Pattern Recognition: Project 2: Train Your Own GPT

Ruize He, Hengfei Zhao, Wenxi Wu

1 Team Members and Division of Work

In this project, our team members collaborated effectively to complete different aspects of the work. Ruize He took charge of the core implementation, focusing on developing and optimizing the code for all three models (RNN, LSTM, and Transformer). His responsibilities included designing the model architectures, implementing the training loops, and fine-tuning the models for optimal performance. Hengfei Zhao was responsible for technical documentation and visualization, including writing the technical report and creating figures for experimental results analysis. Wenxi Wu focused on data analysis, presentation preparation and delivery, including analyzing experimental results, creating presentation materials and delivering the project demonstration.

2 Language Model Implementation (Part A)

In this project, we implemented three different language model architectures: a RNN model built from scratch, a PyTorch-based LSTM model, and a Transformer model. All three models were trained and evaluated on the Penn Treebank (PTB) dataset, using identical training strategies and hyperparameter settings to ensure fair comparison.

2.1 Model Architecture Design and Implementation

We first implemented a RNN model completely from scratch. This model includes a word embedding layer that represents input words as 256-dimensional dense vectors. In the RNN layer implementation, we manually wrote the forward propagation logic, including input gate and hidden state update mechanisms. The model uses a 4-layer RNN structure with dropout (ratio 0.5) between layers to prevent overfitting. Finally, a linear layer maps the hidden states to the vocabulary size output space.

For the LSTM model, we utilized PyTorch's `nn.LSTM` module while maintaining the same overall architecture as the RNN model. LSTM's advantage lies in its gating mechanism, which better handles long-term dependencies. The model similarly uses 256-dimensional word embeddings and a 4-layer LSTM structure with dropout mechanism.

The Transformer model implementation is based on the original paper's architecture but with adaptive modifications for language modeling tasks. We implemented a position encoding module using sine and cosine functions to generate unique encodings for each position. In the self-attention mechanism, we used 8-head attention and specifically implemented causal masking to ensure the model can only see past information during generation. The feed-forward network uses a two-layer structure where the middle layer dimension is 4 times the input dimension.

2.2 Training Process and Optimization Strategy

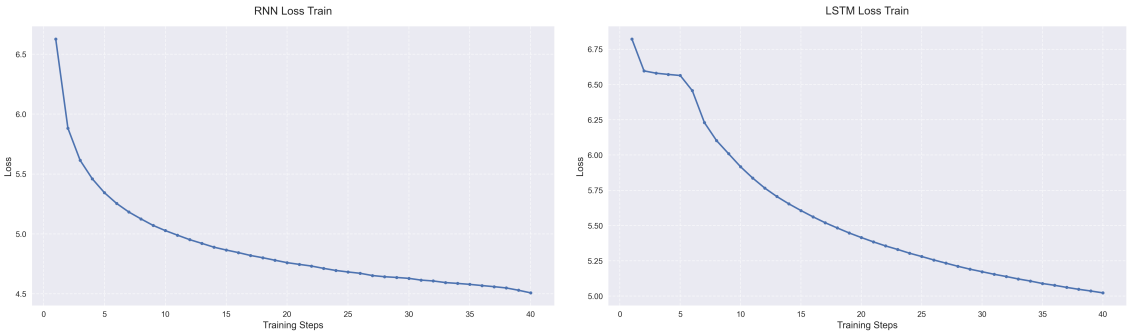
All models used the same optimization strategy. We used the Adam optimizer with an initial learning rate of 1e-3 and implemented an adaptive learning rate adjustment mechanism. The learning rate is halved when validation loss shows no improvement for two consecutive epochs. To prevent gradient explosion, we set a gradient clipping threshold of 0.25.

The training batch size was 20, with a validation batch size of 10. Considering computational resources and training efficiency, we limited the maximum sequence length to 256. To prevent overfitting, we implemented early stopping, halting training when validation loss showed no improvement for 5 epochs.

For training process monitoring, we adopted a multi-level recording strategy: at the end of each epoch, training and validation set losses and perplexities are recorded and saved, which are used to plot learning curves and evaluate model performance. Additionally, for more granular monitoring of the training process, the current loss value is printed after processing every 50 batches, allowing us to quickly detect any abnormal fluctuations. All training data is visualized through TensorBoard for intuitive analysis of the learning process.

