NLP practice pj： 基于预训练transformer模型的wnli分类

本次pj是 基于预训练transformer模型的wnli分类，我最终通过更换模型为 `microsoft/deberta-v3-large` ,使用warmup和cosine策略，以及调整lr，bs，epoch等超参，实现了test的score达到85.6的效果

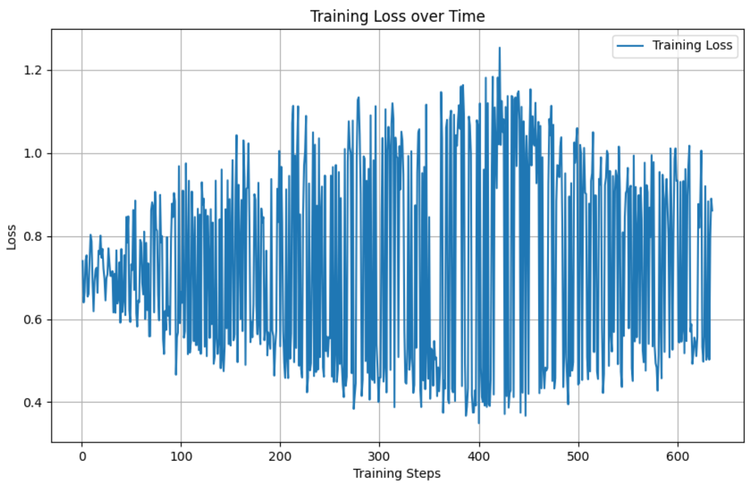
具体的PJ实验内容如下

首先我使用了原模型prajjwal1/bert-tiny进行实验

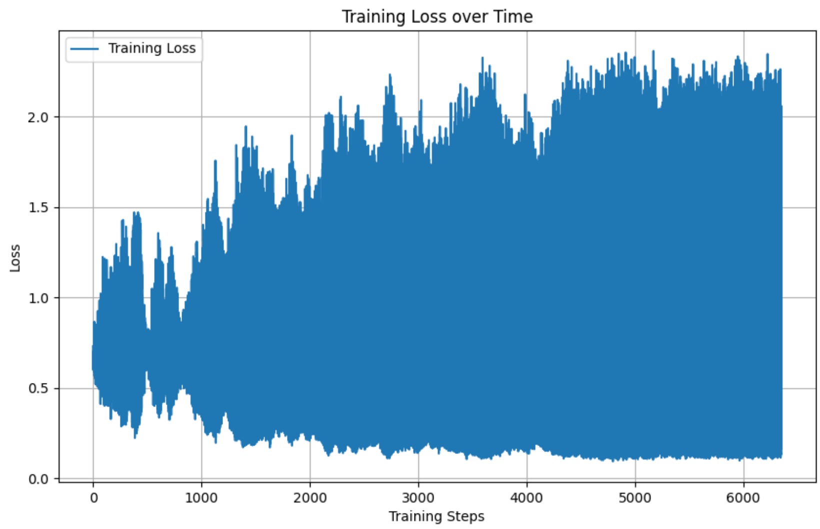
原模型训练之后的结果：

```
'eval_loss': 0.8087890148162842, 'eval_accuracy': 0.43661971830985913
```

