```
! pip install transformers datasets peft
```

```
Requirement already satisfied: transformers in
/usr/local/lib/python3.10/dist-packages (4.47.0)
Requirement already satisfied: datasets in
/usr/local/lib/python3.10/dist-packages (3.3.1)
Requirement already satisfied: peft in /usr/local/lib/python3.10/dist-
packages (0.14.0)
Requirement already satisfied: filelock in
/usr/local/lib/python3.10/dist-packages (from transformers) (3.17.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.24.0 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.29.0)
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.10/dist-packages (from transformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in
/usr/local/lib/python3.10/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.10/dist-packages (from transformers)
(2024.11.6)
Requirement already satisfied: requests in
/usr/local/lib/python3.10/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.21.0)
Requirement already satisfied: safetensors>=0.4.1 in
/usr/local/lib/python3.10/dist-packages (from transformers) (0.4.5)
Requirement already satisfied: tqdm>=4.27 in
/usr/local/lib/python3.10/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: pyarrow>=15.0.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (19.0.1)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in
/usr/local/lib/python3.10/dist-packages (from datasets) (0.3.8)
Requirement already satisfied: pandas in
/usr/local/lib/python3.10/dist-packages (from datasets) (2.2.3)
Requirement already satisfied: xxhash in
/usr/local/lib/python3.10/dist-packages (from datasets) (3.5.0)
Requirement already satisfied: multiprocess<0.70.17 in
/usr/local/lib/python3.10/dist-packages (from datasets) (0.70.16)
Requirement already satisfied: fsspec<=2024.12.0,>=2023.1.0 in
/usr/local/lib/python3.10/dist-packages (from
fsspec[http]<=2024.12.0,>=2023.1.0->datasets) (2024.12.0)
Requirement already satisfied: aiohttp in
/usr/local/lib/python3.10/dist-packages (from datasets) (3.11.12)
Requirement already satisfied: psutil in
/usr/local/lib/python3.10/dist-packages (from peft) (5.9.5)
Requirement already satisfied: torch>=1.13.0 in
/usr/local/lib/python3.10/dist-packages (from peft) (2.5.1+cu121)
Requirement already satisfied: accelerate>=0.21.0 in
/usr/local/lib/python3.10/dist-packages (from peft) (1.2.1)
```

```
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(2.4.6)
Requirement already satisfied: aiosignal>=1.1.2 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(1.3.2)
Requirement already satisfied: async-timeout<6.0,>=4.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(5.0.1)
Requirement already satisfied: attrs>=17.3.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(25.1.0)
Requirement already satisfied: frozenlist>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(6.1.0)
Requirement already satisfied: propcache>=0.2.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(0.2.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in
/usr/local/lib/python3.10/dist-packages (from aiohttp->datasets)
(1.18.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.10/dist-packages (from huggingface-
hub<1.0,>=0.24.0->transformers) (4.12.2)
Requirement already satisfied: mkl_fft in
/usr/local/lib/python3.10/dist-packages (from numpy>=1.17-
>transformers) (1.3.8)
Requirement already satisfied: mkl_random in
/usr/local/lib/python3.10/dist-packages (from numpy>=1.17-
>transformers) (1.2.4)
Requirement already satisfied: mkl_umath in
/usr/local/lib/python3.10/dist-packages (from numpy>=1.17-
>transformers) (0.1.1)
Requirement already satisfied: mkl in /usr/local/lib/python3.10/dist-
packages (from numpy>=1.17->transformers) (2025.0.1)
Requirement already satisfied: tbb4py in
/usr/local/lib/python3.10/dist-packages (from numpy>=1.17-
>transformers) (2022.0.0)
Requirement already satisfied: mkl-service in
/usr/local/lib/python3.10/dist-packages (from numpy>=1.17-
>transformers) (2.4.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(3.4.1)
Requirement already satisfied: idna<4,>=2.5 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
```

```
(3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.10/dist-packages (from requests->transformers)
(2025.1.31)
Requirement already satisfied: networkx in
/usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->peft)
(3.4.2)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->peft)
(3.1.4)
Requirement already satisfied: sympy==1.13.1 in
/usr/local/lib/python3.10/dist-packages (from torch>=1.13.0->peft)
(1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in
/usr/local/lib/python3.10/dist-packages (from sympy==1.13.1-
>torch>=1.13.0->peft) (1.3.0)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas->datasets)
(2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas->datasets)
(2025.1)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.10/dist-packages (from pandas->datasets)
(2025.1)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2-
>pandas->datasets) (1.17.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.13.0-
>peft) (3.0.2)
Requirement already satisfied: intel-openmp>=2024 in
/usr/local/lib/python3.10/dist-packages (from mkl->numpy>=1.17-
>transformers) (2024.2.0)
Requirement already satisfied: tbb==2022.* in
/usr/local/lib/python3.10/dist-packages (from mkl->numpy>=1.17-
>transformers) (2022.0.0)
Requirement already satisfied: tcmlib==1.* in
/usr/local/lib/python3.10/dist-packages (from tbb==2022.*->mkl-
>numpy>=1.17->transformers) (1.2.0)
Requirement already satisfied: intel-cmplr-lib-rt in
/usr/local/lib/python3.10/dist-packages (from mkl_umath->numpy>=1.17-
>transformers) (2024.2.0)
Requirement already satisfied: intel-cmplr-lib-ur==2024.2.0 in
/usr/local/lib/python3.10/dist-packages (from intel-openmp>=2024->mkl-
>numpy>=1.17->transformers) (2024.2.0)
```

