ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Natural Language Processing
Fall 2021

Prof. Ryan Cotterell

Course Assignment Episode 1

January 11, 2022

Deheng Zhang
nethz Username: dezhang
Student ID: 21-950-787

Collaborators:
Junling Wang
Tianyi Liu
Kailin Liu

By submitting this work, I verify that it is my own. That is, I have written my own solutions to each problem for which I am submitting an answer. I have listed above all others with whom I have discussed these answers.

Question 1: Backpropagation

(a) The function can be decomposed in following way:

$$
\begin{aligned}
y &= sigmoid(w_{11}^2 h_1 + w_{21}^2 h_2 + w_{31}^2 h_3) \\
h_1 &= ReLU(w_{11}^1 x_1 + w_{21}^1 x_2) \\
h_2 &= ReLU(w_{12}^1 x_1 + w_{22}^1 x_2) \\
h_3 &= ReLU(w_{13}^1 x_1 + w_{23}^1 x_2)
\end{aligned}
$$

The computational graph is shown in Figure 1:

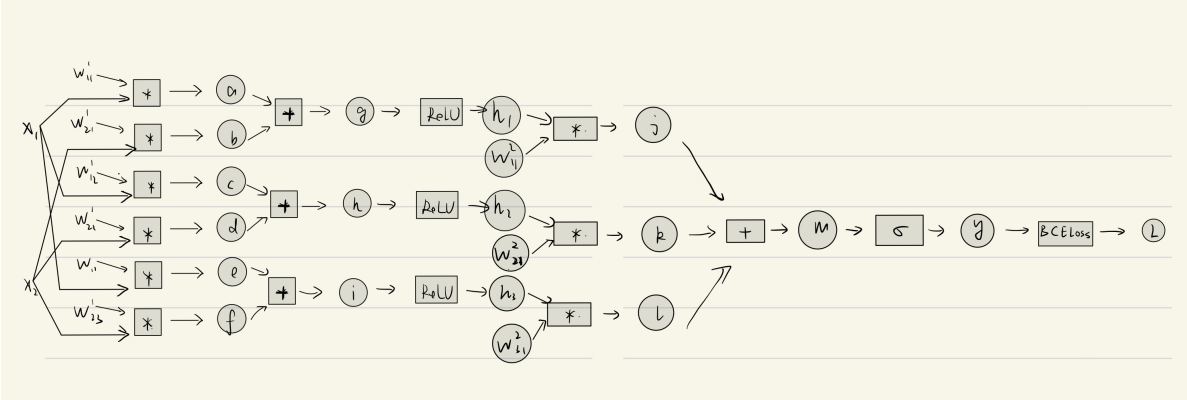

Figure 1: Computational Graph

(b) The forward and backward process is shown in Figure 2(forward in blue, backward in red).

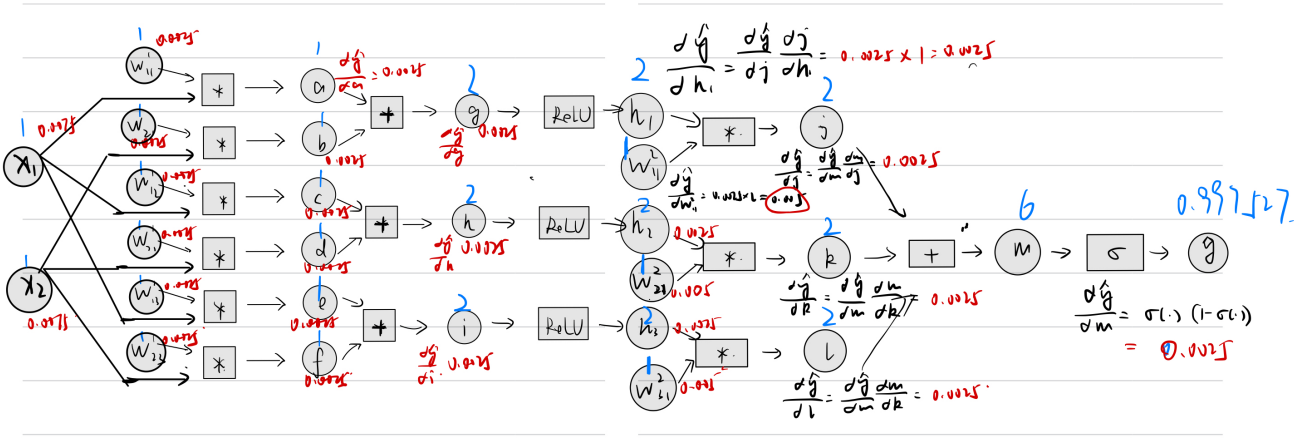

Figure 2: Computational Graph with Forward and Backward

(i) Evaluate the graph:

