CS 236 Homework 1 Solutions

Instructors: Stefano Ermon and Aditya Grover

{ermon,adityag}@cs.stanford.edu

Available: 10/01/2018; Due: 23:59 PST, 10/15/2018

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 $p_\theta(y|x)$ 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

$$\max_\theta f(\theta) \equiv \min_\theta \psi(f(\theta)). \tag{1}$$

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

$$\psi\left(\mathbb{E}_{\hat{p}(x,y)} \log p_\theta(y|x)\right) = \mathbb{E}_{\hat{p}(x)} KL(\hat{p}(y|x)\|p_\theta(y|x)). \tag{2}$$

Note that

$$\mathbb{E}_{\hat{p}(x)} KL(\hat{p}(y|x)\|p_\theta(y|x)) = \mathbb{E}_{\hat{p}(x)}\mathbb{E}_{\hat{p}(y|x)}\left(\log \hat{p}(y|x) - \log p_\theta(y|x)\right) \tag{3}$$

$$= \left(\mathbb{E}_{\hat{p}(x,y)} \log \hat{p}(y|x)\right) - \left(\mathbb{E}_{\hat{p}(x,y)} \log p_\theta(y|x)\right). \tag{4}$$

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

$$\psi(z) = \left(\mathbb{E}_{\hat{p}(x,y)} \log \hat{p}(y|x)\right) - z. \tag{5}$$

Problem 2: Logistic Regression and Naive Bayes (12 points)

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

$$p_\theta(y) = \pi_y \text{ , where } \sum_{y=1}^{k} \pi_y = 1 \tag{6}$$

$$p_\theta(x|y) = \mathcal{N}(x|\mu_y, \sigma^2 I). \tag{7}$$

where we assume a diagonal covariance structure for modeling each of the Gaussians in the mixture. Such a model is parameterized by $\theta = (\pi_1, \pi_2, \ldots, \pi_k, \mu_1, \mu_2, \ldots, \mu_k, \sigma)$, where $\pi_i \in \mathbb{R}_{++}$, $\mu_i \in \mathbb{R}^n$, and $\sigma \in \mathbb{R}_{++}$. Now consider the multi-class logistic regression model for directly predicting y from x as:

$$p_\gamma(y|x) = \frac{\exp(x^\top w_y + b_y)}{\sum_{i=1}^{k} \exp(x^\top w_i + b_i)}, \tag{8}$$

parameterized by vectors $\gamma = \{w_1, w_2, \ldots, w_k, b_1, b_2, \ldots, b_k\}$, where $w_i \in \mathbb{R}^n$ and $b_i \in \mathbb{R}$.
Show that for any choice of θ, there exists γ such that

$$p_\theta(y|x) = p_\gamma(y|x). \tag{9}$$

Solution
Note that

$$p_\theta(y|x) = \frac{p_\theta(x,y)}{p_\theta(x)} \tag{10}$$

$$= \frac{\pi_y \cdot \exp\left(-\frac{1}{2\sigma^2}(x-\mu_y)^\top(x-\mu_y)\right) \cdot Z^{-1}(\sigma)}{\sum_i \pi_i \cdot \exp\left(-\frac{1}{2\sigma^2}(x-\mu_i)^\top(x-\mu_i)\right) \cdot Z^{-1}(\sigma)}, \tag{11}$$

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

$$p_\theta(y|x) = \frac{\exp\left(-\frac{1}{2\sigma^2}(x^\top x - 2x^\top\mu_y + \mu_y^\top\mu_y) + \ln\pi_y\right)}{\sum_i \exp\left(-\frac{1}{2\sigma^2}(x^\top x - 2x^\top\mu_i + \mu_i^\top\mu_i) + \ln\pi_i\right)} \tag{12}$$

$$= \frac{\exp\left(\frac{1}{2\sigma^2}(2x^\top\mu_y - \mu_y^\top\mu_y) + \ln\pi_y\right)}{\sum_i \exp\left(\frac{1}{2\sigma^2}(2x^\top\mu_i - \mu_i^\top\mu_i) + \ln\pi_i\right)} \tag{13}$$

$$= \frac{\exp\left(x^\top\frac{\mu_y}{\sigma^2} + \left[-\frac{\mu_y^\top\mu_y}{2\sigma^2} + \ln\pi_y\right]\right)}{\sum_i \exp\left(x^\top\frac{\mu_i}{\sigma^2} + \left[-\frac{\mu_i^\top\mu_i}{2\sigma^2} + \ln\pi_i\right]\right)}. \tag{14}$$

