CS 236 Homework 1

Instructor: Stefano Ermon

ermon@cs.stanford.edu

Available: 10/02/2023; Due: 23:59 PST, 10/16/2023

Problem 1: Maximum Likelihood Estimation and KL Divergence (10 points)

Can be attempted after Lecture 2

Let $\hat{p}(x, y)$ denote the empirical data distribution over a space of inputs $x \in \mathcal{X}$ and outputs $y \in \mathcal{Y}$. For example, in an image recognition task, x can be an image and y can be whether the image contains a cat or not. Let $p_\theta(y \mid x)$ be a probabilistic classifier parameterized by θ, e.g., a logistic regression classifier with coefficients θ. Show that the following equivalence holds:

$$\arg\max_{\theta \in \Theta} \mathbb{E}_{\hat{p}(x,y)} \left[\log p_\theta(y|x)\right] = \arg\min_{\theta \in \Theta} \mathbb{E}_{\hat{p}(x)} \left[D_{\mathrm{KL}}(\hat{p}(y|x) \| p_\theta(y|x))\right].$$

where D_{KL} denotes the KL-divergence:

$$D_{\mathrm{KL}}(p(x) \| q(x)) = \mathbb{E}_{x \sim p(x)}[\log p(x) - \log q(x)].$$

Problem 2: Logistic Regression and Naive Bayes (10 points)

Can be attempted after Lecture 2

A mixture of k Gaussians specifies a joint distribution given by $p_\theta(\boldsymbol{x}, y)$ where $y \in \{1, \ldots, k\}$ signifies the mixture id and $\boldsymbol{x} \in \mathbb{R}^n$ denotes n-dimensional real valued points. The generative process for this mixture can be specified as:

$$p_\theta(y) = \pi_y \text{ , where } \sum_{y=1}^{k} \pi_y = 1$$
$$p_\theta(\boldsymbol{x} \mid y) = \mathcal{N}(\boldsymbol{x} \mid \mu_y, \sigma^2 I).$$

where we assume a diagonal covariance structure for modeling each of the Gaussians in the mixture. Such a model is parameterized by $\theta = (\pi_1, \pi_2, \ldots, \pi_k, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_k, \sigma)$, where $\pi_i \in \mathbb{R}_{++}$, $\boldsymbol{\mu}_i \in \mathbb{R}^n$, and $\sigma \in \mathbb{R}_{++}$. Now consider the multi-class logistic regression model for directly predicting y from x as:

$$p_\gamma(y \mid \boldsymbol{x}) = \frac{\exp(\boldsymbol{x}^\top \boldsymbol{w}_y + b_y)}{\sum_{i=1}^{k} \exp(\boldsymbol{x}^\top \boldsymbol{w}_i + b_i)},$$

parameterized by vectors $\gamma = \{\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_k, b_1, b_2, \ldots, b_k\}$, where $\boldsymbol{w}_i \in \mathbb{R}^n$ and $b_i \in \mathbb{R}$.

Show that for any choice of θ, there exists γ such that

$$p_\theta(y \mid \boldsymbol{x}) = p_\gamma(y \mid \boldsymbol{x}).$$

Problem 3: Conditional Independence and Parameterization (15 points)

Can be attempted after Lecture 2

Consider a collection of n discrete random variables $\{X_i\}_{i=1}^n$, where the number of outcomes for X_i is $|\mathrm{val}(X_i)| = k_i$.

1. **[3 points]** Without any conditional independence assumptions, what is the total number of independent parameters needed to describe the joint distribution over $(X_1, \ldots, X_n)$?

2. **[3 points]** Under what independence assumptions is it possible to represent the joint distribution $(X_1, \ldots, X_n)$ with $\sum_{i=1}^{n}(k_i - 1)$ total number of independent parameters?