loss曲线



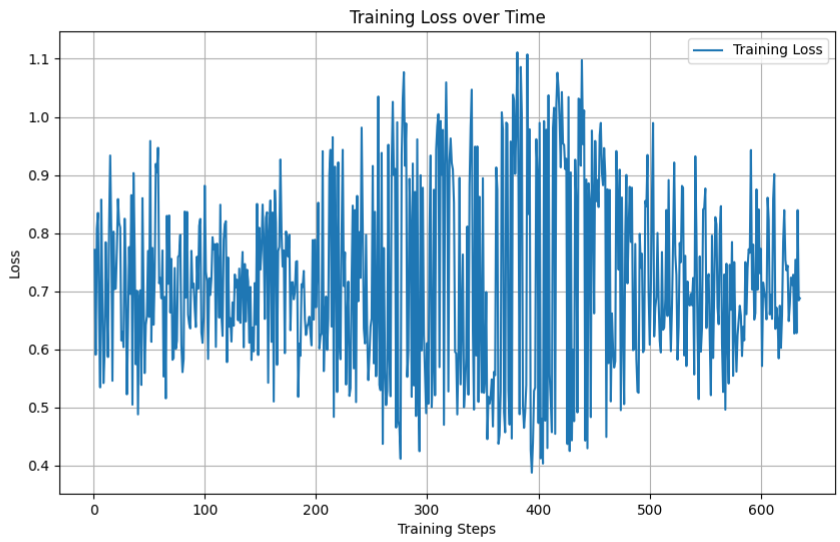
如果设置epoch为10，得到的训练曲线会这样



由曲线看出，模型并没有收敛，可能是学习率设置太大或者是学习率调度器有问题

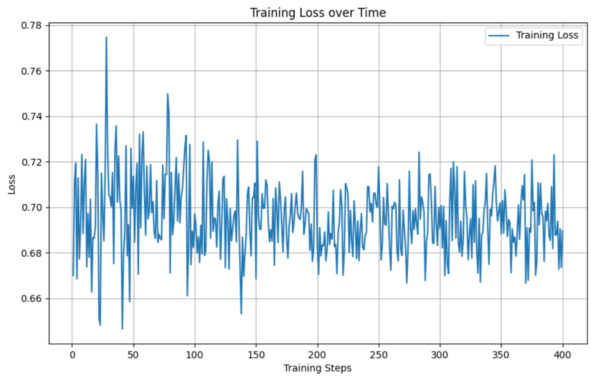
改进

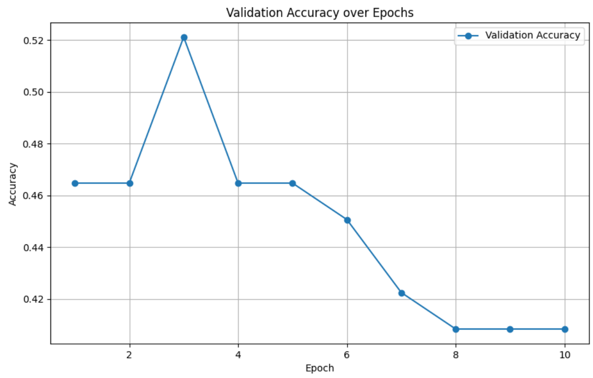
- 增加了warm up预热策略，设置warmup_ratio=0.2，在训练开始的 20% 时间内逐渐提高学习率至设定值。该预热策略可以在训练初期防止梯度爆炸，提高训练的稳定性和最终效果
- 学习率调度器改为余弦退火lr_scheduler_type="cosine"，这种调度方法可以帮助模型更快收敛，并在后期进一步稳定



对比之前在epoch快结束的时候可以观测到loss曲线有收敛的趋势

将epoch设置为10，得到的曲线如下





loss曲线虽然收敛，但是没有明显的下降趋势，valid的accuracy在epoch=3的时候达到峰值，之后逐渐降低，应该是出现了过拟合的现象，最高值是accuracy=0.52

尝试修改学习率和bs，也没有得到很好的效果，最高的accuracy只有0.573

因此我觉得可能的原因在于数据和模型，主要是亮点：

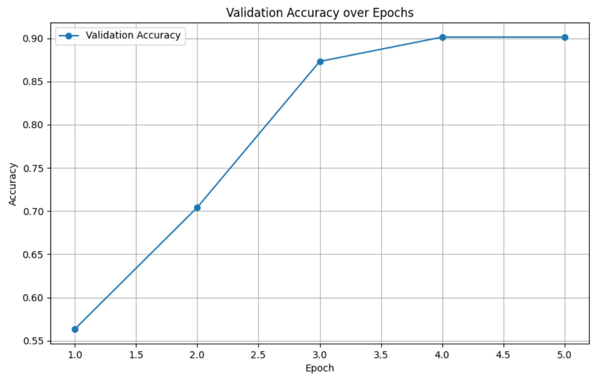
- 数据量太少或太局部以至于模型训练效果不好，无法泛化
- 模型本身性能不好

数据集的扩大较为困难，因此我尝试将模型更换

表格记录了使用的尝试过的模型以及在验证集的最好的效果

model	Best valid accuracy	bs	lr	epoch
prajjwal1/bert-tiny	0.573	16	2e-5	2
microsoft/deberta-v3-small	0.56	16	2e-5	3
albert/albert-base-v2	0.56	16	2e-5	4
xlnet/xlnet-base-cased	0.56	16	2e-5	3
microsoft/deberta-v3-large	0.91	16	2e-5	5

