LightweightFineTuning

August 27, 2025

1 Lightweight Fine-Tuning Project

TODO: In this cell, describe your choices for each of the following

- PEFT technique: LoRA
- Model: roBERTa
- Evaluation approach: epoch
- Fine-tuning dataset: imdb

1.1 Loading and Evaluating a Foundation Model

TODO: In the cells below, load your chosen pre-trained Hugging Face model and evaluate its performance prior to fine-tuning. This step includes loading an appropriate tokenizer and dataset.

[4]:
```python
from datasets import load_dataset
import pandas as pd
import numpy as np

splits=["train","test"]
dataset = load_dataset("imdb", split=splits)
dataset = [dataset.rename_column('label', 'labels') for dataset in dataset]
# Crează un dicționar cu seturile de date
ds = {split: dataset.shuffle(seed=42).select(range(500)) for split, dataset in␣
 ↪zip(splits, dataset)}

#display(ds)
display(dataset)
print("\n")
display(ds)
```

```
[Dataset({
     features: ['text', 'labels'],
     num_rows: 25000
 }),
 Dataset({
     features: ['text', 'labels'],
     num_rows: 25000
 })]
```

```
{'train': Dataset({
     features: ['text', 'labels'],
     num_rows: 500
 }),
 'test': Dataset({
     features: ['text', 'labels'],
     num_rows: 500
 })}
```

[5]:
```
# here i just vizualize the data to understand it

print("tipul lui ds:", type(ds))
#print(dataset.shape)
print("tipul lui dataset:", type(dataset))
print(dataset,"\n")
display(dataset[0][0]) #for i in range(0,2)]
display(ds["train"][0])
```

```
tipul lui ds: <class 'dict'>
tipul lui dataset: <class 'list'>
[Dataset({
    features: ['text', 'labels'],
    num_rows: 25000
}), Dataset({
    features: ['text', 'labels'],
    num_rows: 25000
})]
```

```
{'text': 'I rented I AM CURIOUS-YELLOW from my video store because of all the␣
↪controversy that surrounded it when it was first released in 1967. I also␣
↪heard that at first it was seized by U.S. customs if it ever tried to enter␣
↪this country, therefore being a fan of films considered "controversial" I␣
↪really had to see this for myself.<br /><br />The plot is centered around a␣
↪young Swedish drama student named Lena who wants to learn everything she can␣
↪about life. In particular she wants to focus her attentions to making some␣
↪sort of documentary on what the average Swede thought about certain political␣
↪issues such as the Vietnam War and race issues in the United States. In␣
↪between asking politicians and ordinary denizens of Stockholm about their␣
↪opinions on politics, she has sex with her drama teacher, classmates, and␣
↪married men.<br /><br />What kills me about I AM CURIOUS-YELLOW is that 40␣
↪years ago, this was considered pornographic. Really, the sex and nudity scenes␣
↪are few and far between, even then it\'s not shot like some cheaply made porno.
↪ While my countrymen mind find it shocking, in reality sex and nudity are a␣
↪major staple in Swedish cinema. Even Ingmar Bergman, arguably their answer to␣
↪good old boy John Ford, had sex scenes in his films.<br /><br />I do commend␣
↪the filmmakers for the fact that any sex shown in the film is shown for␣
↪artistic purposes rather than just to shock people and make money to be shown␣
↪in pornographic theaters in America. I AM CURIOUS-YELLOW is a good film for␣
↪anyone wanting to study the meat and potatoes (no pun intended) of Swedish␣
↪cinema. But really, this film doesn\'t have much of a plot.',
 'labels': 0}

{'text': 'There is no relation at all between Fortier and Profiler but the fact␣
↪that both are police series about violent crimes. Profiler looks crispy,␣
↪Fortier looks classic. Profiler plots are quite simple. Fortier\'s plot are␣
↪far more complicated… Fortier looks more like Prime Suspect, if we have to␣
↪spot similarities… The main character is weak and weirdo, but have␣
↪"clairvoyance". People like to compare, to judge, to evaluate. How about just␣
↪enjoying? Funny thing too, people writing Fortier looks American but, on the␣
↪other hand, arguing they prefer American series (!!!). Maybe it\'s the␣
↪language, or the spirit, but I think this series is more English than American.
↪ By the way, the actors are really good and funny. The acting is not␣
↪superficial at all…',
 'labels': 1}
```

