CS 236 Homework 1 Solutions

Instructors: Stefano Ermon and Yang Song

{ermon,yangsong}@cs.stanford.edu

Available: 09/22/2021; Due: 23:59 PST, 10/13/2021

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

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:

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

Solution

We rely on the known property that if ψ is a strictly monotonically decreasing function, then the following two problems are equivalent

$$\max_\theta f(\theta) \equiv \min_\theta \psi(f(\theta)).$$

This property can be proven via proof by contradiction, and we assume familiarity with this property. Now, it suffices to show that there exists a strictly monotonically decreasing ψ such that

$$\psi\Bigg(\mathbb{E}_{\hat{p}(x,y)} \log p_\theta(y|x)\Bigg) = \mathbb{E}_{\hat{p}(x)} \left[D_{\mathrm{KL}}(\hat{p}(y|x) \| p_\theta(y|x))\right].$$

Note that

$$\begin{aligned}\mathbb{E}_{\hat{p}(x)} \left[D_{\mathrm{KL}}(\hat{p}(y|x) \| p_\theta(y|x))\right] &= \mathbb{E}_{\hat{p}(x)}\mathbb{E}_{\hat{p}(y|x)}\Bigg(\log \hat{p}(y|x) - \log p_\theta(y|x)\Bigg) \\ &= \Bigg(\mathbb{E}_{\hat{p}(x,y)} \log \hat{p}(y|x)\Bigg) - \Bigg(\mathbb{E}_{\hat{p}(x,y)} \log p_\theta(y|x)\Bigg).\end{aligned}$$

Since the first term is a constant in our optimization problem (since it does not depend on θ), we simply choose the strictly monotonically decreasing function

$$\psi(z) = \Bigg(\mathbb{E}_{\hat{p}(x,y)} \log \hat{p}(y|x)\Bigg) - z.$$

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

A mixture of k Gaussians specifies a joint distribution given by $p_\theta(\boldsymbol{x}, y)$ where $y \in \{1, \dots, k\}$ 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:

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

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

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

Solution
Note that

$$
\begin{aligned}
p_\theta(y|x) &= \frac{p_\theta(x, y)}{p_\theta(x)} \\
&= \frac{\pi_y \cdot \exp\left(-\frac{1}{2\sigma^2}(x - \mu_y)^\top (x - \mu_y)\right) \cdot Z^{-1}(\sigma)}{\sum_i \pi_i \cdot \exp\left(-\frac{1}{2\sigma^2}(x - \mu_i)^\top (x - \mu_i)\right) \cdot Z^{-1}(\sigma)},
\end{aligned}
$$

where $Z(\sigma)$ is the Gaussian partition function (which is a function of σ). Further algebraic manipulations show that

$$
\begin{aligned}
p_\theta(y|x) &= \frac{\exp\left(-\frac{1}{2\sigma^2}(x^\top x - 2x^\top \mu_y + \mu_y^\top \mu_y) + \ln \pi_y\right)}{\sum_i \exp\left(-\frac{1}{2\sigma^2}(x^\top x - 2x^\top \mu_i + \mu_i^\top \mu_i) + \ln \pi_i\right)} \\
&= \frac{\exp\left(\frac{1}{2\sigma^2}(2x^\top \mu_y - \mu_y^\top \mu_y) + \ln \pi_y\right)}{\sum_i \exp\left(\frac{1}{2\sigma^2}(2x^\top \mu_i - \mu_i^\top \mu_i) + \ln \pi_i\right)} \\
&= \frac{\exp\left(x^\top \frac{\mu_y}{\sigma^2} + \left[-\frac{\mu_y^\top \mu_y}{2\sigma^2} + \ln \pi_y\right]\right)}{\sum_i \exp\left(x^\top \frac{\mu_i}{\sigma^2} + \left[-\frac{\mu_i^\top \mu_i}{2\sigma^2} + \ln \pi_i\right]\right)}.
\end{aligned}
$$

Thus, when $\theta = (\sigma, \pi, \mu_1, \ldots, \mu_k)$, simply set

$$
\begin{aligned}
w_y &= \frac{\mu_y}{\sigma^2} + \alpha \\
b_y &= -\left(\frac{\mu_y^\top \mu_y}{2\sigma^2}\right) + \ln \pi_y + \beta,
\end{aligned}
$$

where α and β are allowed to be any constants (with respect to y).

