SET_CONF

February 11, 2025

[1]:

```
import torch
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM,␣
 ↪DataCollatorForSeq2Seq
from datasets import load_dataset
from torch.optim import AdamW, SGD
from torch.utils.data import DataLoader
from rouge_score import rouge_scorer
from tqdm import tqdm
from torch.cuda.amp import autocast, GradScaler
from sklearn.metrics import precision_recall_fscore_support
```

```
m:\Project\SentimentAna\.venv\Lib\site-packages\tqdm\auto.py:21: TqdmWarning:
IProgress not found. Please update jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

[2]:

```
SAVE_PATH = 'M:/Project/SentimentAna/save'
```

[3]:

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(torch.cuda.is_available())
```

```
True
```

[4]:

```
scaler = torch.amp.GradScaler('cuda')
tokenizer = AutoTokenizer.from_pretrained("facebook/bart-base")
model = AutoModelForSeq2SeqLM.from_pretrained("facebook/bart-base").to(device)
dataset = load_dataset('cnn_dailymail', '3.0.0', split='train')
dataset = dataset.select(range(8000))
```

[5]:

```
def tokenize_function(example):
    inputs = tokenizer(example["article"], max_length=512,␣
 ↪padding="max_length", truncation=True)
    labels = tokenizer(example["highlights"], max_length=128,␣
 ↪padding="max_length", truncation=True)
    return {
        "input_ids": torch.tensor(inputs["input_ids"], dtype=torch.long),
        "attention_mask": torch.tensor(inputs["attention_mask"], dtype=torch.
 ↪long),
        "labels": torch.tensor(labels["input_ids"], dtype=torch.long),
    }
```

[6]:

```
tokenized_dataset = dataset.map(tokenize_function, batched=True,␣
 ↪remove_columns=["article", "highlights"])
tokenized_dataset.set_format(type="torch", columns=["input_ids",␣
 ↪"attention_mask", "labels"])

batch_size = 4
data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)
data_loader = DataLoader(tokenized_dataset, batch_size=batch_size,␣
 ↪shuffle=True, collate_fn=data_collator)
print(len(data_loader))
```

```
2000
```

[7]:

```
def train_model(model, data_loader, max_batches=1000, lr=5e-5):
    model.train()
    optimizer = AdamW(model.parameters(), lr=lr)
    batch_count = 0
    progress_bar = tqdm(data_loader, desc="Training Progress",␣
 ↪total=max_batches)
    for batch in progress_bar:
        if batch_count >= max_batches:
            break
        optimizer.zero_grad()
        batch = {k: v.to(device) for k, v in batch.items()}
        with torch.cuda.amp.autocast():
            outputs = model(**batch)
            loss = outputs.loss
        scaler.scale(loss).backward()
        scaler.step(optimizer)
        scaler.update()
        batch_count += 1
        progress_bar.set_postfix(loss=loss.item())
    print(f"Training completed. Processed {batch_count} batches.")
    model.save_pretrained(SAVE_PATH)
    tokenizer.save_pretrained(SAVE_PATH)
```

[8]:

```
train_model(model, data_loader, max_batches=2000)
```

```
Training Progress:   0%|          | 0/2000 [00:00<?,
?it/s]m:\Project\SentimentAna\.venv\Lib\site-
packages\transformers\data\data_collator.py:657: UserWarning: Creating a tensor
from a list of numpy.ndarrays is extremely slow. Please consider converting the
list to a single numpy.ndarray with numpy.array() before converting to a tensor.
(Triggered internally at C:\actions-
runner\_work\pytorch\pytorch\pytorch\torch\csrc\utils\tensor_new.cpp:257.)
```

```
  batch["labels"] = torch.tensor(batch["labels"], dtype=torch.int64)
C:\Users\iambh\AppData\Local\Temp\ipykernel_28400\2405581.py:11: FutureWarning:
`torch.cuda.amp.autocast(args…)` is deprecated. Please use
`torch.amp.autocast('cuda', args…)` instead.
  with torch.cuda.amp.autocast():
Training Progress: 100%|      | 2000/2000 [04:10<00:00,  7.97it/s,
loss=1.11]
m:\Project\SentimentAna\.venv\Lib\site-
packages\transformers\modeling_utils.py:2758: UserWarning: Moving the following
attributes in the config to the generation config: {'early_stopping': True,
'num_beams': 4, 'no_repeat_ngram_size': 3, 'forced_bos_token_id': 0}. You are
seeing this warning because you've set generation parameters in the model
config, as opposed to in the generation config.
  warnings.warn(
```

```
Training completed. Processed 2000 batches.
```