[6]:
```python
#from transformers import AutoTokenizer
from transformers import RobertaTokenizer, RobertaForSequenceClassification,␣
↪Trainer, TrainingArguments
from transformers import BitsAndBytesConfig
# Load the tokenizer
tokenizer = RobertaTokenizer.from_pretrained('roberta-base')

# Tokenize the dataset
def tokenize_function(examples):
    return tokenizer(examples['text'], truncation=True, padding='max_length')
```

```python
tokenized_ds={}
for split in splits:
    tokenized_ds[split] = ds[split].map(tokenize_function, batched=True)


# Show the first example of the tokenized training set
print(tokenized_ds)
print(tokenized_ds['train'][0])
```

```
{'train': Dataset({
    features: ['text', 'labels', 'input_ids', 'attention_mask'],
    num_rows: 500
}), 'test': Dataset({
    features: ['text', 'labels', 'input_ids', 'attention_mask'],
    num_rows: 500
})}
{'text': 'There is no relation at all between Fortier and Profiler but the fact
that both are police series about violent crimes. Profiler looks crispy, Fortier
looks classic. Profiler plots are quite simple. Fortier\'s plot are far more
complicated… Fortier looks more like Prime Suspect, if we have to spot
similarities… The main character is weak and weirdo, but have "clairvoyance".
People like to compare, to judge, to evaluate. How about just enjoying? Funny
thing too, people writing Fortier looks American but, on the other hand, arguing
they prefer American series (!!!). Maybe it\'s the language, or the spirit, but
I think this series is more English than American. By the way, the actors are
really good and funny. The acting is not superficial at all…', 'labels': 1,
'input_ids': [0, 970, 16, 117, 9355, 23, 70, 227, 3339, 906, 8, 6853, 10329, 53,
5, 754, 14, 258, 32, 249, 651, 59, 4153, 3474, 4, 6853, 10329, 1326, 32042, 6,
3339, 906, 1326, 4187, 4, 6853, 10329, 21258, 32, 1341, 2007, 4, 3339, 906, 18,
6197, 32, 444, 55, 6336, 734, 3339, 906, 1326, 55, 101, 1489, 10471, 13771, 6,
114, 52, 33, 7, 1514, 20097, 734, 20, 1049, 2048, 16, 3953, 8, 7735, 139, 6, 53,
33, 22, 31238, 35246, 2389, 845, 1806, 101, 7, 8933, 6, 7, 1679, 6, 7, 10516, 4,
1336, 59, 95, 6218, 116, 31906, 631, 350, 6, 82, 2410, 3339, 906, 1326, 470, 53,
6, 15, 5, 97, 865, 6, 7594, 51, 6573, 470, 651, 36, 16506, 322, 5359, 24, 18, 5,
2777, 6, 50, 5, 4780, 6, 53, 38, 206, 42, 651, 16, 55, 2370, 87, 470, 4, 870, 5,
169, 6, 5, 5552, 32, 269, 205, 8, 6269, 4, 20, 3501, 16, 45, 34501, 23, 70, 734,
2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0]}
```

[7]:
```python
#from transformers import AutoModelForCausalLM6
import torch

model = RobertaForSequenceClassification.from_pretrained(
    "roberta-base",
    num_labels=2
)

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


training_args = TrainingArguments(
    output_dir='./results',
    learning_rate=1e-4,
    num_train_epochs=5,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    gradient_accumulation_steps=8,
    fp16=True,
    logging_dir='./logs',
    logging_steps=10,
    eval_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
)


# Inițializează trainer-ul
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_ds['train'],
    eval_dataset=tokenized_ds['test'],
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)


# freeze all paramters
for param in model.base_model.parameters():
   param.requires_grad = True

display(model)
```

```
device =  cuda
```

```
Some weights of RobertaForSequenceClassification were not initialized from the
model checkpoint at roberta-base and are newly initialized:
['classifier.dense.bias', 'classifier.dense.weight', 'classifier.out_proj.bias',
'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
C:\ProgramData\Anaconda3\Lib\site-packages\transformers\training_args.py:1545:
FutureWarning: `evaluation_strategy` is deprecated and will be removed in
version 4.46 of  Transformers. Use `eval_strategy` instead
  warnings.warn(
```

```
RobertaForSequenceClassification(
  (roberta): RobertaModel(
    (embeddings): RobertaEmbeddings(
      (word_embeddings): Embedding(50265, 768, padding_idx=1)
      (position_embeddings): Embedding(514, 768, padding_idx=1)
      (token_type_embeddings): Embedding(1, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): RobertaEncoder(
      (layer): ModuleList(
        (0-11): 12 x RobertaLayer(
          (attention): RobertaAttention(
            (self): RobertaSdpaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): RobertaSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): RobertaIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): RobertaOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
  )
  (classifier): RobertaClassificationHead(
    (dense): Linear(in_features=768, out_features=768, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (out_proj): Linear(in_features=768, out_features=2, bias=True)
  )
)
```