Problem 3: Conditional Independence and Parameterization (15 points)

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

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 $(X_1, \ldots, X_n)$.

Solution

1. There are $\prod_{i=1}^n k_i$ unique configurations. Without independence assumptions, the number of independent parameters needed is $(\prod_{i=1}^n k_i) - 1$. Alternate form is $\sum_{i=1}^n ((k_i - 1) * \prod_{j=1}^{i-1} k_j)$
2. If the distribution is fully-factorized (i.e., $p(x_1, \ldots, x_n) = \prod_i p(x_i)$), then we only need $\sum_{i=1}^n (k_i - 1)$ independent parameters.
3. The random variables $\{X_i\}_{i=1}^m$ are part of a complete graph and thus requires $(\prod_{i=1}^m k_i) - 1$ parameters. When $m > i$, each random variable requires $(k_i - 1) \prod_{j=m-i}^{i-1} k_j$ parameters. The total is thus

$$\left(\sum_{i=1}^m (k_i - 1) \prod_{j=1}^{i-1} k_j \right) + \left(\sum_{i=m+1}^n (k_i - 1) \prod_{j=i-m}^{i-1} k_j \right)$$

$$= \left(\prod_{i=1}^m k_i \right) - 1 + \left(\sum_{i=m+1}^n (k_i - 1) \prod_{j=i-m}^{i-1} k_j \right)$$

$$\text{alt. form} = \sum_{i=1}^m ((k_i - 1) * \prod_{j=1}^{i-1} k_j) + \left(\sum_{i=m+1}^n (k_i - 1) \prod_{j=i-m}^{i-1} k_j \right)$$

Problem 4: Autoregressive Models (15 points)

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 $\hat{\sigma}_i : \mathbb{R}^{n-i} \to \mathbb{R}_{++}$:

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

where $x_{>i}$ denotes

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

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 $n = 2$.]

Solution

They do not cover the same hypothesis space. To see why, consider the simple case of describing a joint distribution over (X_1, X_2) using the forward versus reverse factorizations. Consider the forward factorization where

$$p_f(x_1) = \mathcal{N}(x_1 | 0, 1)$$
$$p_f(x_2 | x_1) = \mathcal{N}(x_2 | \mu_2(x_1), \epsilon),$$

for which

$$\mu_2(x_1) = \begin{cases} 0 & \text{if } x_1 \le 0 \\ 1 & \text{otherwise.} \end{cases}$$

(*) This construction makes $p_f(x_2)$ a mixture of two distinct Gaussians, which $p_r(x_2)$ cannot match, since $p_f(x_2)$ is strictly Gaussian. Any counterexample of this form, which makes $p_f(x_2)$ non-Gaussian, suffices for full-credit.

(**) Interestingly, we can also intuit about the distribution $p_f(x_1|x_2)$. If one chooses a very small positive ϵ, then the corresponding $p_f(x_1|x_2)$ will approach a truncated Gaussian distribution, which cannot be approximated by the Gaussian $p_r(x_1|x_2)$.[1]

Optionally, we can prove (*) and a variant of (**) which states that, any $\epsilon > 0$, the distribution

$$p_f(x_1|x_2) = \frac{p_f(x_1, x_2)}{p_f(x_2)}.$$

is a mixture of truncated Gaussians whose mixture weights depend on ϵ.

Proof of (*). We exploit the fact that μ_2 is step function by noting that

$$\begin{aligned} p_f(x_2) &= \int_{-\infty}^{\infty} p_f(x_1, x_2) \mathrm{d}x_1 & (1) \\ &= \int_{-\infty}^{0} p_f(x_1) \mathcal{N}(x_2 | 0, \epsilon) \mathrm{d}x_1 + \int_{0}^{\infty} p_f(x_1) \mathcal{N}(x_2 | 1, \epsilon) \mathrm{d}x_1 & (2) \\ &= \frac{1}{2}(\mathcal{N}_0(x_2) + \mathcal{N}_1(x_2)). & (3) \end{aligned}$$

For notational simplicity, we introduce the notation $\mathcal{N}_\mu$ in Eq. (3). The use of a step function for μ_2 thus partitions the space of x_1 so that the marginal distribution of x_2 is a mixture of two Gaussians.

[1] This observation will be useful when we move on to variational autoencoders $p(z, x)$ (where z is a latent variable) and discuss the importance of having good variational approximations of the true posterior $p(z|x)$.

