Natural Language Processing - Part I
22 June 2023

First and last name, Student ID: ______________________________

Write part I in a <u>different</u> sheet of paper than part II so we can <u>split</u> them when grading.

1. Given the following quote from Albert Einstein:

```
we cannot solve problems with the kind of thinking we employed when we came up with them.
```

(a) Perform token learning with the byte-pair encoding (only 4 iterations). For each iteration, write down: the selected most frequent pair, the dictionary, and the learned rule set. $\boxed{8}$

Solution

Init Step: split words by punctuation, count the word token occurrences, and create the initial vocabulary:

occurrences		dictionary	rule set
3	w e _	_, a, b, c, d, e, h,	
1	c a n n o t _	i, k, l, m, n, o, p,	
1	s o l v e _	r, s, t, u, v, w, y	
1	p r o b l e m s _		
1	w i t h _		
1	t h e _		
1	k i n d _		
1	o f _		
1	t h i n k i n g _		
1	e m p l o y e d _		
1	w h e n _		
1	c a m e _		
1	u p _		
1	w i t h _		
1	t h e m _		

Iteration 1: most frequent pair (**e**, _) :

occurrences		**dictionary**	**rule set**
3	w **e_**	_, a, b, c, d, e, h,	(e, _)
1	c a n n o t _	i, k, l, m, n, o, p,	
1	s o l v **e_**	r, s, t, u, v, w, y,	
1	p r o b l e m s _	**e_**	
1	w i t h _		
1	t h **e_**		
1	k i n d _		
1	o f _		
1	t h i n k i n g _		
1	e m p l o y e d _		
1	w h e n _		
1	c a m **e_**		
1	u p _		
1	w i t h _		
1	t h e m _		

Iteration 2: most frequent pair (**t**, **h**) :

occurrences		dictionary	rule set
3	w **e_**	_, a, b, c, d, e, h,	(e, _)
1	c a n n o t _	i, k, l, m, n, o, p,	(t, h)
1	s o l v **e_**	r, s, t, u, v, w, y,	
1	p r o b l e m s _	**e_**, **th**,	
1	w i t h _		
1	**th** **e_**		
1	k i n d _		
1	o f _		
1	**th** i n k i n g _		
1	e m p l o y e d _		
1	w h e n _		
1	c a m **e_**		
1	u p _		
1	w i **th** _		
1	**th** e m _		

Iteration 3: most frequent pair (**w**, **e_**) :

occurrences		dictionary	rule set
3	**we_**	_, a, b, c, d, e, h,	(e, _)
1	c a n n o t _	i, k, l, m, n, o, p,	(t, h)
1	s o l v **e_**	r, s, t, u, v, w, y,	(w, e_)
1	p r o b l e m s _	**e_**, **th**, **we_**,	
1	w i **th** _		
1	**th** **e_**		
1	k i n d _		
1	o f _		
1	**th** i n k i n g _		
1	e m p l o y e d _		
1	w h e n _		
1	c a m **e_**		
1	u p _		
1	w i **th** _		
1	**th** e m _		

Iteration 4: most frequent pair (**w**, **e_**) :

occurrences		dictionary	rule set
3	**we_**	_, a, b, c, d, e, h,	(e, _)
1	c a n n o t _	i, k, l, m, n, o, p,	(t, h)
1	s o l v **e_**	r, s, t, u, v, w, y,	(w, e_)
1	p r o b l e m s _	**e_**, **th**, **we_**,	
1	w i **th** _		
1	**th** **e_**		
1	k i n d _		
1	o f _		
1	**th** i n k i n g _		
1	e m p l o y e d _		
1	w h e n _		
1	c a m **e_**		
1	u p _		
1	w i **th** _		
1	**th** e m _		

Iteration 5: most frequent pair (**e**, **m**) :

occurrences		**dictionary**	**rule set**
3	**we_**	_, a, b, c, d, e, h,	(e, _)
1	c a n n o t _	i, k, l, m, n, o, p,	(t, h)
1	s o l v **e_**	r, s, t, u, v, w, y,	(w, e_)
1	p r o b l **em** s _	**e_**, **th**, **we_**,	(e, m)
1	w i **th** _	**em**	
1	**th** **e_**		
1	k i n d _		
1	o f _		
1	**th** i n k i n g _		
1	e m p l o y e d _		
1	w h e n _		
1	c a m **e_**		
1	u p _		
1	w i **th** _		
1	**th** **em** _		