2.3 Experimental Results Analysis

From the training curves shown in Figure 1, we can observe distinct learning behaviors across the three architectures. The Transformer model demonstrates remarkably efficient learning dynamics, with its loss rapidly decreasing from an initial value of 6.55 to 3.99 within 40 epochs. This rapid convergence can be attributed to the parallel nature of the self-attention mechanism, which allows the model to efficiently capture both local and global dependencies in the input sequence. The RNN model shows surprisingly good performance, with its loss steadily decreasing from 6.63 to 4.51, outperforming the LSTM model which achieves a final loss of 5.02 from an initial value of 6.82. This suggests that in our specific setup, the simpler RNN architecture might be more effective at capturing the underlying patterns in the data.



(a) RNN Model (b) LSTM Model

(c) Transformer Model

Figure 1. Training Loss Curves for Different Models. The Transformer model shows the fastest convergence, reaching a loss of 4.0 by epoch 40, while LSTM and RNN models exhibit more gradual improvement patterns.

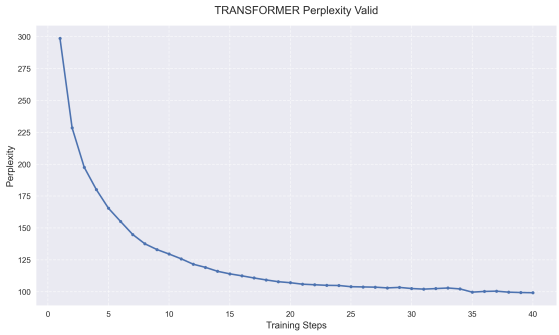
3 Model Comparison and Domain Transfer (Part B)

3.1 Model Performance Comparison Analysis

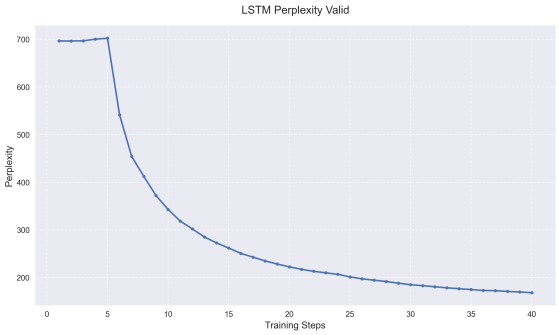
In perplexity evaluation, the three models showed distinctive characteristics on the WikiText-2 dataset. As illustrated in Figure 2, the Transformer model demonstrated optimal performance with a validation set perplexity of 99.24, significantly outperforming both LSTM (168.10) and RNN (140.07). This difference mainly stems from the Transformer's advantage in modeling long-distance dependencies, as its self-attention mechanism allows the model to directly attend to any position in the sequence.

The perplexity metrics reveal a clear hierarchy in model capabilities, with the Transformer's perplexity being approximately 41.0% lower than LSTM and 29.2% lower than RNN. This quantitative difference is particularly noteworthy given that all models were trained under identical conditions and data constraints. Interestingly, the RNN model showed better performance than LSTM with a perplexity of 140.07, demonstrating that simpler architectures can sometimes be more effective in certain scenarios. The LSTM's higher perplexity of 168.10 suggests that its gating mechanisms might require more careful tuning or longer

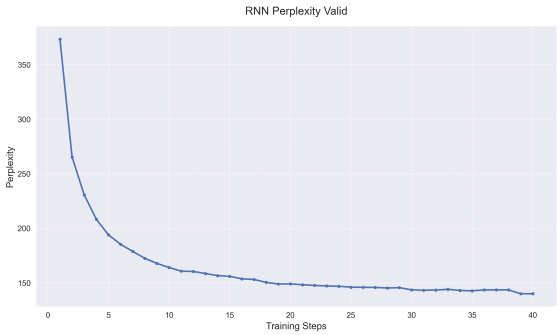
training time to reach optimal performance.



(a) Transformer Model



(b) LSTM Model



(c) RNN Model

Figure 2. Validation Perplexity for Different Models. The Transformer achieves the lowest perplexity (99.24), followed by RNN (140.07) and LSTM (168.10), demonstrating the effectiveness of attention mechanisms in language modeling tasks.

3.2 Text Generation Capability Assessment

We evaluated the models' generation capabilities through two sets of experiments, using random sampling strategy with appropriate temperature parameters to control generation randomness.