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

$$w_y = \frac{\mu_y}{\sigma^2} + \alpha \tag{15}$$

$$b_y = -\left(\frac{\mu_y^\top\mu_y}{2\sigma^2}\right) + \ln\pi_y + \beta, \tag{16}$$

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

Problem 3: Conditional Independence and Parameterization (16 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. **[2 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. **[12 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 m previous 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 $(X_1, \ldots, X_n)$?
3. **[2 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?

Solution

1. There are $\prod_{i=1}^n k_i$ unique configurations. Without independence assumptions, the number of independent parameters needed is $(\prod_{i=1}^n k_i) - 1$.
2. 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

   $$\begin{aligned} &\left(\sum_{i=1}^{m} (k_i - 1) \prod_{j=1}^{i-1} k_j \right) + \left(\sum_{i=m+1}^{n} (k_i - 1) \prod_{j=i-m}^{i-1} k_j \right) \\ &= \left(\prod_{i=1}^{m} k_i \right) - 1 + \left(\sum_{i=m+1}^{n} (k_i - 1) \prod_{j=i-m}^{i-1} k_j \right) \end{aligned} \tag{17}$$
3. 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.

Problem 4: Autoregressive Models (12 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:

$$p_f(x_1, \ldots, x_n) = \prod_{i=1}^n p_f(x_i | x_{<i}) = \prod_{i=1}^n \mathcal{N}(x_i | \mu_i(x_{<i}), \sigma_i^2(x_{<i})), \tag{18}$$

where $x_{<i}$ denotes

$$x_{<i} = \begin{cases} (x_1, \ldots, x_{i-1})^\top & \text{if } i > 1 \\ 0 & \text{if } i = 1. \end{cases} \tag{19}$$

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

$$p_r(x_1, \dots, x_n) = \prod_{i=1}^n p_r(x_i | x_{>i}) = \prod_{i=1}^n \mathcal{N}(x_i | \hat{\mu}_i(x_{>i}), \hat{\sigma}_i^2(x_{>i})), \tag{20}$$

where $x_{>i}$ denotes

$$x_{>i} = \begin{cases} (x_{i+1}, \dots, x_n)^\top & \text{if } i < n \\ 0 & \text{if } i = n. \end{cases} \tag{21}$$

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 counterexample.

[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

$$p_f(x_1) = \mathcal{N}(x_1 | 0, 1) \tag{22}$$
$$p_f(x_2 | x_1) = \mathcal{N}(x_2 | \mu_2(x_1), \epsilon), \tag{23}$$

for which

$$\mu_2(x_1) = \begin{cases} 0 & \text{if } x_1 \le 0 \\ 1 & \text{otherwise.} \end{cases} \tag{24}$$

(*) 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

$$p_f(x_1 | x_2) = \frac{p_f(x_1, x_2)}{p_f(x_2)}. \tag{25}$$

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

$$\begin{aligned} p_f(x_2) &= \int_{-\infty}^{\infty} p_f(x_1, x_2) \mathrm{d}x_1 & (26) \\ &= \int_{-\infty}^{0} p_f(x_1) \mathcal{N}(x_2 | 0, \epsilon) \mathrm{d}x_1 + \int_{0}^{\infty} p_f(x_1) \mathcal{N}(x_2 | 1, \epsilon) \mathrm{d}x_1 & (27) \\ &= \frac{1}{2}(\mathcal{N}_0(x_2) + \mathcal{N}_1(x_2)). & (28) \end{aligned}$$

For notational simplicity, we introduce the notation $\mathcal{N}_\mu$ in Eq. (28). 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

$$p_f(x_1, x_2) = \begin{cases} p_f(x_1)\mathcal{N}_0(x_2) & \text{if } x_1 \le 0 \\ p_f(x_1)\mathcal{N}_1(x_2) & \text{if } x_1 > 0. \end{cases} \tag{29}$$

Combining the numerator and denominator thus yields

$$p_f(x_1|x_2) = \begin{cases} p_f(x_1) \cdot \dfrac{2\mathcal{N}_0(x_2)}{\mathcal{N}_0(x_2) + \mathcal{N}_1(x_2)} & \text{if } x_1 \le 0 \\ \\ p_f(x_1) \cdot \dfrac{2\mathcal{N}_1(x_2)}{\mathcal{N}_0(x_2) + \mathcal{N}_1(x_2)} & \text{if } x_1 > 0, \end{cases} \tag{30}$$

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

$$v_i = \frac{2\mathcal{N}_i(x_2|i, \epsilon)}{\mathcal{N}_0(x_2|0, \epsilon) + \mathcal{N}_1(x_2|1, \epsilon)}. \tag{31}$$

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 $x_2 \ne 0.5$, note that $v_1 \ne v_0$. Thus, when $x_2 \ne 0$, $p_f(x_1|x_2)$ 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 condition on the latent variables. One natural objective for learning a latent variable model is to maximize the marginal likelihood of the observed data given by:

$$p(x) = \int_z p(x, z)\mathrm{d}z. \tag{32}$$

When z is high dimensional, tractable 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:

$$A(z^{(1)}, \dots, z^{(k)}) = \frac{1}{k}\sum_{i=1}^{k} p(x|z^{(i)}), \text{ where } z^{(i)} \sim p(z). \tag{33}$$

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 $\log p(x)$? Explain why or why not.

Solution

The estimator A is unbiased since

$$\begin{aligned} \mathbb{E}_{z^{(1)},\dots,z^{(k)}} A(z^{(1)}, \dots, z^{(k)}) &= \frac{1}{k}\sum_{i=1}^{k} \mathbb{E}_{z^{(i)}} p(x|z^{(i)}) && (34) \\ &= \mathbb{E}_{p(z)} p(x|z) && (35) \\ &= \int p(z)p(x|z)\mathrm{d}z && (36) \\ &= p(x). && (37) \end{aligned}$$

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

$$\begin{aligned}
\mathbb{E}_{z^{(1)},\ldots,z^{(k)}} \log A(z^{(1)},\ldots,z^{(k)}) &\leq \log \mathbb{E}_{z^{(1)},\ldots,z^{(k)}} A(z^{(1)},\ldots,z^{(k)}) && (38)\\
&= \log p(x). && (39)
\end{aligned}$$

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

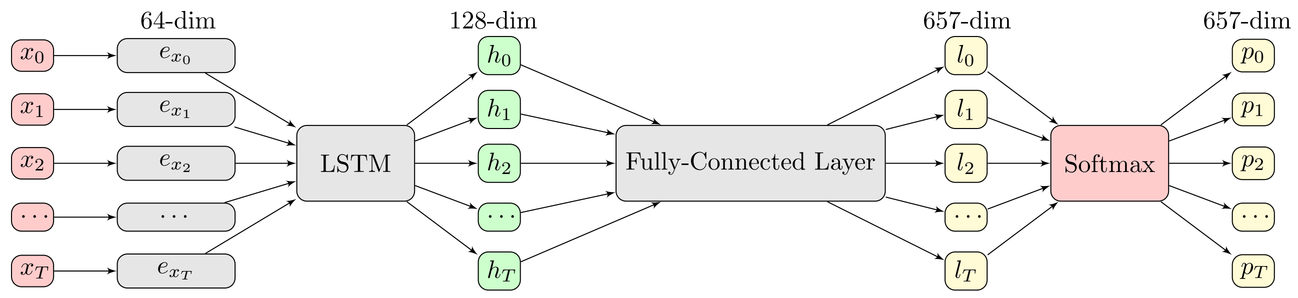

Figure 1: The architecture of our model. T is the sequence length of a given input. x_i is the index token. e_{x_i} is the trainable embedding of token x_i. h_i is the output of LSTMs. l_i is the logit and p_i is the probability. Nodes in gray (please view in color) contain trainable parameters.

