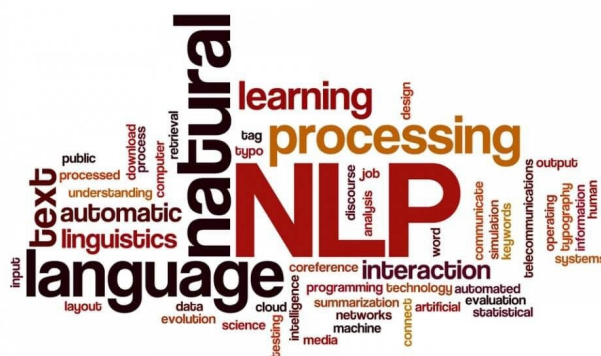


بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ



# دانشکده مهندسی برق و کامپیوتر

## پردازش زبان‌های طبیعی - تمرین چهارم

سید مهدی رضوی

استاد : آقای دکتر فیلی

خرداد ماه ۱۴۰۳

## فهرست مطالب

۳	تمرین اول	۱
۳	Roberta Large Model	۱.۱
۴	Lora Config	۲.۱
۷	P Tuning	۳.۱
۹	تمرین دوم	۲
۹	llama-3-8B	۱.۲
۱۰	In Context Learning (Zero Shot)	۲.۲
۱۳	In Context Learning (One Shot)	۳.۲
۱۴	QLora	۴.۲