HW 8: Low Rank Adaptation (LoRA)

In this assignment, you will learn to implement low-rank adaptation both from scratch and using a library—specifically, with PyTorch and the PEFT library, respectively.

This assignment is divided into two sections:

In the first section, we introduce the parameter-efficient transfer learning (PET) method. We use LoRA to adapt the GPT2 model for the SST-2 dataset. This section will teach you how LoRA works and how to implement it from scratch using forward_hook.

In the second section, we introduce the PEFT library, which allows us to perform LoRA easily.

Part 1: LoRA from Scratch

With the discovery of scaling properties in deep learning models, several researchers tend to increase model size to achieve emergent properties, especially in the natural language processing (NLP) field. For example, GPT-3 contains 175 billion parameters, making it nearly impossible to fine-tune on limited resources. This trend prevents students like us from adapting these enormous foundation models on a single GPU (or with small resources).

To alleviate this problem, researchers have developed new fine-tuning methods, known as parameter-efficient transfer learning, which allow us to train large models with limited resources. The benefits of these methods extend not only to the training process but also to deployment. After fine-tuning, we only need to save a small number of parameters (the LoRA weights), enabling us to deploy the foundation model to various downstream tasks using minimal storage. One of the prevailing methods is Low Rank Adaptation (LoRA).

Another popular option is prompt tuning, where we only train special tokens that are prepended to the input. However, this is not the focus of this homework.

In this section, we will introduce Low-Rank Adaptation. You are assigned to implement LoRA on GPT2 model. We will finetune the model to SST-2 dataset using the traditional and LoRA method.

Load Dataset and Model

In this step, we will prepare the GPT-2 model and the SST-2 dataset.

SST-2 is a widely used dataset for sentiment analysis, extracted from movie reviews, containing sentences labeled as either positive or negative.

```
import torch
import numpy as np
import time
from torch.utils.data import DataLoader
from transformers import GPT2ForSequenceClassification,
GPT2TokenizerFast
```

```python
from datasets import load_dataset
from tqdm.autonotebook import tqdm
from transformers import GPT2Tokenizer, GPT2ForSequenceClassification,
AdamW

# Load the GPT-2 model for sequence classification and its tokenizer
model = GPT2ForSequenceClassification.from_pretrained("gpt2",
num_labels=2)
tokenizer = GPT2TokenizerFast.from_pretrained("gpt2")

# GPT-2 does not have a pad token by default so we set it to the EOS
token.
tokenizer.pad_token = tokenizer.eos_token
model.config.pad_token_id = tokenizer.eos_token_id

# Load the SST-2 dataset
train_dataset_raw = load_dataset("glue", "sst2", split="train")
train_dataset_raw, val_dataset_raw =
train_dataset_raw.train_test_split(test_size=0.2).values()
test_dataset_raw = load_dataset("glue", "sst2", split="validation")

# Preview dataset
print("Sample sentence:")
for data in test_dataset_raw:
    print(data)
    break

def tokenize_function(example):
    return tokenizer(example["sentence"], padding="max_length",
truncation=True, max_length=128)

train_dataset = train_dataset_raw.map(tokenize_function, batched=True)
val_dataset = val_dataset_raw.map(tokenize_function, batched=True)
test_dataset = test_dataset_raw.map(tokenize_function, batched=True)

train_dataset.set_format("torch", columns=["input_ids",
"attention_mask", "label"])
val_dataset.set_format("torch", columns=["input_ids",
"attention_mask", "label"])
test_dataset.set_format("torch", columns=["input_ids",
"attention_mask", "label"])

# Create data loaders
train_dataloader = DataLoader(train_dataset, shuffle=True,
batch_size=16)
val_dataloader = DataLoader(val_dataset, batch_size=16)
test_dataloader = DataLoader(test_dataset, batch_size=16)
```

{"model_id":"e2c4dc20132049babd20735b19b3b3a1","version_major":2,"vers
ion_minor":0}

{"model_id":"0f6e3d0314da433bbf359718094a5f20","version_major":2,"vers
ion_minor":0}

```
Some weights of GPT2ForSequenceClassification were not initialized
from the model checkpoint at gpt2 and are newly initialized:
['score.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
```

{"model_id":"bf9f61f0e43146d4a6cfb4ea837086cf","version_major":2,"vers
ion_minor":0}

{"model_id":"deeada16406f4e1d97cb037c039dc92b","version_major":2,"vers
ion_minor":0}

{"model_id":"1f9b80b2aa434b5b8c29e9618970afb5","version_major":2,"vers
ion_minor":0}

{"model_id":"e3a4bd1f07b34ab680c84363bc9414e7","version_major":2,"vers
ion_minor":0}

{"model_id":"703af2b94d084fafa2a73a64c10bfbce","version_major":2,"vers
ion_minor":0}

{"model_id":"4bdbe201ee1443fc970a2ef97abb6ec5","version_major":2,"vers
ion_minor":0}

{"model_id":"056553c3d1424a52bacff5f1e4af8c5f","version_major":2,"vers
ion_minor":0}

{"model_id":"b74ed38db1e245c08e158423281c52bd","version_major":2,"vers
ion_minor":0}

{"model_id":"a6cc31af5b224fc6abcac02656b5a191","version_major":2,"vers
ion_minor":0}

{"model_id":"5382088a278542cbbe722b837fb05596","version_major":2,"vers
ion_minor":0}

{"model_id":"6ed8505804324aa3bc1dc90eaa41209b","version_major":2,"vers
ion_minor":0}

```
Sample sentence:
{'sentence': "it 's a charming and often affecting journey . ",
'label': 1, 'idx': 0}
```

{"model_id":"43c61916f1b04a1f9808e7bd1694022a","version_major":2,"vers
ion_minor":0}

{"model_id":"721b14149c044c37b8e06b430cfd3c38","version_major":2,"vers
ion_minor":0}

```
{"model_id":"93246bfd6e2f4fa19d0b4b9cad720432","version_major":2,"version_minor":0}
```


Traditional Fine tuning

In the traditional fine-tuning method, the entire model is trained, which is computationally expensive. An alternative approach is to fine-tune only certain layers of the model to reduce resource usage while still adapting the model to a specific task.

To keep the implementation simple, you are assigned to train only the attention weights in the self-attention layers.

The code below displays the names of all layers in the GPT-2 model. This will help you identify which layers to set as trainable or keep frozen. For more details on the attention layers in GPT-2, please refer to the following link: GPT-2 Attention Layer Details.

```
for name, module in model.named_modules():
  print(name, type(module))
```

```
 <class
'transformers.models.gpt2.modeling_gpt2.GPT2ForSequenceClassification'
>
transformer <class 'transformers.models.gpt2.modeling_gpt2.GPT2Model'>
transformer.wte <class 'torch.nn.modules.sparse.Embedding'>
transformer.wpe <class 'torch.nn.modules.sparse.Embedding'>
transformer.drop <class 'torch.nn.modules.dropout.Dropout'>
transformer.h <class 'torch.nn.modules.container.ModuleList'>
transformer.h.0 <class
'transformers.models.gpt2.modeling_gpt2.GPT2Block'>
transformer.h.0.ln_1 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.0.attn <class
'transformers.models.gpt2.modeling_gpt2.GPT2SdpaAttention'>
transformer.h.0.attn.c_attn <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.0.attn.c_proj <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.0.attn.attn_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.0.attn.resid_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.0.ln_2 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.0.mlp <class
'transformers.models.gpt2.modeling_gpt2.GPT2MLP'>
transformer.h.0.mlp.c_fc <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.0.mlp.c_proj <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.0.mlp.act <class
'transformers.activations.NewGELUActivation'>
```

```
transformer.h.0.mlp.dropout <class 'torch.nn.modules.dropout.Dropout'>
transformer.h.1 <class
'transformers.models.gpt2.modeling_gpt2.GPT2Block'>
transformer.h.1.ln_1 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.1.attn <class
'transformers.models.gpt2.modeling_gpt2.GPT2SdpaAttention'>
transformer.h.1.attn.c_attn <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.1.attn.c_proj <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.1.attn.attn_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.1.attn.resid_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.1.ln_2 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.1.mlp <class
'transformers.models.gpt2.modeling_gpt2.GPT2MLP'>
transformer.h.1.mlp.c_fc <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.1.mlp.c_proj <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.1.mlp.act <class
'transformers.activations.NewGELUActivation'>
transformer.h.1.mlp.dropout <class 'torch.nn.modules.dropout.Dropout'>
transformer.h.2 <class
'transformers.models.gpt2.modeling_gpt2.GPT2Block'>
transformer.h.2.ln_1 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.2.attn <class
'transformers.models.gpt2.modeling_gpt2.GPT2SdpaAttention'>
transformer.h.2.attn.c_attn <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.2.attn.c_proj <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.2.attn.attn_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.2.attn.resid_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.2.ln_2 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.2.mlp <class
'transformers.models.gpt2.modeling_gpt2.GPT2MLP'>
transformer.h.2.mlp.c_fc <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.2.mlp.c_proj <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.2.mlp.act <class
'transformers.activations.NewGELUActivation'>
transformer.h.2.mlp.dropout <class 'torch.nn.modules.dropout.Dropout'>
transformer.h.3 <class
'transformers.models.gpt2.modeling_gpt2.GPT2Block'>
```

```
transformer.h.3.ln_1 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.3.attn <class
'transformers.models.gpt2.modeling_gpt2.GPT2SdpaAttention'>
transformer.h.3.attn.c_attn <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.3.attn.c_proj <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.3.attn.attn_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.3.attn.resid_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.3.ln_2 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.3.mlp <class
'transformers.models.gpt2.modeling_gpt2.GPT2MLP'>
transformer.h.3.mlp.c_fc <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.3.mlp.c_proj <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.3.mlp.act <class
'transformers.activations.NewGELUActivation'>
transformer.h.3.mlp.dropout <class 'torch.nn.modules.dropout.Dropout'>
transformer.h.4 <class
'transformers.models.gpt2.modeling_gpt2.GPT2Block'>
transformer.h.4.ln_1 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.4.attn <class
'transformers.models.gpt2.modeling_gpt2.GPT2SdpaAttention'>
transformer.h.4.attn.c_attn <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.4.attn.c_proj <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.4.attn.attn_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.4.attn.resid_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.4.ln_2 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.4.mlp <class
'transformers.models.gpt2.modeling_gpt2.GPT2MLP'>
transformer.h.4.mlp.c_fc <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.4.mlp.c_proj <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.4.mlp.act <class
'transformers.activations.NewGELUActivation'>
transformer.h.4.mlp.dropout <class 'torch.nn.modules.dropout.Dropout'>
transformer.h.5 <class
'transformers.models.gpt2.modeling_gpt2.GPT2Block'>
transformer.h.5.ln_1 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.5.attn <class
```

```
'transformers.models.gpt2.modeling_gpt2.GPT2SdpaAttention'>
transformer.h.5.attn.c_attn <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.5.attn.c_proj <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.5.attn.attn_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.5.attn.resid_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.5.ln_2 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.5.mlp <class
'transformers.models.gpt2.modeling_gpt2.GPT2MLP'>
