In [26]:
```
from datasets import load_dataset, DatasetDict,Dataset
from transformers import (
    AutoTokenizer,
    AutoConfig,
    AutoModelForSequenceClassification,
    DataCollatorWithPadding,
    TrainingArguments,
    Trainer
)
from peft import PeftModel,PeftConfig,get_peft_model,LoraConfig
import evaluate
import torch
import numpy as np
```

It looks like you're importing various modules and classes from the Hugging Face `transformers` library for sequence classification tasks. These imports include:

- `AutoTokenizer`: Used to automatically load the appropriate tokenizer for a given pretrained model.
- `AutoConfig`: Helps in automatically configuring model settings for a pretrained model.
- `AutoModelForSequenceClassification`: Automatically loads a pretrained model for sequence classification.
- `DataCollatorWithPadding`: A utility class for collating and handling padding of sequences during training.
- `TrainingArguments`: Contains settings and configurations for training a model.
- `Trainer`: The training interface provided by the `transformers` library for training models.

With these imports, you have access to tools for easily initializing and training sequence classification models using pretrained models provided by Hugging Face's `transformers` library. You can use these components to set up, train, and fine-tune models for various natural language understanding or sequence classification tasks.

In [27]:
```
model_checkpoint = 'distilbert-base-uncased' #67M base parameter

# define label maps
id2label = {0:"Negative",1:"Positive"}
label2id = {"Negative":0,"Positive":1}

# generate classification model from model_checkpoint
model = AutoModelForSequenceClassification.from_pretrained(
    model_checkpoint,num_labels = 2,id2label=id2label,label2id=label2id
)
```

```
Some weights of DistilBertForSequenceClassification were not initialized from the mod
el checkpoint at distilbert-base-uncased and are newly initialized: ['classifier.weig
ht', 'classifier.bias', 'pre_classifier.weight', 'pre_classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use it for p
redictions and inference.
```

In [28]:
```
model
```

Out[28]:
```
DistilBertForSequenceClassification(
  (distilbert): DistilBertModel(
    (embeddings): Embeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (transformer): Transformer(
      (layer): ModuleList(
        (0-5): 6 x TransformerBlock(
          (attention): MultiHeadSelfAttention(
            (dropout): Dropout(p=0.1, inplace=False)
            (q_lin): Linear(in_features=768, out_features=768, bias=True)
            (k_lin): Linear(in_features=768, out_features=768, bias=True)
            (v_lin): Linear(in_features=768, out_features=768, bias=True)
            (out_lin): Linear(in_features=768, out_features=768, bias=True)
          )
          (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
          (ffn): FFN(
            (dropout): Dropout(p=0.1, inplace=False)
            (lin1): Linear(in_features=768, out_features=3072, bias=True)
            (lin2): Linear(in_features=3072, out_features=768, bias=True)
            (activation): GELUActivation()
          )
          (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        )
      )
    )
  )
  (pre_classifier): Linear(in_features=768, out_features=768, bias=True)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
```

In [29]:
```
# Load dataset
from datasets import load_dataset

dataset = load_dataset("shawhin/imdb-truncated")
dataset
```

Out[29]:
```
DatasetDict({
    train: Dataset({
        features: ['label', 'text'],
        num_rows: 1000
    })
    validation: Dataset({
        features: ['label', 'text'],
        num_rows: 1000
    })
})
```

In [30]:
```
# Create tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_checkpoint,add_prefix_space = True)

# Create tokenize function
def tokenize_function(examples):
    # extract text
    text = examples["text"]
```

```
    #tokenize and truncate text
    tokenizer.truncation_side="left"
    tokenized_inputs = tokenizer(
        text,
        return_tensors="np",
        truncation=True,
        max_length=12
    )

    return tokenized_inputs

# add pad token if none exists
if tokenizer.pad_token is None:
    tokenizer.add_special_tokens({'pad_token':'[PAD]'})
    model.resize_token_embeddings(len(tokenizer))

# tokenizer training and validation datasets
tokenized_dataset = dataset.map(tokenize_function, batched=True)
tokenized_dataset
```