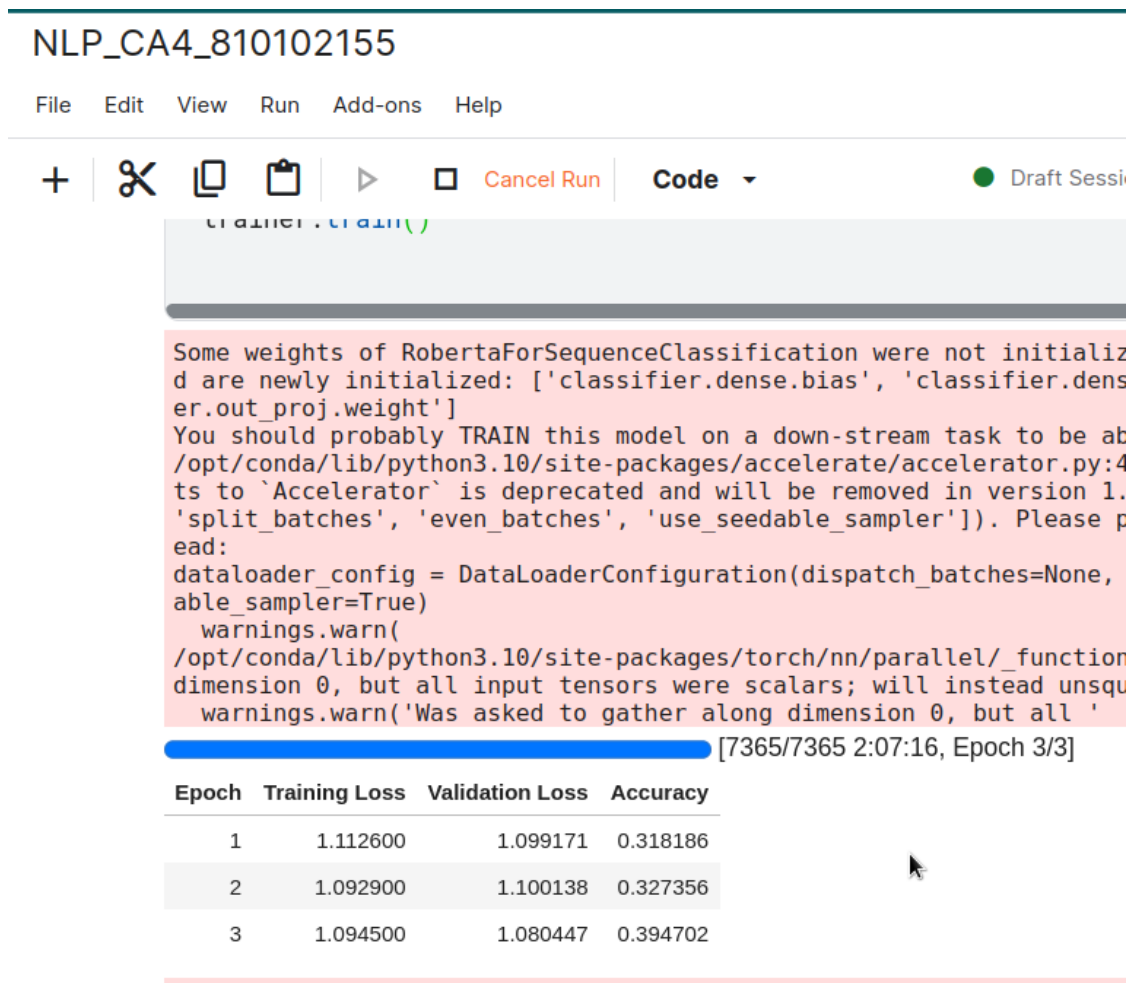
## فهرست تصاویر

۳	Roberta Large Without Tuning	۱
۴	Roberta Large Config	۲
۵	Lora Hyperparameters	۳
۶	Roberta Large Results	۴
۷	PTuning Config	۵
۸	PTuning Hyperparameters	۶
۸	Roberta Large PTuning	۷
۹	Quantization Config Using BitsAndBytes Library	۸
۱۰	نمونه پرامپت‌های بررسی شده در مدل لااما ۳	۹
۱۱	Zero Shot Learning with LLama-3-8B	۱۰
۱۱	Zero Shot Learning with LLama-3-8B(Predict relationship)	۱۱
۱۲	Zero Shot Learning with LLama-3-8B(Report)	۱۲
۱۳	One Shot Learning with LLama-3-8B	۱۳
۱۴	QLora Config	۱۴

## ۱ تمرین اول

## ۱.۱ Roberta Large Model

برای این تمرین ما ابتدا از مدل لارج روبرتا بر روی مجموعه داده MultiNLI استفاده خواهیم کرد. زمان صرف شده برای یادگیری این مدل در ۳ ایپاک حدود دو ساعت بوده است و دقت بیشینه این مدل ۳۹ درصد خواهد بود. تابع ضرر در طول مدت آموزش بر حسب ایپاک ها نزولی بوده است.



شکل ۱: Roberta Large Without Tuning

## ۲.۱ Lora Config

## Low Rank Adaption

کانفیگ ما برای قسمت Lora و همچنین نتایج حاصل از اجرای این مدل به شرح زیر خواهد بود . رویکردهای کارآمد پارامتر برای فاین تیون وجود دارد که اثبات شده است. اگرچه اکثر این رویکردها عملکرد کمتری را به همراه داشته اند، اما انطباق با رتبه پایین (LoRA) به مدل ازپیش آموزش دیده کمک خواهد کرد که نتایج از پیش آموزش دیده در طول مدتی که کمتر از آن استفاده شده است ، باز هم به خاطر مدل بماند و از فراموشی مدل جلوگیری خواهد کرد.

LoRA یک روش فاین تیون بهبود یافته است که در آن به جای تنظیم دقیق همه وزن هایی که ماتریس وزن مدل زبان بزرگ از پیش آموزش داده شده را تشکیل می دهند، دو ماتریس کوچک تر که تقریباً به این ماتریس بزرگ تر نزدیک می شوند، تنظیم دقیق می شوند. این ماتریس ها Adaptor LoRA (تنظیم گر لورا) را تشکیل می دهند. این آداپتور دقیق تنظیم شده سپس در مدل از پیش آموزش دیده بارگذاری شده و برای استنباط استفاده می شود.

