Part 1. Count-based Language Modeling

(a) The likelihood of a corpus under the bigram model is given by

$$p(w_1, w_2, \ldots, w_T) = \prod_{t=1}^{T} p_\theta(w_t|w_{t-1}) = \prod_{i=1}^{T}\prod_{j=1}^{T} p_\theta(w_j|w_i)^{c(w_i,w_j)}$$

where $c(w_i, w_j)$ is the count of the number of times the sequence (w_i, w_j) appears. To find the MLE, we take the log likelihood:

$$\log p(w_1, w_2, \ldots, w_T) = \sum_{i=1}^{T}\sum_{j=1}^{T} p_\theta(w_j|w_i)^{c(w_i,w_j)}$$

Now, in order to maximize $\log p(w_1, w_2, \ldots, w_T)$ subject to the constraints that $\sum_{j=1}^{T} p(w_j|w_i) = 1$ for all $i \in \{1, 2, \ldots T\}$, we introduce Lagrange multipliers λ_i:

$$\mathcal{L}(\theta, \lambda_1, \lambda_2, \ldots, \lambda_T) = \sum_{i=1}^{T}\sum_{j=1}^{T} p_\theta(w_j|w_i)^{c(w_i,w_j)} + \sum_{i=1}^{T} \lambda_i \left(\sum_{j=1}^{T} p(w_j|w_i) - 1 \right)$$

Differentiating with respect to $p(w_k|w_i)$ and setting the derivative to 0, we get

$$\frac{\partial \mathcal{L}}{\partial p(w_k|w_i)} = \frac{c(w_i, w_k)}{p(w_k|w_i)} + \lambda_i = 0$$

$$p(w_k|w_i) = -\frac{c(w_i, w_k)}{\lambda_i}$$

Then, since we have $\sum_{j=1}^{T} p(w_j|w_i) = 1$, we get

$$\sum_{j=1}^{T} p(w_j|w_i) = \sum_{j=1}^{T} -\frac{c(w_i, w_k)}{\lambda_i} = -\frac{\sum_{j=1}^{T} c(w_i, w_k)}{\lambda_i} = 1$$

$$\lambda_i = -\sum_{j=1}^{T} c(w_i, w_k)$$

Lastly, since $p(w_k|w_i) = -\frac{c(w_i,w_k)}{\lambda_i}$, we can plug λ_i back in to get that the MLE of $p(w_k|w_i)$ is

$$\hat{\theta}_{i,k} = \frac{c(w_i, w_k)}{\sum_{j=1}^{T} c(w_i, w_j)}$$

Rewriting this in terms of our bigram language, we get that the MLE of $P(w_t = u|w_{t-1} = v)$ for $u, v \in \mathcal{V}$ is

$$\hat{\theta}_{u,v} = \frac{c(u, v)}{\sum_{u \in \mathcal{V}} c(u, v)}$$

(b) We are given that the density of the Dirichlet distribution is

$$p(\pi;\beta) = \frac{1}{B(\beta)} \prod_{u \in \mathcal{V}} \pi_u^{\beta_u - 1}$$

and using similar logic to part (a), we know that the likelihood of $D = (w_1, \ldots, w_T)$ under the unigram model is

$$p(D|\pi) = p(w_1, \ldots, w_T|\pi) = \prod_{t=1}^{T} \pi_t = \prod_{u \in \mathcal{V}} \pi_u^{c(u)}$$

where $c(u)$ is the count of the number of times u appears. We can multiply these two expressions together to get

$$p(D|\pi)p(\pi;\beta) = \left(\prod_{u \in \mathcal{V}} \pi_u^{c(u)} \right) \left(\frac{1}{B(\beta)} \prod_{u \in \mathcal{V}} \pi_u^{\beta_u - 1} \right) = \frac{1}{B(\beta)} \prod_{u \in \mathcal{V}} \pi_u^{c(u) + \beta_u - 1}$$

which we know is proportional to the posterior via Bayes' rule, since $p(\pi|D;\beta) = \frac{p(D|\pi)p(\pi;\beta)}{p(D)}$. Thus, setting $\beta_u = \alpha$ for all $u \in \mathcal{V}$, we get

$$p(\pi|D;\beta) \propto \prod_{u \in \mathcal{V}} \pi_u^{c(u) + \alpha - 1}$$

Looking at the expression, we see that the posterior is really a Dirichlet distribution with parameters $\beta_u = c(u) + \alpha$ for all $u \in \mathcal{V}$. Then the mean vector of this distribution is given by

$$\mathbb{E}_{p(\pi|D;\beta)}[\pi] = \frac{1}{\sum_{u=1}^{\mathcal{V}} \beta_u} \beta = \frac{1}{\sum_{u=1}^{\mathcal{V}} c(u) + \alpha} \beta = \frac{1}{T + \alpha|\mathcal{V}|} \beta$$

since $\sum_{u=1}^{\mathcal{V}} c(u) = T$ (i.e. the sum of the counts of all words is just the total number of words in the corpus) and $\sum_{u=1}^{\mathcal{V}} \alpha = \alpha|\mathcal{V}|$. Thus, each element π_u of the mean vector is given by

$$\pi_u = \frac{\beta_u}{T + \alpha|\mathcal{V}|} = \frac{c(u) + \alpha}{T + \alpha|\mathcal{V}|}$$

which is exactly the Laplace smoothing estimate θ_u. Therefore, $\mathbb{E}_{p(\pi|D;\beta)}[\pi] = \theta$.

