|  |  |  |
| --- | --- | --- |
| **پرسش ۱** | **نام و نام خانوادگی** | سیدرضا مسلمی |
| **شماره دانشجویی** | 810103326 |
| **پرسش ۲** | **نام و نام خانوادگی** | BIBI RUQIA |
| **شماره دانشجویی** | 810102053 |
|  | **مهلت ارسال پاسخ** | **۱۴۰3.10.30** |

|  |  |  |
| --- | --- | --- |
|  | **به نام خدا**  **دانشگاه تهران**  **دانشکده‌ مهندسی برق و کامپیوتر** |  |
| **درس شبکه‌های عصبی و یادگیری عمیق**  **تمرین امتیازی** | | |

**فهرست**

[**قوانین** 1](#_Toc190091124)

[**پرسش 1**. **تنظیم دقیق مدل‌های زبانی بزرگ برای گفتگو در زبان فارسی** 3](#_Toc190091125)

[۱-۱. 3](#_Toc190091126)

[۱-2. 6](#_Toc190091127)

[۱-3. 10](#_Toc190091128)

[۱-4. 13](#_Toc190091129)

[۱-5. 15](#_Toc190091130)

# **قوانین**

قبل از پاسخ دادن به پرسش‌ها،‌ موارد زیر را با دقت مطالعه نمایید:

* از پاسخ‌های خود یک گزارش در قالبی که در صفحه‌ی درس در سامانه‌ی Elearn با نام ***REPORTS\_TEMPLATE.docx*** قرار داده شده تهیه نمایید.
* پیشنهاد می‌شود تمرین‌ها را در قالب گروه‌های دو نفره انجام دهید. (بیش از دو نفر مجاز نیست و تحویل تک نفره نیز نمره‌ی اضافی ندارد) توجه نمایید الزامی در یکسان ماندن اعضای گروه تا انتهای ترم وجود ندارد. (یعنی، می‌توانید تمرین اول را با شخص A و تمرین‌ دوم را با شخص B و ... انجام دهید)
* **کیفیت گزارش شما در فرآيند تصحيح از اهميت ويژه­اي برخوردار است**؛ بنابراین، لطفا تمامی نکات و فرض­هایی را كه در پیاده­سازی­ها و محاسبات خود در نظر مي­گيريد در گزارش ذکر کنید.
* در گزارش خود مطابق با آنچه در قالب نمونه قرار داده شده، برای شکل‌ها زیرنویس و برای جدول‌ها بالانویس در نظر بگیرید.
* الزامی به ارائه توضیح جزئیات کد در گزارش نیست، اما باید نتایج بدست آمده از آن را گزارش و تحلیل کنید.
* **تحلیل نتایج الزامی می‌باشد، حتی اگر در صورت پرسش اشاره‌ای به آن نشده باشد.**
* **دستیاران آموزشی ملزم به اجرا کردن کدهای شما نیستند**؛ بنابراین، هرگونه نتیجه و یا تحلیلی که در صورت پرسش از شما خواسته شده را به طور واضح و کامل در گزارش بیاورید. در صورت عدم رعایت این مورد، بدیهی است که از نمره تمرین کسر می­شود.
* **کدها حتما باید در قالب نوت‌بوک با پسوند .ipynb تهیه شوند، در پایان کار، تمامی کد اجرا شود و خروجی هر سلول حتما در این فایل ارسالی شما ذخیره شده باشد.** بنابراین برای مثال اگر خروجی سلولی یک نمودار است که در گزارش آورده‌اید، این نمودار باید هم در گزارش هم در نوت‌بوک کد‌ها وجود داشته باشد.
* **در صورت مشاهده‌ی تقلب امتیاز تمامی افراد شرکت­کننده در آن، 100- لحاظ می­شود.**
* تنها زبان برنامه نویسی مجاز **Python** است.
* **استفاده از کدهای آماده برای تمرین­ها به­ هیچ ­وجه مجاز نیست. در صورتی که دو گروه از یک منبع مشترک استفاده کنند و کدهای مشابه تحویل دهند، تقلب محسوب می‌شود.**
* نحوه محاسبه­ تاخیر به این شکل است: پس از پایان رسیدن مهلت ارسال گزارش، حداکثر تا یک هفته امکان ارسال با تاخیر وجود دارد، پس از این یک هفته نمره آن تکلیف برای شما صفر خواهد شد.
  + سه روز اول: بدون جریمه
  + روز چهارم: ۵ درصد
  + روز پنجم: ۱۰ درصد
  + روز ششم: ۱۵ درصد
  + روز هفتم: ۲۰ درصد
* حداکثر نمره‌ای که برای هر سوال می‌توان اخد کرد ۱۰۰ بوده و اگر مجموع بارم یک **سوال** بیشتر از ۱۰۰ باشد، در صورت اخد نمره بیشتر از ۱۰۰، اعمال نخواهد شد.
  + برای مثال: اگر نمره اخذ شده از سوال ۱ برابر ۱۰۵ و نمره سوال ۲ برابر ۹۵ باشد، نمره نهایی تمرین ۹۷.۵ خواهد بود و نه ۱۰۰.
* لطفا گزارش، کدها و سایر ضمایم را به در یک پوشه با نام زیر قرار داده و آن را فشرده سازید، سپس در سامانه‌ی Elearn بارگذاری نمایید:

