```python
# This Python 3 environment comes with many helpful analytics
libraries installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

VERSION = 27
# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save &
Run All"
# You can also write temporary files to /kaggle/temp/, but they won't
be saved outside of the current session
```

```
input/test_unlabelled.pkl
```

```python
# ----------------------------
# 1. Import libraries
# ----------------------------
import torch
from transformers import AutoTokenizer,
AutoModelForSequenceClassification, TrainingArguments, Trainer,
DataCollatorWithPadding
from datasets import load_dataset, load_from_disk
from peft import get_peft_model, LoraConfig, TaskType, AdaLoraConfig
from sklearn.metrics import accuracy_score
import numpy as np
import pandas as pd
import pickle
import nltk

# 2. Use GPU if available
# ----------------------------
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# ----------------------------
# 3. Load and preprocess AGNEWS dataset
```

```python
# ----------------------------
def tokenize_function(examples):
    return tokenizer(examples["text"], truncation=True,
padding="max_length", max_length=256)
dataset = load_dataset("ag_news")
tokenizer = AutoTokenizer.from_pretrained("roberta-base")

#Augmented training set - see create_augment.ipynb
tokenized_train = load_from_disk("tokenized_datasets/synonym_augment")

tokenized_test = dataset['test'].map(tokenize_function, batched=True)
tokenized_test = tokenized_test.rename_column("label", "labels")
tokenized_test.set_format("torch", columns=["input_ids",
"attention_mask", "labels"])


data_collator = DataCollatorWithPadding(tokenizer=tokenizer,
return_tensors="pt")
```

```
Using device: cuda
```

model = AutoModelForSequenceClassification.from_pretrained("roberta-base")

for name, module in model.named_modules(): if any(k in name for k in ["query", "key", "value", "dense", "proj"]): print(name)

```python
from peft.tuners.adalora import SVDLinear
# ----------------------------
# 4. Define training arguments
# ----------------------------
training_args = TrainingArguments(
    output_dir="./results",
    optim="adamw_torch",   #AdamW Optimizer
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-4,
    lr_scheduler_type="cosine",   #Cosine annealing LR scheduling
    warmup_ratio=0.1,
    per_device_train_batch_size=32,
    per_device_eval_batch_size=64,
    num_train_epochs=6,
    adam_beta1=0.9,
    adam_beta2=0.999,
    adam_epsilon=1e-8,
    weight_decay=0.05,
    report_to="none",
    fp16 = True,
    seed = 42,
    load_best_model_at_end=True,
)
```

```
model = AutoModelForSequenceClassification.from_pretrained("roberta-
base", num_labels=4)

# -------
# 5. AdaLoRA
# -------

total_steps = (len(tokenized_train) //
training_args.per_device_train_batch_size) *
training_args.num_train_epochs
total_steps = total_steps // training_args.gradient_accumulation_steps

ada_config = AdaLoraConfig(
    init_r=8,  # Filler value, changed below
    target_r=4,  # Target rank for query, key, value, dense
    tinit=200,   # Initial budgeting
    tfinal=total_steps-300,  #Budgeting at end of training
    total_step=total_steps,
    deltaT=10,  #Smooth AdaLoRA transitions
    beta1=0.85,
    beta2=0.95,
    lora_alpha=16,
    lora_dropout=0.1,   #Reduces overfitting
    target_modules=["query", "key", "value", "dense"],  # Ensure these
layers exist in the model
    bias="none",
    task_type=TaskType.SEQ_CLS,
    layers_to_transform=list(range(6, 12))  # Apply to layers 6-11
)

model = get_peft_model(model, ada_config)

# Manual override for specific layers
rank_overrides = {
    "query": 8,
    "key": 6,
    "value": 6,
    "dense": 4
}
#
----------------------------------------------------------------------
------------
# 6. Asymmetric LoRA - force unfrozen layers to take the override
ranks specified above
#
----------------------------------------------------------------------
------------
```

```python
for name, module in model.named_modules():
    if hasattr(module, "r"):
        # Extract layer type from name
        layer_type = name.split(".")[-1]  # "query", "key", "value", 
or "dense"
        if layer_type in rank_overrides:
            # Modify the attribute
            new_rank = rank_overrides[layer_type]

            # Update the rank attribute
            module.r = new_rank

            in_features = module.in_features
            out_features = module.out_features

            # Reinitialize all LoRA components
            for adapter_name in module.lora_A.keys():
                module.lora_A[adapter_name] = torch.nn.Parameter(
                    torch.randn((new_rank, in_features), 
device=model.device)
                )
                module.lora_B[adapter_name] = torch.nn.Parameter(
                    torch.zeros((out_features, new_rank), 
device=model.device)
                )
                module.lora_E[adapter_name] = torch.nn.Parameter(
                    torch.ones((new_rank, 1), device=model.device)  # 
Singular values
                )

            # Force re-registration of parameters
            module.to(model.device)

print("=" * 50)
print("Layers with AdaLoRA Ranks (r):")
print("=" * 50)
for name, module in model.named_modules():
    if hasattr(module, "r"):  # Check if it's a LoRA layer
        print(f"{name}: r = {module.r}")# (target_r = 
{module.target_r})")
print("=" * 50)

model.print_trainable_parameters()

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

```
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.