$$h_1 = ReLU(1 \times 1 + 1 \times 1) = 2$$
$$h_2 = ReLU(1 \times 1 + 1 \times 1) = 2$$
$$h_3 = ReLU(1 \times 1 + 1 \times 1) = 2$$
$$\hat{y} = Sigmoid(2 \times 1 + 2 \times 1 + 2 \times 1) = \frac{1}{1 + e^{-6}} \approx 0.997527$$

(ii) Compute the partial derivative for the network weights using backpropagation evaluated at $\mathbf{x_0}$.

$$\frac{\delta \hat{y}}{\delta m} = \sigma(m)(1 - \sigma(m)) \approx 0.0025$$
$$\frac{\delta \hat{y}}{\delta j} = \frac{\delta \hat{y}}{\delta k} = \frac{\delta \hat{y}}{\delta l} = \frac{\delta \hat{y}}{\delta m}\frac{\delta \hat{m}}{\delta j} \approx 0.0025$$
$$\frac{\delta \hat{y}}{\delta h_1} = \frac{\delta \hat{y}}{\delta h_2} = \frac{\delta \hat{y}}{\delta h_3} = \frac{\delta \hat{y}}{\delta j}\frac{\delta j}{\delta h_1} \approx 0.0025 \times 1 = 0.0025$$
$$\frac{\delta \hat{y}}{\delta w_{11}^2} = \frac{\delta \hat{y}}{\delta w_{21}^2} = \frac{\delta \hat{y}}{\delta w_{31}^2} = \frac{\delta \hat{y}}{\delta j}\frac{\delta j}{\delta w_{11}^2} \approx 0.0025 \times 2 = 0.005$$
$$\frac{\delta \hat{y}}{\delta g} = \frac{\delta \hat{y}}{\delta h} = \frac{\delta \hat{y}}{\delta i} = \frac{\delta \hat{y}}{\delta h_1}\frac{\delta h_1}{\delta g} \approx 0.0025 \times 1 = 0.0025$$
$$\frac{\delta \hat{y}}{\delta a} = \frac{\delta \hat{y}}{\delta b} = \frac{\delta \hat{y}}{\delta c} = \frac{\delta \hat{y}}{\delta d} = \frac{\delta \hat{y}}{\delta e} = \frac{\delta \hat{y}}{\delta f} == \frac{\delta \hat{y}}{\delta g}\frac{\delta g}{\delta a} \approx 0.0025 \times 1 = 0.0025$$
$$\frac{\delta \hat{y}}{\delta w_{11}^1} = \frac{\delta \hat{y}}{\delta w_{21}^1} = \frac{\delta \hat{y}}{\delta w_{12}^1} = \frac{\delta \hat{y}}{\delta w_{22}^1} = \frac{\delta \hat{y}}{\delta w_{12}^1} = \frac{\delta \hat{y}}{\delta w_{23}^1} = \frac{\delta \hat{y}}{\delta a}\frac{a}{\delta w_{11}^1} \approx 0.0025 \times 1 = 0.0025$$
$$\frac{\delta \hat{y}}{\delta x_1} = \frac{\delta \hat{y}}{\delta x_2} = \frac{\delta \hat{y}}{\delta a}\frac{\delta a}{\delta x_1} + \frac{\delta \hat{y}}{\delta c}\frac{\delta c}{\delta x_1} + \frac{\delta \hat{y}}{\delta e}\frac{\delta e}{\delta x_1} \approx 0.0025 \times 1 + 0.0025 \times 1 + 0.0025 \times 1 = 0.0075$$

(iii) Compute the calues of L_{BCE} and $\frac{\delta L_{BCE}}{\delta \hat{y}}$ at the point $(\mathbf{x_0}, y_0)$.