Total for Question 1: 8

2. Given the corpus $D = \{d_1, d_2, d_3\}$, where:
d_1="never gonna give you up"
d_2="never gonna let you down"
d_3="never gonna run around and desert you"

(a) Write the tfidf formula, and explain the idea behind the formula: $\boxed{4}$

Solution:
$tfidf(w_i, d_j) = tf(w_i, d_f) \times idf(w_i)$
$tf(w_i, d_j) = \frac{\#\ of\ occurrences\ of\ w_i\ in\ d_j}{\#\ number\ of\ word\ tokens\ in\ d_j}$
$idf(w_i) = log_{10}(\frac{|D|}{|d : w_i \in d, d \in D|})$

(b) For each document, compute the tfidf for the words "never" and "up": $\boxed{4}$

Solution: $tf(never, d_1) = \frac{1}{5}$
$tf(never, d_2) = \frac{1}{5}$
$tf(never, d_3) = \frac{1}{7}$
$idf(never) = log_{10}(\frac{3}{3})$
$tfidf(never, d_1) = tf(never, d_1) \times idf(never) = \frac{1}{5} \times log_{10}(\frac{3}{3})$
$tfidf(never, d_2) = tf(never, d_2) \times idf(never) = \frac{1}{5} \times log_{10}(\frac{3}{3})$
$tfidf(never, d_3) = tf(never, d_3) \times idf(never) = \frac{1}{7} \times log_{10}(\frac{3}{3})$

$tf(up, d_1) = \frac{1}{5}$
$tf(up, d_2) = \frac{0}{5}$
$tf(up, d_3) = \frac{0}{7}$
$idf(up) = log_{10}(\frac{3}{1})$
$tfidf(up, d_1) = tf(up, d_1) \times idf(up) = \frac{1}{5} \times log_{10}(\frac{3}{1})$
$tfidf(up, d_2) = tf(up, d_2) \times idf(up) = \frac{0}{5} \times log_{10}(\frac{3}{1})$
$tfidf(up, d_3) = tf(up, d_3) \times idf(up) = \frac{0}{7} \times log_{10}(\frac{3}{1})$

Total for Question 2: 8

Natural Language Processing - Part II
22 June 2023

3. Answer the following questions by reporting the mathematical procedure, if needed. If you have to compute the actual value, please write the procedure that leads you to the numerical values.

   Write part I in a <u>different</u> sheet of paper than part II so we can <u>split</u> them when grading.

   (a) When building NLP applications is important to be aware to which extent different NLP machinery can capture short or long-term dependencies. Below you find different NLP tools we encountered in the course. Connect each tool with the number of word token relationships they capture at inference time. That is, `10` means the tool captures relationships with word token 10 word apart. $\boxed{1}$

Elman RNN	LSTM	word2vec	GPT2
100	1	10	2^{10}

   Solution: word2vec with 1; Elman RNN with 10; LSTM RNN with 100; GPT2 with 2^{10}.

   (b) Let's assume that you have a vocabulary V made of a million word tokens and you have a word embedding matrix $\mathbf{E} \in \mathbb{R}^{D\times|V|}$ where D is the embedding dimension. Let us also assuming that the word `salt` is at index $i = 100$ of V. Describe with linear algebra how you can get the word embedding of the word `salt` from $\mathbf{E}$. $\boxed{1}$

   Solution: This can be done by creating a one-hot encoding vector $\mathbf{x}$ of all zeros except a 1 at position i of $\mathbf{x}$. That is $\mathbf{x}[i] = 1$ and $\mathbf{x}[j] = 0 \quad \forall j \in |V|\ j \neq i$.
   Given the shape of $\mathbf{E}$, $\mathbf{x}$ as to be $\in \mathbb{R}^{|V|\times 1}$. We can retrieve $\mathbf{e}_{\text{salt}} = \mathbf{E}\mathbf{x}$ with matrix multiplication. For each new component $d \in D$, this will zero out all the other word-embedding dimensions except a 1 on the salt embedding. The matmul will repeat this $\forall d \in D$ times automatically so that $\mathbf{e}_{\text{salt}} \in \mathbb{R}^{D\times 1}$.

