人工智能实践作业二

傅申 PB20000051

目录

1. 实验内容

本次实验需要完成文本分类任务.

1.1. 数据集

本次实验提供了训练集和测试集, 均为 Excel 表格文件. 其中, 每个表格包含两栏, 第一栏为中文文本数据, 第二栏为标签 (包含 “0” 和 “1” 两类). 训练集中包含 1599 条样例, 测试集中包含 401 条样例. 数据集目录结构如下:

```
data
├── test.xlsx
└── train.xlsx
```

1.2. 实验要求

在样本不足的情况下完成对 Transformer 模型的训练, 尽可能正确分类文本.

2. 模型原理

本次实验选择了 HuggingFace Transformers 中的 `BertForSequenceClassification` 模型来进行文本分类任务. 该模型是在 BERT 模型的基础上对文本进行分类的.

2.1. Transformer 和 Attention [1]

Transformer 是一种采用自注意力机制 (Self Attention) 的深度学习模型, 由 Google 在 2017 年提出, 用于解决序列到序列 (Sequence to Sequence, Seq2Seq) 问题.

Self Attention 和 Multi-head Attention 是 Transformer 模型的核心. Self Attention 通过使用 Query, Key 和 Value 三个向量来计算输入序列中每个元素的输出, 从而实现对输入序列的理解. 具体而言, 对于输入序列 $X = \{x_1, x_2, ..., x_n\}$, 首先计算 Query, Key 和 Value 向量:

$$Q = XW_Q, K = XW_K, V = XW_V$$

然后计算 Attention 分数:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

这样, 就得到了输入序列的输出. 而 Multi-head Attention 是将多个 Self Attention 模块并行地进行计算, 并将其结果拼接在一起, 从而得到更好的结果. 图 1 展示了这两种机制的计算过程.

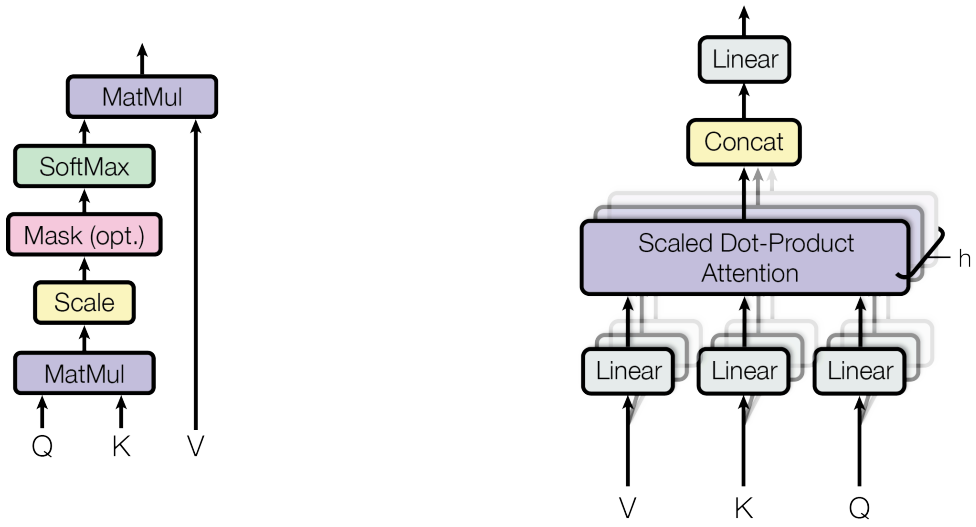

图 1 Self Attention (左) 和 Multi-head Attention (右)

Transformer 模型的结构如图 2 所示. 其中, Encoder 和 Decoder 分别由若干个 Encoder Layer 和 Decoder Layer 组成. Encoder Layer 和 Decoder Layer 的结构类似, 都是由 Multi-head Attention, Feed Forward 和 Layer Normalization 组成, 用于 Encoder 和 Decoder 的信息提取和传递. 不过, Decoder Layer 还包含一个 Masked Multi-head Attention 模块, 用于确保在预测时, 每个位置只能依赖于当前位置之前的信息, 从而避免模型在预测时使用未来信息.