Proof of (**) variant. The numerator is simply

$$p_f(x_1, x_2) = \begin{cases} p_f(x_1)\mathcal{N}_0(x_2) & \text{if } x_1 \le 0 \\ p_f(x_1)\mathcal{N}_1(x_2) & \text{if } x_1 > 0. \end{cases}$$

Combining the numerator and denominator thus yields

$$p_f(x_1|x_2) = \begin{cases} p_f(x_1) \cdot \dfrac{2\mathcal{N}_0(x_2)}{\mathcal{N}_0(x_2) + \mathcal{N}_1(x_2)} & \text{if } x_1 \le 0 \\ \\ p_f(x_1) \cdot \dfrac{2\mathcal{N}_1(x_2)}{\mathcal{N}_0(x_2) + \mathcal{N}_1(x_2)} & \text{if } x_1 > 0, \end{cases}$$

where $p_f(x_1)$ is multiplied by the weighting term

$$v_i = \frac{2\mathcal{N}_i(x_2|i, \epsilon)}{\mathcal{N}_0(x_2|0, \epsilon) + \mathcal{N}_1(x_2|1, \epsilon)}.$$

Note that $v_i/2$ can be interpreted as the posterior probability of the i^{th} Gaussian mixture component when x_2 is observed. For any choice of $x_2 \neq 0.5$, note that $v_1 \neq v_0$. Thus, when $x_2 \neq 0$, $p_f(x_1|x_2)$ will experience a sudden density transition when x_1 crosses 0. One should be able to see that $p_f(x_1|x_2)$ is an unevenly-weighted mixture of two truncated Gaussian distributions, which $p_r(x_1|x_2)$ cannot match. Furthermore, as $\epsilon \to 0$, we see that (v_0, v_1) approaches $(0, 1)$, which in turn causes $p_f(x_1|x_2 = 1)$ to approach a truncated Gaussian.

Problem 5: Monte Carlo Integration (10 points)

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]

Solution

The estimator A is unbiased since

$$\begin{aligned} \mathbb{E}_{z^{(1)},\ldots,z^{(k)}} A(z^{(1)}, \ldots, z^{(k)}) &= \frac{1}{k}\sum_{i=1}^{k} \mathbb{E}_{z^{(i)}} p(x|z^{(i)}) \\ &= \mathbb{E}_{p(z)} p(x|z) \\ &= \int p(z)p(x|z)\mathrm{d}z \\ &= p(x). \end{aligned}$$

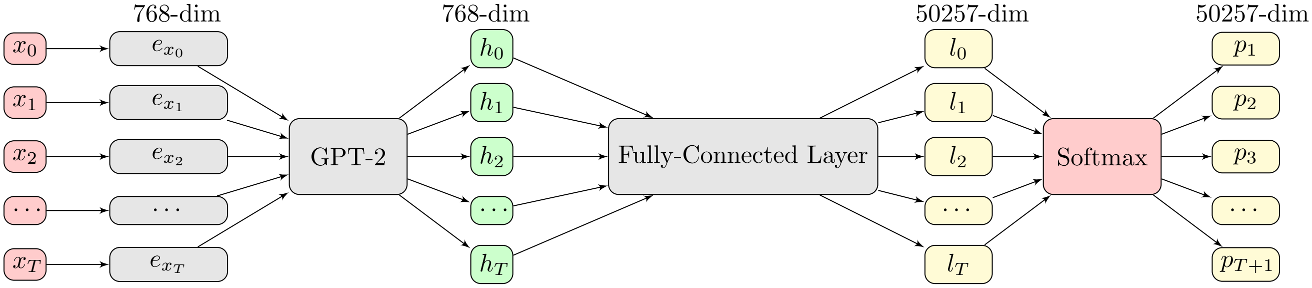

Figure 1: The architecture of our model. T is the sequence length of a given input. x_i is the index token. e_{x_i} is the trainable embedding of token x_i. h_i is the output of GPT-2. l_i is the logit and p_i is the probability. Nodes in gray (please view in color) contain trainable parameters.

The estimator $\log A$ is not guaranteed to be unbiased since, by Jensen's inequality,

$$\begin{aligned}\mathbb{E}_{z^{(1)},\ldots,z^{(k)}} \log A(z^{(1)},\ldots,z^{(k)}) &\leq \log \mathbb{E}_{z^{(1)},\ldots,z^{(k)}} A(z^{(1)},\ldots,z^{(k)}) \\ &= \log p(x).\end{aligned}$$

Note that since log is strictly concave, equality holds if and only if the random variable A is deterministic.