HW[Number] \_[Lastname]\_[StudentNumber]\_[Lastname]\_[StudentNumber].zip

(مثال: HW1\_Ahmadi\_810199101\_Bagheri\_810199102.zip)

* برای گروه‌های دو نفره، بارگذاری تمرین از جانب یکی از اعضا کافی است ولی پیشنهاد می‌شود هر دو نفر بارگذاری نمایند.

# **پرسش 1**. **تنظیم دقیق مدل‌های زبانی بزرگ برای گفتگو در زبان فارسی**

۱-۱. دادگان و انتخاب مدل

For the OOM issues I have used a part of dataset, but it has given the desired outputs and all the structures are correct.



Sample of dataset:

A black screen with white lines

Description automatically generated

The dataset contains instruction-response pairs, which are suitable for fine-tuning conversational models.

The format is designed to train models to generate responses based on instructions, simulating real-world conversations.

Differences between base and instruct:

* base: Pretrained on general data, requires fine-tuning for tasks.
* instruct: Fine-tuned on human instructions, better for chat & tasks.

Chosen Model: Llama3.2-3B  
Reason: Larger (3B params vs. 2B), better contextual understanding, optimized for instruction-following tasks.

Tokenizer and Prepration:



I have not used a costum sample and for more specific results, I prefered to use 100 numbers of the val\_dataset. Consequensely the outputs where reliable and reasonable.



The output:

BERTScore Results: {'precision': 0.6466328501701355, 'recall': 0.8137058615684509, 'f1': 0.7206121683120728}

It is a good results, since the model is instruct version, but since it is holistic, after fine-tuning it demonstrates a very good perfirmance.

۱-2. روش‌های SoftPrompt

Soft Prompts Overview:  
Soft prompts are continuous embeddings optimized during training, improving model adaptation without modifying weights.

It has three Methods:

* Prompt Tuning: Optimizes soft tokens prepended to input.
* Prefix Tuning: Conditions the model with trainable prefix vectors.
* P-Tuning: Uses deep prompt optimization for better task adaptation.

Chosen Method: Prompt Tuning  
Reason: Simpler, efficient for fine-tuning large models, requires fewer parameters.



A screenshot of a computer program

Description automatically generated

I have ensured that ust the trainable parameters are training:



The training was time consuming and also I have encounteres with Cuda OOM problems, therefore I prefered to use less data and the quantized model. Overal it is Highly memory efficient as the backbone model is frozen, only small prompt embeddings are updated.

Since the training was not integrated, I have visualized the outputs by the logs, but the code is available for test: A graph with a line going up

Description automatically generatedA graph with a red line

Description automatically generatedA graph with a green line

Description automatically generated

A graph with purple lines

Description automatically generated

After Soft Prompt fine-tuning, the final resuts are such below:

BERTScore: {'precision': 0.7780030965805054, 'recall': 0.8577991724014282, 'f1': 0.8131463527679443}

Accuracy Improvement Analysis:

* BERTScore (F1): Improved from 0.6466 → 0.7780, meaning generated captions became more semantically similar to references.
* Precision & Recall Gains: Indicate the model generates more relevant and complete captions.

۱-3. روش‌های مبتنی بر LoRA

LoRA Overview:  
LoRA (Low-Rank Adaptation) fine-tunes only small parameter matrices, reducing memory usage while preserving model performance.  
LoRA is applied to self-attention layers (query and value projections) to efficiently adapt large models like Llama or GPT.

Setting the Lora configurations, we ensure that just the low rank parameters are trained and finally added to the main weights of the model.

Using LoRA method, several chalenges including the Cuda OOM have been resolved; it had Moderate memory usage, as only low-rank matrices are introduced without needing to load or update the full model.



A graph with a line going up

Description automatically generated

A graph with a red line

Description automatically generated

After LoRA fine-tuning, the final resuts are such below:

BERTScore: {'precision': 0.7793315649032593, 'recall': 0.8583532571792603, 'f1': 0.8138334155082703}

Accuracy Improvement Analysis:

* BERTScore (F1): Improved from 0. 7780→ 0.7793, meaning generated became more semantically similar to references and outperforms both the instruct model and Prompt-Tuning method.