Part 2. (a) We use the RNN hidden state update equation

$$h_t = f(Wx_t + Uh_{t-1}), \qquad W = \begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix},\ U = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

where $h_t \in \mathbb{R}^2$ are 2-dimensional vectors (with h_0 initialized to $[0, 0]$), $x_t \in ([1, 0], [0, 1])$ are one-hot encoded vectors (with "(" corresponding to $[1, 0]$ and ")" corresponding to $[0, 1]$), and f is defined as

$$f\left(\begin{bmatrix} x \\ y \end{bmatrix}\right) = \begin{cases} \begin{bmatrix} x \\ y \end{bmatrix} & \text{if } x \geq 0 \\ \begin{bmatrix} x \\ -1 \end{bmatrix} & \text{if } x < 0 \end{cases}$$

Lastly, we define a decision function that maps values of h_t to True or False as follows: if the second element $h_{t,2} = -1$, then the decision function outputs False because the string $x_1 x_2 \ldots x_t$ is not in BPL; otherwise, if $h_{t,2} \neq -1$, then the decision function outputs True because the string is in BPL.

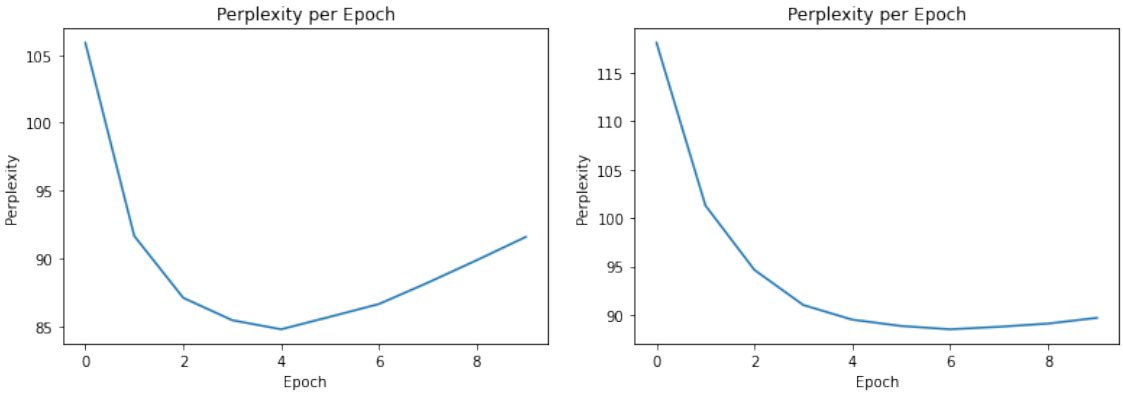
The decision function returns the correct output at any time. This works because for each prefix $x_1 x_2 \ldots x_k$, the first element in the hidden vector, $h_{k,1}$, keeps track of the running balance of parentheses, where each "(" contributes $+1$ and each ")" contributes -1. We can see this from the product of Wx_t:

$$W\begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1 \qquad W\begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} 0 \\ 1 \end{bmatrix} = -1$$

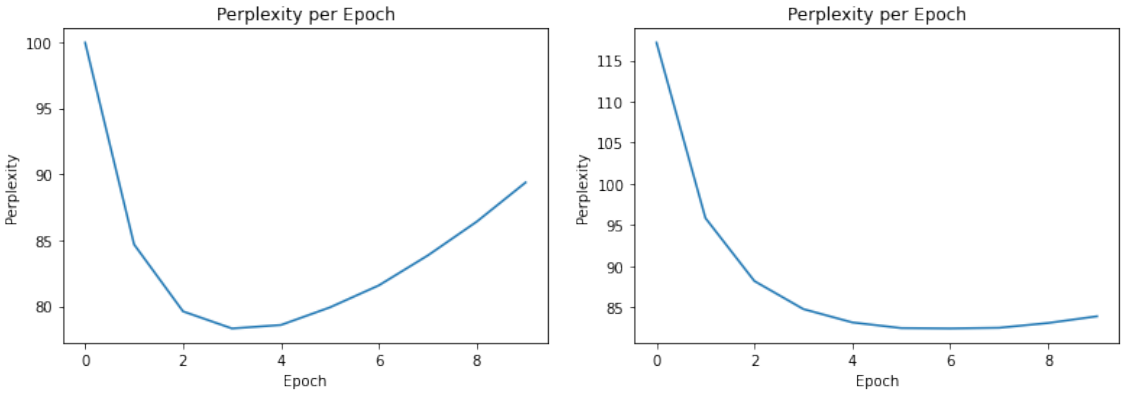
This, combined with U being the identity matrix such that $Uh_{k-1} = h_k$, maintains the sum in $h_{k,1}$ as each character is read on each iteration and its value added to the sum. If this sum ever becomes negative, this means that there are more ")" characters than "(" characters, and the current string (as well as its successors) should be classified as not in BPL, and the update function f accordingly sets the flag $h_{k,2}$ to -1. Otherwise, if the balance never goes below 0, then since h_0 is initialized to $[0, 0]$, f will never modify $h_{t,2}$, so the decision function will always output True (i.e. "in BPL").