3. **[9 points]** Let $1, 2, \ldots, n$ denote the topological sort for a Bayesian network for the random variables $X_1, X_2, \ldots, X_n$. Let m be a positive integer in $\{1, 2, \ldots, n-1\}$. Suppose, for every $i > m$, the random variable X_i is conditionally independent of all ancestors given the previous m ancestors in the topological ordering. Mathematically, we impose the independence assumptions

$$p(X_i|X_{i-1}, X_{i-2}, \ldots X_2, X_1) = p(X_i|X_{i-1}, X_{i-2}, \ldots X_{i-m})$$

for $i > m$. For $i \leq m$, we impose no conditional independence of X_i with respect to its ancestors.

Derive the total number of independent parameters to specify the joint distribution over $(X_1, \ldots, X_n)$.

Problem 4: Autoregressive Models (15 points)

Can be attempted after Lecture 3

Consider a set of n univariate *continuous* real-valued random variables $(X_1, \ldots, X_n)$. You have access to powerful neural networks $\{\mu_i\}_{i=1}^n$ and $\{\sigma_i\}_{i=1}^n$ that can represent any function $\mu_i : \mathbb{R}^{i-1} \to \mathbb{R}$ and $\sigma_i : \mathbb{R}^{i-1} \to \mathbb{R}_{++}$. We shall, for notational simplicity, define $\mathbb{R}^0 = \{0\}$. You choose to build the following Gaussian autoregressive model in the *forward* direction:

$$p_f(x_1, \ldots, x_n) = \prod_{i=1}^{n} p_f(x_i|x_{<i}) = \prod_{i=1}^{n} \mathcal{N}(x_i|\mu_i(x_{<i}), \sigma_i^2(x_{<i})),$$

where $x_{<i}$ denotes

$$x_{<i} = \begin{cases} (x_1, \ldots, x_{i-1})^\top & \text{if } i > 1 \\ 0 & \text{if } i = 1. \end{cases}$$

Your friend chooses to factor the model in the *reverse* order using equally powerful neural networks $\{\hat{\mu}_i\}_{i=1}^n$ and $\{\hat{\sigma}_i\}_{i=1}^n$ that can represent any function $\hat{\mu}_i : \mathbb{R}^{n-i} \to \mathbb{R}$ and $\hat{\sigma}_i : \mathbb{R}^{n-i} \to \mathbb{R}_{++}$:

$$p_r(x_1, \ldots, x_n) = \prod_{i=1}^{n} p_r(x_i|x_{>i}) = \prod_{i=1}^{n} \mathcal{N}(x_i|\hat{\mu}_i(x_{>i}), \hat{\sigma}_i^2(x_{>i})),$$

where $x_{>i}$ denotes

$$x_{>i} = \begin{cases} (x_{i+1}, \ldots, x_n)^\top & \text{if } i < n \\ 0 & \text{if } i = n. \end{cases}$$

Do these models cover the same hypothesis space of distributions? In other words, given any choice of $\{\mu_i, \sigma_i\}_{i=1}^n$, does there always exist a choice of $\{\hat{\mu}_i, \hat{\sigma}_i\}_{i=1}^n$ such that $p_f = p_r$? If yes, provide a proof. Else, provide a concrete counterexample, including mathematical definitions of the modeled functions, and explain why.

[Hint: Consider the case where $n = 2$.]

Problem 5: Monte Carlo Integration (10 points)

Can be attempted after Lecture 4

A latent variable generative model specifies a joint probability distribution $p(x, z)$ between a set of observed variables $x \in \mathcal{X}$ and a set of latent variables $z \in \mathcal{Z}$. From the definition of conditional probability, we can express the joint distribution as $p(x, z) = p(z)p(x|z)$. Here, $p(z)$ is referred to as the prior distribution over z

and $p(x|z)$ is the likelihood of the observed data given the latent variables. One natural objective for learning a latent variable model is to maximize the marginal likelihood of the observed data given by:

$$p(x) = \int_z p(x, z)\mathrm{d}z.$$

When z is high dimensional, evaluation of the marginal likelihood is computationally intractable even if we can tractably evaluate the prior and the conditional likelihood for any given x and z. We can however use Monte Carlo to estimate the above integral. To do so, we sample k samples from the prior $p(z)$ and our estimate is given as:

$$A(z^{(1)}, \ldots, z^{(k)}) = \frac{1}{k}\sum_{i=1}^{k} p(x|z^{(i)}), \text{ where } z^{(i)} \sim p(z).$$