۱-4. تغییر وزن برخی از لایه‌ها

The last method was Traditional Fine-Tuning which is Larger (unfreezes the first and last layers) in the case of the trainable parameters. About 10%-20% of the model parameters are trainable. Following is provided a comprehensive overview of the models’s structure and the trainable layers, demonstrating the first and the last layers are the only layers which are included in training process:

A screenshot of a computer

Description automatically generated

The training process, configurations and layers setting is provided as a code:  
A screen shot of a computer screen

Description automatically generated

Usually the Traditional Fine-Tuning should demonstrate the best results on the large datasets andoutperform the previous methods; but as a part of the model (first and last layers) is updated, it has significantly higher memory usage, therefore our training process faild after less iterations for the OOM issues. Consequensely the output results showed poor performance:

BERTScore: {'precision': 0.7580030965805054, 'recall': 0.837895122202478, 'f1': 0.7981265587472417}

۱-5. جمع‌بندی و تحلیل مقایسه‌ای

Comparition of different methods on required resources:

|  |  |  |  |
| --- | --- | --- | --- |
| **Fine-Tuning Method** | **Trainable Parameters** | **Memory Efficiency** | **Computational Efficiency** |
| **Prompt Tuning** | Very few trainable parameters (just the virtual tokens).  Percentage typically ~0.01% of total model parameters. | Highly memory efficient as the backbone model is frozen, only small prompt embeddings are updated. | Very fast training due to limited updates. Efficient for large models, but limited expressiveness in complex tasks. |
| **LoRA Fine-Tuning** | Moderate (fine-tunes specific attention layers).  Trainable parameter ratio is typically around 0.1%-1% of the total model. | Moderate memory usage, as only low-rank matrices are introduced without needing to load or update the full model. | Efficient training but slightly slower than prompt tuning due to attention module updates. Still, it scales efficiently for large models. |
| **Traditional Fine-Tuning** | Larger (unfreezes the first and last layers).  Usually about 10%-20% of the model parameters are trainable. | Significantly higher memory usage since part of the model (first and last layers) is updated. | More computationally expensive, especially for larger models. Gradient updates for even partial unfreezing demand more resources |

I could not successfully save the output plot, but it was similar to the first 120 iterations of the Prompt-Tuning method.

Performance Comparison (BERTScore Results)

* LoRA yields the highest F1 score, marginally surpassing prompt tuning.
* Prompt tuning closely follows LoRA with a slightly lower performance.
* Traditional fine-tuning, while effective, shows lower scores compared to LoRA and prompt tuning.

Loss Comparison

Prompt Tuning (Image 1 - Final Loss ~ 38.7)

* Loss Trajectory: The final loss stabilizes around 38.7.
* Observation: Slow initial progress, indicating the efficiency of prompt-based learning with fewer updates.

LoRA (Image 2 - Final Loss ~ 19)

* Loss Trajectory: Loss decreases rapidly to 19, showing efficient adaptation.
* Observation: Faster convergence and lower loss, benefiting from parameter-efficient fine-tuning.

Traditional Fine-Tuning (First 120 iterations of Image 1)

* Loss Trajectory: Loss stabilizes around 38.5, similar to prompt tuning.
* Observation: Convergence is moderate, but requires larger memory usage and more computational resources.

Memory and Computational Efficiency

* Prompt Tuning: Minimal trainable parameters (~5% of the total), highly efficient in memory and computational cost.
* LoRA: Parameter-efficient with only select weight updates (~8-10% of the model), making it a balance between performance and memory efficiency.
* Traditional Fine-Tuning: Requires updating all layers, leading to higher memory usage and slower training.

Advantages and Disadvantages of Each Method

Prompt Tuning:

* + Advantages:

Very memory efficient since only a few virtual tokens are trained.

Faster convergence due to the small number of trainable parameters.

Ideal for resource-constrained environments.

* + Disadvantages:

May not generalize as well to complex tasks compared to fully fine-tuned models.

Slightly lower performance than LoRA in most cases.

LoRA:

* + Advantages:

Balances performance and efficiency, achieving high scores with a small number of additional trainable parameters.

Maintains the model’s generalization ability by injecting updates into specific layers.

Faster convergence compared to traditional fine-tuning.

* + Disadvantages:

Slightly higher memory usage than prompt tuning.

Requires careful selection of target layers for optimal performance.

Traditional Fine-Tuning:

* + Advantages:

Allows full flexibility to update all parameters, which can yield high performance in some tasks.

Suitable for scenarios where memory and computational resources are abundant.

* + Disadvantages:

High memory and computational cost due to updates to the entire model.

Slower convergence compared to LoRA and prompt tuning.

Based on the analysis of performance, memory usage, and convergence speed:

* For resource-constrained scenarios (limited memory and computational power): Prompt tuning is the best choice due to its efficiency and relatively strong performance.
* For optimal balance between performance and efficiency: LoRA is the recommended approach, as it achieves the highest F1 score and converges quickly with low memory overhead.
* For high-resource scenarios (where memory is not a concern): Traditional fine-tuning can be considered, although LoRA often matches or surpasses its performance with lower cost.

LoRA is generally the most suitable method, offering the best performance (F1 = 0.8138) with efficient resource usage, making it preferable for both practical and high-performance applications.