   (c) Consider you have two corpora $\mathcal{C}_x$ and $\mathcal{C}_y$ of text of the same length: N word tokens. They share the same vocabulary V. A corpus is generated randomly selecting word tokens in V randomly with uniform distribution with replacement. The second corpus is a common dataset used to train `Continuous Bag-of-Words CBOW word2vec`. Assume you have the parameters of `word2vec` trained on this common dataset. You also know the window-size T used to train it. $\boxed{2}$
   Describe a procedure to automatically discern the two corpora.

   Solution: There might be multiple ways to do it but one is to exploit the fact that word2vec learned the distribution of the common dataset that is encoded in its parameters, while if we try to assess the likelihood of word2vec on the random dataset, it may have low likelihood (or else high loss). We being to recall that `CBOW w2v` has two set of parameters, the context vector embedding $\boldsymbol{\theta}_C$ and center word $\boldsymbol{\theta}_W$. $\forall i$ word token in the text, given a generic position i, we compute the average vector selecting from $\boldsymbol{\theta}_C$ the context embedding of the window, i.e.

$$\widehat{\boldsymbol{\theta}_C} = \frac{1}{2T-1} \sum_{\substack{j\in[i-T,i+T] \\ j\neq i}} \boldsymbol{\theta}_{Cj}.$$

   We then matmul $\mathbf{z} = \boldsymbol{\theta}_W \widehat{\boldsymbol{\theta}_C}$ and pass $\mathbf{z}$ to softmax $\mathbf{p}$ and compute the likelihood $\log(\mathbf{p})_i$ where i selects the index of the current center world. So to decide simply compute $\ell_x = \sum_{n=1}^{N} \log(\mathbf{p})_i$ over the first corpus and compute $\ell_y = \sum_{n=1}^{N} \log(\mathbf{p})_i$ over the second corpus. N runs over the total word tokens of the texts.
   Then classify the corpora as: If $\ell_x > \ell_y$ then first corpus is $\mathcal{C}_x$ is common dataset and $\mathcal{C}_y$ random; else $\mathcal{C}_x$ is random and $\mathcal{C}_y$ the common dataset.

Total for Question 3: 4

4. We are given a **Elmann RNN** trained at the character level. The input squence of character is shown in Fig. 1. The vocabulary is `['i','o','c','a','!']` and the associated char embeddings are `[-2,-1,2,1,0]`. The details of the RNN are:

- hidden layer $W_h = -1$, hidden layer input $W_x = 2$, hidden layer classification $W_y = -1$. The activation functions are all $\sigma(x) = x$. There are NO biases.
- RNN is trained with self-supervision similar to GPT2. Instead of cross-entropy, it regresses the char embedding of the next char token with a loss as $\big(f(z) - y\big)^2$ where $f(z)$ is the output of the RNN and y is the next char embedding. The final loss is $\ell = \sum_{i=1}^{4} \ell_i$

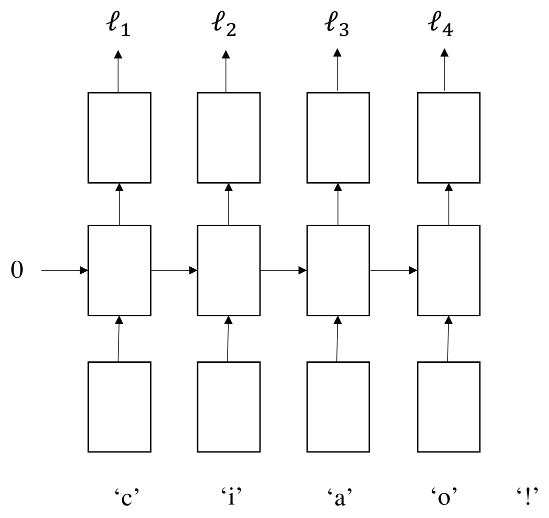

Figure 1: Elman network considered.

(a) $\boxed{4}$
- Write down the generic equation for all the state update for the model shown in Fig. 1 at a generic time step.
- Compute the numerical value of $\frac{\partial \ell}{\partial W_h}$, that is the partial derivative of the loss wrt to the hidden layer W_h, showing all the derivation with a computational graph by breaking it down to forward pass and backward pass. I strongly suggest you rebuild the graph on paper.