به عنوان مثال ، در کانفیگ زیر ، به جای تنظیم یک ماتریس با ابعاد  $V * V$  که نیاز به حدوداً وی به توان دو عملیات کاهش گرادیان و تنظیم کردن خواهد داشت ،

ما ماتریس را به حاصل ضرب دو ماتریس  $V * 16$  تبدیل خواهیم کرد و سپس به تنظیم کردن پارامترها می پردازیم. به شدت این عملیات به کاهش میزان عملیات تنظیم کردن پارامتر و کاهش گرادیان کمک خواهد کرد.

NLP\_CA4\_Q1\_a

File Edit View Run Add-ons Help

+ [Icons] Run All Code

● Draft Session off (run a cell to start) [Icons]

```
import torch
from datasets import load_dataset
from transformers import RobertaTokenizer, RobertaForSequenceClassification, TrainingArguments, Trainer
from peft import LoraConfig, get_peft_model

dataset = load_dataset("nyu-mll/multi_nli")

tokenizer = RobertaTokenizer.from_pretrained("roberta-large")
model = RobertaForSequenceClassification.from_pretrained("roberta-large", num_labels=3)

peft_config = LoraConfig(
    lora_alpha=8,
    lora_dropout=0.1,
    r=16,
    bias="none",
    task_type="CAUSAL_LM",
)

model = get_peft_model(model, peft_config)
```

شکل ۲: Roberta Large Config

همانطور که از نتایج مشخص است ، این بار آموزش با دقت بالاتر و در مدت زمان کمتری شکل گرفته است. در ایپاک سوم ، به دقت حدود ۶۸ درصد دست یافته ایم. همچنین این فرضیه به شدت واضح است که اگر در پارامترهای آموزش میزان نرخ یادگیری را کاهش بدهیم ، دقت در همین ایپاک های پایین نیز رشد محسوس تری خواهد داشت.

NLP\_CA4\_Q1\_a Draft saved

File Edit View Run Add-ons Help

+ ✂ 📄 📌 ▶ ▶▶ Run All Code ▾ ● Draft

```
predictions = np.argmax(logits, axis=-1)
return {'accuracy': accuracy_score(labels, predictions)}

training_args = TrainingArguments(
    output_dir='./results',
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset_reduced,
    eval_dataset=encoded_dataset['validation_matched'],
    compute_metrics=compute_metrics
)

trainer.train()
```

شکل ۳: Lora Hyperparameters

## NLP\_CA4\_810102155

Draft saved

File Edit View Run Add-ons Help

+ ✂ 📄 📌 ▶ ▶▶ Run All Code ▾ ● Draft Ses

```
trainer.train()
```

Some weights of RobertaForSequenceClassification were not initialized and are newly initialized: ['classifier.dense.bias', 'classifier.dense.out\_proj.weight']  
You should probably TRAIN this model on a down-stream task to be able to use it without these weight initializations.

/opt/conda/lib/python3.10/site-packages/accelerate/accelerator.py:101: DeprecationWarning: `Accelerator` is deprecated and will be removed in version 0.15.0. Please use `Accelerator` instead.  
[7365/7365 1:37:38, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy
1	1.085200	1.070879	0.394396
2	0.955700	0.919808	0.606521
3	0.857600	0.803203	0.681406

شکل ۴: Roberta Large Results

## ۳.۱ P Tuning

تنظیم پرامپت ، که فقط پرامپت‌های پیوسته را با یک مدل زبان ثابت تنظیم می‌کند، به طور قابل توجهی ذخیره‌سازی هر وظیفه و استفاده از حافظه در زمان آموزش را کاهش می‌دهد.

تنظیم پرامپت برای فاین تیون کردن به ما این امکان را می‌دهد که دیگر به صورت دستی به طراحی پرامپت نپردازیم . همچنین این روش با استفاده از یک رمزکننده یک پرامپت را به بردارهای Embedding اضافه خواهیم کرد.

