AI_HW2_repo_111550076

notion link:

AI_HW2_repo_111550076

Part 0:Implement a different preprocessing method

Briefly explain the method you implemented and give an example in the report.

- **remove_stopwords():** Transpose all characters in the text to lower text and eliminate terms in stopword list in nltk.
- **preprocessing_function():** preprocesses a given text by removing stopwords, replacing `<br />` tags with spaces, converting to lowercase, removing punctuation, tokenizing, stemming the words using SnowballStemmer, and finally joining the words back into a string.

  E.g.

  text=”It is important to have some understanding of the French society of Today to really enjoy the humor of this movie
”

  remove_stopwords=”important understanding French society Today really enjoy humor movie”

  preprocessed_text=”import understand french societi today realli enjoy humor movi”

```python
def preprocessing_function(text: str) -> str:
    preprocessed_text = remove_stopwords(text)

    # TO-DO 0: Other preprocessing function attemption
    # Begin your code
    preprocessed_text = preprocessed_text.replace("<br />", " ").lower()
    preprocessed_text = re.sub(r'[^\w\s]','',preprocessed_text)
    words = nltk.word_tokenize(preprocessed_text)
    stemmer = nltk.SnowballStemmer(language='english')
    words = [stemmer.stem(word) for word in words]
    preprocessed_text = ' '.join(words)
    # End your code

    return preprocessed_text
```

Part 1: Implement the Bi-Gram language model

Briefly explain the concept of perplexity in report and discuss how it will be influenced.

In NLP, perplexity measures a language model's prediction capability. Lower perplexity signifies a better-performing model, as it indicates less surprise when encountering new data.

In mathematics, the perplexity (PP) of a sequence of words $W = w_1, w_2, \ldots, w_N$ for a model M is defined as:

$$PP(W) = P(w_1, w_2, \ldots, w_N)^{-\frac{1}{N}}$$

where $P(w_1, w_2, \ldots, w_N)$ is the probability of the word sequence W given by the model M, and N is the length of the sequence. This formula essentially calculates the inverse probability of the sequence, normalized by the number of words, which gives us a measure of how well the model predicts the sequence. Lower values of perplexity indicate better predictive performance.

Screenshot the outputs and tell your observations about the differences in the perplexity caused by the preprocessing methods

- **No preprocessing**

```
!python main.py --model_type ngram --preprocess 0 --part 2
# no preprocess

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Perplexity of ngram: 116.26046015880357
F1 score: 0.7051, Precision: 0.7082, Recall: 0.7059
```

- **Using remove_stopword() function:**

```
[ ] !python main.py --model_type ngram --preprocess 1 --part 2

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Perplexity of ngram: 195.43245350997685
F1 score: 0.6745, Precision: 0.6902, Recall: 0.6792
```

- **Using preprocessing_function() function:**

```
!python main.py --model_type ngram --preprocess 1 --part 2
# preprocessing_function

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
Perplexity of ngram: 158.8042876611314
F1 score: 0.7028, Precision: 0.7181, Recall: 0.7067
```

Observation:

When using remove stopwords function, it may increase the perplexity of a model, yet, this action can improve the method's effectiveness as reflected by a higher F1 score. Therefore, there is no direct correlation between perplexity and F1 score; higher complexity does not automatically imply lower accuracy.

Part 2 Implement the BERT model

Briefly explain the two pre-training steps in BERT.

1. **Masked language model**

   During this process, BERT randomly masks portions of the input tokens and endeavors to predict these masked words by considering their surrounding context. This compels BERT to cultivate a profound comprehension of both sentence structure and context, enhancing its understanding and processing capabilities.

2. **Next sentence prediction**

   BERT is trained to grasp the intricate relationships between sentences. Throughout its training phase, it is presented with pairs of sentences and is tasked with determining whether the second sentence in each pair genuinely follows the first in the original document. This method trains BERT to discern the logical flow and connection between sentences, improving its ability to comprehend and predict textual continuity.

Briefly explain four different BERT application scenarios

1. **Sentiment analysis**

   Determining if text is positive, negative, or neutral in sentiment is key for analyzing customer feedback or social media opinion, aiding in understanding public or consumer attitudes.

2. **Question answering**

   BERT excels in question answering systems, where it can pinpoint and extract answers from a text corpus in response to a specific question. This capability is particularly beneficial for chatbots and virtual assistants, enhancing their ability to provide accurate and relevant information on demand.

3. **Text Summarization**

   BERT efficiently condenses large texts into key summaries, aiding quick comprehension of detailed content like news or research, saving time in our fast-paced world.

4. **Named Entity Recognition**

BERT excels in recognizing and classifying specific entities within text, a capability central to extracting information from documents. This enhances search algorithms and improves content organization, making it easier to find and categorize information efficiently.

Discuss the difference between BERT and distilBERT?

1. **Model Size and Efficiency**

   BERT is a large model with 110 million parameters, making it computationally intensive, whereas DistilBERT, a distilled version, has 40% fewer parameters, offering similar performance but with greater speed and efficiency.

