COMP3361 Assignment 3 Submission

Name: <Anonymized>
University Number: 0000000000

Spring 2024

1 Written Problems (50%)

1.1 Multi-Choice (20%)

Question Number	Selected Option
1	C
2	D
3	C, D
4	B
5	B
6	D
7	C
8	C
9	D
10	A

Table 1: Selected Options for Multi-Choice Questions

1. Select the answer that includes all the skip-gram (word, context) training pairs for the sentence the cat ran away, for a window size k = 2 from target.

   A) [the, cat], [the, ran], [cat, ran], [cat, away], [ran, away]

   B) [the], [cat], [ran], [away]

   C) [the, cat], [the, ran], [cat, the], [cat, ran], [cat, away], [ran, the], [ran, cat], [ran, away], [away, cat], [away, ran]

   D) [the, ran], [ran, cat], [cat, away]

   E) [the, cat ran], [cat, the], [cat, ran away], [ran, the cat], [ran, away], [away, cat ran]

2. What if gradients become too large or small?

   A) If too large, the model will become difficult to converge

   B) If too small, the model can't capture long-term dependencies

   C) If too small, the model may capture a wrong recent dependency

   D) All of the above

3. Which of the following methods do not involve updating the model parameters? Select all that apply.

   A) Fine-tuning

   B) Transfer learning

   C) In-context learning

   D) Prompting

4. What range of values can cross entropy loss take?

   A) 0 to 1

   B) 0 to ∞

   C) -1 to 1

   D) $-\infty$ to 0

5. Can we use bidirectional RNNs in the following tasks? (1) text classification, (2) code generation, (3) text generation

   A) Yes, Yes, Yes

   B) Yes, No, No

   C) Yes, Yes, No

   D) No, Yes, No

6. Select the models that are capable of generating dynamic word embeddings, which can change depending on the surroundings of a word in a sentence.

   A) Bag of Words (BoW)

   B) Word2Vec

   C) GloVe

   D) T5

7. What makes Span Corruption a unique pretraining objective compared to traditional MLM?

   A) It involves masking and predicting individual tokens rather than spans of tokens.

   B) It only uses punctuation marks as indicators for span boundaries.

   C) It masks contiguous spans of text and trains the model to predict the masked spans, encouraging understanding of longer context.

   D) It requires the model to correct grammatical errors within the span.

8. In the transformer model architecture, positional encodings are added to the input embeddings to provide the model with information about the position of tokens in the sequence. Given a sequence length of L and a model dimension of D, which of the following PyTorch code snippets correctly implements the calculation of sinusoidal positional encodings?

   A) `def positional_encoding(L, D):`

```
    position = torch.arange(L).unsqueeze(1)
    div_term = torch.exp(torch.arange(0, D, 2) * -(np.log(10000.0) / D))
    pe = torch.zeros(L, D)
    pe[:, 0::2] = torch.sin(position * div_term)
    pe[:, 1::2] = torch.cos(position * div_term)
    return pe
```

B)
```
def positional_encoding(L, D):
    position = torch.arange(L).unsqueeze(1)
    div_term = torch.exp(torch.arange(0, D, 2) * -(np.log(10000.0) / D))
    pe = torch.zeros(L, D)
    pe[:, 0::2] = torch.sin(position / div_term)
    pe[:, 1::2] = torch.cos(position / div_term)
    return pe
```

C)
```
def positional_encoding(L, D):
    position = torch.arange(L, dtype=torch.float).unsqueeze(1)
    div_term = 1 / (10000 ** (2 * torch.arange(D // 2) / D))
    pe = torch.zeros(L, D)
    pe[:, 0::2] = torch.sin(position * div_term)
    pe[:, 1::2] = torch.cos(position * div_term)
    return pe
```

D)
```
def positional_encoding(L, D):
    position = torch.arange(L, dtype=torch.float).unsqueeze(1)
    div_term = torch.exp(torch.arange(0, D, 2) * -(np.log(10000.0) / L))
    pe = torch.zeros(L, D)
    pe[:, 0::2] = torch.sin(position * div_term)
    pe[:, 1::2] = torch.cos(position / div_term)
    return pe
```

9. In instruction tuning, what is the primary benefit of using natural language instructions for model fine-tuning?

   A) Reduces the need for labeled data in supervised learning tasks.

   B) Enables the model to perform zero-shot or few-shot learning on tasks it was not explicitly trained for.

   C) Significantly decreases the computational resources needed for training large models.

   D) Allows the model to improve its performance on specific tasks without fine-tuning.

10. How does Low-Rank Adaptation (LoRA) efficiently fine-tune large pre-trained models for specific tasks?

    A) By freezing the original parameters and training a small set of new parameters introduced as adapters at certain layers, significantly reducing computational needs.

    B) By applying low-rank matrices to adjust the attention weights, enabling task-specific tuning without extensively retraining the original parameters.

    C) By pruning less important neurons based on initial assessments, simplifying the model for specific tasks with minimal performance impact.

    D) By adding task-specific tokens to the model's vocabulary and fine-tuning their embeddings only, leveraging the existing model for seamless integration.