در واقع P-tuning . امبدینگ‌های سریع قابل آموزش را به ورودی اضافه می‌کند که توسط یک رمزگذار سریع برای یافتن یک پرامپت بهتر بهینه شده است.

```

NLP_CA4_Q1_a Draft saved
File Edit View Run Add-ons Help

+ ✂ 📄 📌 ▶ ▶▶ Run All Code ▾ Draft Session off (run a cell to st

from transformers import TrainingArguments, Trainer
from sklearn.metrics import accuracy_score
from peft import get_peft_model, PromptEncoderConfig

dataset = load_dataset("nyu-mll/multi_nli")

tokenizer = RobertaTokenizer.from_pretrained("roberta-large")
model = RobertaForSequenceClassification.from_pretrained("roberta-large", num_labels=3)

peft_config = PromptEncoderConfig(
    task_type="SEQ_CLS",
    num_virtual_tokens=20,
    encoder_hidden_size=model.config.hidden_size
)

model = get_peft_model(model, peft_config)

def preprocess_function(examples):
    return tokenizer(
        examples["premise"],
        examples["hypothesis"],
        truncation = "only_first",
        padding="max_length",
        max_length=128 ,
        return_overflowing_tokens=True,
        stride = 0
    )

```

شکل ۵: PTuning Config

همانطور که از تصویر نتایج نیز مشخص است ، الگوریتم با میزان دقت ۴۳ درصد در ایپاک ۳ خواهد بود. باز هم در صورت کاهش میزان نرخ یادگیری کمی مدت زمان آموزش افزایش خواهد یافت اما دقت ارزیابی (Validation) افزایش خواهد یافت.

تمپرچر Temperature یک فرامتر از LSTM ها (و به طور کلی شبکه های عصبی) است که برای کنترل تصادفی بودن پیش بینی ها با مقیاس گذاری لاجیت ها قبل از اعمال تصمیم softmax استفاده می شود. مقیاس بندی دما به طور گسترده ای برای بهبود عملکرد برای کارهای NLP که از لایه تصمیم Softmax استفاده می کنند، استفاده شده است.

پرامپت سخت: پرامپت گسسته طراحی شده توسط محققان به عنوان متن واضح.

پرامپت نرم: یک بردار پیوسته قابل آموزش.

تنظیم پرامپت: بردارهای پیوسته قابل آموزش پرامپت های نرم در فاین تیون به روز می شوند.

NLP\_CA4\_Q1\_a

Draft saved

File Edit View Run Add-ons Help







 Run All
 **Code**


 Draft Sess

```
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return {'accuracy': accuracy_score(labels, predictions)}

training_args = TrainingArguments(
    output_dir='./results',
    evaluation_strategy="epoch",
    save_strategy="steps",
    learning_rate=2e-5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset_reduced,
    eval_dataset=encoded_dataset['validation_matched'],
    compute_metrics=compute_metrics
)

trainer.train()
```

شكل ٦: PTuning Hyperparameters







NLP\_CA4\_Q1\_b

Draft saved

File Edit View Run Add-ons Help

 Share

Save Version 0






 Run All
 
 Code

- Draft Session (1h:24m)

```

Please pass an accelerator.DataLoaderConfiguration instead:
data_loader_config = DataLoaderConfiguration(dispatch_batches=None, split_batches=False, even_batches=True, use_seedable_sampler=True)
warnings.warn(
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit:

```

```
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb version 0.17.0 is available! To upgrade, please run: $ pip install wandb --upgrade
Tracking run with wandb version 0.16.6
```

Run data is saved locally in /kaggle/working/wandb/run-20240530\_194147-ueh7efp4

Syncing run **apricot-silence-5** to **Weights & Biases** (docs)

View project at [https://wandb.ai/tehran\\_university/huggingface](https://wandb.ai/tehran_university/huggingface)  
View run at [https://wandb.ai/tehran\\_university/huggingface/runs/ueh7efp4](https://wandb.ai/tehran_university/huggingface/runs/ueh7efp4)

[14727/14727 1:15:54, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy
1	1.114500	1.097582	0.341111
2	1.079600	1.083682	0.419969
3	1.040300	1.076257	0.438207

```
[5]: TrainOutput(global_step=14727, training_loss=1.0937511657530417, metrics={'train_runtime': 4703.5227, 'train_samples_per_second': 25.047, 'train_steps_per_second': 3.131, 'total_flos': 2.754304523131392e+16, 'train_loss': 1.0937511657530417, 'epoch': 3.0})
```

