1 Answer1

1.1 a)

Given the vectors:

$$v1 = [1, 2, -3]$$
$$q1 = [1, -1, 0]$$
$$k1 = [1, 0, 0]$$

$$v2 = [2, 2, 0]$$
$$q2 = [1, -1, -2]$$
$$k2 = [0, 2, 1]$$

$$v3 = [0, 3, -1]$$
$$q3 = [1, 4, -3]$$
$$k3 = [1, -3, 2]$$

For the attention weights **For token 2:**

$$(\alpha_{12}, \alpha_{22}, \alpha_{32}) = softmax(q_2 \cdot k_1, q_2 \cdot k_2, q_2 \cdot k_3)$$

$$(\alpha_{12}, \alpha_{22}, \alpha_{32}) = \text{softmax}\left(\begin{bmatrix}1\\-1\\-2\end{bmatrix} \cdot \begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}1\\-1\\-2\end{bmatrix} \cdot \begin{bmatrix}0\\2\\1\end{bmatrix}, \begin{bmatrix}1\\-1\\-2\end{bmatrix} \cdot \begin{bmatrix}1\\-3\\2\end{bmatrix}\right)$$

$$(\alpha_{12}, \alpha_{22}, \alpha_{32}) = softmax(1, -4, 0)$$

Next, we apply the softmax function:

$$\begin{aligned}\text{softmax}(1, -4, 0) &= \left[\frac{e^1}{e^1 + e^{-4} + e^0}, \frac{e^{-4}}{e^1 + e^{-4} + e^0}, \frac{e^0}{e^1 + e^{-4} + e^0}\right] \\ &= [0.7275, 0.0049, 0.2676]\end{aligned}$$

$$(\alpha_{12}, \alpha_{22}, \alpha_{32}) = (0.7275, 0.0049, 0.2676)$$

The input y_2 to the second layer for token 2 is:

$$y_2 = (\alpha_{12}v_1, \alpha_{22}v_2, \alpha_{32}v_3)$$

$$y_2 = (0.7275v_1, 0.0049v_2, 0.2676v_3)$$

$$y_2 = (0.7275X \begin{bmatrix}1\\2\\-3\end{bmatrix}, 0.0049X \begin{bmatrix}2\\2\\0\end{bmatrix}, 0.2676X \begin{bmatrix}1\\-3\\2\end{bmatrix})$$

Given the weighted vectors:
For $\alpha_{12}v_1$:

$$\alpha_{12}v_1 = \begin{bmatrix} 0.7275 \\ 1.455 \\ -2.1825 \end{bmatrix}$$

For $\alpha_{22}v_2$:

$$\alpha_{22}v_2 = \begin{bmatrix} 0.0098 \\ 0.0098 \\ 0 \end{bmatrix}$$

For $\alpha_{32}v_3$:

$$\alpha_{32}v_3 = \begin{bmatrix} 0 \\ 0.8028 \\ -0.2676 \end{bmatrix}$$

Summing them component-wise:
For the first component:

$$0.7275 + 0.0098 + 0 = 0.7373$$

For the second component:

$$1.455 + 0.0098 + 0.8028 = 2.2676$$

For the third component:

$$-2.1825 + 0 - 0.2676 = -2.4501$$

Thus, the resultant vector is:

$$y_2 = \begin{bmatrix} 0.7373 \\ 2.2676 \\ -2.4501 \end{bmatrix}$$

For token 3:

$$(\alpha_{13}, \alpha_{23}, \alpha_{33}) = softmax(q_3 \cdot k_1, q_3 \cdot k_2, q_3 \cdot k_3)$$

$$(\alpha_{12}, \alpha_{22}, \alpha_{32}) = \text{softmax}\left(\begin{bmatrix} 1 \\ 4 \\ -3 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 4 \\ -3 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 4 \\ -3 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ -3 \\ 2 \end{bmatrix} \right)$$

$$(\alpha_{12}, \alpha_{22}, \alpha_{32}) = softmax(1, -4, 0)$$

Next, we apply the softmax function:

$$\begin{aligned} \text{softmax}(1, 5, -17) &= \left[\frac{e^1}{e^1 + e^5 + e^{-17}}, \frac{e^5}{e^1 + e^5 + e^{-17}}, \frac{e^{-17}}{e^1 + e^5 + e^{-17}} \right] \\ &= \left[(0.0180, 0.9820, 2.74 \times 10^{-10}) \right] \end{aligned}$$

$$(\alpha_{13}, \alpha_{23}, \alpha_{33}) = (0.0180, 0.9820, 2.74 \times 10^{-10})$$