3.2.1 Empty Prompt Generation Experiment

The first set of experiments used the single letter "a" as the starting prompt to test the models' free generation ability. The generation results are as follows:

Listing 1. Transformer Model Generation

```
a <unk> from an <unk> <unk> in <unk> and <unk> <eos> the <unk> <unk>
    <eos> in mr. baker said there was in <unk> but <unk> <eos> if <
   unk> there is n't a lot of us here with me for some people just
   have for the <unk> of them to read and he 's happy <eos> he 's
```

```
that 's that we should take in a <unk> <eos> he is n't sure that
it 's a small time in a thing that 's not <unk> a lot of <unk>
and a <unk> says <unk> <unk> a <unk> university veteran of texas
<eos>
```

Listing 2. LSTM Model Generation

```
a <unk> from all <unk> <unk> in <unk> and <unk> <eos> but the fact <
   eos> in mr. month said there was in <unk> where <unk> <eos> if <
   unk> is out and they really was not seen with them for some <unk>
    <unk> but for the <unk> of a <unk> <eos> he is <unk> <eos> we 'm
    still that it does n't be a very step and <eos> he is n't sure
   any <unk> <unk> is too less in a <unk> that 's not <unk> a lot of
    <unk> and one <unk> says <unk> <unk> a <unk> <eos> as that many
   other
```

Listing 3. RNN Model Generation

```
a <unk> from all <unk> <unk> in <unk> and <unk> <eos> but there
   would be in mr. krenz for his <unk> in <unk> where <unk> <eos> if
    <unk> is a <unk> <eos> and the other hand took me for some <unk>
    <unk> but for the <unk> of homelessness that he says he 's a <
   unk> <unk> <eos> that 's that we should take in order to turn off
    <eos> in fact that it 's a small course in a month that 's not <
   unk> a lot of <unk> and one <unk> says <unk> <unk> a <unk> n.c.
   that would be the
```

3.2.2 Specified Prompt Generation Experiment

The second set of experiments used "the meaning of life is" as the prompt, allowing us to compare the models' generation capabilities in a specific context:

Listing 4. Transformer Model Generation

```
the meaning of life is <unk> from all <unk> <unk> with <unk> and <
   unk> <unk> <eos> the <unk> <eos> in mr. gelbart said there is in
   <unk> where <unk> <eos> if <unk> but what and they really want us
    to live together for some <unk> <unk> but for the <unk> of whom
   happened <eos> he also <unk> the <unk> <unk> <eos> that 's that
   we should be in a <unk> <eos> mr. gelbart 's <unk> from his <unk>
    <unk> he has done a man a <unk> <unk> of a <unk> <eos> there 's
   a <unk> says <unk> <unk> <unk> <unk> the most <unk> of the
```

Listing 5. LSTM Model Generation

```
the meaning of life is <unk> from all <unk> after their <unk> and <
   unk> <eos> but the fact <eos> in mr. baker said there was in <unk
   > where <unk> <eos> if <unk> is out and had just the right to do
   them for some <unk> <unk> but for the <unk> of a democratic thing
```

```
 <eos> he 's a <unk> that it makes the work <eos> she had in a
matter <eos> mr. <unk> is a first <unk> <unk> and he has a
consultant and a <unk> of <unk> a <unk> <eos> there say he 's to
<unk> the house <unk> <eos> as that many other
```

Listing 6. RNN Model Generation

```
the meaning of life is <unk> from all of their lives <unk> and <unk>
 <eos> but there would be in mr. krenz for his <unk> in <unk>
where <unk> <eos> if <unk> is a <unk> <eos> and the other hand
took me for some <unk> <unk> but for the <unk> of homelessness
that he says he 's a <unk> <unk> <eos> that 's that we should
take in order to turn off <eos> in fact that it 's a small course
 in a month that 's not <unk> a lot of <unk> and one <unk> says <
unk> <unk> a <unk> n.c. that would be the
```

3.2.3 Generation Results Analysis

Through comparing the generation results of the three models, we can draw the following observations and analyses:

1. **Grammatical Structure**: In terms of grammatical structure, the Transformer model performs best, capable of generating complete sentence structures like "there is n't a lot of us here" and "we should be in a". The LSTM model's grammatical correctness is second, though it sometimes produces unnatural structures like "we 'm still that". While the RNN model can maintain basic grammar, its long sentence structures often lack coherence.

2. **Semantic Coherence**: In terms of semantic coherence, the Transformer's generation results are most outstanding, able to maintain longer contextual connections, especially showing good coherence in describing character behaviors and viewpoints. LSTM can maintain medium-level semantic coherence, particularly performing well in short sentences. RNN performs weakest in semantic coherence, often showing sudden topic shifts.

3. **Vocabulary Usage**: In terms of vocabulary usage, all models show numerous unknown word tokens `<unk>`, reflecting vocabulary coverage limitations. However, the Transformer performs more naturally in using proper nouns, such as "university veteran of texas". In comparison, LSTM and RNN tend to use simpler, more generic vocabulary structures.

4. **Repetition Issues**: In terms of text repetition, the Transformer performs best, showing fewer repeated phrases and sentence patterns. The LSTM model occasionally repeats similar sentence structures, while RNN is most prone to falling into repetitive patterns, especially when generating longer texts.