+ Code

شکل ۷: Roberta Large PTuning



## ۲ تمرین دوم

## ۱.۲ llama-3-8B

به مدل لااما ۳ می‌رسیم. با توجه به هدف این تمرین می‌توانیم این‌گونه بیان‌کنیم که مهم‌ترین هدف از این تمرین این است که ما با رویکردهایی آشنا شویم که بتوانیم مدل‌های زبانی بزرگ را در حافظه ثانویه که همان RAM می‌باشد ذخیره‌کنیم. با توجه به امکانات سخت‌افزاری موجود در سایت kaggle ما به صورت مستقیم، نتوانستیم این مدل را در GPU بارگذاری کنیم. به همین منظور این مدل را کوانتیزه کردیم و سپس بارگذاری نمودیم. از کتابخانه بیتس اند بایتس برای این منظور استفاده نمودیم.

## NLP\_CA4\_Q2\_a\_Version(2)

Draft saved

File Edit View Run Add-ons Help

+ | ✂ | 📄 | ▶ | ☐ Cancel Run | Code ▾



```
from transformers import BitsAndBytesConfig
import torch

bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_use_double_quant=False
    # load_in_8bit_fp32_cpu_offload=True
)

device_map = {
    'transformer.h.0': 'cuda:1',
    'transformer.h.1': 'cuda:2',
}

model_name = 'meta-llama/Meta-Llama-3-8B'
tokenizer = AutoTokenizer.from_pretrained(model_name)

model = AutoModelForCausalLM.from_pretrained(
    model_name,
    quantization_config=bnb_config,
    device_map='auto'
)
```

شکل ۸: Quantization Config Using BitsAndBytes Library

مطابق با تنظیمات تصویر ۸ مدل را با داده‌های ۴ بیتی بارگذاری خواهیم کرد.

## ۲.۲ In Context Learning (Zero Shot)

در موضوع In Context Learning ما باید به این موضوع بپردازیم که اگر مدل بر روی داده‌های ما آموزش ندیده باشد، به چه صورت می‌تواند پیش‌بینی کند. در نتیجه ما در این قسمت مدل را دیگر آموزش نمی‌دهیم. فقط برجسب‌های پیش‌بینی شده توسط مدل لاما ۳ به وسیله پرامپت ما را بررسی خواهیم کرد. در تصویر شماره ۱۰ نتایج طبقه‌بندی را مشاهده خواهیم کرد. همانطور که پیش‌بینی هم می‌شد، دقت پیش‌بینی برجسب‌ها نیز خیلی ضعیف خواهد بود. نتایج خروجی توسط مدل به صورت ۳ نوع کاراکتر که عبارت خواهند بود از =، ، :، o آخرین کاراکتر خروجی مدل لاما به پرسش پرامپت ما یکی از ۳ نوع کاراکتر بالا خواهند بود که متناظرا آن‌ها را به صفر و یک و دو نظیر کردیم.

```
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
from datasets import load_dataset
from torch.utils.data import DataLoader
import gc
from sklearn.metrics import classification_report

# Load dataset
dataset = load_dataset("nyu-ml1/multi_nli")

# Model and tokenizer
model_name = 'meta-llama/Meta-Llama-3-8B'
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(model_name)

# Format prompt function
def format_prompt(premise, hypothesis):
    description = (
        "relationship of contradicts means that the premise contradicts the hypothesis "
        "Entails means that the premise entails the hypothesis "
        "neutral means that neither is necessarily true. "
        "Premise is : " + premise + " Hypothesis is : " + hypothesis +
        ". if Premise and Hypothesis have entailment relation return 0. "
        "else if Premise and Hypothesis have contradict relation return 2. "
        "else if Premise and Hypothesis have neutral relation return 1. Thank you so much."
    )
    return description
```