[9]:

```
#  Load the fine-tuned model
model_path = "save/"
tokenizer = AutoTokenizer.from_pretrained(model_path)
model = AutoModelForSeq2SeqLM.from_pretrained(model_path).to(device)

#  Load test dataset (using a small sample)
datasetGlobal = load_dataset("cnn_dailymail", "3.0.0")  # Use a subset for␣
 ↪quick evaluation
dataset = datasetGlobal['test'].select(range(1000))
```

[10]:

```
#  Define ROUGE scorer
scorer = rouge_scorer.RougeScorer(["rouge1", "rouge2", "rougeL"],␣
 ↪use_stemmer=True)

#  Evaluation function
def evaluate_model(model, tokenizer, dataset):
    model.eval()
    scores = {"rouge1": [], "rouge2": [], "rougeL": []}

    for sample in tqdm(dataset, desc="Evaluating"):
        article = sample["article"]
        reference_summary = sample["highlights"]

        # Tokenize input
        inputs = tokenizer(article, return_tensors="pt", max_length=512,␣
 ↪truncation=True).to(model.device)

        # Generate summary
        with torch.no_grad():
            output_ids = model.generate(**inputs, max_length=128, num_beams=4,␣
 ↪early_stopping=True)
```

```
        generated_summary = tokenizer.decode(output_ids[0],␣
↪skip_special_tokens=True)

        # Compute ROUGE scores
        rouge_scores = scorer.score(reference_summary, generated_summary)
        for key in scores.keys():
            # print(key, rouge_scores[key])
            scores[key].append(rouge_scores[key])
    return scores
```

[11]:

```
#  Run evaluation
rouge_results = evaluate_model(model, tokenizer, dataset)
print("ROUGE Scores:", len(rouge_results))
```

```
Evaluating: 100%|      | 1000/1000 [07:11<00:00,  2.32it/s]

ROUGE Scores: 3
```

[12]:

```
print("\nEvaluation Metrics:")
for rouge_type, scores in rouge_results.items():
    precision = 0.0
    recall = 0.0
    f1 = 0.0
    # print(scores[0])
    for score in scores:
        precision += score.precision
        recall += score.recall
        f1 += score.fmeasure
    print(f"{rouge_type.upper()} - Precision: {precision/len(scores)}, Recall:␣
↪{recall/len(scores)}, F1-score: {f1/len(scores)}")
```

```
Evaluation Metrics:
ROUGE1 - Precision: 0.28289406307855414, Recall: 0.37609117913634954, F1-score:
0.3152299116667997
ROUGE2 - Precision: 0.1054578594264912, Recall: 0.14104918621257406, F1-score:
0.11763983277090026
ROUGEL - Precision: 0.19906350405502668, Recall: 0.266200112771452, F1-score:
0.2222821589157812
```

