# 某个小组　情感分析大作业报告

祝昊焜 52001910058，郑航 520021911347，闫泉林 520021910226

2022 年 12 月 15 日

## 目录

| 不同版本的准确率对比 | | |
|---|---|---|
| version | ACU | |
| | 中文 | 英文 |
| 1 | 86.03 | |
| 2 | 84.8 | |
| 3 | 83.35 | 92.65 |
| 4 | 86.62 | 95.14 |

图 1: Accuracy

# 1 任务

本次大作业完成的是 Positive and Negative Sentimental Analysis 二元情感分析问题。文本情感分析在社交媒体、舆情监测等领域有着广泛的应用，如商品评价正负面的分析、公司网络评价正负面的监测等。因此，实现高效准确的情感分析具有现实意义。

目前，业界的情感分析任务包括词级别的情感分析、句子或篇章级别的情感分析，目标级的情感分析（分析针对某个对象的情感观点）等，通常会聚焦于特定的领域，且情感分析的细分维度较多，通常不止进行简单的二分类。

本次实验任务具体为，设计并训练模型，使得给定一份文本，模型能对文本的情感倾向（积极或消极）进行二元的预测分类。给定的训练集中，中英文语料各有积极和消极的评价 5000 条，共 20000 条数据，数据量偏小且语料领域较为单一。任务的评测标准为模型在测试集上进行预测的 Accuracy 正确率。

# 2 信息收集

## 2.1 Transformer 模型

过往主要的序列转换模型是基于复杂的循环或卷积神经网络，其中包括一个编码器和一个解码器。性能最好的模型还通过一种注意机制将编码器和解码器连接起来。但是在Vaswani et al., “Attention is All you Need”一文中提出了剔除递归和卷积神经网络而完全基于注意力机制的网络架构 Transformer。实验表明 Transformer 在质量上更优越，同时更并行，需要的训练时间明显更少。

### 2.1.1 编码器与解码器结构

**编码器**由六个相同的层组成，每个层内有两个子层：

- 第一个子层是 multi-head self-attention mechanism 多头自注意力层，用来计算输入的自注意力。
- 第二个子层是简单的基于位置的全连接网络。

对于每一个子层，都采用残差连接，即每个子层的输出都是 $LayerNorm(x + Sub - layer(x))$，其中所有子层的输入与输出都与 embedding layer 输出维度相同为 512 维。

**解码器**也由六个相同的层构成，每层包括 3 个子层：

- 第一个子层是带遮罩的多头自注意力层，也是用于计算输入的自注意力，但是因为在生成过程中的时刻 i 时，应该只考虑 i 之前的输入，所以要加上一个遮罩 Mask。
- 第二个子层是一个基于位置的全连接网络。
- 第三个子层是将编码器的输出作为键值而将解码器第二个子层的输出作为查询计算注意力。