transformer.h.5.mlp.c_fc <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.5.mlp.c_proj <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.5.mlp.act <class
'transformers.activations.NewGELUActivation'>
transformer.h.5.mlp.dropout <class 'torch.nn.modules.dropout.Dropout'>
transformer.h.6 <class
'transformers.models.gpt2.modeling_gpt2.GPT2Block'>
transformer.h.6.ln_1 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.6.attn <class
'transformers.models.gpt2.modeling_gpt2.GPT2SdpaAttention'>
transformer.h.6.attn.c_attn <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.6.attn.c_proj <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.6.attn.attn_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.6.attn.resid_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.6.ln_2 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.6.mlp <class
'transformers.models.gpt2.modeling_gpt2.GPT2MLP'>
transformer.h.6.mlp.c_fc <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.6.mlp.c_proj <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.6.mlp.act <class
'transformers.activations.NewGELUActivation'>
transformer.h.6.mlp.dropout <class 'torch.nn.modules.dropout.Dropout'>
transformer.h.7 <class
'transformers.models.gpt2.modeling_gpt2.GPT2Block'>
transformer.h.7.ln_1 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.7.attn <class
'transformers.models.gpt2.modeling_gpt2.GPT2SdpaAttention'>
transformer.h.7.attn.c_attn <class
'transformers.pytorch_utils.Conv1D'>
```

```
transformer.h.7.attn.c_proj <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.7.attn.attn_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.7.attn.resid_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.7.ln_2 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.7.mlp <class
'transformers.models.gpt2.modeling_gpt2.GPT2MLP'>
transformer.h.7.mlp.c_fc <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.7.mlp.c_proj <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.7.mlp.act <class
'transformers.activations.NewGELUActivation'>
transformer.h.7.mlp.dropout <class 'torch.nn.modules.dropout.Dropout'>
transformer.h.8 <class
'transformers.models.gpt2.modeling_gpt2.GPT2Block'>
transformer.h.8.ln_1 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.8.attn <class
'transformers.models.gpt2.modeling_gpt2.GPT2SdpaAttention'>
transformer.h.8.attn.c_attn <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.8.attn.c_proj <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.8.attn.attn_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.8.attn.resid_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.8.ln_2 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.8.mlp <class
'transformers.models.gpt2.modeling_gpt2.GPT2MLP'>
transformer.h.8.mlp.c_fc <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.8.mlp.c_proj <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.8.mlp.act <class
'transformers.activations.NewGELUActivation'>
transformer.h.8.mlp.dropout <class 'torch.nn.modules.dropout.Dropout'>
transformer.h.9 <class
'transformers.models.gpt2.modeling_gpt2.GPT2Block'>
transformer.h.9.ln_1 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.9.attn <class
'transformers.models.gpt2.modeling_gpt2.GPT2SdpaAttention'>
transformer.h.9.attn.c_attn <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.9.attn.c_proj <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.9.attn.attn_dropout <class
```

```
'torch.nn.modules.dropout.Dropout'>
transformer.h.9.attn.resid_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.9.ln_2 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.9.mlp <class
'transformers.models.gpt2.modeling_gpt2.GPT2MLP'>
transformer.h.9.mlp.c_fc <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.9.mlp.c_proj <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.9.mlp.act <class
'transformers.activations.NewGELUActivation'>
transformer.h.9.mlp.dropout <class 'torch.nn.modules.dropout.Dropout'>
transformer.h.10 <class
'transformers.models.gpt2.modeling_gpt2.GPT2Block'>
transformer.h.10.ln_1 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.10.attn <class
'transformers.models.gpt2.modeling_gpt2.GPT2SdpaAttention'>
transformer.h.10.attn.c_attn <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.10.attn.c_proj <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.10.attn.attn_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.10.attn.resid_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.10.ln_2 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.10.mlp <class
'transformers.models.gpt2.modeling_gpt2.GPT2MLP'>
transformer.h.10.mlp.c_fc <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.10.mlp.c_proj <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.10.mlp.act <class
'transformers.activations.NewGELUActivation'>
transformer.h.10.mlp.dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.11 <class
'transformers.models.gpt2.modeling_gpt2.GPT2Block'>
transformer.h.11.ln_1 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.11.attn <class
'transformers.models.gpt2.modeling_gpt2.GPT2SdpaAttention'>
transformer.h.11.attn.c_attn <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.11.attn.c_proj <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.11.attn.attn_dropout <class
'torch.nn.modules.dropout.Dropout'>
```

```
transformer.h.11.attn.resid_dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.h.11.ln_2 <class
'torch.nn.modules.normalization.LayerNorm'>
transformer.h.11.mlp <class
'transformers.models.gpt2.modeling_gpt2.GPT2MLP'>
transformer.h.11.mlp.c_fc <class 'transformers.pytorch_utils.Conv1D'>
transformer.h.11.mlp.c_proj <class
'transformers.pytorch_utils.Conv1D'>
transformer.h.11.mlp.act <class
'transformers.activations.NewGELUActivation'>
transformer.h.11.mlp.dropout <class
'torch.nn.modules.dropout.Dropout'>
transformer.ln_f <class 'torch.nn.modules.normalization.LayerNorm'>
score <class 'torch.nn.modules.linear.Linear'>
```