1. **[5 points]** An estimator $\hat{\theta}$ is an unbiased estimator of θ if and only if $\mathbb{E}[\hat{\theta}] = \theta$. Show that A is an unbiased estimator of $p(x)$.

2. **[5 points]** Is $\log A$ an unbiased estimator of $\log p(x)$? Prove why or why not. [Hint: The proof is short, using the definition of an unbiased estimator and Jensen's Inequality]

Problem 6: Programming assignment (40 points)

Can be attempted after Lecture 3

In this programming assignment, we will use an autoregressive generative model to create text. We will generate samples from the recent OpenAI GPT-2 model[1]. It is a model based on the Transformer, which is a special kind of autoregressive model built on self-attention blocks, and has become the backbone of many recent state-of-the-art sequence models in Natural Language Processing. See this blog post for a friendly introduction.

You will be asked to implement the sampling procedure for this model and compute likelihoods. In Figure 1, we show an illustration on how this model (roughly) works. Consider a sequence of tokens $x_0, x_1, \cdots, x_M$. For each token x_i, it has 50257 possible values. First, for each possible value of a token, we use a 768-dimensional trainable vector as its embedding, which results in a total of 50257 different embedding vectors. Next, we feed the embeddings into a GPT-2 network. The output vectors of the GPT-2 network are finally passed through a fully-connected layer to form a 50257-way softmax representing the probability distribution of the next token p_{i+1}. See Figure 1 for an illustration.

Training such models can be computationally expensive, requiring specialized GPU hardware. In this particular assignment, we provide a smaller pretrained model. It should be feasible to run this model without using any GPUs. After loading this pretrained model into `PyTorch`, you are expected to implement and answer the following questions.

1. **[4 points]** Suppose we wish to find an efficient bit representation for the 50257 tokens. That is, every token is represented as $(a_1, a_2, \cdots, a_n)$, where $a_i \in \{0, 1\}, \forall i = 1, 2, \cdots, n$. What is the minimal n that we can use?

2. **[6 points]** If the number of possible tokens increases from 50257 to 60000, what is the increase in the number of parameters? Give an exact number and explain your answer. [Hint: The number of parameters in the GPT-2 module in Fig. 1 does not change.]

   Note: For the following questions, you will need to complete the starter code in designated areas. After the code is completed, run `main.py` to provide related files for submission. Run the script `bash make_submission.sh` to generate `cs236_hw1_2023.zip` and upload it to GradeScope.

3. **[8 points]** In this question, we will try to generate paper abstracts using the GPT-2 model. We will first implement the sampling procedure for GPT-2. Then, we will choose 5 sentences from the abstracts of some NeurIPS[2] 2015 papers and see how GPT-2 generates the rest of the abstract. You will need to complete the method `sample` in `questions/sample.py` in the starter code.

   [Hint: The text generated should look like a technical paper.]

[1] This is the same model used in Lecture 1 to generate advice on how to get an A in CS236.

[2] Neural Information Processing Systems (NeurIPS) is a machine learning conference.