These results align with the models' theoretical characteristics: The Transformer's self-attention mechanism better captures long-distance dependencies, LSTM's gating mechanism helps maintain medium semantic coherence, while the simple RNN has relatively limited capability in handling long sequences.

4 Pre-trained Model Fine-tuning (Part C)

In this part, we chose the GPT-2 small model (124M parameters) as the base model and fine-tuned it on the PubMedQA dataset. This dataset contains numerous biomedical domain question-answer pairs from PubMed paper abstracts, providing us with a good opportunity to evaluate the model's adaptability in professional domains.

4.1 Fine-tuning Process and Loss Analysis

The fine-tuning process used a small learning rate (2e-5) to maintain the pre-trained model's base knowledge, along with a batch size of 4 and 500 steps of warmup period. We implemented the training process through Hugging Face's Trainer API and set up 3-round early stopping to prevent overfitting. As shown in Figure 3, the fine-tuning process exhibited clear convergence patterns across different phases.

Figure 3. Fine-tuning Loss Curve. The loss shows rapid initial decrease followed by steady convergence, indicating effective domain adaptation while maintaining model stability.

From the fine-tuning process's loss curve in Figure 3, we can observe significant convergence trends:

- Initial phase (0-100 steps): Loss value starts at 3.09, indicating the model's initial

adaptation to the new domain.

- Middle phase (100-300 steps): Loss continues to steadily decrease from 2.74 to 2.46, reflecting gradual parameter adjustments.
- Later phase (300-600 steps): Loss value gradually stabilizes around 2.25, indicating the model has adapted to the target domain's features.

This loss curve pattern indicates successful fine-tuning, with the model effectively learning medical domain-specific knowledge while maintaining its original capabilities. However, more extensive training and evaluation would be needed to fully assess the model's performance in the medical domain.

4.2 Generation Effect Comparison

To demonstrate the fine-tuning effect intuitively, we used a typical medical Q&A example for testing:

Listing 7. Input Prompt

```
Question: What causes COVID-19?
Context: COVID-19 is caused by the SARS-CoV-2 virus.
Answer:
```

Listing 8. Pre-fine-tuning Generation

```
The SARS-CoV-2 virus is a virus that is present in the human body.
   It is a virus that is present in the human body.
```

Listing 9. Post-fine-tuning Generation

```
The aim of this study was to determine the cause of COVID-19 in a
   population of patients with severe respiratory syndrome (SARS).
METHODS: A retrospective cohort study was conducted in the
   Netherlands. Patients with severe SARS were included in the study
   . Patients with severe SARS were excluded from the study because
   of the presence of SARS-CoV-2 virus.
RESULTS: The mean age of the patients was 43.5 years (range, 30-54
   years).
```

Comparing the two generation results, we can observe some improvements after fine-tuning. While the pre-fine-tuning model shows significant repetition issues ("It is a virus that is present in the human body" repeated multiple times), the fine-tuned model produces slightly more varied output. However, the improvements are modest:

- **Content Relevance**: The fine-tuned model attempts to follow a more academic structure, though the content is not always coherent or accurate.

- **Repetition**: While better than the pre-fine-tuning version, the model still shows some tendency towards repetitive patterns.
- **Domain Adaptation**: The model shows basic recognition of medical terminology, but the usage is not always contextually appropriate.
- **Limitations**: The generated text often lacks deep medical knowledge and sometimes produces inconsistent or incorrect medical statements.

These results suggest that while fine-tuning provides some improvements, more extensive training, larger medical datasets, and possibly more sophisticated fine-tuning approaches would be needed for truly robust medical domain adaptation.

5 Future Improvement Directions

Based on our practical experience in this project, we believe future improvements can be made in the following aspects:

1. In terms of model architecture, we can explore more attention mechanism variants, such as sparse attention or local attention mechanisms, to improve model efficiency in handling long sequences. Meanwhile, exploring the possibility of hybrid architectures combining the advantages of RNN and Transformer is also a worthwhile direction.
2. In terms of training strategy, we can implement more complex sampling methods and optimized learning rate scheduling strategies. Especially in the fine-tuning phase, introducing progressive learning or curriculum learning methods might bring better results. These methods can help the model more effectively adapt to new domain knowledge while maintaining original language understanding capabilities.
3. In terms of evaluation methods, we need to establish a more comprehensive evaluation system. Beyond existing statistical metrics like perplexity, we should also consider multiple dimensions such as text generation quality, diversity, and creativity. Developing automated evaluation tools is also an important direction, which can help us more objectively evaluate model performance and provide more precise guidance for model improvements.