

**序列标注编程作业：**

**基于Transformer的命名体识别**

**自然语言处理课程实验报告**

班级：110

姓名：耿翊中

学号：2021213382

2024年5月

1. 实验要求

简述本次作业的目的和要求，包括采用 Transformer 架构的命名实体识别的具体任务。

强调使用 BERT 模型的选择理由。

2. 方法

2.1 数据集描述

描述数据集的组成，包括训练集、发展集和测试集的细节。

统计并列出不同实体标签及其分布。

2.2 模型架构

详细介绍 BERT 模型的架构和如何适应命名实体识别任务。

讨论使用的模型变种（如 BERT base 或 BERT large）。

2.3 字向量初始化

描述字向量的来源（如随机初始化或预训练词向量）和向量的维数。

2.4 训练细节

详述训练算法、学习率、批次大小、训练轮数等。

如适用，讨论任何特殊的训练技巧或策略（如学习率调度、正则化策略等）。

3. 实验

3.1 训练过程

描述模型的训练过程，包括监控的损失函数和性能指标。

展示训练损失和发展集性能（准确率）的变化曲线。

3.2 模型评估

介绍在测试集上的性能评估方法和结果。

可能包括准确率、召回率和 F1 分数等指标。

3.3 结果讨论

分析和讨论实验结果，包括模型的强项和弱点。

讨论可能影响性能的因素，如模型架构的选择、数据的质量和训练过程的特定方面。

4. 结论与未来工作

总结实验结果和学到的教训。

探讨可能的改进方向和未来的研究工作。

# 实验要求

作业内容：

基于 train.txt 和 train\_TAG.txt 数据训练一个以 Transformer 为基础结构的命名实体识别模型，进而为 test.txt 进行序列标注，输出标签文件，标签文件输出格式与 train\_TAG.txt 相同。

即保持 test.txt 中的行次序、分行信息以及行内次序，行内每个字的标签之间用空格分隔。

输出文件命名方式：学号.txt。

关于模型设计：

模型必须以 Transformer 为基础结构，可以单独采用 Transformer 编码器(典型的如 BERT)， 也可以尝试单独采用 Transformer 解码器(典型的如 GPT)，或者编码器+解码器的组合(典型的如 T5)。Transformer 的层数等超参数自选，字向量可以随机初始化后训练，也可以采用已 有开放词向量作为初始值，维度自选。是否采用类 CRF 模块建模序标关联可选。

所有输出文本均采用 Unicode(UTF-8)编码、算法采用 Python (3.0 以上版本) 实现

此次作业需要提交的材料：

1. 基于 Transformer 结构的序标模型和训练算法的文本说明，提交 doc(或 pdf)文本说明至少包括：

a)给出标注集(标签集)。序列标注的标签集是 train\_TAG.txt 中的所有不同的标签组成的集合，请自行统计获得，标签集是后续训练模型和标注的基础，请注意统计完整。

b)对模型和执行细节进行说明。模型和执行细节应至少包含：模型的结构及其结构参数(如用什么 transformer、堆叠层数、输入 token 数等等)、所用初始字向量的来源、向量维数、训练算法以及学习率、训练批次大小、训练轮数等；

c)给出训练损失和发展集性能随时间变化的曲线：每轮记录训练 loss，同时每一轮在发展集上进行测试，获得其标注性能(准确率)。给出所选择用来进行测试的训练轮次。

d)参考文献。给出参考的论文、网站、代码链接等等，说明是否使用了大模型辅助，辅助程

度如何。

1. 提交完整的实现代码，其中关键部分需要进行注释说明：与文本说明中的参数和执行细节对应。
2. 对 test.txt 进行序列标注得到的标注文件

提交 txt 文件，文件命名方式：学号.txt；

注：评分时 1 占 40%、2 占 30%、3 占 30%

# 方法（Method)

## 2.1数据预处理

### 2.1.1数据概况

本实验使用的语料库包括训练集、发展集和测试集，主要用于训练和评估基于 Transformer 的命名实体识别（NER）模型。

训练集：包括 train.txt 和 train\_TAG.txt。train.txt 文件由字或符号序列组成，每个字或符号作为一个单独的实体，例如“人 民 网 1 月 1 日 讯 据 《 纽 约 时 报 》 报 道 ，”。对应的 train\_TAG.txt 文件包含了每个字或符号的实体标签，标签示例如“O O O B\_T I\_T I\_T I\_T O O O B\_LOC I\_LOC O O O O O O”，其中“O”表示该字或符号不是命名实体的一部分，“B\_T”表示一个时间实体的开始，“I\_T”表示时间实体的中间部分，“B\_LOC”和“I\_LOC”分别标记地点实体的开始和中间部分。

发展集：由 dev.txt 和 dev\_TAG.txt 组成，结构与训练集相同。发展集用于模型训练过程中的性能调整和模型选择，以确保模型在未见数据上也能表现良好。

测试集：test.txt，用于最终评估模型的性能。测试集的结构类似于训练集，但不包含标签，模型需要预测这些数据的实体标签。

### 2.1.2读取和处理数据

这部分的函数均在utils.py中实现

1. 读取标签集：函数 **read\_tags** 被用来从标签文件中读取唯一的标签集合。这是通过读取每一行的标签，使用 split() 方法将其分割开，并将分割后的标签添加到一个集合中，从而确保标签的唯一性。这一步骤帮助我们了解所有可能的实体标签，为模型输出层的设计提供指导。

创建标签索引：**create\_tag\_to\_ix** 函数生成一个从标签到索引的映射，这对于模型的训练阶段将标签转换为可处理的数值格式是必要的。

最终得到标签集：

1. tag\_to\_ix = {'I\_LOC': 0, 'B\_LOC': 1, 'I\_T': 2, 'O': 3, 'I\_ORG': 4, 'B\_T': 5, 'I\_PER': 6, 'B\_PER': 7, 'B\_ORG': 8}

2. 读取训练和发展数据：**read\_data** 函数用于读取文本文件和相应的标签文件。

文本行和标签行被同时读取并按空格分割。

接着，通过迭代每一行的单词和对应标签，我们构造了句子和相应的标签序列。

对于bert模型，有句子长度限制，所以我选择了用标点符号对一行中的文本进行切分，并再utils.py中作验证，确保训练，验证，测试三个文件的文本句子经过处理后都没有大于max\_length=128的句子。

具体实现方式是在句子结束标识符（如逗号、句号等）出现时，将当前句子及其标签添加到列表中。如果行结束时仍有剩余单词，这些单词被视为一个新的句子。

1. **for** text\_line, tag\_line **in** zip(text\_lines, tag\_lines):
2. sentence = []
3. sentence\_tags = []
4. **for** word, tag **in** zip(text\_line, tag\_line):
5. sentence.append(word)
6. sentence\_tags.append(tag)
7. **if** word **in** {',', '。', '、', ';', ':', '('}:
8. texts.append(sentence)
9. tags.append(sentence\_tags)
10. sentence = []
11. sentence\_tags = []

3.读取测试数据：通过**read\_test\_data**函数实现。与读取训练数据类似，但不涉及标签的读取。此函数只是简单地将测试文本分割成单词并按句子进行组织。

### 2.1.3构建数据类

此类函数在dataset.py文件中实现

为了支持命名实体识别任务，设计并实现了一个自定义数据集类 **NERDataset**，该类继承自 PyTorch 的 Dataset 类。NERDataset 类的主要职责是处理文本数据和相应的标签，并将文本通过 tokenizer 转换成模型可处理的格式。

主要功能:

初始化: 接受文本列表、标签列表和 tokenizer。每个文本输入通过 tokenizer 转换成模型需要的格式，包括输入 ID 和注意力掩码。

样本获取: 通过索引获取特定的样本，其中包括处理后的输入 ID、注意力掩码和相应的标签。

数据处理:

文本和标签被转换为固定的最大长度，超出部分进行截断，不足部分用填充处理。

标签同样经过转换，以适应模型的训练需求，非标签部分用特殊值（-100）填充以在训练时忽略。

## 2.2模型结构（bert-base-chinese)

此模型BertForNer在model.py文件中实现

开始实验时，想到的就是用预训练的bert-base-chinese模型加一个线性层来完成这样一个任务。所以选用了预训练的bert-base-chinese，调用Transformers库中的AutoModel方法直接导入bert模型来完成任务。

1. 结构参数：

堆叠层数：BERT-base-Chinese 模型包含 12 层 Transformer 堆叠。

隐藏层大小：每个隐藏层具有 768 个隐藏单元。

自注意力头数：模型在每个 Transformer 层中配置了 12 个自注意力头。

输入 token 数：设置为128

1. 初始字向量的来源与向量维数

字向量来源：在 BERT-base-Chinese 中，字向量是通过在大规模中文文本上预训练得到的。这些向量能够捕捉字词间的语义和句法关系，为下游任务如命名实体识别提供了有用的先验知识。

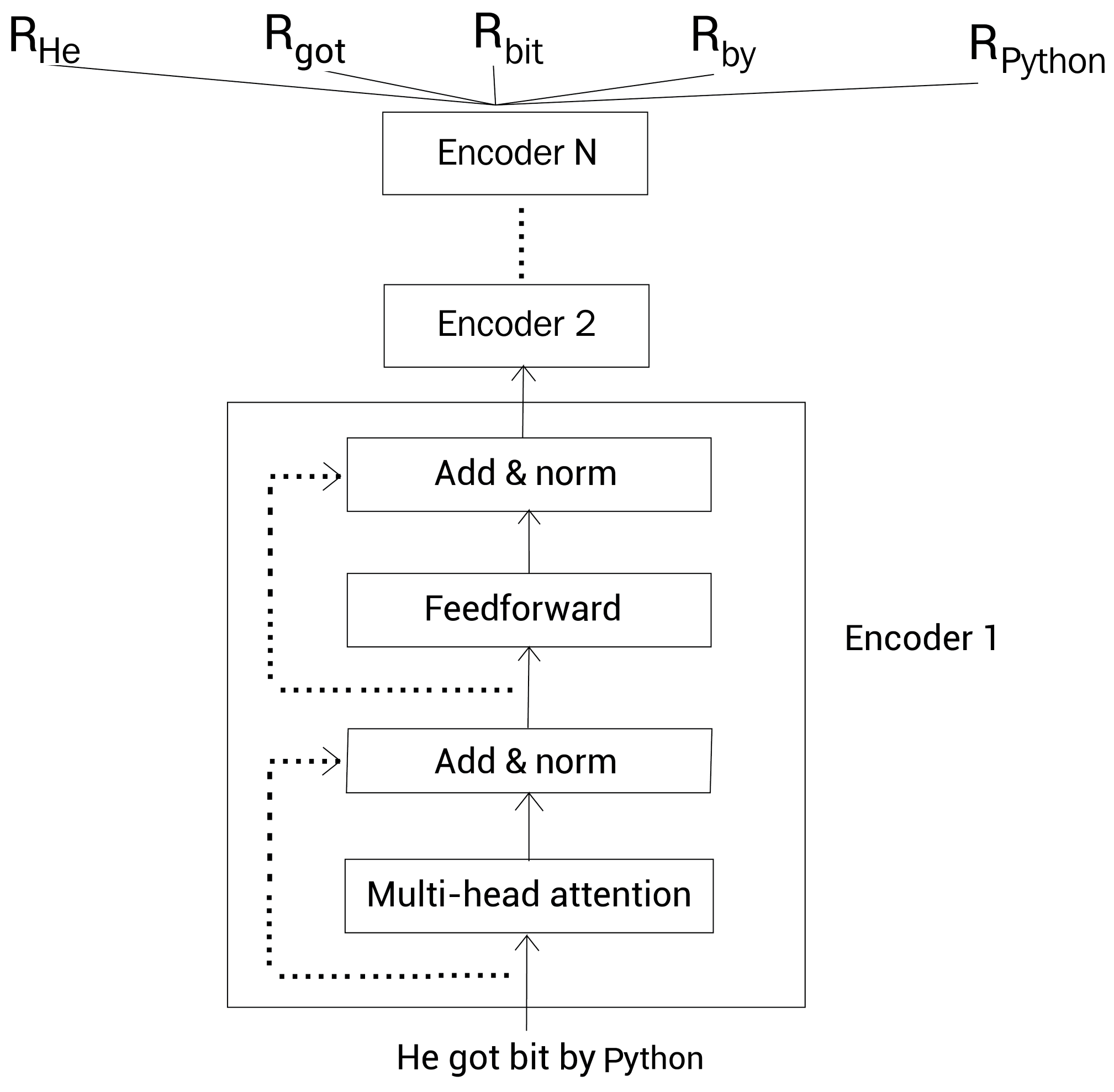
向量维数：每个字的向量维度为 768，这与模型的隐藏层大小一致。

## 2.3模型结构（自实现bert）

此模型代码在self\_bert\_model.py文件中实现

在调用bert-base-chinese后，我又尝试根据bert的原理结构，自己构建bert架构来完成任务。

bert的基本架构包括嵌入层、位置嵌入层、多个Transformer编码器层和输出分类器。每个编码器层由多头注意力和前馈网络组成，每层的输出都经过层归一化。



根据上述架构，要实现bert模型，首先要实现Encoder结构，所以实现了两个Encoder的组成类：

多头注意力模块 (**EnhancedMultiHeadAttention**): 本模块利用多个注意力头并行处理信息，允许模型在不同的表示子空间中学习信息。每个头部分别学习输入的不同方面，然后通过一个线性层整合这些信息。

前馈网络 (**FeedForwardLayer**): 每个Transformer编码器层包含一个前馈网络，该网络由两个线性变换组成，中间加入ReLU激活函数和Dropout，增强模型的非线性表达能力和泛化性。