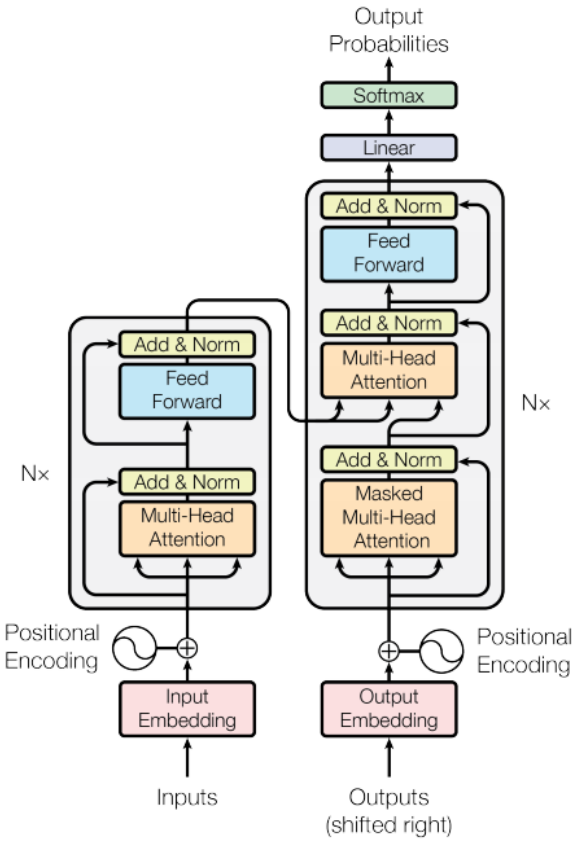


图 2: Transformer 模型结构

### 2.1.2 注意力机制

**归一化的点积注意力**

在 Transformer 中使用的是注意力是 Scaled Dot-Product Attention, 即归一化的点积注意力，假设输入的查询 q，键的维度 $d_k$，值的维度 $d_v$，那么就计算查询与每个键的点乘操作，并除以 $\sqrt{d_k}$，在应用 Softmax 函数计算权重。

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{1}$$

**多头注意力**

在实践中，当给定相同的查询、键和值的集合时，我们希望模型可以基于相同的注意力机制学习到不同的行为，然后将不同的行为作为知识组合起来，捕获序列内各种范围的依赖关系（例如，短距离依赖和长距离依赖关系）。因此，允许注意力机制组合使用查询、键和值的不同子空间表示（representation subspaces）可能是有益的。

为此，与其只使用单独一个注意力汇聚，我们可以用独立学习得到的 $h$ 组不同的线性投影（linear projections）来变换查询、键和值。然后，这 $h$ 组变换后的查询、键和值将并行地送到注意力汇聚中。最后，将这 $h$ 个注意力汇聚的输出拼接在一起，并且通过另一个可以学习的线性投影进行变换，以产生最终输出。这种设计被称为多头注意力（multihead attention）。

在 Transformer 模型中仅仅计算一次注意力是不够的，因此提出了多头注意力机制，操作如下：

- 首先对 Q、K、V 做一次线性映射，将输入维度均为 $d_{\text{model}}$ 的 Q, K, V 矩阵映射到 $Q \in R^{m\times d_k}, K \in R^{m\times d_k}, V \in R^{m\times d_v}$
- 然后在采用 Scaled Dot-Product Attention 计算出结果
- 多次进行上述两步操作，然后将得到的结果进行合并

- 将合并的结果进行线性变换

总结来说，计算公式如下：

$$\text{Attention}(Q,K,V) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O \quad \text{head}_i \text{ head}_i \text{ head}_i \text{ head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \quad (2)$$

其中第 1 步的线性变换参数为 $W_i^Q \in R^{d_{\text{model}} \times d_k}, W_i^K \in R^{d_{\text{model}} \times d_k}$ $W_i^V \in R^{d_{\text{model}} \times d_v}$ 4 $W^O \in R^{hd_v \times d_{\text{model}}}$ . 而第三步计算的次数是 h 。

在Vaswani et al., "Attention is All you Need"中取 $d_{\text{model}} = 512$, 表示每个时刻的输入维度和输出维度，h=8 表示 8 次 Attention 操作，$d_k = d_v = \frac{d_{\text{model}}}{h} = 64$ 表示经过线性变换之后、进行 Attention 操作之前的维度。那么进行一次 Attention 之后输出的矩阵维度是 $R^{m \times d_v} = R^{m \times 64}$, 然后进行 h = 8 次操作合并之后输出的结果是 $R^{m \times (h \times d_v)} = R^{m \times 512}$ ，因此输入和输出的矩阵维度相同。这样输出的矩阵 $R^{m \times 512}$ ，每行的向量都是对 V 向量中每一行 $v_i$ 的加权，示意图如下所示