2.2. BERT 模型 [2]

BERT 是 Google 在 2018 年提出的基于 Transformers 的预训练模型, 其全称为 Bidirectional Encoder Representations from Transformers. 该模型是一个双向的, Encoder-only 的 Transformer 模型, 用于解决 NLP 任务.

BERT 的训练过程如图 3 所示, 所示分为两个阶段: 预训练和微调. 预训练阶段使用无标签的数据进行, 通过 Masked Language Model (MLM) 和 Next Sentence Prediction (NSP) 两个任务, 学习语言的表示. 其中, MLM 任务是将输入序列中的某些词随机替换为 `[MASK]` 符号, 要求模型预测被替换

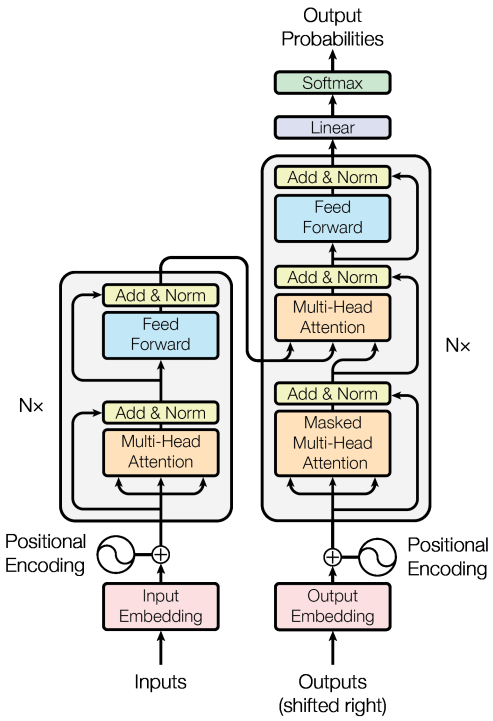

图 2 transformer 模型结构

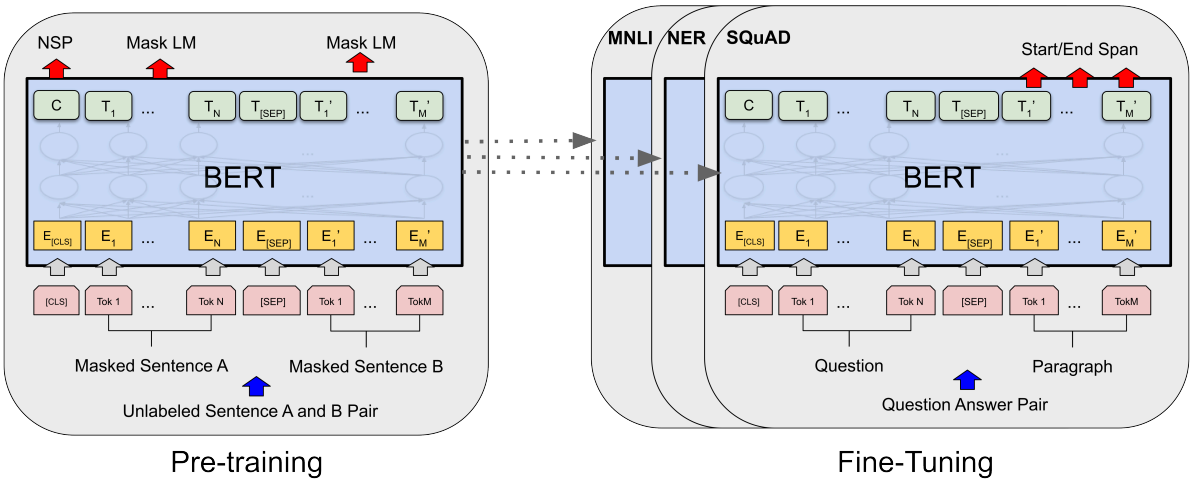

图 3 BERT 模型的训练过程

的词; NSP 任务是判断两个输入序列是否是连续的. 预训练完成后, 可以将模型用于微调, 用于解决特定的 NLP 任务.