通过这两个类，组合实现编码器层**TransformerEncoderLayer：**应用增强多头注意力和前馈网络，通过层归一化和残差连接增强训练稳定性。

而后通过Encoder组合成基础的bert模型结构，在这里还要另外实现位置嵌入层，另外的词嵌入层已经有nn.embedding函数来实现了。

词位编码模块 (**TokenPositionalEncoding**): 该模块生成正弦波形的位置编码，帮助模型理解词元在序列中的相对位置。通过使用正弦和余弦函数的组合来为每个位置生成唯一的编码，这是Transformer模型的一个关键特性。

最终组成成为**BertModel**类，可以自定义bert模型的具体参数

1. **class** BertModel(nn.Module):
2. """基于BERT的语言模型，适用于多种NLP任务。
4. 包括词嵌入层、位置嵌入层、片段嵌入层和多个编码器层。
5. """
6. **def** \_\_init\_\_(self, n\_layers, d\_model, d\_ff, n\_heads,
7. max\_seq\_len, vocab\_size, pad\_id, dropout=0.1):
8. super(BertModel, self).\_\_init\_\_()
9. self.model\_dim = d\_model
10. self.pad\_token\_id = pad\_id
11. self.token\_embedding = nn.Embedding(vocab\_size, d\_model, padding\_idx=pad\_id)
12. self.positional\_encoding = TokenPositionalEncoding(d\_model, max\_seq\_len)
13. self.segment\_embedding = nn.Embedding(2, d\_model, padding\_idx=0)
14. self.embedding\_dropout = nn.Dropout(dropout)
15. self.encoder\_layers = nn.ModuleList(
16. [TransformerEncoderLayer(d\_model, d\_ff, n\_heads, dropout) **for** \_ **in** range(n\_layers)])
18. **def** forward(self, input\_ids, token\_type\_ids, mask, return\_attention=False):
19. """模型前向传播，支持返回注意力权重。
21. 参数:
22. input\_ids (Tensor): 输入的词ID。
23. token\_type\_ids (Tensor): 词的类型ID。
24. mask (Tensor): 输入的掩码。
25. return\_attention (bool): 是否返回注意力权重。
26. """
27. output = self.token\_embedding(input\_ids) + self.positional\_encoding(input\_ids) + self.segment\_embedding(token\_type\_ids)
28. output = self.embedding\_dropout(output)
29. attention\_weights = []
30. **for** layer **in** self.encoder\_layers:
31. output, attention = layer(output, mask)
32. **if** return\_attention:
33. attention\_weights.append(attention)
35. output = (output, attention\_weights) **if** return\_attention **else** output
36. **return** output
37. 结构参数：

堆叠层数：可自定义有多少层Transformer堆叠，后续实验中选择为2层

隐藏层大小：可自定义，后续实验中选择每个隐藏层具有 768 个隐藏单元。

自注意力头数：可自定义，后续实验中选择模型在每个 Transformer 层中配置 6 个自注意力头。

输入 token 数：设置为128

1. 初始字向量的来源与向量维数

字向量来源：在 BERT-base-Chinese 中，字向量是通过在大规模中文文本上预训练得到的。这些向量能够捕捉字词间的语义和句法关系，为下游任务如命名实体识别提供了有用的先验知识。

向量维数：每个字的向量维度为 768

## 2.4训练细节

训练部分详细代码可以在train\_self\_model.py文件中找到

1. 训练算法

优化器：模型使用 AdamW 优化器，一种基于梯度下降的优化算法，适用于权重衰减正则化，特别是在大规模数据集和参数量较大的模型中。AdamW 通过调整权重衰减方式，有助于控制模型在训练过程中的过拟合。

2. 学习率

初始学习率：设置为 5e-5。通过调查，这是一个常用的学习率，对于BERT类模型的微调是标准的选择，因为它足够小，可以精细调整预训练模型的权重，避免在训练初期造成过大扰动。

3. 批次大小和训练轮数

批次大小：设置为 4096

训练轮数：设置为进行 20 轮训练。

4. 正则化策略

Dropout：模型中使用了 0.1 的dropout率，用于减少复杂模型中的过拟合。通过在训练过程中随机丢弃部分网络连接，dropout迫使模型学习更加鲁棒的特征。

5. 模型评估和保存：

模型评估：在每个训练周期后，通过开发集来评估模型的性能。

模型保存：每个周期结束后，都将当前模型的状态保存下来，这不仅方便随时恢复训练，也便于选择最优模型进行最终的评估和部署。

# 实验

## 3.1训练过程