1.2 Short Answer (30%)

Question 3.1: A trigram language model is also often referred to as a second-order Markov language model. It has the following form:

$$P\left(X_1 = x_1, \ldots, X_n = x_n\right) = \prod_{i=1}^{n} P\left(X_i = x_i \mid X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1}\right)$$

Question 3.1a: Could you briefly explain the advantages and disadvantages of a high-order Markov language model compared to the second-order one?

An advantage of using a high-order Markov language model is that it can capture longer n-grams. For example, pronouns in English can refer to specific names or words, which is usually located more than two or three words apart.

Another advantage is that these language models can capture longer phrases, specifically names and entities. For example, HKU can be represented by the phrase "The University of Hong Kong", which contains five words. This can be effectively captured by a language model with order 5 or above.

A disadvantage of a high-order Markov language model is that these models suffer from the curse of dimensionality. The number of n-grams (thus features) increases exponentially with the order of the chosen language model, as the space complexity of the model is $O(|V|^n)$, where $|V|$ is the vocabulary size.

Another disadvantage of a high-order language model is that they cause data sparsity. It is expected that only a few n-grams out of all possible n-grams will appear in a particular sentence, which makes the feature vector more sparse compared to a model with a lower order, for instance, the unigram model.

Question 3.1b: Could you give some examples in English where English grammar suggests that the second-order Markov assumption is clearly violated?

An example would be "He is a handsome man" where the word "man" depends on the word "He", which is three words apart. This cannot be captured by trigrams, which violates the second-order Markov assumption.

Another example would be "Bob is a talented student who achieves a 4.0 GPA" where the word "who" refers to the word "Bob", which are four words apart from each other. This also violates the second-order Markov assumption.

Question 3.2 We'd like to define a language model with $V = \{\text{the, a, dog}\}$, and $p(x_1 \dots x_n) = \gamma \times 0.5^n$ for any $x_1 \dots x_n$, such that $x_i \in V$ for $i = 1 \dots (n-1)$, and $x_n = \text{STOP}$, where γ is some expression (which may be a function of n).

Which of the following definitions for γ give a valid language model? Please choose the answer and prove it.

(Hint: recall that $\sum_{n=1}^{\infty} 0.5^n = 1$)

1. $\gamma = 3^{n-1}$
2. $\gamma = 3^n$
3. $\gamma = 1$
4. $\gamma = \frac{1}{3^n}$
5. $\gamma = \frac{1}{3^{n-1}}$

The answer: 5. $\gamma = \frac{1}{3^{n-1}}$ gives a valid language model, and the proof is given as follows:

In order to define a valid language model, the probabilities of all sequence of words with any word length should sum up to 1.

For each sentence with length n, there are 3^{n-1} possible arrangements of the words, as the first $n-1$ words can be any word taken from the vocabulary, and the last word is the STOP token.

Since $p(x_1, \dots, x_n)$ represents the probability of observing a sequence of words, we have

$$\begin{aligned}
&\sum_{n=1}^{\infty} \sum_{j=1}^{3^{n-1}} p(x_1, \dots, x_n) = 1 \\
\Rightarrow &\sum_{n=1}^{\infty} 3^{n-1} p(x_1, \dots, x_n) = 1 \\
\Rightarrow &\sum_{n=1}^{\infty} 3^{n-1} \cdot \gamma \cdot 0.5^n = 1 \\
\Rightarrow &\sum_{n=1}^{\infty} 3^{n-1} \cdot \gamma \cdot 0.5^n = \sum_{n=1}^{\infty} 0.5^n
\end{aligned}$$

By comparing like terms and solving for the unknown γ, we have $\gamma = \dfrac{1}{3^{n-1}}$, which corresponds to option (5).

Question 3.3 Given a small document corpus D consisting of two sentences: {“i hug pugs", “hugging pugs is fun"} and a desired vocabulary size of N=15, apply the Byte Pair Encoding (BPE) algorithm to tokenize the documents.

Assume the initial vocabulary includes individual characters and spaces as separate tokens. The BPE algorithm should merge the most frequent adjacent pairs of tokens iteratively until the vocabulary size reaches N=15.

Question 3.3a What is the final list of the desired vocabulary tokens?

The final list of desired vocabulary tokens is {"i", " ", "h", "u", "g", "p", "s", "n", "f", "ug", "hug", " p", "pug", "ugs", "s "} which has vocabulary size N=15.

We first start with the initial list with the distinct characters in the sentences, which is {"i", " ", "h", "u", "g", "p", "s", "n", "f"} and has vocabulary size N=9.

Next, we count the adjacent pairs of tokens, which are shown in Table 2.

Bigram	Counts	Bigram	Counts	Bigram	Counts
"i "	1	"pu"	2	"s "	2
" h"	1	"gs"	2	" i"	1
"hu"	2	"gg"	1	"is"	1
"ug"	4	"gi"	1	" f"	1
"g "	2	"in"	1	"fu"	1
" p"	2	"ng"	1	"un"	1

Table 2: Count of bigram frequencies at the first iteration of the BPE algorithm

From Table 2, we observe that the most frequent bigram is "ug", which has four occurrences. The token "ug" is then added to the list of vocabulary tokens. Currently, the list contains {"i", " ", "h", "u", "g", "p", "s", "n", "f", "ug"} and has size N=10.