شکل ۹: نمونه پرامپت‌های بررسی شده در مدل لاما ۳

NLP\_CA4\_Q2\_a\_Version(2) Draft saved

File Edit View Run Add-ons Help

+ ✂ 📄 📋 ▶ ▶▶ Run All Code ▾ ● Draft

hypothesis : Lorenzo and Giuliano were related to one another .

	precision	recall	f1-score	support
0	0.37	0.57	0.45	75
1	0.33	0.06	0.10	50
2	0.35	0.35	0.35	75
accuracy			0.36	200
macro avg	0.35	0.33	0.30	200
weighted avg	0.35	0.36	0.32	200

+ Code + Markdown

شکل ۱۰: Zero Shot Learning with LLama-3-8B

NLP\_CA4\_Q2\_a\_Version(2) Draft saved

File Edit View Run Add-ons Help

+ ✂ 📄 📋 ▶ ▶▶ Run All Code ▾ ● Draft Session off (run a cell to start) ⏻

```
def predict_relation(premise, hypothesis):
    prompt = format_prompt(premise, hypothesis)
    inputs = tokenizer(prompt, return_tensors="pt", max_length=512 ,
                        truncation=True, padding="max_length")
    tokenizer.pad_token = tokenizer.eos_token
    tokenizer.add_eos_token = True

    with torch.no_grad():
        outputs = model(**inputs)
    logits = outputs.logits
    # logits shape : torch.Size([1, 512, 128256])

    relation_logits = logits[0, -1, :] # Shape: [128256]

    # Get the prediction for the relation
    prediction = torch.argmax(relation_logits).item()

    if random.randint(1, 50) < 20:
        print(f'prediction : {prediction}')
        print(tokenizer.decode(prediction))
        print(f'premise : {premise}')
        print(f'hypothesis : {hypothesis}')
        print('-----')

    # 0 entailment
    # 1 neural
    # 2 contradict
    results = {284 : 0 , 25:1 , 355:2 , 278 : 1}
```

شکل ۱۱: Zero Shot Learning with LLama-3-8B(Predict relationship)

NLP\_CA4\_Q2\_a\_Version(2) Draft saved

File Edit View Run Add-ons Help

+ | ✂ | 📄 | 📋 | ▶ ▶▶ Run All | Code ▾

● Draft Sess

## Zero Shot Learning

```
[ ]: from sklearn.metrics import classification_report

premises = dataset['train']['premise'][:200]
hypotheses = dataset['train']['hypothesis'][:200]
labels = dataset['train']['label'][:200]

predictions = []

for item in range(200):
    premise = premises[item]
    hypothesis = hypotheses[item]

    model.eval()
    with torch.no_grad():
        predictions.append(predict_relation(premise, hypothesis))

print(classification_report(labels , predictions))
```

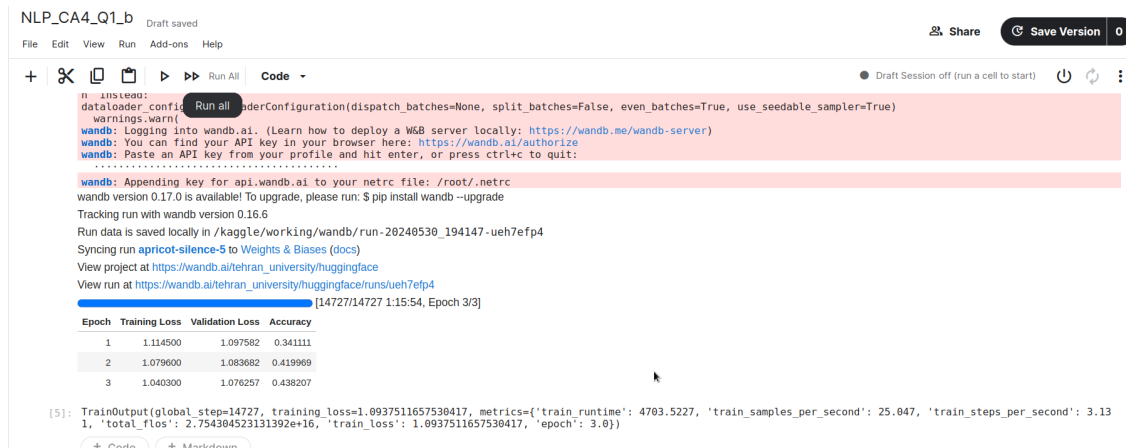
+ Code

+ Markdown

شکل ۱۲: Zero Shot Learning with LLama-3-8B(Report)