Problem 6: Programming assignment (40 points)

In this programming assignment, we will use an autoregressive generative model to generate text from machine learning papers. In particular, we will train a character-based recurrent neural network (RNN) to generate paragraphs. The training dataset consists of all papers published in NIPS 2015.[2] The model used in this assignment is a four-layer Long Short-Term Memory (LSTM) network. LSTM is a variant of RNN that performs better in modeling long-term dependencies. See this blog post for a friendly introduction.

There are a total of 657 different characters in NIPS 2015 papers, including alphanumeric characters as well as many non-ascii symbols. During training, we first convert characters to a number in the range 0 to 656. Then for each number, we use a 64-dimensional trainable vector as its embedding. The embeddings are then fed into a four-layer LSTM network, where each layer contains 128 units. The output vectors of the LSTM network are finally passed through a fully-connected layer to form a 657-way softmax representing the probability distribution of the next token. See Figure 1 for an illustration.

Training such models can be computationally expensive, requiring specialized GPU hardware. In this particular assignment, we provide a pretrained generative model. 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 657 characters. That is, every character 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: 10.

2. [**6 points**] If the size of vocabulary increases from 657 to 900, what is the increase in the number of parameters? [Hint: The number of parameters in the LSTM module in Fig. 1 does not change.]

   Solution:

$$\underbrace{(900-657)\times 64}_{\text{embeddings}} + \underbrace{(900-657)\times 128 + \underbrace{900-657}_{\text{bias}}}_{\text{fully-connected layer}} = 46899.$$

   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 `./make_submission.sh` to generate `hw1.zip` and upload it to GradeScope.

3. [**10 points**] In the starter code, complete the method `sample` in `model.py` to generate 5 paragraphs each of length 1000 from this model.

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

Solutions:

Code:

```
def sample(self, seq_len):
    """
    Sample a string of length `seq_len` from the model.
    :param seq_len [int]: String length
    :return [list]: A list of length `seq_len` that contains the index of each generated
                                                    character.
    """
    voc_freq = self.dataset.voc_freq
    with torch.no_grad():
        h_prev = None
        texts = []
        x = np.random.choice(voc_freq.shape[0], 1, p=voc_freq)[None, :]
        x = torch.from_numpy(x).type(torch.int64).to(self.device)
        # TODO: Complete the code here.
        for i in range(seq_len):
            logits, h_prev = self.forward(x, h_prev)
            np_logits = logits[-1, :].to('cpu').numpy()
            probs = np.exp(np_logits) / np.sum(np.exp(np_logits))
            ix = np.random.choice(np.arange(self.vocab_size), p=probs.ravel())
            x = torch.tensor(ix, dtype=torch.int64)[None, None].to(self.device)
            texts.append(ix)
    return texts
```

4. **[10 points]** Complete the method `compute_prob` in `model.py` to compute the log-likelihoods for each string. Plot a separate histogram of the log-likelihoods of strings within each file.