We then perform the next iteration of the BPE algorithm, to count the adjacent pairs of tokens with the newly added token "ug". The count results are shown in Table 3.

Bigram	Counts	Bigram	Counts	Bigram	Counts
"i "	1	"ugs"	2	"s "	2
" h"	1	"ugg"	1	" i"	1
"hug"	2	"gi"	1	"is"	1
"ug "	1	"in"	1	" f"	1
" p"	2	"ng"	1	"fu"	1
"pug"	2	"g "	1	"un"	1

Table 3: Count of bigram frequencies at the second iteration of the BPE algorithm

From table 3, we observe that the five most frequent tokens are "hug", " p", "pug", "ugs" and "s ", which are then added to the list of vocabulary tokens. Currently, the list contains {"i", " ", "h", "u", "g", "p", "s", "n", "f", "ug", "hug", " p", "pug", "ugs", "s "} and has size N=15.

Since we have reached the desired vocabulary size N=15, the algorithm stops. The final list of the desired vocabulary tokens is {"i", " ", "h", "u", "g", "p", "s", "n", "f", "ug", "hug", " p", "pug", "ugs", "s "}.

Question 3.3b What is the final list of document tokens after reaching the desired vocabulary size?

In order to obtain the document tokens, we first encode the vocabulary token list into unique indices. The mapping is shown in Table 4.

Token	Index	Token	Index	Token	Index
"i"	1	"p"	6	"hug"	11
" "	2	"s"	7	" p"	12
"h"	3	"n"	8	"pug"	13
"u"	4	"f"	9	"ugs"	14
"g"	5	"ug"	10	"s "	15

Table 4: Mapping between tokens and its index

Based on the mapping obtained in Table 4, the obtained list of document tokens is {[1, 2, 11, 12, 14], [11, 5, 1, 8, 5, 12, 14, 2, 1, 15, 9, 4, 8]}, which corresponds to the sentences {"i hug pugs", "hugging pugs is fun"} respectively.

Question 3.4 Let $\mathbf{Q} \in \mathbb{R}^{N \times d}$ denote a set of N query vectors, which attend to M key and value vectors, denoted by matrices $\mathbf{K} \in \mathbb{R}^{M \times d}$ and $\mathbf{V} \in \mathbb{R}^{M \times c}$ respectively. For a query vector at position n, the softmax attention function computes the following quantity:

$$\mathrm{Attn}\left(\mathbf{q}_n, \mathbf{K}, \mathbf{V}\right) = \sum_{m=1}^{M} \frac{\exp\left(\mathbf{q}_n^\top \mathbf{k}_m\right)}{\sum_{m'=1}^{M} \exp\left(\mathbf{q}_n^\top \mathbf{k}_{m'}\right)} \mathbf{v}_m^\top := \mathbf{V}^\top \mathrm{softmax}\left(\mathbf{K}\mathbf{q}_n^\top\right)$$

which is an average of the set of value vectors $\mathbf{V}$ weighted by the normalized similarity between different queries and keys.

Please briefly explain what is the time and space complexity for the attention computation from query $\mathbf{Q}$ to $\mathbf{K}$, $\mathbf{V}$, using the big O notation.

The time complexity for the attention computation is $O(N \cdot M \cdot (c + d))$.

For computing the attention for a single query vector, the matrix-vector product $\mathbf{K}\mathbf{q}_n^\top$ needs to be first computed. This takes $O(M \cdot d)$ time as the resulting vector has dimension M-by-1, and each value in the matrix requires $O(d)$ time to calculate. After obtaining $\mathbf{K}\mathbf{q}_n^\top$ which is a M-by-1 vector, the softmax function is applied which takes $O(M)$ time. Afterwards, multiplying $\mathbf{V}^\top$, which is a c-by$M$ matrix to $\mathrm{softmax}\left(\mathbf{K}\mathbf{q}_n^\top\right)$ takes $O(c \cdot M)$ time, as the resulting vector is of size c-by-1 and each value requires $O(M)$ time to compute. Thus, the attention computation for a single query vector is $O(M \cdot d + M + c \cdot M)$, which is equivalent to $O\left(M \cdot (c + d)\right)$. Since the attention computation requires computing N query vectors in the matrix $\mathbf{Q}$, the time complexity is multiplied by N, which is $O(N \cdot M \cdot (c + d))$.

The space complexity is $O(N \cdot d + M \cdot d + M \cdot c)$. As the only storage cost involves storing the query ($\mathbf{Q}$), key ($\mathbf{K}$) and value ($\mathbf{V}$) matrices, the space complexity is equal to the total number of weights stored in these matrices. Since $\mathbf{Q}$ is an N-by-d matrix, $\mathbf{K}$ is an M-by-d matrix, and $\mathbf{V}$ is an M-by-c matrix, the number of weights stored are $N \cdot d$, $M \cdot d$ and $M \cdot c$ respectively. Overall, the space complexity is $O(N \cdot d + M \cdot d + M \cdot c)$.