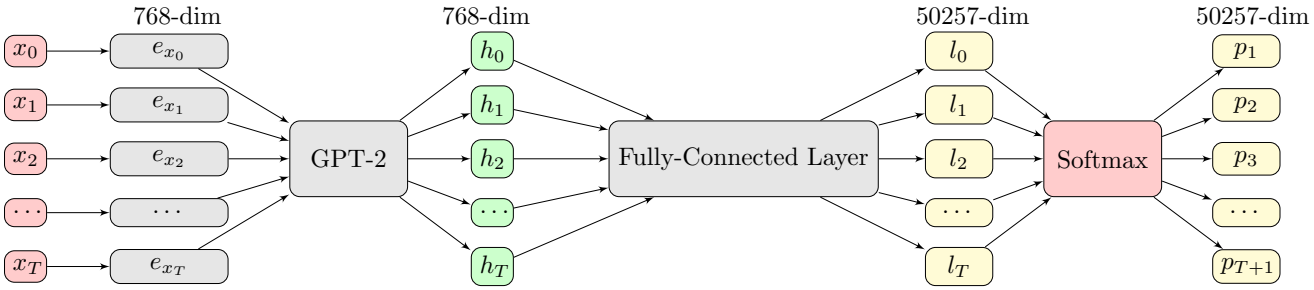

Figure 1: The architecture of our model. T is the sequence length of a given input. x_i is the index token. e_{x_i} is the trainable embedding of token x_i. h_i is the output of GPT-2. l_i is the logit and p_i is the probability. Nodes in gray (please view in color) contain trainable parameters.

4. **[8 points]** Complete the function `log_likelihood` in `questions/likelihood.py` to compute the log-likelihoods for each string. Plot a separate histogram of the log-likelihoods of strings within each file.

   [Hint: What do you think is the probability that a model successfully trained on abstracts will generate completely random text? What about the probability that it will generate well-written abstracts?]

5. **[7 points]** We now provide new texts in `snippets.pkl`. The texts are either taken from NeurIPS 2015 papers or Shakespeare's work, or generated randomly. Try to infer **whether the text is random**. You will need to complete the function `classification` in `questions/classifier.py`.

   [Hint: Look carefully at the plots you generated for Question 7. Is there a simple representation space in which the random data is separable from the non-random data?]

6. **[7 points]** Temperature scaling is a commonly used technique for adjusting the likelihood of the next token. We pick a scalar temperature $T > 0$ and divide the next token logits by T:

$$p_T(x_i|x_{<i}) \propto e^{\log p(x_i|x_{<i})/T}$$

   The model p is the GPT-2 model used in question 6.3, and the model p_T is the temperature-scaled model. For $T < 1$, we can see that p_T induces a *sharper* distribution than p, since it makes likely tokens even more likely.

   Complete the function `temperature_scale` in `questions/sample.py` only for the case when `temperature_horizon=1` to perform temperature scaling during sampling.

7. **[Bonus: 10 points]** In the previous question, we performed temperature scaling only over the next token, i.e., with $T < 1$ we made likely next-tokens even more likely. What if we want to make likely *sentences* even more likely? In this case, we should consider scaling the *joint temperature*.

$$p_T^{\text{joint}}(x_0 x_1 \dots x_M) \propto e^{\log p(x_0 x_1 \dots x_M)/T}$$

   a) Does applying chain rule with single-token temperature scaling recover joint temperature scaling? In other words, determine if the following equation holds for arbitrary T:

$$\prod_{i=0}^{M} p_T(x_i|x_{<i}) \stackrel{?}{=} p_T^{\text{joint}}(x_0 x_1 \dots x_M)$$

   b) Next, we will implement temperature scaling over more than than one token (for simplicity, we will do temperature scaling over two tokens).

$$p_T^{\text{joint-2}}(x_i x_{1+1}|x_{<i}) \propto e^{\log p(x_i x_{i+1}|x_{<i})/T}$$
$$p_T^{\text{joint-2}}(x_i|x_{<i}) = \sum_j p_T^{\text{joint-2}}(x_i, x_{i+1} = a_j|x_{<i})$$

Complete the function `temperature_scale` in `questions/sample.py` for the case when `temperature_horizon=2` to perform temperature scaling over a horizon of two tokens.