$$L_{BCE} = -ylog\hat{y} - (1 - y)log(1 - \hat{y}) \approx -log(1 - 0.997527) \approx 6.002476$$
$$\frac{\delta L_{BCE}}{\delta \hat{y}} = -y\frac{1}{\hat{y}} + (1 - y)\frac{1}{1 - \hat{y}} \approx 404.428793$$

(iv) Run a step of weight optimization over f following a standard gradient descent optimization procedure with learning rate $\eta = 0.1$:

$$w_{11}^{1\ *} = w_{12}^{1\ *} = w_{21}^{1\ *} = w_{22}^{1\ *} = w_{31}^{1\ *} = w_{32}^{1\ *} = w_{11}^1 - \eta\frac{\delta L}{\delta w_{11}^1}$$
$$= w_{11}^1 - \eta\frac{\delta L}{\delta \hat{y}}\frac{\delta \hat{y}}{\delta w_{11}^1} \approx 1 - 0.1 \times 404.428793 \times 0.0025 \approx 0.898893$$
$$w_{11}^{2\ *} = w_{21}^{2\ *} = w_{31}^{2\ *} = w_{11}^2 - \eta\frac{\delta L}{\delta w_{11}^2}$$
$$= w_{11}^2 - \eta\frac{\delta L}{\delta \hat{y}}\frac{\delta \hat{y}}{\delta w_{11}^2} = 1 - 0.1 \times 404.428793 \times 0.005 \approx 0.797786$$

(c) In this question, we can merge the sigmoid function and binary cross entropy according to the

following equation:

$$
\begin{aligned}
& -ylog(\frac{1}{1+e^{-m}}) - (1-y)log(\frac{e^{-m}}{1+e^{-m}}) \\
&= -y(log1 - log(1+e^{-m})) - (1-y)(-m - log(1+e^{-m})) \\
&= ylog(1+e^{-m}) + (1-y)m + (1-y)log(1+e^{-m}) \\
&= log(1+e^{-m}) + (1-y)m
\end{aligned}
$$

The modified computational graph is shown in Figure 3, the advantage of the modification is that the computational graph is easier to be understood by human, and the memory for dynamic programming is reduced (since we have less parameters). Besides, with less float number computation, the precision can be also increased. However, this computational graph cannot output the prediction $\hat{y}$, therefore, it can only be used during training, and for testing, we still need the original graph.

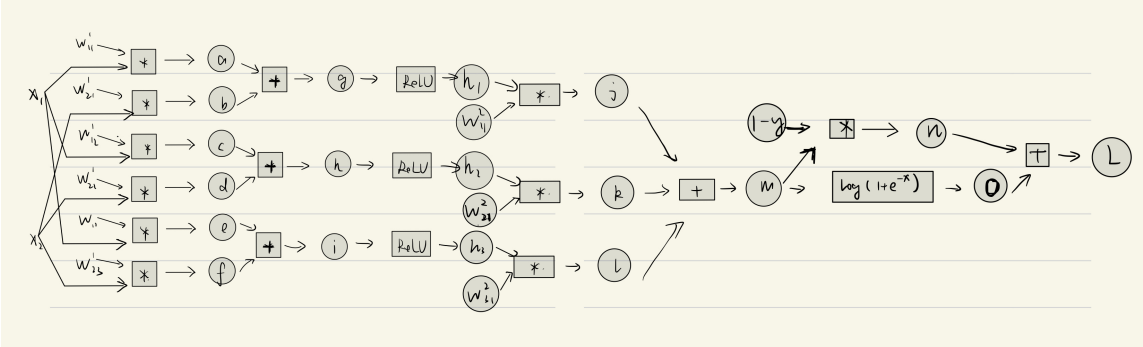

Figure 3: Modified Computational Graph

Question 2: Log-linear models

(a) Implementation(link):

- Feature function: the feature function is similar to the lecture note, but I add an homogeneous dimension for the intercept.

$$
f(x,1) = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix}, \qquad f(x,0) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}
$$

- The negative log likelihood loss(BCE) is implemented as the following equation:

$$
\sum_{i=1}^{N}(-\theta f(x_i, y_i) + log(\sum_{y' \in \mathcal{Y}} exp(\theta f(x_i, y_i'))))
$$

- The gradient of negative log likelihood is implemented as the following equation:

$$\sum_{i=1}^{N}(-f(x_i, y_i) + \sum_{y' \in \mathcal{Y}} f(x_i, y_i') \frac{exp(\theta f(x_i, y_i'))}{\sum_{y'' \in \mathcal{Y}} exp(\theta f(x_i, y_i''))})$$

- fixed learning rate η: after parameter tuning, I found 0.0001 is a great choise.