2. **Training Process**

   BERT, created by Google, learns from a vast text corpus to predict masked tokens and sentence relationships, while DistilBERT uses knowledge distillation to mimic BERT's capabilities, achieving efficient language understanding.

3. **Performance**

   BERT delivers top-tier results in various NLP tasks, while DistilBERT, though smaller, retains about 97% of BERT's efficacy, with BERT sometimes outperforming it in more complex tasks due to its larger size.

4. **Use Cases**

   BERT suits high-performance needs without resource constraints, while DistilBERT's efficiency makes it ideal for limited hardware or faster processing demands, like in mobile or web applications.

Screenshot the required test F1-score.

```
!python  main.py  --model_type  BERT  --preprocess  1  --part  2

[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
tokenizer_config.json: 100% 28.0/28.0 [00:00<00:00, 119kB/s]
config.json: 100% 483/483 [00:00<00:00, 2.88MB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 4.27MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 2.73MB/s]
model.safetensors: 100% 268M/268M [00:01<00:00, 204MB/s]
100% 5000/5000 [26:33<00:00,  3.14it/s]
100% 10000/10000 [01:26<00:00, 115.08it/s]
Epoch: 0, F1 score: 0.9212, Precision: 0.9214, Recall: 0.9212, Loss: 0.2664
```

Explain the relation of the Transformer and BERT and the core of the Transformer.

- **BERT’s relation to the Transformer**

  BERT is a revolutionary model in NLP developed by Google. It significantly advances the understanding of context in text by analyzing words in relation to all the other words in a sentence, rather than in isolation or in a single direction. This bidirectional approach allows for a more nuanced understanding of language nuances, making BERT highly effective for a range of NLP tasks such as question answering, sentiment analysis, and language inference.

- **Core of the Transformer**

  The Transformer architecture, foundational to BERT, centers around the self-attention mechanism. This enables dynamic emphasis on various input segments, assessing each word's relevance within its context, for efficient, simultaneous processing. Unlike earlier models dependent on recurrent or convolutional layers, it excels in handling language data. Additionally, positional encoding provides essential word order awareness, vital for interpreting structured language.

Part 3 Implement the LSTM model

Briefly explain the difference between vanilla RNN and LSTM.

- **Vanilla RNNs:**

Vanilla RNNs process sequences with a simple structure good for short-term dependencies but falter on long-term data due to the vanishing gradient problem, hindering learning from distant inputs.

- **LSTM:**

LSTMs improve upon RNNs with a gated architecture that efficiently manages long sequences, overcoming the vanishing gradient problem and excelling at capturing long-term dependencies.

- **Summary**

While both RNNs and LSTMs are designed for sequential data, LSTMs provide a robust solution to the limitations of vanilla RNNs by efficiently managing long-term information, thanks to their gating mechanisms.

Please explain the meaning of each dimension of the input and output for each layer in the model.

1. **Embedding Layer (self.embedding)**
    - nn.Embedding()
        - Input Dimension: [batch_size, sequence_length]
        - Output Dimension: [batch_size, sequence_length, embedding_dim]
2. **LSTM Layer(self.encoder)**
    - nn.LSTM()
        - Input Dimension: [sequence_length, batch_size, input_size]
        - Output Dimension: a tuple (output, (h_n,c_n)):
            - output: [sequence_length, batch_size, num_directions *hidden_size]
            - h_n (hidden state): [num_layers * num_directions, batch_size, hidden_size]

        - c_n(cell state): [num_layers * num_directions, batch_size, hidden_size]

3. **Linear Layer(self.classifier)**
    - nn.Drouput()
        - Input and Output Dimension: The same. The dropout layer randomly zero out elements of the input tensor with probability `p` (where `p` is the dropout rate specified during layer creation) during training. This helps prevent overfitting by reducing the model's dependency on specific weights.
    - nn.Linear()
        - Input Dimension: [batch_size, additional_dimension, input_size]
        - output Dimension: [batch_size, additional_dimension, output_size]
    - nn.sigmoid():
        - Input and Output Dimension: The same.

        This layer applies the sigmoid function element-wise, which is defined as $f(x) = 1/1 + exp(-x)$ mapping the input values to a range between 0 and 1.