[8]:
```python
print("Trainer va folosi device:", training_args.device)
print("Modelul este pe device:", next(model.parameters()).device)

trainer.train()
results = trainer.evaluate()
print(results)
```

```
Trainer va folosi device: cuda:0
Modelul este pe device: cuda:0
```

```
C:\ProgramData\Anaconda3\Lib\site-
packages\transformers\models\roberta\modeling_roberta.py:370: UserWarning:
1Torch was not compiled with flash attention. (Triggered internally at
C:\actions-runner\_work\pytorch\pytorch\builder\windows\pytorch\aten\src\ATen\na
tive\transformers\cuda\sdp_utils.cpp:555.)
  attn_output = torch.nn.functional.scaled_dot_product_attention(
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

{'eval_loss': 0.27025726437568665, 'eval_accuracy': 0.898, 'eval_runtime':
5.7027, 'eval_samples_per_second': 87.677, 'eval_steps_per_second': 21.919,
'epoch': 4.8}
```

[ ]:

```

```

1.2 Performing Parameter-Efficient Fine-Tuning

TODO: In the cells below, create a PEFT model from your loaded model, run a training loop, and save the PEFT model weights.

[12]:

```
from peft import get_peft_model, LoraConfig, TaskType

peft_config = LoraConfig(
    task_type = TaskType.SEQ_CLS,
    inference_mode = False,
    r = 8,
    lora_alpha=16,
    lora_dropout=0.1
)
```

[14]:

```
lora_model = get_peft_model(model, peft_config)
lora_model.print_trainable_parameters()
```

```
trainable params: 887,042 || all params: 125,534,212 || trainable%: 0.7066
```

[15]:

```
# Inițializează trainer-ul
lora_trainer = Trainer(
    model=lora_model,
    args=training_args,
    train_dataset=tokenized_ds['train'],
    eval_dataset=tokenized_ds['test'],
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)

lora_trainer.train()
```

```
<IPython.core.display.HTML object>
```

[15]:

```
TrainOutput(global_step=75, training_loss=0.11738499402999877,
metrics={'train_runtime': 118.7226, 'train_samples_per_second': 21.057,
'train_steps_per_second': 0.632, 'total_flos': 638006516121600.0, 'train_loss':
0.11738499402999877, 'epoch': 4.8})
```

[17]:
```
lora_results = lora_trainer.evaluate()
print(lora_results)
```

```
<IPython.core.display.HTML object>

{'eval_loss': 0.27178093791007996, 'eval_accuracy': 0.904, 'eval_runtime':
6.4978, 'eval_samples_per_second': 76.949, 'eval_steps_per_second': 19.237,
'epoch': 4.8}
```

1.2.1 IMPORTANT

Due to workspace storage constraints, you should not store the model weights in the same directory but rather use `/tmp` to avoid workspace crashes which are irrecoverable. Ensure you save it in /tmp always.

[19]:
```
# Saving the model
#model.save("/tmp/your_model_name")
lora_model.save_pretrained("./lora_model")
```

1.3 Performing Inference with a PEFT Model

TODO: In the cells below, load the saved PEFT model weights and evaluate the performance of the trained PEFT model. Be sure to compare the results to the results from prior to fine-tuning.

[24]:
```
from peft import AutoPeftModelForSequenceClassification

# Încarcă modelul LoRA salvat
lora_model = AutoPeftModelForSequenceClassification.from_pretrained("./
 ↪lora_model")
```

```
Some weights of RobertaForSequenceClassification were not initialized from the
model checkpoint at roberta-base and are newly initialized:
['classifier.dense.bias', 'classifier.dense.weight', 'classifier.out_proj.bias',
'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

[26]:
```
print("Base Model Accuracy:", results["eval_accuracy"])
print("LoRA Model Accuracy:", lora_results["eval_accuracy"])
```

```
Base Model Accuracy: 0.898
LoRA Model Accuracy: 0.904
```

[ ]:

[ ]:

[ ]:

[ ]: