Deep Generative Models

Lecture 3: Maximum Likelihood Learning

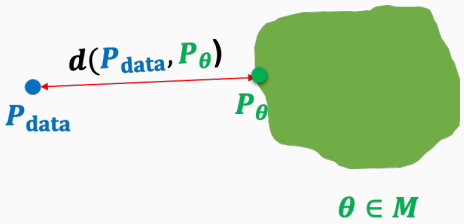
Aditya Grover

UCLA

Learning a generative model

- **Given:** a training set of examples, e.g., images of dogs

$x^{(j)} \sim P_{\text{data}}$
$j = 1, 2, \ldots, m$



- **Goal:** learn a probability distribution $p(x)$ over images x
  - **Generation:** If we sample $x_{new} \sim p(x)$, 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*)
- First question: how to represent $p_\theta(x)$. Second question: **how to learn it**.

Setting

- 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 the variables, e.g., $(X_{\text{bank}} = 1, X_{\text{dollar}} = 0, ..., Y = 1)$ 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 parameters θ of some "good" model $P_\theta \in \mathcal{M}$
- For example, a FVSBN for all possible choices of the logistic regression parameters. $\mathcal{M} = \{P_\theta, \theta \in \Theta\}$, $\theta =$ concatenation of all logistic regression coefficients

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

KL-divergence

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

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

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

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

- Notice that KL-divergence is **asymmetric**, i.e., $D(p\|q) \neq D(q\|p)$

Detour on KL-divergence

- Knowledge of the data distribution aids compression
- 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's more efficient to uses 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 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:

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

- $\mathbf{D}(P_{\text{data}}||P_\theta) = 0$ 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:

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

- The first term does not depend on P_θ.
- Then, *minimizing* KL divergence is equivalent to *maximizing* the **expected log-likelihood**
  - Asks that P_θ assign high probability to instances sampled from P_{data}, so as to reflect the true distribution
  - Because of log, samples $\mathbf{x}$ where $P_\theta(\mathbf{x}) \approx 0$ weigh heavily in objective
- Although we can now compare models, since we are ignoring $\mathbf{H}(P_{\text{data}})$, we don't know how close we are to the optimum
- Problem: In general we do not know P_{data}.

Maximum likelihood estimation

- Approximate the expected log-likelihood

$$\mathbf{E}_{\mathbf{x}\sim P_{\text{data}}}\left[\log P_\theta(\mathbf{x})\right]$$

  with the *empirical log-likelihood*:

$$\mathbf{E}_{\mathcal{D}}\left[\log P_\theta(\mathbf{x})\right] = \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}} \log P_\theta(\mathbf{x})$$

- Learning using **Maximum likelihood estimation**:

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

- Note: Equivalent to maximizing joint likelihood of the data under i.i.d. $P_\theta(\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}) = \prod_{\mathbf{x}\in\mathcal{D}} P_\theta(\mathbf{x})$

Example

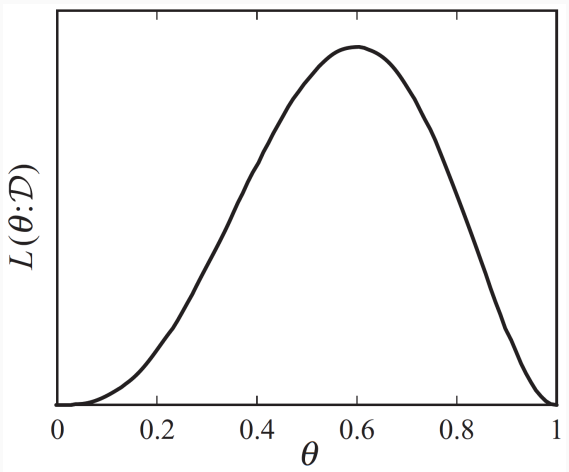
Single variable example: A biased coin

- 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

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

$\theta = (\theta_1, \cdots, \theta_n)$ are the parameters of all the conditionals. Training data $\mathcal{D} = \{\mathbf{x}^{(1)}, \cdots, \mathbf{x}^{(m)}\}$. 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

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

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

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

1. Initialize $\theta^0 = (\theta_1, \cdots, \theta_n)$ 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: Gradient Descent

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

- Discrete x_i's (e.g., language models): Equivalent to classification where we predict label of X_i given $x_{<i}$. Reduces to cross-entropy loss for categorical $p_{\text{neural}}(x_i | \mathbf{x}_{<i}; \theta_i)$
- Continuous x_i's (e.g., speech): Equivalent to regression where we predict label of X_i given $x_{<i}$. Reduces to mean-squared error (+ variance terms) for Gaussian $p_{\text{neural}}(x_i | \mathbf{x}_{<i}; \theta_i)$

What is the gradient with respect to θ_i?

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

Each conditional $p_{\text{neural}}(x_i | \mathbf{x}_{<i}; \theta_i)$ 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: Same tricks as supervised learning

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

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

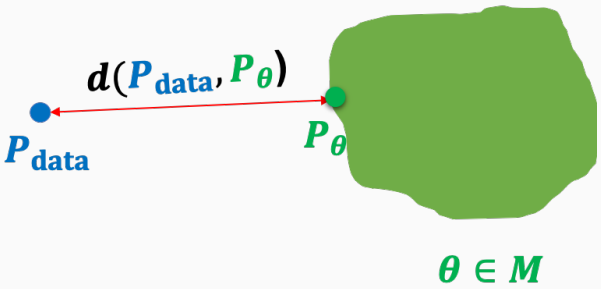
$$\nabla_\theta \ell(\theta) = \sum_{j=1}^{m} \sum_{i=1}^{n} \nabla_\theta \log p_{\text{neural}}(x_i^{(j)} | \mathbf{x}_{<i}^{(j)}; \theta_i)$$

- What if $m = |\mathcal{D}|$ is huge? Use mini-batches
- What if the model overfits? Intuition for overfitting: High likelihood on training set, low likelihood on test set (e.g., new images of dogs outside $\mathcal{D}$)

How to avoid overfitting?

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

$x^{(j)} \sim P_{\text{data}}$
$j = 1, 2, \ldots, m$



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

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

- Evaluate generalization performance on a held-out validation set

Conditional generative models

- Suppose we want to generate a set of variables $\mathbf{X}$ given some others $\mathbf{Y}$, e.g., text to speech, image captioning, machine translation
- We model $P_\theta(\mathbf{x} \mid \mathbf{y})$, and use a **conditional** loss function

$$-E_{(\mathbf{x},\mathbf{y})\sim D}[\log P_\theta(\mathbf{x} \mid \mathbf{y})].$$

- Since the loss function only depends on $P_\theta(\mathbf{x} \mid \mathbf{y})$, suffices to estimate the conditional distribution, not the joint
- We can factorize the joint distribution autoregressively:

$$P_\theta(\mathbf{x} \mid \mathbf{y}) = \prod_{i=1}^{n} P_\theta(x_i \mid x_{<i}, \mathbf{y})$$