(b) Comparison

(i) Computation time: the result is shown in Figure 4. My implementation is slower than the sklearn version. I think this is because sklearn use less parameters comparing to my implementation. My implementation use an extended feature function, but for binary classification, the model only need one line to separate two classes.

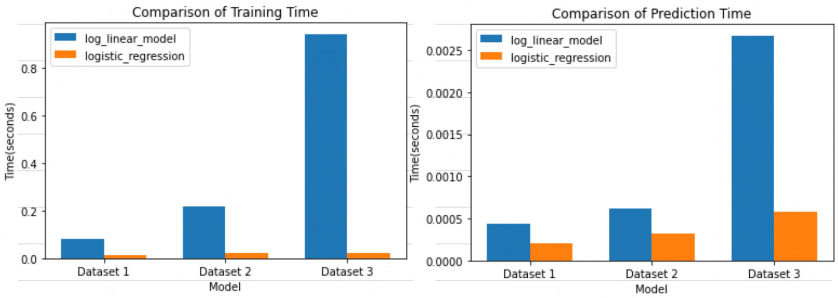

Figure 4: Comparison of running time

(ii) In-sample accuracy: The result is shown in Figure 5. We can find the difference of accuracy decrease as the size of dataset increases. This may be caused by the number of parameters. Since my model has a larger set of parameter, so it cannot converge when the data is not enough.

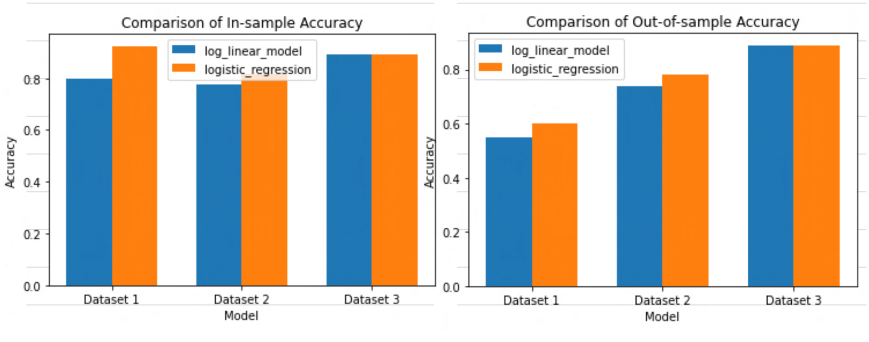

Figure 5: Comparison of accuracy

(iii) Out-of sample accuracy: The result is shown in Figure 5. We can see the similar effect as in-sample accuracy. As we can observe, both model overfits the dataset when the number of data is small.

(iv) Coefficient values: The result is shown in Figure 6. By observation, we can find the difference of parameters decrease as the size of the dataset increase. This is also because the models overfits the small dataset.

Question 3: Language modeling

Drawing token from vocabulary V independently and uniformly at random.

(a) Mary draws n tokens to form her corpus.

(i) What is the expected number of distinct tokens that appear?

Answer: Let W_i be the random variable that represents the existence of i-th word ($W_i = 1$ if the i-th word exists in the corpus). N represents the number of distinct words.

$$P(W_i = 0) = \left(\frac{|V|-1}{|V|}\right)^n, P(W_i = 1) = 1 - \left(\frac{|V|-1}{|V|}\right)^n$$

$$E[W_i] = 1 - \left(\frac{|V|-1}{|V|}\right)^n, where\ N = \sum_i^{|V|} W_i$$

$$E[N] = E[\sum_i^{|V|} W_i] = \sum_i^{|V|} E[W_i] = |V|(1 - \left(\frac{|V|-1}{|V|}\right)^n)$$

(ii) What is the probability that all $|V|$ of the words from the vocabulary have appeared?