The input y_2 to the second layer for token 3 is:

$$y_3 = (\alpha_{13}v_1, \alpha_{23}v_2, \alpha_{33}v_3)$$

$$y_3 = (0.0180v_1, 0.9820v_2, 2.74 \times 10^{-10}v_3)$$

$$y_3 = (0.0180X \begin{bmatrix} 1 \\ 2 \\ -3 \end{bmatrix}, 0.9820X \begin{bmatrix} 2 \\ 2 \\ 0 \end{bmatrix}, 2.74 \times 10^{-10}X \begin{bmatrix} 1 \\ -3 \\ 2 \end{bmatrix})$$

Given the weighted vectors:
For $\alpha_{12}v_1$:

$$\alpha_{13}v_1 = \begin{bmatrix} 0.0180 \\ 0.0360 \\ -0.0540 \end{bmatrix}$$

For $\alpha_{22}v_2$:

$$\alpha_{23}v_2 = \begin{bmatrix} 1.9640 \\ 1.9640 \\ 0 \end{bmatrix}$$

For $\alpha_{32}v_3$:

$$\alpha_{33}v_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Summing them component-wise:
For the first component:

$$0.0180 + 1.9640 + 0 = 1.9820$$

For the second component:

$$0.0360 + 1.9640 + 0 = 2$$

For the third component:

$$-0.0540 + 0 + 0 = -0.0540$$

Thus, the resultant vector is:

$$y_3 = \begin{bmatrix} 1.9820 \\ 2.0000 \\ -0.0540 \end{bmatrix}$$

1.2 b)

Attention matrix is here:

$$\alpha = \begin{bmatrix} 0.047, 0.002, 0.95 \\ 0.7275, 0.0049, 0.2676 \\ 0.018, 0.982, 0 \end{bmatrix}$$

$$\alpha_{12} \neq \alpha_{21} \; 0.002 \neq 0.7275$$

$$\alpha_{13} \neq \alpha_{31} \; 0.954 \neq 0.018$$

$$\alpha_{23} \neq \alpha_{32} \; 0.2676 \neq 0.982$$

so we can say attention weight is not symmetric.

1.3 c)

If

$$(\alpha_{12}, \alpha_{22}, \alpha_{32}) = \text{softmax}(q_1 \cdot q_1, q_1 \cdot q_2, q_1 \cdot q_3)$$

and we don't use softmax function Our attention matrix will be:

$$\alpha = \begin{bmatrix} q_1 \cdot q_1, q_1 \cdot q_2, q_1 \cdot q_3 \\ q_2 \cdot q_1, q_2 \cdot q_2, q_2 \cdot q_3 \\ q_3 \cdot q_1, q_3 \cdot q_2, q_3 \cdot q_3 \end{bmatrix}$$

and we can say yes it is symmetric. But we use softmax function and numerators can be same for a_ij and a_ji but demonitors will be different because:
$q_{11} + q_{12} + q_{13} \neq q_{21} + q_{22} + q_{23}$ in every time. But it has more similar values.
If we use this attention mechanism, we can loss diversity and flexibility. In the original attention mechanism, the queries represent the "questions" or the "requests for information", while the keys represent the "labels" or "handles" on the information. By having both, we allow the network to develop a rich set of representations. If we eliminate the keys and just use the queries, we're essentially making the assumption that the best way to "label" or "handle" information is the same way we request it. This limits the diversity of representations. And the power of the transformer's attention mechanism comes from its ability to weigh the relevance of different input tokens differently based on both the query and key. By removing the key vectors, we're reducing the network's ability to create varied and adaptive patterns of attention based on the content of the input.
In addition it effect our information flow because using only the query vectors to compute attention weights means that the patterns of information flow are entirely determined by the current state's queries. This could lead to scenarios where certain states or tokens consistently dominate the attention weights, especially if their associated query vectors have large magnitudes.

1.4 d)

In RNNs, information about the order of the inputs was inherently built into the model's structure. However, the same isn't true for transformers. This means that if you don't mind the order of the inputs in the attention computation, we get exactly the same result.

This is the primary reason why transformers for language add positional encodings to the input vectors in the first layer. Without positional encodings, when analyzing a sentence using a transformer, it would treat the sentence as a "bag of words" and lose the order of the words. The order of words is critical for many language processing tasks. For instance, the sentences "The cat chased the dog" and "The dog chased the cat" have different meanings, but without positional encodings, a transformer would perceive these two sentences as identical.

To address this issue, the transformer model modifies the input embeddings by combining them with positional embeddings specific to each position in an input sequence. These positional embeddings start with randomly initialized embeddings corresponding to each possible input position up to a certain maximum length. These embeddings are learned alongside other parameters during training.

In conclusion, the absence of positional encodings would prevent the transformer from analyzing sentences correctly. Hence, for language, transformers add positional encodings to the input vectors in the first layer.

2 Answer 2

2.1 a)

An input embedding for a transformer is a dense vector that represents a specific word or token from the input sequence. It converts words into a format suitable for neural network processing. These embeddings capture the semantic essence of words, and in transformers, they are combined with positional encodings to inform the model about the position of each token in a sequence. When using sub-word tokenization, even parts of words (like prefixes or suffixes) have their own embeddings.

2.2 b)

Given:

- Number of tokens: $T = 30,000$
- Size of each input embedding: $S = 768$

Each embeding vector length is 768, The total number of trainable parameters for the input embeddings is:

$$P = T \times S$$

$$P = 30,000 \times 768 = 23,040,000$$

BERT requires 23,040,000 trainable parameters for input embeddings.

2.3 c)

BERT and other transformers incorporate individual characters as tokens in their token-set, especially when trained on large datasets, for the following reasons:

1. **Handling Out-of-Vocabulary Words:** The model can represent any word it hasn't seen before by breaking it down into its constituent characters.
2. **Morphologically Rich Languages:** Character-level tokenization allows transformers to efficiently handle languages with many word forms derived from a single root.
3. **Flexibility in Tokenization:** Individual characters provide flexibility, ensuring the model can process any word or symbol it encounters.
4. **Efficiency in Training:** Character-level tokens allow the model to generalize better, reducing the number of required embeddings.

Thus, individual characters as tokens ensure robust performance across diverse datasets and languages.

2.4 d)

How is an input text tokenized?

Transformers tokenize input text using sub-word tokenization methods, notably Byte-Pair Encoding (BPE) or WordPiece. This process starts at the character level and iteratively merges frequent character combinations.

Is the tokenization unique?

Yes, tokenization is typically unique. Tokenizers will greedily select the longest sub-word token from the vocabulary that matches a prefix of the remaining input text.

Why is unique tokenization desirable?

- **Tutorial Consistency**: Unique tokenization ensures that the same text is always represented in the same manner, leading to consistent model predictions.

- **Training Stability**: During training, seeing the same text represented in different ways can complicate the learning process and reduce the model's generalization ability. Unique tokenization ensures a stable learning experience.
- **Data Efficiency**: If a text can be tokenized in multiple ways, it might require more data for the model to learn the same information. Unique tokenization aids in efficient learning.
- **Interpretability and Debugging**: Knowing how input text is tokenized is crucial when analyzing model behavior or debugging issues. Unique tokenization makes this process more predictable and manageable.

Unique tokenization aids in the effective training, prediction, and analysis of the model.

3 Answer 3

3.1 a)

What is a Winograd Schema?

Winograd Schema problems, named after Terry Winograd who first introduced them in his dissertation, are coreference resolution problems designed to be too challenging to be solved by conventional resolution methods described in literature. They have become a benchmark challenge for natural language processing. Consider the task of determining the correct antecedent of the pronoun *they* in the following example:

> The city council denied the demonstrators a permit because
> a. they feared violence.
> b. they advocated violence.

Determining the correct antecedent for the pronoun *they* requires understanding that the second clause is intended as an explanation for the first, and also general knowledge such as city councils being more likely to fear violence and demonstrators being more likely to advocate for it. Solving Winograd Schema problems necessitates finding ways to represent or uncover the required real-world knowledge.

Why is it difficult to use NLP to resolve coreference in a Winograd Schema?

The resolution of Winograd Schema problems relies not just on linguistic knowledge but also on understanding that goes beyond language, often requiring general knowledge. For instance, in the aforementioned example, determining the correct reference for the pronoun *they* requires knowledge like city councils being more likely to fear violence compared to demonstrators.

Why are Winograd Schemata proposed as a test for NLP models' understanding of a sentence?

Winograd Schemata are proposed as tests because they challenge the model's ability to truly understand context and semantics beyond just the linguistic structure. If a model can correctly resolve the coreference in a Winograd Schema, it suggests that the model has a deeper understanding of the sentence beyond just its surface meaning.

3.2 b)

1. Winograd Schema Problems

- **Description**: Winograd Schema problems are coreference resolution challenges designed to be particularly difficult for conventional resolution methods. They require deep semantic understanding and real-world knowledge. An example is determining the correct antecedent of the pronoun "they" in: "The city council denied the demonstrators a permit because they feared violence."
- **Difficulty**: Resolving these problems requires understanding that goes beyond linguistic cues, necessitating the representation or uncovering of real-world knowledge.