Problem 6: Programming assignment (40 points)

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[2]. 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 $x_0, x_1, \cdots, x_T$. 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?

   Solution: 16.

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.]

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

Solution:

$$\underbrace{(60000-50257)\times 768}_{\text{embeddings}}+\underbrace{(60000-50257)\times 768+\underbrace{60000-50257}_{\text{bias}}}_{\text{fully-connected layer}}=14974991.$$

It is also fine to exclude the bias term. The answer will be

$$\underbrace{(60000-50257)\times 768}_{\text{embeddings}}+\underbrace{(60000-50257)\times 768}_{\text{fully-connected layer}}=14965248.$$

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_2019.zip` and upload it to GradeScope.

3. [**10 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[3] 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.]

   Solutions:
   Code:

```
def sample(model, start_text, config):
    length = config.n_ctx // 2

    current_text = start_text
    past = None
    output = [start_text]
    with torch.no_grad():
        for _ in trange(length):
            logits, past = model(current_text, past=past)
            # logits: a tensor of shape (batch_size, length_of_current_text,
                                                    size_of_vocabulary)

            current_logits = logits[:, -1, :]
            logits = top_k_logits(current_logits, k=config.top_k)

            ##TODO:
            ## 1) sample using the given ‘logits‘ tensor;
            ## 2) append the sample to the list ‘output‘;
            ## 3) update ‘current_text‘ so that sampling can continue.

            probs = F.softmax(logits, dim=-1)
            current_text = torch.multinomial(probs, num_samples=1)
            output.append(current_text)

        output = torch.cat(output, dim=1)
        return output
```

4. [**10 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?]

   Solutions:
   Code:

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

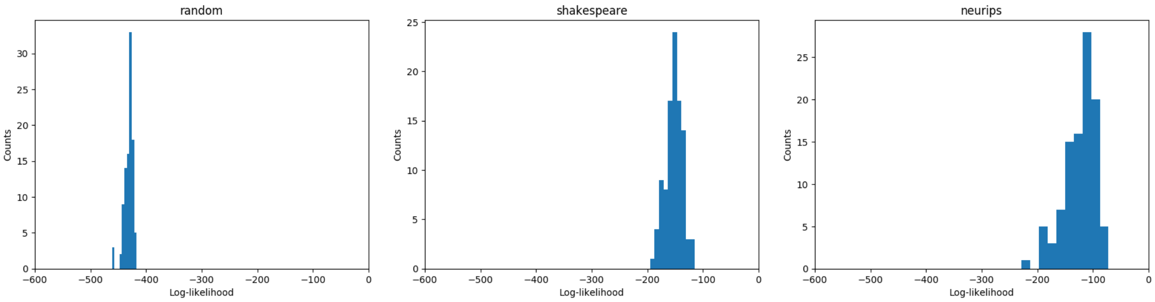

Figure 2

```
def log_likelihood(model, text):
    """
    Compute the log-likelihoods for a string ‘text‘
    :param model: The GPT-2 model
    :param texts: A tensor of shape (1, T), where T is the length of the text
    :return: The log-likelihood. It should be a Python scalar.
    """

    with torch.no_grad():
        ## TODO:
        ##  1) Compute the logits from ‘model‘;
        ##  2) Return the log-likelihood of the ‘text‘ string. It should be a Python
                                                        scalar.
        ##       NOTE: for simplicity, you can ignore the likelihood of the first token in
                                                        ‘text‘

        logits, past = model(text, past=None)
        loss = nn.CrossEntropyLoss(reduction=’sum’)
        return -loss(logits[0][:-1], text[0][1:]).item()
```

Figures: See Figure 2

5. [**10 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 4. Is there a simple representation space in which the random data is separable from the non-random data?]

Solutions:

Code:

```
def classification(model, text):
    """
    Classify whether the string ‘text‘ is randomly generated or not.
    :param model: The GPT-2 model
    :param texts: A tensor of shape (1, T), where T is the length of the text
    :return: True if ‘text‘ is a random string. Otherwise return False
    """

    with torch.no_grad():
        ## TODO: Return True if ‘text‘ is a random string. Or else return False.

        from questions.likelihood import log_likelihood
        ll = log_likelihood(model, text)
        return True if ll < -300 else False
```