TODO 1: Freeze the Model and Train Only Attention Weights

You are assigned to freeze the entire model, except for the last two attention weights and the classification head. Note that, in this context, the attention weights do not include the projection layer of the transformer. Instead, they refer only to the weights of the query, key, and value.

HINT: `c_proj` is projection layer.

```
for n, p in model.named_parameters():
    # TODO 1: freeze every layer except the last two attention weights
and classification head

    # By default, set requires_grad to False (freeze all parameters)
    p.requires_grad = False

    # Unfreeze the classification head (score layer)
    if 'score' in n:
        p.requires_grad = True

    # Unfreeze the attention weights (q, k, v) in the last two layers
    # GPT-2 has 12 layers (0-11), so we want layers 10 and 11
    if any(layer in n for layer in ['h.10.', 'h.11.']) and any(weight
in n for weight in ['attn.c_attn.weight', 'attn.c_attn.bias']):
        p.requires_grad = True
```

Check Your Answer: The number of learnable parameters is around 3545088.

```
pytorch_total_params = sum(p.numel() for p in model.parameters() if
p.requires_grad)
print("Number of Trainable Parameters:", pytorch_total_params)

Number of Trainable Parameters: 3545088
```

You are assigned to train the GPT-2 model on the SST-2 dataset. Due to the long training time, you will train the model for only 3 epochs. Your model should have around 86-88% accuracy.

```
optimizer = AdamW(model.parameters(), lr=5e-5)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device:", device)
model.to(device)

num_epochs = 3
for epoch in tqdm(range(num_epochs)):
    model.train()
    for batch in tqdm(train_dataloader):
        input_ids = batch["input_ids"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        labels = batch["label"].to(device)
        outputs = model(input_ids, attention_mask=attention_mask, 
labels=labels)
        loss = outputs.loss

        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for batch in test_dataloader:
            input_ids = batch["input_ids"].to(device)
            attention_mask = batch["attention_mask"].to(device)
            labels = batch["label"].to(device)
            outputs = model(input_ids, attention_mask=attention_mask)
            predictions = torch.argmax(outputs.logits, dim=-1)
            correct += (predictions == labels).sum().item()
            total += labels.size(0)

    accuracy = correct / total
    print(f"Epoch {epoch + 1}/{num_epochs} - Accuracy: 
{accuracy:.4f}")
```

```
Device: cuda

/usr/local/lib/python3.10/dist-packages/transformers/
optimization.py:591: FutureWarning: This implementation of AdamW is 
deprecated and will be removed in a future version. Use the PyTorch 
implementation torch.optim.AdamW instead, or set 
`no_deprecation_warning=True` to disable this warning
  warnings.warn(
```

```
{"model_id":"4bfe005983984fb3ad294dd54350377b","version_major":2,"version_minor":0}

{"model_id":"241c0b20a2d343d0b7776869dce3dfa0","version_major":2,"version_minor":0}

Epoch 1/3 - Accuracy: 0.8406

{"model_id":"1c583a25b4644749be2384bd44642d7e","version_major":2,"version_minor":0}

Epoch 2/3 - Accuracy: 0.8830

{"model_id":"90e110bfb048460cb9408c50ed84a891","version_major":2,"version_minor":0}

Epoch 3/3 - Accuracy: 0.8865
```

As you can see, fine-tuning in the traditional way takes a long time to complete and also requires a high-computation GPU to fine-tune the entire model. Therefore, it is not feasible for most people.

In the next part, we will introduce a better method: parameter-efficient learning, which requires lower computation. We will focus on the state-of-the-art method, Low-Rank Adaptation (LoRA).

Low Rank Adaptation

The concept of LoRA is that we are going to estimate the gradient (adaptation matrix) with two smaller matrices (A and B):

$$\text{Adaptation Matrix} = B \times A$$

where $\text{Adaptation Matrix} \in R^{m \times n}$, $A \in R^{r \times n}$, and $B \in R^{m \times r}$. We could make this approximation based on the assumption that Adaptation Matrix has a rank of r. Therefore, the fine-tuned weight becomes:

$$W = W_0 + \Delta W$$

$$¿ W_0 + \frac{\alpha}{r} B A$$

where W denotes the fine-tuned weight, W_0 represents pre-trained weight, ΔW is the gradient and α can be seen as a learning rate. A is initialized using a common initialization, like Kaiming initialization, during the initialization process. On the other hand, B is set to 0 such that the model's output remains the same after injecting LoRA, resulting in a stabilized training process.

To summarize, when injecting LoRA into a layer, we insert new parameters called matrix A and B and initialize them using the above description. Then, we modify the forward pass with `forward_hook` such that the output becomes:

$$h = W x + \frac{\alpha}{r} B A x$$

where x and h are the input and output, respectively. We recommend you read this blog to learn more about `forward_hook`.

LoRA on Linear Layer

- TODO 2: initialize A and B to ones (every entry in the matrix is one), such that we can verify your forward pass after attaching the hook.
- TODO 3: implement the forward hook such that new output h is

$$h = W x + \frac{\alpha}{r} B A x$$

Hint: When you want to declare and initialize a parameter, you can use `torch.nn.Parameter` and `torch.nn.init`, respectively.

```
import math
import torch.nn as nn
import torch.nn.functional as F
# Initialize LoRA and attach a hook.
def attach_lora(layer, r, lora_alpha, in_features, out_features):
    assert r > 0, "rank must greater than 0."
    # TODO 2: Declare A and B matrices and initialize A and B to ones.
    layer.lora_A = nn.Parameter(torch.ones((r, in_features)))
    layer.lora_B = nn.Parameter(torch.ones((out_features, r)))

    scaling = lora_alpha / r

    def hook(model, input, output):
        assert len(input) == 1, "The length of the input must be 1."
        # TODO 3: Compute adapatation matrix (BA) and modify the
forward pass.
        lora_update = torch.matmul(layer.lora_B, layer.lora_A) *
scaling
        output += torch.matmul(input[0], lora_update.T) # input[0] =
get item in batch -> x

    return hook
```

To test your `forward_hook`, we will check the difference of the output before and after injecting the LoRA when you initialize matrices A and B with ones.

```
from transformers.modeling_utils import Conv1D
# The Conv1D layer from the Transformer library is actually a linear
layer.
(https://github.com/huggingface/transformers/blob/main/src/transformer
s/pytorch_utils.py#L100)

class DummyLinear(nn.Module):
  def __init__(self):
    super().__init__()
```

```
    self.linear = Conv1D(20, 10)

  def forward(self, x):
    return self.linear(x)

r, lora_alpha = 1, 4
input_ = torch.arange(10, dtype=torch.float32).unsqueeze(0)
dummy_linear = DummyLinear()
output_before = dummy_linear(input_)
for name, module in dummy_linear.named_modules():
  if isinstance(module, Conv1D):
    in_features, out_features = module.weight.shape
    h = module.register_forward_hook(attach_lora(module, r,
lora_alpha, in_features, out_features))
output_after = dummy_linear(input_)

if torch.all(torch.isclose(output_after - output_before, lora_alpha *
input_.sum() * torch.ones_like(output_before))):
  print("Your forward hook seems to be correct.")
else:
  print("There is something wrong with your forward hook.")
```

```
Your forward hook seems to be correct.
```