Screenshot the required test F1-score.

```
[ ]  !python  main.py  --model_type  RNN  --preprocess  1  --part  2

     [nltk_data] Downloading package stopwords to /root/nltk_data...
     [nltk_data]   Package stopwords is already up-to-date!
     [nltk_data] Downloading package punkt to /root/nltk_data...
     [nltk_data]   Package punkt is already up-to-date!
     100% 5000/5000 [02:13<00:00, 37.50it/s]
     100% 10000/10000 [00:11<00:00, 869.22it/s]
     Epoch: 0, F1 score: 0.724, Precision: 0.7256, Recall: 0.7244, Loss: 0.6801
     100% 5000/5000 [02:12<00:00, 37.65it/s]
     100% 10000/10000 [00:12<00:00, 819.86it/s]
     Epoch: 1, F1 score: 0.8175, Precision: 0.8263, Recall: 0.8186, Loss: 0.5147
     100% 5000/5000 [02:13<00:00, 37.59it/s]
     100% 10000/10000 [00:12<00:00, 823.95it/s]
     Epoch: 2, F1 score: 0.8452, Precision: 0.8453, Recall: 0.8452, Loss: 0.4561
     100% 5000/5000 [02:12<00:00, 37.66it/s]
     100% 10000/10000 [00:11<00:00, 848.70it/s]
     Epoch: 3, F1 score: 0.8566, Precision: 0.8573, Recall: 0.8567, Loss: 0.422
     100% 5000/5000 [02:12<00:00, 37.62it/s]
     100% 10000/10000 [00:12<00:00, 820.59it/s]
     Epoch: 4, F1 score: 0.8426, Precision: 0.8488, Recall: 0.8432, Loss: 0.4027
     100% 5000/5000 [02:12<00:00, 37.60it/s]
     100% 10000/10000 [00:11<00:00, 872.11it/s]
     Epoch: 5, F1 score: 0.863, Precision: 0.8639, Recall: 0.8631, Loss: 0.3908
     100% 5000/5000 [02:12<00:00, 37.66it/s]
     100% 10000/10000 [00:12<00:00, 831.60it/s]
     Epoch: 6, F1 score: 0.855, Precision: 0.8559, Recall: 0.8551, Loss: 0.3772
     100% 5000/5000 [02:12<00:00, 37.61it/s]
     100% 10000/10000 [00:12<00:00, 825.56it/s]
     Epoch: 7, F1 score: 0.8573, Precision: 0.8631, Recall: 0.8578, Loss: 0.3701
     100% 5000/5000 [02:12<00:00, 37.63it/s]
     100% 10000/10000 [00:11<00:00, 834.52it/s]
     Epoch: 8, F1 score: 0.8709, Precision: 0.871, Recall: 0.8709, Loss: 0.3612
     100% 5000/5000 [02:12<00:00, 37.62it/s]
     100% 10000/10000 [00:12<00:00, 826.97it/s]
     Epoch: 9, F1 score: 0.8641, Precision: 0.8651, Recall: 0.8642, Loss: 0.357
```

Discussion:

Discuss the innovation of the NLP field and your thoughts of why the technique is evolving from n-gram -> LSTM -> BERT.

The evolution of Natural Language Processing (NLP) techniques has been marked by a transition from simpler models like n-grams to more complex

architectures such as LSTM networks and BERT. Initially, n-gram models provided a basic method of analyzing word sequences by counting their frequency. However, they struggled to capture the intricate contextual relationships present in natural language due to their limited scope.

In response to the limitations of n-gram models, LSTM networks were introduced to the NLP landscape. These networks utilized memory cells and gating mechanisms to better model sequential data, allowing them to capture longer-term dependencies. While LSTM networks represented a significant improvement over n-grams, they still faced challenges in understanding complex linguistic structures and capturing nuanced context.

The emergence of BERT marked a major breakthrough in NLP. BERT leverages bidirectional attention mechanisms and pretrained representations to achieve a deep contextual understanding of language. By considering the entire context of a word in a sentence, BERT can capture intricate semantic relationships and nuances, leading to state-of-the-art performance across a wide range of NLP tasks. Thus, the evolution from n-gram models to LSTM networks and BERT represents a continual quest for better contextual understanding and improved performance in NLP.

Describe problems you meet and how you solve them.

1. Don’t how to start building a model, and don’t know the meaning of the parameter in the model.

   solution: Read through the pytorch tutor website and learn the syntax，and search the resource on the internet to understand how to use.

2. Code running is far too slow

   solution: run on colab and use T4 GPU resource

In-context learning

Try to use the LLMs (such as ChatGPT, Gemini, etc) with in-context learning to complete the task above.

---

1. **Few-shot**

- **Few-shot learning** involves supplying the Language Model (LLM) with a limited number of examples illustrating the task it needs to perform.
- https://chat.openai.com/share/dbf52d6a-b405-4f60-b910-ff1f2ae33d14

2. Chain of Thought
    - **Chain of Thought** entails directing the model to articulate the step-by-step reasoning process leading to a conclusion.
    - https://chat.openai.com/share/eab757e4-0fb4-4988-a89e-08331bb86934
3. Socratic Method
    - **Socratic Method** guides the model to a conclusion through a series of questions and answers, fostering critical thinking and deeper understanding.
    - https://chat.openai.com/share/44663495-880d-4410-a0e3-0a3861fac52b

Summary

1. Few-shot learning: Adapts rapidly with minimal data, mastering new tasks.
2. Chain of thought: Guides step-by-step reasoning for clarity and coherence.
3. Socratic method: Provokes critical thought through interactive questioning.