[13]:

```
import requests
from bs4 import BeautifulSoup

def get_article_text(url):
    headers = {"User-Agent": "Mozilla/5.0"}
    response = requests.get(url, headers=headers)
    if response.status_code != 200:
        return "Failed to retrieve article."

    soup = BeautifulSoup(response.text, "html.parser")

    # Extract text from paragraphs
    paragraphs = soup.find_all("p")
    article_text = " ".join([p.get_text() for p in paragraphs])

    return article_text[:2000]  # Limit text to avoid exceeding model's input␣
 ↪length

def summarize_article(url):
    article_text = get_article_text(url)

    if "Failed" in article_text:
        return "Error fetching article."

    #  Tokenize input
    inputs = tokenizer(article_text, return_tensors="pt", max_length=512,␣
 ↪truncation=True).to(model.device)

    #  Generate summary
    with torch.no_grad():
        output_ids = model.generate(**inputs, max_length=128, num_beams=4,␣
 ↪early_stopping=True)

    summary = tokenizer.decode(output_ids[0], skip_special_tokens=True)
    return summary

#  Example Usage
article_url = "https://www.hindustantimes.com/trending/
 ↪beerbiceps-sparks-outrage-with-crass-joke-on-india-s-got-latent-watch-your-parents-have-sex
 ↪html"
summary = summarize_article(article_url)
print("Summary:\n", summary)
```

```
Summary:
 Ranveer Allahbadia is facing flak for a crass joke on show India's Got Latent .
The comedian appeared on the show alongside Samay Raina, Ashish Chanchlani,
Apoorva Mukhija aka The Rebel Kid .
He has hosted several Indian politicians on his podcast .
```

[14]:

```
# dataset = datasetGlobal['test'].select(range(1000))
meta_train = datasetGlobal['train'].select(range(6000))  # 75% for meta-train
meta_test = datasetGlobal['test'].select(range(2000))   # 25% for meta-test
print(len(meta_test))
```

```
2000
```

[15]:

```
meta_train = meta_train.map(tokenize_function, batched=True,␣
 ↪remove_columns=["article", "highlights"])
meta_test = meta_test.map(tokenize_function, batched=True,␣
 ↪remove_columns=["article", "highlights"])
meta_train.set_format(type="torch", columns=["input_ids", "attention_mask",␣
 ↪"labels"])
meta_test.set_format(type="torch", columns=["input_ids", "attention_mask",␣
 ↪"labels"])

#  DataLoader
batch_size = 2
meta_train_loader = DataLoader(meta_train, batch_size=batch_size, shuffle=True)
meta_test_loader = DataLoader(meta_test, batch_size=batch_size, shuffle=True)
print(len(meta_train_loader))
```

```
3000
```

[16]:

```
import higher

def maml_train(model, meta_train_loader, meta_lr=1e-3, outer_lr=5e-5,␣
 ↪inner_steps=3, meta_steps=500, batch_size=2):
    model.train()
    meta_optimizer = AdamW(model.parameters(), lr=outer_lr)
    scaler = torch.amp.GradScaler(device='cuda')  # Mixed Precision for Memory␣
 ↪Optimization
    progress_bar = tqdm(range(meta_steps), desc="MAML Training")

    for step in progress_bar:
        try:
            task_batch = next(iter(meta_train_loader))  # Sample batch
            task_batch = {k: v.to(device) for k, v in task_batch.items()}

            # Clone model for task-specific updates (memory efficient)
            with higher.innerloop_ctx(model, SGD(model.parameters(),␣
 ↪lr=meta_lr), copy_initial_weights=False) as (fast_model, inner_optimizer):
                for _ in range(inner_steps):
                    with torch.amp.autocast(device_type='cuda'):  # Enable␣
 ↪mixed precision
                        inner_outputs = fast_model(**task_batch)
                        inner_loss = inner_outputs.loss
                    inner_optimizer.step(inner_loss)  #  No need for␣
 ↪zero_grad()

                # Compute meta-loss (final step)
                with torch.amp.autocast(device_type='cuda'):
                    meta_outputs = fast_model(**task_batch)
                    meta_loss = meta_outputs.loss
            #  Only backpropagate once in outer loop
            meta_optimizer.zero_grad()
            scaler.scale(meta_loss).backward()
            scaler.step(meta_optimizer)
            scaler.update()

            progress_bar.set_postfix(loss=meta_loss.item())

        except RuntimeError as e:
            if "CUDA out of memory" in str(e):
                print(f"Skipping step {step} due to OOM error.")
                torch.cuda.empty_cache()  # Free up memory
                continue  # Skip current step

    print("MAML Training Completed!")
    return model
```