Out[30]:

```
Map:   0%|          | 0/1000 [00:00<?, ? examples/s]
DatasetDict({
    train: Dataset({
        features: ['label', 'text', 'input_ids', 'attention_mask'],
        num_rows: 1000
    })
    validation: Dataset({
        features: ['label', 'text', 'input_ids', 'attention_mask'],
        num_rows: 1000
    })
})
```

In [31]:

```
# Create data collator
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

`DataCollatorWithPadding` is a utility class from the Hugging Face `transformers` library used for collating and processing data during training in cases where input sequences have varying lengths. This class specifically handles padding of sequences to ensure uniform lengths within a batch.

When working with sequence data for NLP tasks, inputs often have different lengths. However, during training, it's efficient to process inputs in batches, which requires sequences within a batch to have the same length. This is where padding comes in—to ensure uniformity within a batch.

`DataCollatorWithPadding` performs the following tasks:

- It collates examples into batches.
- It pads sequences within each batch to the same length, using a padding token (usually an ID representing padding) to fill in the gaps.

Here's an example of how you might use `DataCollatorWithPadding`:

```
from transformers import DataCollatorWithPadding
from transformers import Trainer, TrainingArguments
```

```
# Initialize a DataCollatorWithPadding instance
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

# Create your training arguments and other necessary objects
training_args = TrainingArguments(...)
trainer = Trainer(
    model=model,
    args=training_args,
    data_collator=data_collator,  # Set the data collator for handling
padding
    train_dataset=train_dataset,
    eval_dataset=eval_dataset
)

# Start the training
trainer.train()
```

In this example, `tokenizer` is an instance of the Hugging Face tokenizer that you're using for your model. `DataCollatorWithPadding` is initialized with the tokenizer to handle padding of sequences during training.

By using `DataCollatorWithPadding` within the `Trainer`, you ensure that your model processes batches of sequences efficiently by automatically padding them to the same length, allowing for smoother training without having to manually handle padding logic in your training loop.

Evaluation metrics

In [32]:
```
# import accuracy evaluation metric
accuracy = evaluate.load("accuracy")

# define an evaluation function to pass into trainer later
def compute_metrics(p):
    predictions, labels = p
    predictions = np.argmax(predictions,axis=1)

    return {"accuracy":accuracy.compute(predictions=predictions,
                                        references=labels)}
```

Untrained model performance

In [33]:
```
### define list of examples
text_list = ["It was good","Not a fan,dont recommend","Better than first one","this or

print("Untrained model prediction")
print("----------------------------------------------------")

for text in text_list:
    # tokenize text
    inputs = tokenizer.encode(text,return_tensors="pt")
    # Compute logits
    logits = model(inputs).logits
    # Convert logits to labels
```

```
    prediction = torch.argmax(logits)

    print(text + " - " +id2label[prediction.tolist()])
```

```
Untrained model prediction
--------------------------------------------------
It was good - Positive
Not a fan,dont recommend - Negative
Better than first one - Negative
this one is pass - Positive
```

PEFT (Parameterized Entangled Friendly Transformations):

PEFT in the context of language models like LLMs involves techniques related to entanglement and transformation of parameters within the model architecture. It's a method that aims to improve the expressiveness and efficiency of language models by introducing entangled transformations that enable more effective parameter interactions within the model. PEFT is a technique that leverages quantum-inspired principles to enhance the learning dynamics or performance of language models.

LoRA (Low Rank Adaptation):

LoRA, in the context of language models, specifically refers to "Low Rank Adaptation," which focuses on adapting and compressing the parameters of large language models. This technique aims to reduce the computational overhead and memory requirements of LLMs by leveraging low-rank approximations. By approximating the parameters using low-rank matrices or tensors, LoRA helps in making these models more efficient while preserving their performance to a significant extent.

Both PEFT and LoRA are strategies or methodologies used to address challenges related to efficiency, computational requirements, and performance enhancement in large language models like LLMs. While PEFT involves entangled transformations to improve expressiveness, LoRA focuses on low-rank adaptations for parameter compression and efficiency without sacrificing performance significantly.

These techniques aim to optimize and improve the capabilities of language models, offering ways to make them more computationally efficient, easier to train, or more scalable for various NLP applications.

Fine Tuning with LoRa

In [34]:
```
peft_config = LoraConfig(task_type="SEQ_CLS", # for seuqence classification
                r=4, # intrinsic rank of traininable weight matrix,
                lora_alpha=32, # this is like a learning rate
                lora_dropout=0.01, # probabilty of dropout
                target_modules= ['q_lin'],) # we apply lora to query layer
```

In [35]:
```
model = get_peft_model(model,peft_config)
model.print_trainable_parameters()
```