Solution: For the equation of RNN see NLP slides. For the computational graph is similar to what we have see in class in the last lecture. The tricky part is to remember that W_h has four arrows entering the weights, because W_h is used 4 times. so in total we have to sum up 4 gradients over W_h. All proceeds similar to the esample in class, just remember that when you compute $\frac{\partial \ell}{\partial h_t}$ it receives two gradients as input (from the regression layer above and from the next h_{t+1} from the right. Once you have summed those you have $\frac{\partial \ell}{\partial h_t}$. So the local gradient at time t on W_h is $\frac{\partial \ell}{\partial W_h} = \frac{\partial \ell}{\partial h_t}\frac{\partial h_t}{\partial W_h}$ where $\frac{\partial h_t}{\partial W_h} = \frac{\partial \sigma(W_h h_{t-1} + W_x x_t)}{\partial W_h} = h_{t-1}$ but you have to add it four times for each branch.

(b) Assume you have a vocabulary defined as `['i','o','c','a','!']`. You are at the end of classification layer of the RNN at time step n. The RNN this time models vectors and works with cross-entropy loss and softmax trained with self-supervision. You observe that the gradient of the loss at the branch n respect to the logit $\mathbf{z}$ is $\nabla_{\mathbf{z}} \ell_n(w_1, \ldots, w_n; \mathbf{z}) = [0.1, 0.1, 0.2, -0.65, 0.25]$. $\boxed{1}$
- State if you can recover which is the **ground-truth char token** set as supervision to the RNN at this step. Motivate and describe the process to recover it, if possible.
- Moreover, describe how you could retrieve and compute the value of the loss at this time step, if possible.

Solution: The sign of the gradient $\nabla_{\mathbf{z}} \ell_n(w_1, \ldots, w_n; \mathbf{z})$ will tell us which is the next char token to predict, in particular the negative sign. So in this case the ground-truth char token is `'a'`. The probability of `'a'` is simply $-0.65 + 1$ which is 0.35. From this we can recover the loss at that branch that is $-\log(0.35)$. The minus is coming from deriving the cross-entropy loss and softmax wrt to the logit $\mathbf{z}$. The gradient is the probabilities itself on all the value except in the ground-truth index: there the gradient is g=-1+p so to recover p simply g+1=p. Given:

$$\begin{aligned}
\text{Logit:}\quad & z \\
\text{Softmax:}\quad & \text{softmax}(z)_j = \frac{e^{z_j}}{\sum_k e^{z_k}} \\
\text{Cross-Entropy Loss:}\quad & \text{CE}(p,q) = -\sum_j p_j \log(q_j)
\end{aligned}$$

The gradient of the loss with respect to the logit z is:

$$\begin{aligned}
\frac{\partial \text{CE}(p, \text{softmax}(z))}{\partial z_i} &= \frac{\partial}{\partial z_i}\left(-\sum_j p_j \log\left(\frac{e^{z_j}}{\sum_k e^{z_k}}\right)\right) \\
&= -\sum_j p_j \frac{\partial}{\partial z_i}\left(\log\left(\frac{e^{z_j}}{\sum_k e^{z_k}}\right)\right) \\
&= -\sum_j p_j \frac{\partial}{\partial z_i}\left(\log(e^{z_j}) - \log\left(\sum_k e^{z_k}\right)\right) \\
&= -\sum_j p_j \left(\frac{\partial}{\partial z_i} z_j - \frac{\partial}{\partial z_i}\log\left(\sum_k e^{z_k}\right)\right) \\
&= -p_i + \frac{e^{z_i}}{\sum_k e^{z_k}} \cdot \sum_j p_j \\
&= \text{softmax}(z)_i - p_i
\end{aligned}$$

Hence, the gradient of the cross-entropy loss with respect to the logit z is given by $\text{softmax}(z)_i - p_i$. In our case p_i is one-hot encoding over `'a'`. Another possible solution that I read in solution from students is to consider definition of the gradient and look at the negative sign, so perturb positively the 'a' logit makes the loss go down.