Answer: If we choose random variable N to represent the number of distinct items. For probability of $N = k$ we select a set of k distinct words in $\binom{|V|}{k}$. If we only choose from the set, the probability is $\binom{k}{k}(\frac{k}{|V|})^n$. In order to get the probability that all item in the set existed, we can define the recursive relationship:

$$P(N = k) = \binom{|V|}{k}(P(n\ draws\ from\ set\ k) - P(n\ draws\ from\ true\ subset\ of\ k))$$

According to the inclusion-exclusion rule:

$$P(N = k) = \binom{|V|}{k} \sum_{d=0}^{k} (-1)^d \binom{k}{k-d} \left(\frac{k-d}{|V|}\right)^n$$

Thus, the probability of all existence equals to:

$$P(N = |V|) = \sum_{d=0}^{|V|} (-1)^d \binom{|V|}{|V|-d} \left(\frac{|V|-d}{|V|}\right)^n$$

(b) Mary wants the bigram *work hard* to appear in the corpus.

(i) What is the expected number of tokens that will be drawn from the vocabulary before this bigram appears?

Figure 6: Comparison of coefficient

Answer: In Figure 7, we can see the state transformation of the token. State O repre-

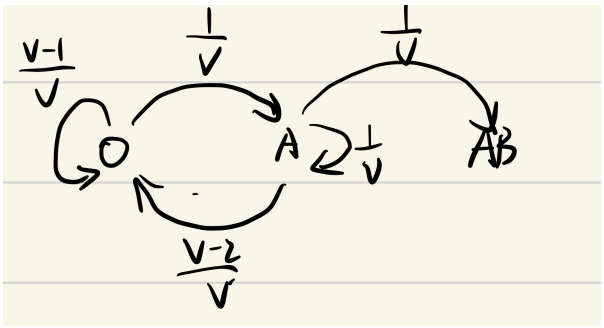

Figure 7: Markov chain for "work hard"

sents the first word is not *work*. State A represents the first word is *work*, but the second word is not sure. State AB represents the first word is *work* and the second word is *hard*. If we assume the expected number of token before the existence of two consecutive words is x, the expected number of token from state A to state AB is y, we have:

$$x = \frac{|V|-1}{|V|}(x+1) + \frac{1}{|V|}(y+1)$$
$$y = \frac{|V|-2}{|V|}(x+1) + \frac{1}{|V|}(y+1) + \frac{1}{|V|}1$$
$$E[N_{draws}] = x = |V|^2$$

(ii) Mary decided that this number is quite big. Instead, she would at least like to ensure that the word work appears in the text with probability $\geq 95\%$. What is the number of tokens she needs to draw before this happens?

Answer: If we choose random variable N to be the number of tokens drawn before this happens:

$$p(N \leq k) = 1 - \left(\frac{|V|-1}{|V|}\right)^k$$
$$1 - \left(\frac{|V|-1}{|V|}\right)^k \geq 0.95$$
$$\left(\frac{|V|-1}{|V|}\right)^k \leq 0.05$$
$$k\log\frac{(|V|-1)}{|V|} \leq \log 0.05$$
$$k \geq \frac{\log 0.05}{\log(|V|-1) - \log|V|}$$

(c) One thing Mary dislikes is when the same words appear next to each other in her corpus.

(i) What is the expected number of draws before two of the same tokens appear next to each other?

Answer: If we assume the expected number of token before the existence of two same

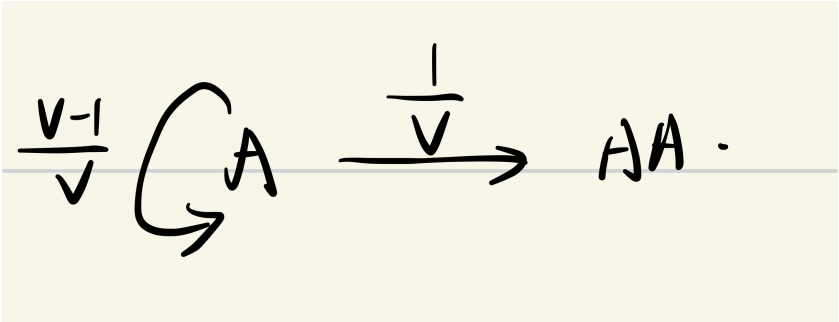

Figure 8: Markov chain for two consecutive words

words is $x + 1$, we have:

$$x = \frac{|V| - 1}{|V|}(x + 1) + \frac{1}{|V|}(1)$$
$$x = |V|$$
$$E[N_{draws}] = |V| + 1$$