```
trainable params: 628,994 || all params: 67,584,004 || trainable%: 0.9306847223789819
```

In [36]:
```python
# hyperparameters
lr = 1e-3 #size of optimization step
batch_size = 4 # number of examples processed per optimization step
num_epoch = 10 # number of times model runs through training data

# define training arguments
training_args = TrainingArguments(
    output_dir=model_checkpoint + "-lora-text-classification",
    learning_rate=lr,
    per_device_train_batch_size=batch_size,
    per_device_eval_batch_size=batch_size,
    num_train_epochs=num_epoch,
    weight_decay=0.01,
    evaluation_strategy="epoch",
    save_strategy="epoch",
    load_best_model_at_end=True,
)
```

In [37]:
```python
# creater trainer object
trainer = Trainer(
    model=model, # our peft model
    args=training_args, # hyperparameter
    train_dataset=tokenized_dataset["train"], # training data
    eval_dataset=tokenized_dataset["validation"], # validation data
    tokenizer=tokenizer, # define tokenizer
    data_collator=data_collator, # this will dynamically pad examples in each batch to
    compute_metrics=compute_metrics, # evaluate model using compute_metrics
)

# train model
trainer.train()
```

```
  0%|          | 0/2500 [00:00<?, ?it/s]
You're using a DistilBertTokenizerFast tokenizer. Please note that with a fast tokeni
zer, using the `__call__` method is faster than using a method to encode the text fol
lowed by a call to the `pad` method to get a padded encoding.
  0%|          | 0/250 [00:00<?, ?it/s]
Trainer is attempting to log a value of "{'accuracy': 0.701}" of type <class 'dict'>
for key "eval_accuracy" as a metric. MLflow's log_metric() only accepts float and int
types so we dropped this attribute.
Trainer is attempting to log a value of "{'accuracy': 0.701}" of type <class 'dict'>
for key "eval/accuracy" as a scalar. This invocation of Tensorboard's writer.add_scal
ar() is incorrect so we dropped this attribute.
Checkpoint destination directory distilbert-base-uncased-lora-text-classification\che
ckpoint-250 already exists and is non-empty.Saving will proceed but saved results may
be invalid.
{'eval_loss': 0.5780891180038452, 'eval_accuracy': {'accuracy': 0.701}, 'eval_runtim
e': 2.0084, 'eval_samples_per_second': 497.92, 'eval_steps_per_second': 124.48, 'epoc
h': 1.0}
{'loss': 0.6095, 'learning_rate': 0.0008, 'epoch': 2.0}
  0%|          | 0/250 [00:00<?, ?it/s]
```