Instruction

TODO 4: Change the initialization of A and B where A is initialized with Kaiming Uniform (a = sqrt(5)), and B is set to 0.

```
# Initialize LoRA and attach a hook.
def attach_lora(layer, r, lora_alpha, in_features, out_features):
    assert r > 0, "rank must greater than 0."
    # TODO 4: initialize A with kaiming uniform with a = sqrt(5) and
initialize B to 0.
    layer.lora_A = nn.Parameter(torch.empty(r, in_features))
    layer.lora_B = nn.Parameter(torch.zeros(out_features, r))

    nn.init.kaiming_uniform_(layer.lora_A, a=math.sqrt(5))

    scaling = lora_alpha / r

    def hook(model, input, output):
        assert len(input) == 1, "The length of the input must be 1."
        # Copy from TODO 3
        lora_update = torch.matmul(layer.lora_B, layer.lora_A) *
scaling
        output += torch.matmul(input[0], lora_update.T)

    return hook
```

Similar to TODO 1, You are assigned to inject lora into the last two attention weights.

TODO 5: inject lora into the last two attention weights

```
r, lora_alpha = 1, 4
def attach_lora_to_maskformer(model, r, lora_alpha):
    hooks = []
    transformer_layers = model.transformer.h
    num_layers = len(transformer_layers)

    # Target only the last two layers
    target_layer_indices = range(num_layers - 2, num_layers)

    for layer_idx in target_layer_indices:
        # Access attention mechanism
        attention_module = transformer_layers[layer_idx].attn
        attention_layer = attention_module.c_attn

        # Get layer dimensions
        in_features, out_features = attention_layer.weight.shape

        # Attach LoRA and register hook
        hook_fn = attach_lora(
            layer=attention_layer,
            r=r,
            lora_alpha=lora_alpha,
            in_features=in_features,
            out_features=out_features
        )
        hook_handle = attention_layer.register_forward_hook(hook_fn)
        hooks.append(hook_handle)
    return hooks

model = GPT2ForSequenceClassification.from_pretrained("gpt2",
num_labels=2)
model.config.pad_token_id = tokenizer.eos_token_id
hooks = attach_lora_to_maskformer(model, r, lora_alpha)
```

```
Some weights of GPT2ForSequenceClassification were not initialized
from the model checkpoint at gpt2 and are newly initialized:
['score.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
```

TODO 6: Freeze the Model and Train Only LoRA Weights

You are assigned to freeze the entire model, except for the bias of the last two attention weights, LoRA weights, and the classification head.

```
for n, p in model.named_parameters():
    # TODO 6: freeze every layer except the bias of the last two
attention weights, LoRA weights, and classification head.

    # By default, freeze all parameters
    p.requires_grad = False

    # Unfreeze the classification head (score layer)
    if 'score' in n:
        p.requires_grad = True

    # Unfreeze the bias of the last two attention weights (c_attn.bias
in layers 10 and 11)
    if any(layer in n for layer in ['h.10.', 'h.11.']) and
'attn.c_attn.bias' in n:
        p.requires_grad = True

    # Unfreeze LoRA weights (lora_A and lora_B)
    if 'lora_A' in n or 'lora_B' in n:
        p.requires_grad = True
```

Check Your Answer: The number of learnable parameters is around 12288.

```
pytorch_total_params = sum(p.numel() for p in model.parameters() if
p.requires_grad)
print(pytorch_total_params)

12288

optimizer = AdamW(model.parameters(), lr=5e-5)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device:", device)
model.to(device)

num_epochs = 3
for epoch in tqdm(range(num_epochs)):
    model.train()
    for batch in tqdm(train_dataloader):
        input_ids = batch["input_ids"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        labels = batch["label"].to(device)
        outputs = model(input_ids, attention_mask=attention_mask,
labels=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

    model.eval()
```

```
    correct = 0
    total = 0
    with torch.no_grad():
        for batch in test_dataloader:
            input_ids = batch["input_ids"].to(device)
            attention_mask = batch["attention_mask"].to(device)
            labels = batch["label"].to(device)
            outputs = model(input_ids, attention_mask=attention_mask)
            predictions = torch.argmax(outputs.logits, dim=-1)
            correct += (predictions == labels).sum().item()
            total += labels.size(0)

    accuracy = correct / total
    print(f"Epoch {epoch + 1}/{num_epochs} - Accuracy:
{accuracy:.4f}")
```

```
Device: cuda
```