从表格中得出，使用 `microsoft/deberta-v3-large` 效果最好，训练5个epoch的loss曲线和验证集的accuracy如下：



测试结果：test accuracy score = 85.6

Score: 58.8

PRIMARY　　DIAGNOSTICS

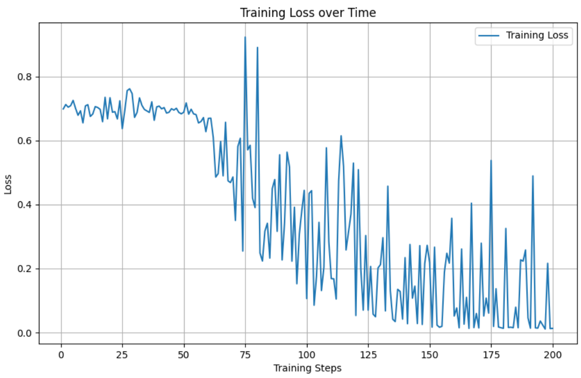
Task	Metric	Score
The Corpus of Linguistic Acceptability	Matthew's Corr	0.0
The Stanford Sentiment Treebank	Accuracy	80.0
Microsoft Research Paraphrase Corpus	F1 / Accuracy	81.5/73.4
Semantic Textual Similarity Benchmark	Pearson-Spearman Corr	61.2/59.1
Quora Question Pairs	F1 / Accuracy	51.4/79.1
MultiNLI Matched	Accuracy	56.0
MultiNLI Mismatched	Accuracy	56.4
Question NLI	Accuracy	50.4
Recognizing Textual Entailment	Accuracy	54.1
Winograd NLI	Accuracy	85.6
Diagnostics Main	Matthew's Corr	9.2

测试结果的链接：

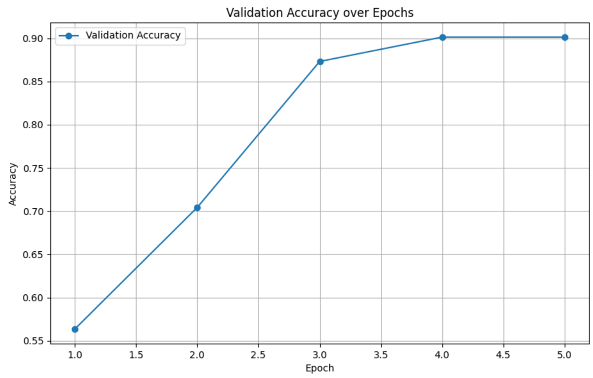
https://gluebenchmark.com/submission/uW4I3J9YPZRIaLXWwlipxvcl9dh1/-OACoSSW1POtzrjU_GMH

最后，我做了一组消融实验来比对学习率调度方式对模型性能的影响

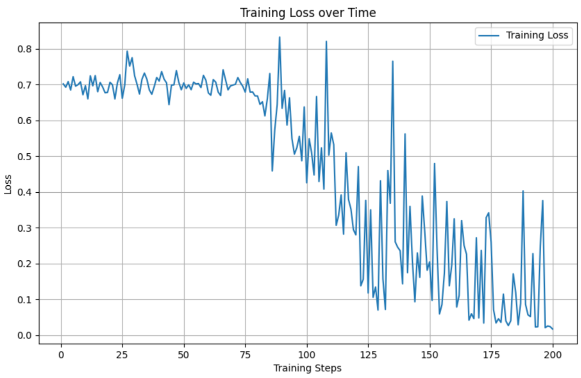
以下是实验结果，由上至下分别为，使用了cosine+warmup策略，使用了linear+warmup策略，只使用linear策略，这三种方式的模型训练曲线图，左图为loss曲线，右图为每个epoch的valid accuracy曲线



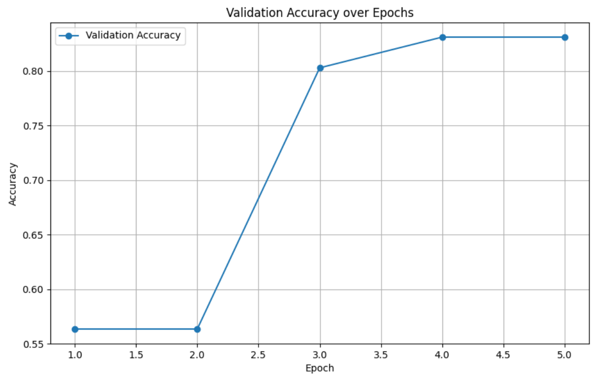
cosine+warm up



cosine+warmup

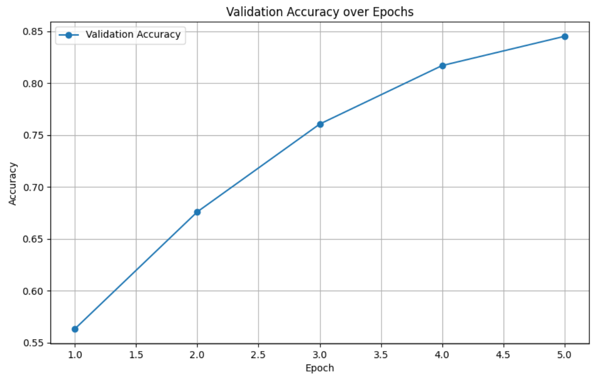


linear+warmup



linear+warmup

linear



linear

从上述图中可以看出，使用了warmup策略后模型训练可以更快的收敛，在第3个epoch的时候就已经达到了0.8以上的accuracy，使用了cosine策略之后模型的性能进一步提升，accuracy达到了0.9以上