==================================================
Layers with AdaLoRA Ranks (r):
==================================================
base_model.model.roberta.encoder.layer.6.attention.self.query: r = 8
base_model.model.roberta.encoder.layer.6.attention.self.key: r = 6
base_model.model.roberta.encoder.layer.6.attention.self.value: r = 6
base_model.model.roberta.encoder.layer.6.attention.output.dense: r = 4
base_model.model.roberta.encoder.layer.6.intermediate.dense: r = 4
base_model.model.roberta.encoder.layer.6.output.dense: r = 4
base_model.model.roberta.encoder.layer.7.attention.self.query: r = 8
base_model.model.roberta.encoder.layer.7.attention.self.key: r = 6
base_model.model.roberta.encoder.layer.7.attention.self.value: r = 6
base_model.model.roberta.encoder.layer.7.attention.output.dense: r = 4
base_model.model.roberta.encoder.layer.7.intermediate.dense: r = 4
base_model.model.roberta.encoder.layer.7.output.dense: r = 4
base_model.model.roberta.encoder.layer.8.attention.self.query: r = 8
base_model.model.roberta.encoder.layer.8.attention.self.key: r = 6
base_model.model.roberta.encoder.layer.8.attention.self.value: r = 6
base_model.model.roberta.encoder.layer.8.attention.output.dense: r = 4
base_model.model.roberta.encoder.layer.8.intermediate.dense: r = 4
base_model.model.roberta.encoder.layer.8.output.dense: r = 4
base_model.model.roberta.encoder.layer.9.attention.self.query: r = 8
base_model.model.roberta.encoder.layer.9.attention.self.key: r = 6
base_model.model.roberta.encoder.layer.9.attention.self.value: r = 6
base_model.model.roberta.encoder.layer.9.attention.output.dense: r = 4
base_model.model.roberta.encoder.layer.9.intermediate.dense: r = 4
base_model.model.roberta.encoder.layer.9.output.dense: r = 4
base_model.model.roberta.encoder.layer.10.attention.self.query: r = 8
base_model.model.roberta.encoder.layer.10.attention.self.key: r = 6
base_model.model.roberta.encoder.layer.10.attention.self.value: r = 6
base_model.model.roberta.encoder.layer.10.attention.output.dense: r =
4
base_model.model.roberta.encoder.layer.10.intermediate.dense: r = 4
base_model.model.roberta.encoder.layer.10.output.dense: r = 4
base_model.model.roberta.encoder.layer.11.attention.self.query: r = 8
base_model.model.roberta.encoder.layer.11.attention.self.key: r = 6
base_model.model.roberta.encoder.layer.11.attention.self.value: r = 6
base_model.model.roberta.encoder.layer.11.attention.output.dense: r =
4
base_model.model.roberta.encoder.layer.11.intermediate.dense: r = 4
base_model.model.roberta.encoder.layer.11.output.dense: r = 4
==================================================
```

```
trainable params: 999,364 || all params: 125,648,108 || trainable%:
0.7954

# -----------------------------
# 7. Train the model
# -----------------------------
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_test,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics,
    data_collator=data_collator,
)

trainer.train()
```

```
/tmp/ipykernel_16942/2805326384.py:4: FutureWarning: `tokenizer` is
deprecated and will be removed in version 5.0.0 for
`Trainer.__init__`. Use `processing_class` instead.
  trainer = Trainer(
No label_names provided for model class
`PeftModelForSequenceClassification`. Since `PeftModel` hides base
models input arguments, if label_names is not given, label_names can't
be set automatically within `Trainer`. Note that empty label_names
list will be used instead.

<IPython.core.display.HTML object>

TrainOutput(global_step=45000, training_loss=69.42143246815999,
metrics={'train_runtime': 4477.2289, 'train_samples_per_second':
321.628, 'train_steps_per_second': 10.051, 'total_flos':
1.9165387456512e+17, 'train_loss': 69.42143246815999, 'epoch': 6.0})
```

```
# -----------------------------
# 8. Evaluate the model
# -----------------------------
eval_results = trainer.evaluate()
print("Final Evaluation Accuracy:", eval_results["eval_accuracy"])
```

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

Final Evaluation Accuracy: 0.9473684210526315
```

```
# -----------------------------
# 9. Check trainable parameter count
# -----------------------------
trainable_params = sum(p.numel() for p in model.parameters() if
p.requires_grad)
print(f"Trainable parameters: {trainable_params}")
```

```
Trainable parameters: 999364
```

```python
from datasets import Dataset
from torch.utils.data import DataLoader

# Load dataset object
with open("input/test_unlabelled.pkl", "rb") as f:
    test_dataset = pickle.load(f)

# Convert to HuggingFace Dataset (already is, but this helps
formatting)
test_dataset = Dataset.from_dict({"text": test_dataset["text"]})

# Tokenize function
def preprocess_function(examples):
    return tokenizer(examples["text"], truncation=True,
padding="max_length", max_length=256)

# Apply tokenizer
tokenized_test_dataset = test_dataset.map(preprocess_function,
batched=True)
tokenized_test_dataset.set_format(type="torch", columns=["input_ids",
"attention_mask"])

# Create PyTorch DataLoader for batching
test_dataloader = DataLoader(tokenized_test_dataset, batch_size=64)

# Prediction loop
model.eval()
all_predictions = []

with torch.no_grad():
    for batch in test_dataloader:
        batch = {k: v.to(device) for k, v in batch.items()}
        outputs = model(**batch)
        preds = torch.argmax(outputs.logits, dim=-1)
        all_predictions.extend(preds.cpu().numpy())
```


```
{"model_id":"e77a0391ac8142e586bc83fee4c2d5e7","version_major":2,"vers
ion_minor":0}
```


```python
# ----------------------------
# 10. Save predictions to CSV
# ----------------------------
df = pd.DataFrame({
    "ID": list(range(len(all_predictions))),   # ID □
    "label": all_predictions
})
df.to_csv(f"submission_v{VERSION}.csv", index=False)
print(f"□ Batched predictions complete. Saved to
submission_v{VERSION}.csv.")
```

```
 Batched predictions complete. Saved to submission_v27.csv.
```