2.3. 将 BERT 模型用于文本分类

通过将 BERT 模型的输出传入一个全连接层, 就可以将 BERT 模型用于文本分类任务. 下面的代码展示了 `BertForSequenceClassification` 模型的结构.

```
class BertForSequenceClassification:
    def __init__(self, num_classes: int, dropout: float):
        self.bert = BertModel()
        self.dropout = nn.Dropout(dropout)
        self.linear = nn.Linear(config.hidden_size, num_classes)

```

```
    def forward(self, input_ids: Tensor, ...):
      outputs = self.bert(input_ids, ...)
      pooled_output = outputs.pooler_output
      pooled_output = self.dropout(pooled_output)
      logits = self.linear(pooled_output)
      return logits
```

可以看到, 该模型将 BERT 的输出传入一个 Dropout 层, 然后再传入一个全连接层, 最后输出分类结果.

3. 实验过程

实验的代码位于 `code` 目录下, 目录结构如下:

```
code
├── arg_parser.py
├── data_loader.py
├── early_stopping.py
├── logger.py
├── main.py
├── models.py
└── procedures.py
```

实验中使用 PyTorch 框架来训练并测试模型.

3.1. 数据预处理

对于训练集和测试集中的数据, 由于它们都是文本, 无法直接作为 BERT 模型的输入, 因此需要进行预处理. 具体而言, 对于每个文本输出, 使用 Tokenizer 将其转换为 Token ID 序列 (并获得其他的信息), 然后对于较短/较长的序列, 进行 Padding/截断操作, 使得所有序列的长度相同.

对数据进行预处理的代码位于 `code/data_loader.py` 中, 该部分除了对每个文本进行上述的操作, 还将数据集组织成了 `torch.utils.data.DataLoader`, 方便后续的训练和测试. 对于训练数据集, 代码还将其划分为训练集和验证集, 以便于在训练过程中进行验证. 以加载测试集的代码为例, 如下所示:

```
def get_test_data_loader(
    tokenizer: PreTrainedTokenizerBase,
    batch_size: int = 32,
    max_length: int = 30,
) -> DataLoader:
    """
    Each batch is a dict with the following keys:
    - input_ids
    - token_type_ids
    - attention_mask
    - label
    """
    # load from xlsx
    if not os.path.exists(TEST_XLSX_PATH):
        raise FileNotFoundError(f"Cannot find {TEST_XLSX_PATH}")
    df = pd.read_excel(TEST_XLSX_PATH)
    # tokenize inputs
    encoded_inputs = tokenizer(
        df["数据"].tolist(),
```

```
        padding=True,
        truncation=True,
        max_length=max_length,
        return_attention_mask=True,
        return_tensors="pt",
    )
    encoded_inputs["label"] = torch.tensor(df["标签"].tolist())
    # generate datasets
    dataset = Dataset.from_dict(encoded_inputs).with_format("torch")
    # data loaders
    data_loader = DataLoader(dataset, batch_size=batch_size, shuffle=False)
    return data_loader
```

3.2. 模型训练

本次实验使用的 `BertForSequenceClassification` 是基于 bert-base-chinese 预训练模型的, 因此在训练之前需要加载预训练模型的参数. 代码如下 (位于 `code/models.py`):

```
def new_model(
    num_labels: int, device: torch.device
) -> tuple[BertForSequenceClassification, PreTrainedTokenizerBase]:
    model = BertForSequenceClassification.from_pretrained(
        "bert-base-chinese", num_labels=num_labels
    ).to(device)
    tokenizer = BertTokenizerFast.from_pretrained("bert-base-chinese")
    return model, tokenizer
```

对模型进行训练和测试的代码位于 `code/procedures.py` 中. 在训练过程中, 我选择了 `torch.optim.AdamW` 优化器来优化模型的参数. 在优化器的设置中, `bias`, `LayerNorm.weight` 和 `LayerNorm.bias` 这三类参数不进行权重衰减. 同时, 我还使用了 Early Stopping 策略, 当验证集上的损失连续多次没有下降时, 可以认为模型已经收敛, 因此可以提前结束训练.