2. Event Coreference

- **Description**: This involves deciding whether two event mentions refer to the same event, such as determining if "buy" and "acquisition" in different sentences refer to the same event.
- **Difficulty**: Event mentions can be both verbal and nominal, making them harder to detect than entity mentions. Once detected, the inherent variability and context-dependence of events pose challenges.

3. Discourse Deixis

- **Description**: In discourse deixis, an anaphor refers back to a discourse segment, which can be challenging to delimit or categorize. For instance, in "According to Soleil, Beau just opened a restaurant. But that turned out to be a lie.", the referent of "that" is a speech act.
- **Difficulty**: The challenge lies in the variability of what the anaphor can refer to, ranging from a speech act to a proposition or a manner of description. Identifying the correct type and its boundaries in the discourse is non-trivial.

3.3 c)

I chose to test ChatGPT's coreference resolution capabilities using the *Winograd Schema problems*. These problems are designed to challenge NLP systems by presenting sentences where a pronoun can refer to multiple entities, and its reference is determined by the context.

1. **Sentence**: The trophy didn't fit in the suitcase because it was too big.
    - **Question**: What was too big?
    - **ChatGPT's Response**: The trophy was too big to fit in the suitcase. So, "the trophy" was too big.
2. **Sentence**: The trophy didn't fit in the suitcase because it was too small.
    - **Question**: What was too small?
    - **ChatGPT's Response**: In this case, "the suitcase" was too small to accommodate the trophy. So, "the suitcase" was too small.
3. **Sentence**: The delivery truck zoomed by the school bus because it was going so fast.
    - **Question**: What was going fast?
    - **ChatGPT's Response**: In the sentence, the pronoun "it" refers to the "delivery truck." So, the delivery truck was going fast.
4. **Sentence**: The man couldn't lift his son because he was so weak.
    - **Question**: Who was weak?
    - **ChatGPT's Response**: In the sentence, the pronoun "he" refers to "the man." So, the man was weak.
5. **Sentence**: Frank felt malaise because he ate the entire pizza.
    - **Question**: Who ate the pizza?
    - **ChatGPT's Response**: In the sentence, the pronoun "he" refers to "Frank." So, Frank ate the pizza.
6. **Sentence**: The cake ended up in the trash because it was so stale.
    - **Question**: What was stale?
    - **ChatGPT's Response**: In the sentence, the pronoun "it" refers to "the cake." So, the cake was stale.

Based on the conversation, ChatGPT demonstrated a strong capability in resolving coreferences in the presented sentences. It correctly identified the referents of the pronouns in each case, showcasing its understanding of the context in which the pronouns were used.

4 Answer 4)

Deep Language Models: Knowledge and Coherence

Language models, particularly those of significant depth and scale like GPT-3, have made remarkable strides in the realm of Natural Language Processing (NLP). However, understanding their capabilities and limitations is crucial to discerning their true nature.

1. In what sense does a deep language model have knowledge of the world, and in what sense does it not?

Deep language models are trained on vast amounts of data, often encompassing a significant portion of the internet. This data, while extensive, is not always representative of the world's diverse perspectives. For instance, the paper "On the dangers of Stochastic Parrots" highlights that internet data can overrepresent hegemonic viewpoints, leading to biases in the models. For example, data sourced from platforms like Reddit might overrepresent younger users from developed countries, thereby skewing the model's "knowledge."

Furthermore, the "Octopus Test" from the second paper serves as a thought experiment illustrating the challenges of learning meaning from form alone. An octopus, despite being adept at detecting statistical patterns in a conversation between two individuals, lacks the context and real-world experiences to truly understand the meaning behind the words. Similarly, while a language model might predict text based on patterns, it doesn't genuinely "understand" the world or the deeper meanings and nuances behind the text.

2. In what sense can a language model such as GPT-3 generate coherent text, and in what sense can it not?

GPT-3 and similar models can generate coherent text in the sense that they can produce grammatically correct and contextually relevant sentences based on the patterns they've learned from their training data. They can mimic human-like text generation to a remarkable degree, often fooling readers into believing they're interacting with a human.

However, their coherence is limited to the patterns they've observed. They lack genuine comprehension and reasoning. As the "Octopus Test" suggests, while the octopus (or a language model) might replicate a conversation based on patterns, it might falter when faced with nuanced or context-heavy discussions. The model doesn't truly "understand" the conversation; it's merely predicting the next likely word or phrase based on its training.

In conclusion, while deep language models are powerful tools capable of impressive text generation, it's essential to recognize their limitations. Their "knowledge" is based on patterns and data, not genuine understanding, and while they can produce coherent text, they lack the depth and nuance of true comprehension.