{"model_id":"f63c07bf22cb4b9a924e49559948d16a","version_major":2,"version_minor":0}

{"model_id":"3dee1488a91c49358e89fb104fc433ff","version_major":2,"version_minor":0}

```
Epoch 1/3 - Accuracy: 0.7546
```

{"model_id":"be9118e194934d80be22a2c50c238a61","version_major":2,"version_minor":0}

```
Epoch 2/3 - Accuracy: 0.8314
```

{"model_id":"4a704dcdc1534c8384bd51ee4e9003a2","version_major":2,"version_minor":0}

```
Epoch 3/3 - Accuracy: 0.8578
```

Part 2: PEFT Library

In the first part, you learned how to implement LoRA from scratch. However, in real-world applications, we can simplify this process by using pre-built libraries. One such library is `peft`, which allows us to inject LoRA into a model more efficiently. By declaring the injected modules in the LoRAConfig, we can easily integrate LoRA without having to implement it ourselves. In this section, you will use the `peft` library to apply LoRA to the model.

TODO 7-8: Initialize the LoRA Config and Set trainable parameters

Your task is to initialize `LoRAConfig` using the same hyperparameters as in TODO 6 (`r=1`, `lora_alpha=4`). Apply LoRA only to the last two attention layers. Then, make sure to freeze the entire model except for the LoRA weights, the bias in the LoRA-injected layers, and the classification head. You only need to set the classification head to be trainable where the rest parameters are already set according to our `LoRAConfig`.

HINT: The total number of trainable parameters should match the result from Part 1 (TODO 6).

```
from peft import LoraConfig, get_peft_model

# TODO 7: Initialize LoRAConfig
layers = ["transformer.h.10.attn.c_attn",
"transformer.h.11.attn.c_attn"]

lora_config = LoraConfig(
    r=1,                    # Rank of the adaptation matrices (same as
TODO 6)
    lora_alpha=4,          # Scaling factor (same as TODO 6)
    target_modules=layers,  # Only apply to last two layers (10 and
11)
    modules_to_save=["score"],  # Ensure classification head is
trainable
    bias="lora_only"            # We'll handle bias training
separately
)

model = GPT2ForSequenceClassification.from_pretrained("gpt2",
num_labels=2)
model.config.pad_token_id = tokenizer.eos_token_id

model = get_peft_model(model, lora_config)
model = model.to(device)

# TODO 8: Set classification head to trainable
for n, p in model.named_parameters():
    # Unfreeze classification head
    if n.startswith('score'):
        p.requires_grad = True
```

```
Some weights of GPT2ForSequenceClassification were not initialized
from the model checkpoint at gpt2 and are newly initialized:
['score.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
/usr/local/lib/python3.10/dist-packages/peft/tuners/lora/layer.py:1264
: UserWarning: fan_in_fan_out is set to False but the target module is
```

```
`Conv1D`. Setting fan_in_fan_out to True.
  warnings.warn(
```

```
pytorch_total_params = sum(p.numel() for p in model.parameters() if
p.requires_grad)
print(pytorch_total_params)
```

```
12288
```

```
optimizer = AdamW(model.parameters(), lr=5e-5)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device:", device)
model.to(device)

num_epochs = 3
for epoch in tqdm(range(num_epochs)):
    model.train()
    for batch in tqdm(train_dataloader):
        input_ids = batch["input_ids"].to(device)
        attention_mask = batch["attention_mask"].to(device)
        labels = batch["label"].to(device)
        outputs = model(input_ids, attention_mask=attention_mask,
labels=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()

    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for batch in test_dataloader:
            input_ids = batch["input_ids"].to(device)
            attention_mask = batch["attention_mask"].to(device)
            labels = batch["label"].to(device)
            outputs = model(input_ids, attention_mask=attention_mask)
            predictions = torch.argmax(outputs.logits, dim=-1)
            correct += (predictions == labels).sum().item()
            total += labels.size(0)

    accuracy = correct / total
    print(f"Epoch {epoch + 1}/{num_epochs} - Accuracy:
{accuracy:.4f}")
```

```
Device: cuda
```

{"model_id":"319bdc6f253e499eaee3973f869d4461","version_major":2,"vers
ion_minor":0}

{"model_id":"23b5473c94384992a31d80c1db7e8208","version_major":2,"version_minor":0}

Epoch 1/3 - Accuracy: 0.7970

{"model_id":"e4eb51f83f0d41dca077a9a0b9ee31b8","version_major":2,"version_minor":0}

Epoch 2/3 - Accuracy: 0.8303

{"model_id":"38673381b12e45c19cad33e96c8a9879","version_major":2,"version_minor":0}

Epoch 3/3 - Accuracy: 0.8486