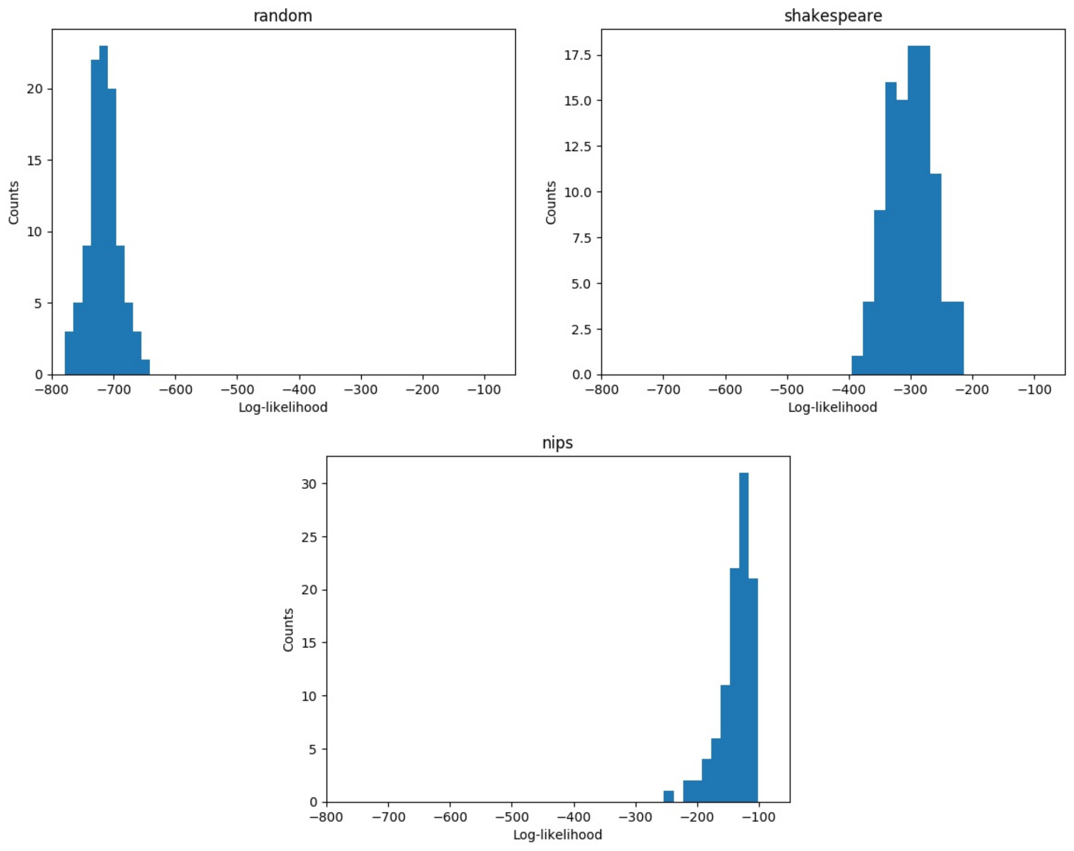

Figure 2

Solutions:
Code:

```
def compute_prob(self, strings):
"""
Compute the probability for each string in ‘strings‘
:param strings [np.ndarray]: an integer array of length N
:return [float]: the log-likelihood
"""
voc_freq = self.dataset.voc_freq
with torch.no_grad():
    h_prev = None
    x = strings[None, 0, None]
    x = torch.from_numpy(x).type(torch.int64).to(self.device)
    ll = np.log(voc_freq[strings[0]])
    # TODO: Complete the code here
    for i in range(len(strings) - 1):
        logits, h_prev = self.forward(x, h_prev)
        log_softmax = F.log_softmax(logits, dim=1)
        ll += log_softmax[-1, strings[i + 1]].item()
        x = strings[None, i + 1, None]
        x = torch.from_numpy(x).type(torch.int64).to(self.device)
    return ll
```

Figures: See Figure. 2

5. **[10 points]** Can you determine the category of an input string by only looking at its log-likelihood? We now provide new strings in `snippets.pkl`. Try to infer whether the string is generated randomly, copied from Shakespeare's work or retrieved from NIPS publications. You will need to complete the code in `main.py`.

Solutions:
Code:

```
for snippet in snippets:
    ll = rnn.compute_prob(np.asarray([dataset.char2idx[c] for c in snippet]))
    if ll < - 600:
        lbls.append(0)
    elif ll < -200:
        lbls.append(1)
    else:
        lbls.append(2)
```