(c) Assume you have an RNN model similar to the one in Fig. 1 that works over word tokens and is NOT a toy example. You need to train it to predict if an exam is perceived easy or difficult by student—binary classification—given textual comments of students about previous exams. $\boxed{1}$

Consider the case where you have a dataset in which cases similar to the following, or variations thereof, appear often: `''The computer science exam was insanely easy''`.

Explain what would you change in the RNN model.

Solution: The RNN should be changed so to incorporate also **bidirectional context** so that the model naturally captures dependency in the text not only from left to right

```
science exam was insanely |--> easy
```

but also from right to left:

```
science exam was insanely <--| easy
```

Update: in my previous solution I did not noticed that you need to change also the loss so that you have a single loss at the end of the network or you do pooling of all the features. I gave extra points to those who wrote that.

Total for Question 4: 6

5. Consider a transformers network with the learnable self-attention mechanism with a single head with residual connection without layer normalization. It receives as input word embeddings $\mathbf{X} \in \mathbb{R}^{D\times N}$ where D is the word embedding dimensions and N is the number of tokens.

   (a) Write the equation of the learnable self-attention mechanism with a single head with liner algebra, in function of the given input $\mathbf{X}$, writing down all the parameters that you need to learn and explain what is their role. Pay attention to the fact that matrices size should match. $\boxed{2½}$

   Solution: There are variations of solutions. First solution is:

$$\underbrace{\mathbf{O}}_{N\times D} = \text{softmax}\left(\underbrace{\mathbf{X}^\top \mathbf{W}_q \mathbf{W}_k \mathbf{X}}_{N\times N}\right) \underbrace{\mathbf{X}^\top \mathbf{W}_v}_{N\times D} + \mathbf{X}^\top$$

   Another way could be:

$$\underbrace{\mathbf{O}}_{D\times N} = \underbrace{\mathbf{W}_v \mathbf{X}}_{D\times N} \text{softmax}\left(\underbrace{\mathbf{X}^\top \mathbf{W}_v \mathbf{W}_q \mathbf{X}}_{N\times N}\right) + \mathbf{X}$$

   (b) Based on the definition that you gave before, state if your self-attention matrix should sum up to 1 across rows or across columns or both, explaining the motivation. $\boxed{1½}$

   Solution: In the first solution the self-attention matrix given by softmax should sum up to 1 by columns: if we fix a row, and run across columns and sum that should be one. The second solution should sum up across rows. The motivation for both is that by doing that for each of the dimension D you take a convex linear combination given but the self-attention weights.

   (c) A transformer is used also in the Contrastive Language-Image Pre-training (CLIP) model. Let's assume that you have an image $\mathtt{x}$ of a scene and need to use CLIP to make an automatic decision if a particular object `<object>` is in the image. You can assume the `<object>` size is big enough so that its visual features can be captured well by CLIP. $\boxed{2}$

   - State if you can use CLIP to solve this problem.
   - If you answered yes then briefly describe why and how can you get a prediction if the object is in the scene.
   - If you replied no, then say why CLIP cannot do it.

   Solution: Yes, CLIP can be used for this purpose and the automatic decision may work. Recall that CLIP has a visual component $\boldsymbol{\theta}_v$ and a textual component $\boldsymbol{\theta}_t$. The textual part takes the form of a transformer and thus can take arbitrary sequence length as input such as text prompt. Recalling that CLIP is trained with contrastive learning to maximize the similarity between pairs of `<images, text>` that correlate while pairs that do not correlated should have low similarity. Correlate means that a text description is pertinent with the visual feature of the image and vice versa. A method to classify if an object is in the image could be:

   - first extract once the embedding of the image $\mathtt{x}$ as $\boldsymbol{\theta}_v(x)$.
   - then build two other embedding with $\boldsymbol{\theta}_t$. One embedding with the text t_1 = `''Yes, the <object> is clearly visible in the image''` and another one t_2 = `''No, the <object> is not visible in the image.''`
   - simply decide based on $\arg\max_{i=1,2}\left(\boldsymbol{\theta}_v(x)^\top \boldsymbol{\theta}_t(t_i)\right)$

Total for Question 5: 6

You can use this space for writing. The summary of points is at the bottom.

Question:	1	2	3	4	5	Total
Points:	8	8	4	6	6	32
Score:						