总结

本次实验，我使用了warmup和cosine策略，使用了更换模型的方法，顺利完成了本次pj

以下贴一个config.json

```
{
```

```json
  "_name_or_path": "microsoft/deberta-v3-large",
  "architectures": [
    "DebertaV2ForSequenceClassification"
  ],
  "attention_probs_dropout_prob": 0.1,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 1024,
  "initializer_range": 0.02,
  "intermediate_size": 4096,
  "layer_norm_eps": 1e-07,
  "max_position_embeddings": 512,
  "max_relative_positions": -1,
  "model_type": "deberta-v2",
  "norm_rel_ebd": "layer_norm",
  "num_attention_heads": 16,
  "num_hidden_layers": 24,
  "pad_token_id": 0,
  "pooler_dropout": 0,
  "pooler_hidden_act": "gelu",
  "pooler_hidden_size": 1024,
  "pos_att_type": [
    "p2c",
    "c2p"
  ],
  "position_biased_input": false,
  "position_buckets": 256,
  "relative_attention": true,
  "share_att_key": true,
  "torch_dtype": "float32",
  "transformers_version": "4.45.2",
  "type_vocab_size": 0,
  "vocab_size": 128100
}

```

训练超参

- lr=2e-5
- epoch=5
- bs=16
- lr_scheduler_type="cosine"
- warmup_ratio=0.2
- gradient_accumulation_steps=1

test的结果链接

https://gluebenchmark.com/submission/uW4I3J9YPZRIaLXWwlipxvcl9dh1/-OACoSSW1POtzrjU_GMH

train使用的代码

```
import argparse

import evaluate
import numpy as np
from datasets import load_dataset
from transformers import (
    AutoModelForSequenceClassification,
    AutoTokenizer,
    EvalPrediction,
    Trainer,
    TrainingArguments,
)
import matplotlib.pyplot as plt

def main(args):
    dataset = load_dataset("SetFit/wnli")
    tokenizer = AutoTokenizer.from_pretrained(args.model, trust_remote_code=True)

    def preprocess_function(examples):
        return {
            **tokenizer(examples["text1"], examples["text2"]),
            "label": examples["label"],
        }

    dataset = dataset.map(preprocess_function, batched=True)
    print(tokenizer.decode(dataset["train"]["input_ids"][0]))

    model = AutoModelForSequenceClassification.from_pretrained(
        args.model,
        num_labels=2,
        trust_remote_code=True,
    )
    print(sum(p.numel() for p in model.parameters()))

    metric = evaluate.load("accuracy")

    def compute_metrics(p: EvalPrediction):
        preds = np.argmax(p.predictions, axis=-1)
        result = metric.compute(predictions=preds, references=p.label_ids)
```

```
        return result

    trainer = Trainer(
        model=model,
        tokenizer=tokenizer,
        train_dataset=dataset["train"],
        eval_dataset=dataset["validation"],
        compute_metrics=compute_metrics,
        args=TrainingArguments(
            output_dir=args.output,
            eval_strategy="epoch",
            save_strategy="epoch",
            logging_steps=1,
            learning_rate=args.lr,
            per_device_train_batch_size=args.bs,
            per_device_eval_batch_size=args.bs,
            gradient_accumulation_steps=args.accum,
            num_train_epochs=args.epoch,
            lr_scheduler_type="cosine",
            warmup_ratio=0.2,
        ),
    )
    trainer.train()

    # 提取损失值和准确率
    losses = [log["loss"] for log in trainer.state.log_history if "loss" in log]
    accuracies = [log["eval_accuracy"] for log in trainer.state.log_history if "eval_accuracy" in log]
    epochs = [log["epoch"] for log in trainer.state.log_history if "eval_accuracy" in log]

    # 绘制损失曲线并保存
    plt.figure(figsize=(10, 6))
    plt.plot(range(1, len(losses) + 1), losses, label="Training Loss")
    plt.xlabel("Training Steps")
    plt.ylabel("Loss")
    plt.title("Training Loss over Time")
    plt.legend()
    plt.grid()
    plt.savefig("training_loss_curve.png")

    # 绘制准确率曲线并保存
    plt.figure(figsize=(10, 6))
    plt.plot(epochs, accuracies, label="Validation Accuracy", marker='o')
    plt.xlabel("Epoch")
    plt.ylabel("Accuracy")
```

```
    plt.title("Validation Accuracy over Epochs")
    plt.legend()
    plt.grid()
    plt.savefig("validation_accuracy_curve.png")

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--model", type=str, default="prajjwal1/bert-tiny")
    parser.add_argument("--lr", type=float, default=2e-5)
    parser.add_argument("--epoch", type=int, default=1)
    parser.add_argument("--bs", type=int, default=1)
    parser.add_argument("--accum", type=int, default=1)
    parser.add_argument("--output", type=str, default="output")
    args = parser.parse_args()
    main(args)
```