(b) Implementation: our vanilla RNN language model consists of an RNNCell with an input size of 256 and a hidden size of 256. On the forward step, we initialize a hidden state (which is all zeroes) and look up the input embeddings using the input IDs. Then we pass the inputs and hidden state to the RNNCell in batches, and store the intermediate hidden states in a list. Finally, we return the hidden states as a stacked tensor.

Using an RNNCell with 256 hidden states results in a minimum perplexity of 84.770456 after 4 epochs. If we lower the number of hidden states to 128, we get a minimum perplexity of 88.514003 after 7 epochs. Finally, if we use a GRUCell instead of an RNNCell, we get a minimum perplexity of 78.330324 after 4 epochs, or — with 128 hidden states — a minimum perplexity of 82.412507 after 7 epochs.



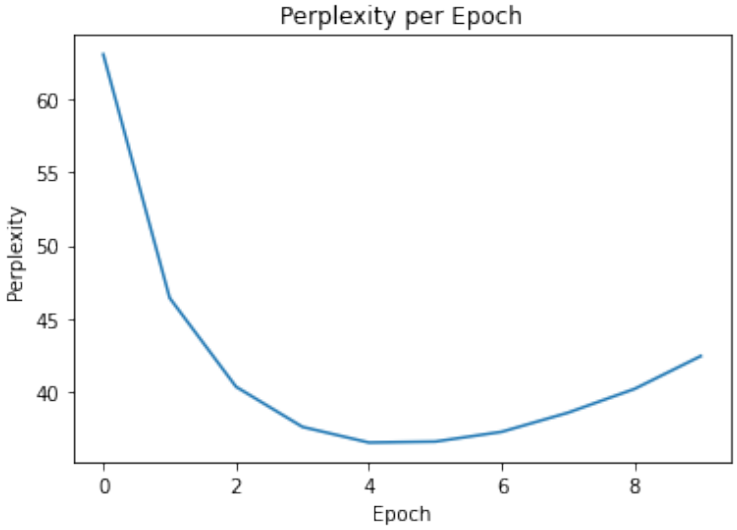
Hình 1: RNNCell Perplexity, left: 256 hidden states, right: 128 hidden states.



Hình 2: GRUCell Perplexity, left: 256 hidden states, right: 128 hidden states.

Part 3. (a) Implementation: our vanilla Seq2seq model consists of an encoder and decoder, both of which are RNNs. The encoder is a single-layer bidirectional GRU. The decoder consists of a single bidirectional GRU layer as well as a linear layer with `input size + hidden size` input features and `hidden size` output features, and it uses the final hidden state of the encoder to input as the initial hidden state for its GRU layer. Lastly, an encoder-decoder module wraps everything together. We use a batch size of 128, a hidden size of 256, and the Adam optimizer.

The minimum validation perplexity of the model is 36.536428 after 5 epochs, and the BLEU score is 6.8087. The perplexity per epoch is shown below.



(b) **Example 1:**

- Src: Họ viết gần 1000 trang về chủ đề này.
- Trg: They wrote almost a thousand pages on the topic.
- Pred: They invited a thousand TEDTalks.

In this example, "viết" (wrote) is wrongly translated as "invited", and "trang về chủ đề này" (pages on the topic) is wrong translated as "TEDTalks". It seems like the model does not have any words in its dictionary to capture the meaning of "pages on the topic", and the closest match is "TEDTalks", which are indeed lectures about specific topics. If the model had more training data, it might be able to differentiate between "pages on a topic" and "talks on a topic". This could also affect the other error; if the model had correctly translated "pages", it might have predicted "wrote", which is more likely for "pages", rather than "invited".

Example 2:

- Src: Và tất cả các trang đều được xem xét bởi 400 khoa học gia và nhà phê bình khác từ <unk> quốc gia.
- Trg: And all of those pages were reviewed by another <unk> scientists and <unk> , from <unk> countries.

- Pred: And all the time were being done by the British and the <unk> of the United Kingdom of <unk>.

In this example, "khoa học gia" (scientists) is mistranslated to "the British", and "quốc gia" (countries) is mistranslated to "the United Kingdom". The mistake here has two layers: the first is that the model couldn't figure out that "quốc gia" meant countries in general, not a specific country (i.e. it could detect that it meant a country, just not which one); the second is that it got the relationship between "scientists" and "countries" wrong. While it is true that scientists are from countries, and British people are from the United Kingdom, the two pairs do not have quite the same relationship. Perhaps the phrase "British people from the United Kingdom" appeared multiple times in the training data and skewed the model's results, or perhaps "scientists" appeared very infrequently, so the model didn't know how to recognize it. In either case, providing more training data (from a variety of sources) might help.