minimization can easily **overfit** the data
 - Extreme example: The data is the model (remember all training data).
- Generalization: the data is a sample, usually there is vast amount of samples that you have never seen. Your model should generalize well to these “never-seen” samples.
- 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 better the data
 - When we have small amount of data, multiple models can fit well, or even better than the true model. Moreover, small perturbations on \mathcal{D} will result in very different estimates
 - This limitation is call the **variance**.

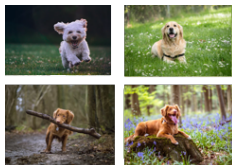
Bias-Variance trade off

- There is an inherent **bias-variance trade off** when selecting the hypothesis class. Error in learning due to both things: bias and variance.
- Hypothesis space: linear relationship
 - Does it fit well? Underfits
- Hypothesis space: high degree polynomial
 - Overfits
- Hypothesis space: low degree polynomial
 - Right tradeoff

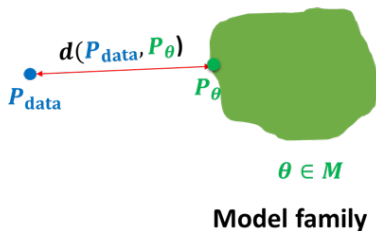


How to avoid overfitting?

- Hard constraints, e.g. by selecting a less expressive model family:
 - Smaller neural networks with less parameters
 - Weight sharing



$$x_i \sim P_{\text{data}} \\ i = 1, 2, \dots, n$$



- Soft preference for “simpler” models: **Occam Razor**.
- Augment the objective function with **regularization**:

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

- Evaluate generalization performance on a held-out validation set

Conditional generative models

- Suppose we want to generate a set of variables Y given some others X , e.g., text to speech
- We concentrate on modeling $p(Y|X)$, and use a **conditional** loss function

$$-\log P_{\theta}(y | x).$$

- Since the loss function only depends on $P_{\theta}(y | x)$, suffices to estimate the conditional distribution, not the joint



Input: image



Brown horse in
grass field

Output: caption

- For autoregressive models, it is easy to compute $p_{\theta}(x)$
 - Ideally, evaluate in parallel each conditional $\log p_{\text{neural}}(x_i^{(j)} | x_{<i}^{(j)}; \theta_i)$.
Not like RNNs.
- Natural to train them via maximum likelihood
- Higher log-likelihood doesn't necessarily mean better looking samples
- Other ways of measuring similarity are possible (Generative Adversarial Networks, GANs)