```
Trainer is attempting to log a value of "{'accuracy': 0.692}" of type <class 'dict'>
for key "eval_accuracy" as a metric. MLflow's log_metric() only accepts float and int
types so we dropped this attribute.
Trainer is attempting to log a value of "{'accuracy': 0.692}" of type <class 'dict'>
for key "eval/accuracy" as a scalar. This invocation of Tensorboard's writer.add_scal
ar() is incorrect so we dropped this attribute.
Checkpoint destination directory distilbert-base-uncased-lora-text-classification\che
ckpoint-500 already exists and is non-empty.Saving will proceed but saved results may
be invalid.
{'eval_loss': 0.6857352256774902, 'eval_accuracy': {'accuracy': 0.692}, 'eval_runtim
e': 2.1827, 'eval_samples_per_second': 458.141, 'eval_steps_per_second': 114.535, 'ep
och': 2.0}
  0%|          | 0/250 [00:00<?, ?it/s]
Trainer is attempting to log a value of "{'accuracy': 0.694}" of type <class 'dict'>
for key "eval_accuracy" as a metric. MLflow's log_metric() only accepts float and int
types so we dropped this attribute.
Trainer is attempting to log a value of "{'accuracy': 0.694}" of type <class 'dict'>
for key "eval/accuracy" as a scalar. This invocation of Tensorboard's writer.add_scal
ar() is incorrect so we dropped this attribute.
Checkpoint destination directory distilbert-base-uncased-lora-text-classification\che
ckpoint-750 already exists and is non-empty.Saving will proceed but saved results may
be invalid.
{'eval_loss': 0.972552478313446, 'eval_accuracy': {'accuracy': 0.694}, 'eval_runtim
e': 2.0695, 'eval_samples_per_second': 483.198, 'eval_steps_per_second': 120.799, 'ep
och': 3.0}
{'loss': 0.3927, 'learning_rate': 0.0006, 'epoch': 4.0}
  0%|          | 0/250 [00:00<?, ?it/s]
Trainer is attempting to log a value of "{'accuracy': 0.704}" of type <class 'dict'>
for key "eval_accuracy" as a metric. MLflow's log_metric() only accepts float and int
types so we dropped this attribute.
Trainer is attempting to log a value of "{'accuracy': 0.704}" of type <class 'dict'>
for key "eval/accuracy" as a scalar. This invocation of Tensorboard's writer.add_scal
ar() is incorrect so we dropped this attribute.
Checkpoint destination directory distilbert-base-uncased-lora-text-classification\che
ckpoint-1000 already exists and is non-empty.Saving will proceed but saved results ma
y be invalid.
{'eval_loss': 1.1614502668380737, 'eval_accuracy': {'accuracy': 0.704}, 'eval_runtim
e': 2.1779, 'eval_samples_per_second': 459.166, 'eval_steps_per_second': 114.791, 'ep
och': 4.0}
  0%|          | 0/250 [00:00<?, ?it/s]
Trainer is attempting to log a value of "{'accuracy': 0.708}" of type <class 'dict'>
for key "eval_accuracy" as a metric. MLflow's log_metric() only accepts float and int
types so we dropped this attribute.
Trainer is attempting to log a value of "{'accuracy': 0.708}" of type <class 'dict'>
for key "eval/accuracy" as a scalar. This invocation of Tensorboard's writer.add_scal
ar() is incorrect so we dropped this attribute.
Checkpoint destination directory distilbert-base-uncased-lora-text-classification\che
ckpoint-1250 already exists and is non-empty.Saving will proceed but saved results ma
y be invalid.
{'eval_loss': 1.3681780099868774, 'eval_accuracy': {'accuracy': 0.708}, 'eval_runtim
e': 2.1277, 'eval_samples_per_second': 469.988, 'eval_steps_per_second': 117.497, 'ep
och': 5.0}
{'loss': 0.2128, 'learning_rate': 0.0004, 'epoch': 6.0}
  0%|          | 0/250 [00:00<?, ?it/s]
```