(ii) Mary doesn't trust her computation from the previous part and decides to im- plement a neural network that, at each step, will output the number of bigrams encountered so far that consist of two of the same token. She proceeds as follows. First, she builds a network that will output 1 whenever the current bigram (consisting of the previous word and the current one) contains two of the same token. She uses the architecture from the figure below. Your job is to help her find the unknowns. Formally, we have

$$h_t^0 = f(w_0 x_{t-1} + w_1 x_t + w_2 h_{t-1}^0 + b_0)$$
$$y_t = g(w_3 h_t^0 + b_1)$$

Find all of f, g, w_i for $i \in \{1, 2, 3\}$ and b_j for $j \in \{0, 1\}$. What kind of activation function f is needed and why? Can we set f to be a linear function of the input?

Answer: Here we set:

$$w_0 = 1, w_1 = -1, w_2 = 0, w_3 = 1, b_0 = 0, b_1 = 0, g(x) = x$$
$$f = \begin{cases} 1 & x = 0 \\ 0 & \text{otherwise} \end{cases}$$

f should be a non-linear function to add non-linearity to the MLP, otherwise it would be equivalent to a single layer perceptron. We cannot set f to be a linear equation generally. However, in this particular problem, we can swap g and f and the result remains the same (not work in question iii).

(iii) Now she would actually like to finish her problem and build what is needed using the architecture from the figure below. As in the previous problem, your job is to find the unknowns. Where we can formally define the components of the network as

$$h_t^0 = f(w_0 x_{t-1} + w_1 x_t + w_2 h_{t-1}^0 + b_0)$$
$$h_t^1 = g(w_3 h_{t-1}^1 + w_4 h_t^0 + b_1)$$
$$y_t = h(w_5 h_t^1 + b_2)$$

Answer: Here we set:

$$w_0 = 1, w_1 = -1, w_2 = 0, w_3 = 1, w_4 = 1, w_5 = 1$$
$$b_0 = 0, b_1 = 0, b_2 = 0, g(x) = x, h(x) = x$$
$$f = \begin{cases} 1 & x = 0 \\ 0 & \text{otherwise} \end{cases}$$

(iv) Mary is considering having a non-uniform unigram distribution to avoid the problem of seeing two of the same tokens next to each other. Consider a single bigram. Which model has a bigger chance of sequentially drawing two of the same token, a uniform unigram language model or a non-uniform unigram language model?

Answer: The non-uniform unigram model has a bigger chance of sequentially drawing two of the same token. If we think about the extreme case, the probability of one word is 1, and for all other words are 0. Then the chance of sequentially drawing two of the same token is highest. Actually for non-uniform distribution, we focus on fewer number of words compared to uniform distribution, thus has a bigger chance.
Proof: Define two vectors that land on the Simplex $\mathbf{p_1}, \mathbf{p_2} \in \Delta^{|V|-1}$, such that:

$$\mathbf{p_1} = [\frac{1}{|V|}, \frac{1}{|V|}, \frac{1}{|V|}, ..., \frac{1}{|V|}]$$
$$\mathbf{p_2} = [P(w_1), P(w_2), P(w_3), ..., P(w_{|V|})]$$

Then, we have:

$$(\mathbf{p_1}\mathbf{p_2})^2 \leq |\mathbf{p_1}|^2 |\mathbf{p_2}|^2$$
$$(\frac{1}{|V|}\sum_{i=1}^{|V|} P(w_i))^2 \leq \sum_{i=1}^{|V|} \frac{1}{|V|^2} \sum_{i=1}^{|V|} P^2(w_i)$$
$$\sum_{i=1}^{|V|} P^2(w_i) \geq \frac{1}{|V|}$$

The equality is obtained when $\mathbf{p_1} = \mathbf{p_2}$ (i.e. the distribution is uniform). Therefore, the probability of drawing two same token is higher when the distribution is non-uniform.