训练部分的代码如下:

```
for epoch in range(n_epochs):
    train_loss, train_acc = __train_one_epoch(
        model, data_loaders.train, optimizer, device
    )
    train_losses.append(train_loss)

    val_loss, val_acc = __val_one_epoch(model, data_loaders.val, device)
    val_losses.append(val_loss)
    # early stopping
    if early_stopping is None:
        continue
    early_stopping(val_loss, model)
    if early_stopping.early_stop:
        LOGGER.info("Early stopping")
        break
```

3.3. 主程序

本实验代码的主程序为 `code/main.py`, 它会解析命令行参数, 并根据命令行参数决定执行的步骤. 命令行参数的部分可以查看 `code/arg_parser.py` 或运行 `python main.py --help`, 这里不再展开.

4. 实验结果

4.1. 实验环境

本次实验使用 BitaHub 平台的 GPU 和镜像环境对模型进行训练和测试. 具体的软硬件环境如下:

- GPU: NVIDIA Tesla V100
  - 驱动版本 470.63.01
  - CUDA 版本为 11.8
- 软件环境
  - Ubuntu 20.04.6 LTS (Docker 镜像)
  - Python 3.10
  - PyTorch 2.1.0
  - HuggingFace Transformers 4.36.2

4.2. 训练过程

由于使用的是预训练模型, 并且启用了 Early Stopping 策略, 模型在较少的 epoch 之后就收敛了. 训练过程中的 loss 如图 4 所示.

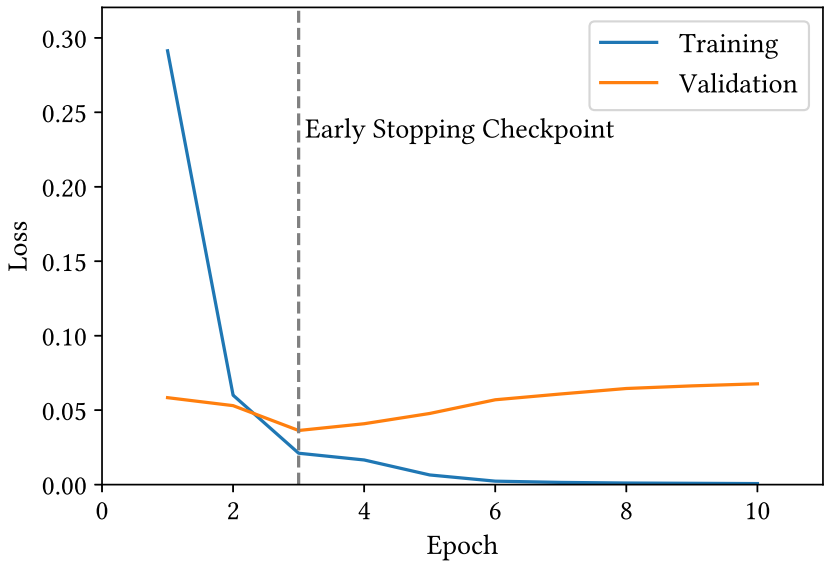

图 4 训练过程中的 loss 曲线

可以看到, 随着训练的不断进行, 模型在训练集上的 loss 不断下降, 最后达到一个很小的值. 但是第三个 epoch 之后, 模型在测试集上的 loss 不降反升, 说明模型已经过拟合了. 这时, Early Stopping 策略介入, 提前结束训练, 并选择第三个 epoch 的模型 (checkpoint) 作为最终的模型.

4.3. 测试结果

在测试集上对模型进行测试, 得到模型在测试集上的 loss 为 0.08010, 准确率为 98.50%. 这说明模型在测试集上的表现很好, 能够很好地进行文本分类.

参考文献

1. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., Polosukhin, I.: Attention Is All You Need, (2023)

2. Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, (2019)