```
Trainer is attempting to log a value of "{'accuracy': 0.705}" of type <class 'dict'>
for key "eval_accuracy" as a metric. MLflow's log_metric() only accepts float and int
types so we dropped this attribute.
Trainer is attempting to log a value of "{'accuracy': 0.705}" of type <class 'dict'>
for key "eval/accuracy" as a scalar. This invocation of Tensorboard's writer.add_scal
ar() is incorrect so we dropped this attribute.
Checkpoint destination directory distilbert-base-uncased-lora-text-classification\che
ckpoint-1500 already exists and is non-empty.Saving will proceed but saved results ma
y be invalid.
{'eval_loss': 1.6282109022140503, 'eval_accuracy': {'accuracy': 0.705}, 'eval_runtim
e': 2.0456, 'eval_samples_per_second': 488.843, 'eval_steps_per_second': 122.211, 'ep
och': 6.0}
  0%|          | 0/250 [00:00<?, ?it/s]
Trainer is attempting to log a value of "{'accuracy': 0.697}" of type <class 'dict'>
for key "eval_accuracy" as a metric. MLflow's log_metric() only accepts float and int
types so we dropped this attribute.
Trainer is attempting to log a value of "{'accuracy': 0.697}" of type <class 'dict'>
for key "eval/accuracy" as a scalar. This invocation of Tensorboard's writer.add_scal
ar() is incorrect so we dropped this attribute.
Checkpoint destination directory distilbert-base-uncased-lora-text-classification\che
ckpoint-1750 already exists and is non-empty.Saving will proceed but saved results ma
y be invalid.
{'eval_loss': 1.7728582620620728, 'eval_accuracy': {'accuracy': 0.697}, 'eval_runtim
e': 2.0668, 'eval_samples_per_second': 483.85, 'eval_steps_per_second': 120.963, 'epo
ch': 7.0}
{'loss': 0.119, 'learning_rate': 0.0002, 'epoch': 8.0}
  0%|          | 0/250 [00:00<?, ?it/s]
Trainer is attempting to log a value of "{'accuracy': 0.698}" of type <class 'dict'>
for key "eval_accuracy" as a metric. MLflow's log_metric() only accepts float and int
types so we dropped this attribute.
Trainer is attempting to log a value of "{'accuracy': 0.698}" of type <class 'dict'>
for key "eval/accuracy" as a scalar. This invocation of Tensorboard's writer.add_scal
ar() is incorrect so we dropped this attribute.
Checkpoint destination directory distilbert-base-uncased-lora-text-classification\che
ckpoint-2000 already exists and is non-empty.Saving will proceed but saved results ma
y be invalid.
{'eval_loss': 1.9462685585021973, 'eval_accuracy': {'accuracy': 0.698}, 'eval_runtim
e': 2.1016, 'eval_samples_per_second': 475.836, 'eval_steps_per_second': 118.959, 'ep
och': 8.0}
  0%|          | 0/250 [00:00<?, ?it/s]
Trainer is attempting to log a value of "{'accuracy': 0.697}" of type <class 'dict'>
for key "eval_accuracy" as a metric. MLflow's log_metric() only accepts float and int
types so we dropped this attribute.
Trainer is attempting to log a value of "{'accuracy': 0.697}" of type <class 'dict'>
for key "eval/accuracy" as a scalar. This invocation of Tensorboard's writer.add_scal
ar() is incorrect so we dropped this attribute.
Checkpoint destination directory distilbert-base-uncased-lora-text-classification\che
ckpoint-2250 already exists and is non-empty.Saving will proceed but saved results ma
y be invalid.
{'eval_loss': 2.0068695545196533, 'eval_accuracy': {'accuracy': 0.697}, 'eval_runtim
e': 2.2825, 'eval_samples_per_second': 438.109, 'eval_steps_per_second': 109.527, 'ep
och': 9.0}
{'loss': 0.0587, 'learning_rate': 0.0, 'epoch': 10.0}
  0%|          | 0/250 [00:00<?, ?it/s]
```

```
Trainer is attempting to log a value of "{'accuracy': 0.702}" of type <class 'dict'>
for key "eval_accuracy" as a metric. MLflow's log_metric() only accepts float and int
types so we dropped this attribute.
Trainer is attempting to log a value of "{'accuracy': 0.702}" of type <class 'dict'>
for key "eval/accuracy" as a scalar. This invocation of Tensorboard's writer.add_scal
ar() is incorrect so we dropped this attribute.
Checkpoint destination directory distilbert-base-uncased-lora-text-classification\che
ckpoint-2500 already exists and is non-empty.Saving will proceed but saved results ma
y be invalid.
{'eval_loss': 2.029104232788086, 'eval_accuracy': {'accuracy': 0.702}, 'eval_runtim
e': 2.2051, 'eval_samples_per_second': 453.488, 'eval_steps_per_second': 113.372, 'ep
och': 10.0}
{'train_runtime': 79.2352, 'train_samples_per_second': 126.207, 'train_steps_per_seco
nd': 31.552, 'train_loss': 0.2785419151306152, 'epoch': 10.0}
```

Out[37]:
```
TrainOutput(global_step=2500, training_loss=0.2785419151306152, metrics={'train_runti
me': 79.2352, 'train_samples_per_second': 126.207, 'train_steps_per_second': 31.552,
'train_loss': 0.2785419151306152, 'epoch': 10.0})
```

Generate Prediction

In [38]:
```
model.to('cpu')

print("Trained model predictions:")
print("--------------------------")
for text in text_list:
    inputs = tokenizer.encode(text, return_tensors="pt").to("cpu")

    logits = model(inputs).logits
    predictions = torch.max(logits,1).indices

    print(text + " - " + id2label[predictions.tolist()[0]])
```

```
Trained model predictions:
--------------------------
It was good - Positive
Not a fan,dont recommend - Negative
Better than first one - Negative
this one is pass - Negative
```

In [ ]:
```

```