Question 4: Dijkstra's algorithm

(a) Decoding problem:

$$\mathbf{y}^* \longleftarrow \mathbf{argmax}_{\mathbf{y}' \in \mathcal{Y}}\ \mathbf{score}(\mathbf{y}', \mathbf{x})$$

(i) Consider a first order CRF that outputs scores over sequences in the output space $\mathcal{Y}$ of length N, where the sequence consists of elements from some set $\mathcal{Y}$, e.g. POS-tags:

$$p(\mathbf{y}|\mathbf{x}) = \frac{exp\{\sum_{n=1}^{N} score(< y_{n-1}, y_n >, \mathbf{x})\}}{\sum_{\mathbf{y}' \in \mathcal{Y}} exp\{\sum_{n=1}^{N} score(< y'_{n-1}, y'_n >, \mathbf{x})\}}$$

Answer: We denote the set of possible tags as $\mathcal{Y}$, then we the vertices, edges, and weights can be represented as following:

$$
\begin{aligned}
&V_{BOS}, V_{EOS} : \textit{the starting and ending vertex.}\\
&V_j^{(i)} : \mathbf{y_i} = \mathbf{t_j}, \mathbf{t_j} \in \mathcal{Y}(\textit{tag of the ith word is } t_j)\\
&E = \{(V_j^{(i)}, V_k^{(i+1)}) | i \in \{0,1,...,N-1\}; j \in \{0,1,...,|\mathcal{Y}|-1\}; k \in \{0,1,...,|\mathcal{Y}|-1\}\}\\
&\quad \cup \{(V_{BOS}, V_j^{(0)}) | j \in \{0,1,...,|\mathcal{Y}|-1\}\} \cup \{(V_j^{(N-1)}, V_{EOS}) | j \in \{0,1,...,|\mathcal{Y}|-1\}\}\\
&W((V_j^{(i)}, V_k^{(i+1)})) = exp\{-score(< y^i = t_j, y_{i+1} = t_k >, \mathbf{x})\}\\
&W((V_{BOS}, V_j^{(0)})) = W((V_j^{(N-1)}, V_{EOS}) = 1
\end{aligned}
$$

In this way, the decoding problem ($argmax$) can be transformed to a shortest path problem ($argmin$), and the path length equals to the multiplication of the edge weights, the algorithm is shown below:

Algorithm 1 Prioritized Dijkstra's Algorithm for Decoding

```
function DIJKSTRA(vertices, adjacency_list, source)
    priority_queue ← []
    distance ← {}
    parent ← {}
    for vertex in vertices do
        distance[vertex] ← ∞
        parent[vertex] ← None
        priority_queue.enqueue(vertex)
    end for
    distance[source] ← 1
    while priority_queue not empty do
        current_vertex ← vertex in queue with min distance[vertex]
        priority_queue.dequeue(current_vertex)
        for neighbor in adjacency_list[current_vertex] and priority queue do
            new_distance = distance[current_vertex] × edge_weight(current_vertex, neighbor)
            if new_distance < distance[neighbor] then
                distance[neighbor] ← new_distance
                parent[neighbor] ← current_vertex
                priority_queue.enqueue(neighbor)
            end if
        end for
    end while
     return distance, parent
```

(ii) The time complexity of the algorithm in terms of the cardinality of the sets V and E? Compare this to the time complexity of the Viterbi algorithm. Is either strictly better than the other?

Answer: The time complexity of Dijkstra algorithm:

- Initialization and enqueue $O(|V|log|V|)$.

- Dequeue for each vertex $O(|V|log|V|)$.
- We need to iterate all edges, for each edge, it is possible insert a new vertex into the queue $O(|E|log|V|)$.
- $O((|V|+|E|)log|V|)$ in total, but can be improved to $O(|V|+|E|log|V|)$ with Fibonacci heap.
- The number of vertices is $N|\mathcal{Y}|+2$, and the number of edges is $2|\mathcal{Y}|+(N-1)|\mathcal{Y}|^2$. Therefore, the complexity is $O(N|\mathcal{Y}|^2 log(N|\mathcal{Y}|))$

The time complexity of Viterbi algorithm is $O(N|\mathcal{Y}|^2)$. Generally, $O(log(N|\mathcal{Y}|)) > O(1)$, therefore, the performance of Viterbi is strictly better than Dijkstra.