[17]:

```
#  Train Model using MAML
model = maml_train(model, meta_train_loader)

#  Save Model
model.save_pretrained("maml_bart")
tokenizer.save_pretrained("maml_bart")
```

```
MAML Training: 100%|      | 500/500 [20:59<00:00,  2.52s/it]

MAML Training Completed!
```

[17]:

```
('maml_bart\\tokenizer_config.json',
 'maml_bart\\special_tokens_map.json',
 'maml_bart\\vocab.json',
 'maml_bart\\merges.txt',
 'maml_bart\\added_tokens.json',
 'maml_bart\\tokenizer.json')
```

[18]:

```
maml_model_path = "maml_bart/"
maml_tokenizer = AutoTokenizer.from_pretrained(maml_model_path)
maml_model = AutoModelForSeq2SeqLM.from_pretrained(maml_model_path).to(device)
```

[19]:

```
def evaluate_model(model, data_loader, tokenizer):
    model.eval()
    scorer = rouge_scorer.RougeScorer(["rouge1", "rouge2", "rougeL"],␣
 ↪use_stemmer=True)

    total_rouge = {
        "rouge1": {"precision": 0, "recall": 0, "f1": 0},
        "rouge2": {"precision": 0, "recall": 0, "f1": 0},
        "rougeL": {"precision": 0, "recall": 0, "f1": 0},
    }
    num_samples = 0

    with torch.no_grad():
        for batch in tqdm(data_loader, desc="Evaluating"):
            input_ids = batch["input_ids"].to(model.device)
            attention_mask = batch["attention_mask"].to(model.device)
            labels = batch["labels"]

            # Generate summaries
            generated_ids = model.generate(input_ids,␣
↪attention_mask=attention_mask, max_length=128)
            generated_texts = tokenizer.batch_decode(generated_ids,␣
↪skip_special_tokens=True)
            reference_texts = tokenizer.batch_decode(labels,␣
↪skip_special_tokens=True)

            # Compute ROUGE scores
            for ref, gen in zip(reference_texts, generated_texts):
                scores = scorer.score(ref, gen)
                for key in total_rouge.keys():
                    total_rouge[key]["precision"] += scores[key].precision
                    total_rouge[key]["recall"] += scores[key].recall
                    total_rouge[key]["f1"] += scores[key].fmeasure
                num_samples += 1

    # Compute the average scores
    avg_rouge = {
        key: {
            "precision": total_rouge[key]["precision"] / num_samples,
            "recall": total_rouge[key]["recall"] / num_samples,
            "f1": total_rouge[key]["f1"] / num_samples,
        }
        for key in total_rouge.keys()
    }

    # Print results in traditional format
    print("\nROUGE Scores:")
    for metric, scores in avg_rouge.items():
        print(f"{metric.upper()} - Precision: {scores['precision']:.4f}, Recall:
↪ {scores['recall']:.4f}, F1-score: {scores['f1']:.4f}")

    return avg_rouge
```

[20]:
```
#  Run evaluation
rouge_results = evaluate_model(maml_model, meta_test_loader, maml_tokenizer)
```

```
Evaluating: 100%|      | 1000/1000 [28:53<00:00,  1.73s/it]
```

```
ROUGE Scores:
ROUGE1 - Precision: 0.3535, Recall: 0.3877, F1-score: 0.3578
ROUGE2 - Precision: 0.1459, Recall: 0.1577, F1-score: 0.1464
ROUGEL - Precision: 0.2425, Recall: 0.2691, F1-score: 0.2467
```

[22]:

```
rouge_results
```

[22]:

```
{'rouge1': {'precision': 0.35353451806337854,
  'recall': 0.3877160705813453,
  'f1': 0.35783459653524324},
 'rouge2': {'precision': 0.14585591533322279,
  'recall': 0.15771790213848147,
  'f1': 0.1464047717008126},
 'rougeL': {'precision': 0.2425044543898082,
  'recall': 0.2690844932607845,
  'f1': 0.2467378872672124}}
```