## In Context Learning (One Shot) ۳.۲

جدول زیر نتایج حاصل از آموزش مدل توسط چند نمونه پرامپت و سپس محاسبه میزان دقت مدل ما می باشد.  
این روش کمی دقت بهتری از حالت دارد که در تصویر نیز مشخص است.



```
from __future__ import annotations
import logging
import os
import sys
import time
import warnings
from dataclasses import dataclass
from typing import Any, Dict, List, Optional, Union

from tqdm import tqdm
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM, Seq2SeqTrainer, Seq2SeqTrainingArguments

warnings.warn(
    "The `Seq2SeqTrainer` class is deprecated. Please use the `Seq2SeqTrainer` class instead.",
    DeprecationWarning,
)

def dataloader_config(dispatch_batches=None, split_batches=False, even_batches=True, use_seedable_sampler=True):
    """
    Configuration for the data loader.
    """
    return Seq2SeqTrainerDataLoaderConfig(
        dispatch_batches=dispatch_batches,
        split_batches=split_batches,
        even_batches=even_batches,
        use_seedable_sampler=use_seedable_sampler,
    )

wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter, or press ctrl+c to quit:
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb version 0.17.0 is available! To upgrade, please run: $ pip install wandb --upgrade
Tracking run with wandb version 0.16.6
Run data is saved locally in /kaggle/working/wandb/run-20240530_194147-ueh7efp4
Syncing run apricot-silence-5 to Weights & Biases (docs)
View project at https://wandb.ai/tehran_university/huggingface
View run at https://wandb.ai/tehran_university/huggingface/runs/ueh7efp4

[14727/14727 1:15:54, Epoch 3/3]

Epoch   Training Loss   Validation Loss   Accuracy
-----
1         1.114500         1.097582         0.341111
2         1.079600         1.083682         0.419969
3         1.040300         1.076257         0.438207

[5]: TrainOutput(global_step=14727, training_loss=1.0937511657538417, metrics={'train_runtime': 4703.5227, 'train_samples_per_second': 25.047, 'train_steps_per_second': 3.13
1, 'total_flos': 2.754384523131392e+16, 'train_loss': 1.0937511657538417, 'epoch': 3.0})
```

شکل ۱۳: One Shot Learning with LLaMA-3-8B

## QLora ۴.۲

روش QLora که ترکیب مدل کوانتیزه شده و Lora خواهد بود. به نظرمی‌رسد که با استفاده از این رویکرد می‌توانیم به دقت و سرعت بالاتری در آموزش برسیم.

```
NLP_CA4_Q2_a_Version(2) Draft saved
File Edit View Run Add-ons Help

+ | ✂ | 📄 | 📌 | ▶ | ⏮ | ⏭ | Run All | Code ▾

bnb_4bit_compu Run all torch.float16,
bnb_4bit_use_double_quant= False
# load_in_8bit_fp32_cpu_offload=True
)

device_map = {
    'transformer.h.0': 'cuda:1',
    'transformer.h.1': 'cuda:2',
}

model_name = 'meta-llama/Meta-Llama-3-8B'
tokenizer = AutoTokenizer.from_pretrained(model_name)

model = AutoModelForCausalLM.from_pretrained(
    model_name,
    quantization_config= bnb_config,
    device_map = 'auto'
)

peft_config = LoraConfig(
    lora_alpha=8,
    lora_dropout=0.1,
    r=16,
    bias="none",
    task_type="CAUSAL_LM",
)

model = get_peft_model(model, peft_config)
```



شکل ۱۴: QLora Config