(b) Semiring-ify Dijkstra's algorithm:

(i) Rewrite the pseudocode from part a(i) in semiring notation and identify the semiring that was used.
Answer: The generalized Dijkstra algorithm on semi-ring $(A, +, \times, \bar{0}, \bar{1})$. Notice that the edge weight between vertices and BOS/EOS should be defined as $\bar{1}$ according to the specific semi-ring.

Algorithm 2 Generalized Dijkstra's Algorithm for Decoding

function DIJKSTRA(vertices, adjacency_list, source)
  $priority\_queue \leftarrow []$
  $distance \leftarrow \{\}$
  for `vertex in vertices` **do**
    $distance[vertex] \leftarrow \bar{0}$
    $parent[vertex] \leftarrow None$
    $priority\_queue.enqueue(vertex)$
  end for
  $distance[source] \leftarrow \bar{1}$
  while `priority_queue not empty` **do**
    $current\_vertex \leftarrow$ *vertex in queue with min* $distance[vertex]$
    $priority\_queue.dequeue(current\_vertex)$
    for `neighbor in adjacency_list[current_vertex] and priority queue` **do**
      $new\_distance = distance[current\_vertex] \bigotimes edge\_weight(current\_vertex, neighbor)$
      if $\bigoplus(new\_distance, distance[neighbor]) \neq distance[neighbor]$ **then**
        $distance[neighbor] \leftarrow new\_distance \bigoplus distance[neighbor]$
        $parent[neighbor] \leftarrow current\_vertex$
        $priority\_queue.enqueue(neighbor)$
      end if
    end for
  end while
  return *distance*

The semi-ring used in (a) is $(\mathbb{R}^+ \cup \{\infty\}, min, \times, \infty, 1)$. To prove this is a semi-ring:

i. $(\mathbb{R}^+ \cup \{\infty\}, min, \infty)$ is a commutative monoid.

ii. $(\mathbb{R}^+ \cup \{\infty\}, \times, 1)$ is a monoid

iii. for $a, b, c \in \mathbb{R}^+ \cup \{\infty\}$

$$min(a,b) \times c = min(a \times c, b \times c)$$
$$c \times min(a,b) = min(c \times a, c \times b)$$

iv. for $a \in \mathbb{R}^+ \cup \{\infty\}$, $\infty \times a = a \times \infty = \infty$

(ii) *Answer*: The longest path can be found by the following semi-ring:

$$(\mathbb{R}^- \cup \{-\infty\}, max, +, -\infty, 0), s.t.$$
$$(\mathbb{R}^- \cup \{-\infty\}, max, -\infty) \textit{ is a commutative monoid.}$$
$$(\mathbb{R}^- \cup \{-\infty\}, +, 0) \textit{ is a monoid.}$$
$$\forall a, b, c \in \mathbb{R}^- \cup \{-\infty\}\ max(a,b) \times c = max(a \times c, b \times c)$$
$$\forall a \in \mathbb{R}^- \cup \{-\infty\}, -\infty + a = a + (-\infty) = -\infty$$

Notice that the order in priority queue should be reversed (larger distance has higher priority).

(iii) *Answer*: The widest path can be found by following semi-ring:

$$(\mathbb{R}^+, max, min, 0, \infty), s.t.$$
$$(\mathbb{R}^+, max, 0) \textit{ is a commutative monoid.}$$
$$(\mathbb{R}^+, min, \infty) \textit{ is a monoid.}$$
$$\forall a, b, c \in \mathbb{R}^+\ min(max(a,b), c) = max(min(a,c), min(b,c))$$
$$c > a, c > b : min(max(a,b), c) = max(min(a,c), min(b,c)) = max(a,b)$$
$$c < a, c < b : min(max(a,b), c) = max(min(a,c), min(b,c)) = c$$
$$a < c < b, b < c < a : min(max(a,b), c) = max(min(a,c), min(b,c)) = c$$
$$\forall a \in \mathbb{R}^+, min(a,0) = min(0,a) = 0$$

For each path, the weight is calculated as the minimum edge weight, and the maximum path weight is selected. The path information is recorded in the *parent* array.