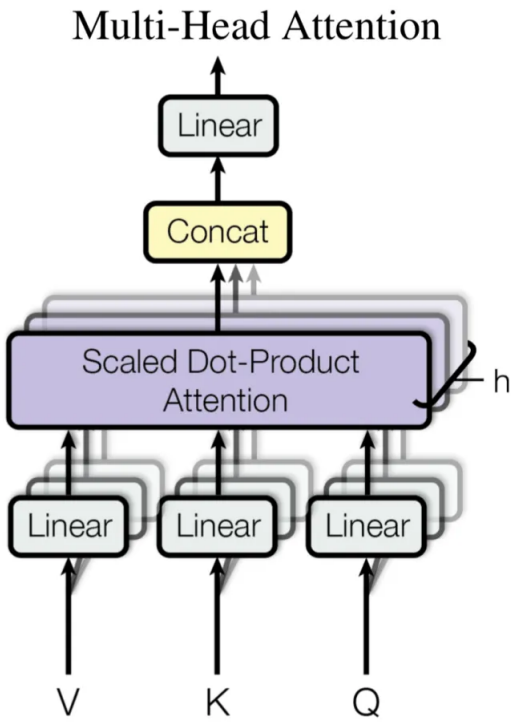


图 3: 多头注意力机制示意图

## 2.2 Bert 模型

在Devlin et al., "Bert: Pre-training of deep bidirectional transformers for language understanding"一文中提出了 Bert 预训练模型。BERT，即 Bidirectional Encoder Representations from Transformers，是一个基于 transformer 的 encoder 模块的预训练语言模型，其预训练来自于无标记的文本，双向指的是可以联合利用左右两个方向的全部文本信息。BERT 在一系列的 NLP 任务上都表现得很好，在 11 个自然语言处理任务上达到了 SOTA，如将 GLUE 分数提高至 80.5%。

### 2.2.1 Bert 的使用框架

BERT 的使用主要分为两步：预训练（pre-training）和微调（fine-tuning）。预训练阶段就是使用大量的无标记文本对模型参数进行训练，微调阶段就是根据不同的任务，在预训练完成的模型上使用新任务的一些带标记数据进行训练，对参数进行微调。

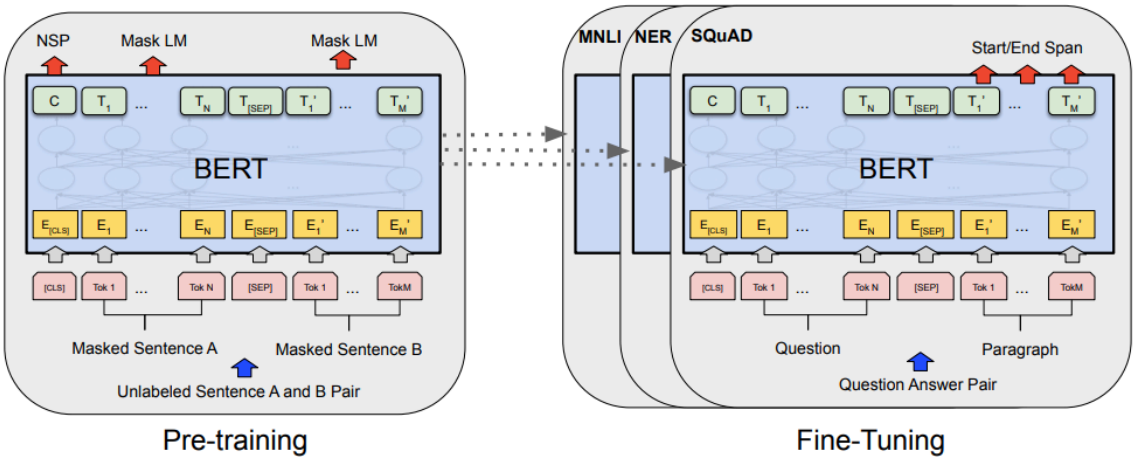


图 4: Bert 的使用框架

### 2.2.2 Bert 的模型结构

BERT 的模型结构是一个多层的、双向的 Transformer 的 Encoder 模块，其 Encoder 的实现与 Transformer 中的原始实现是几乎完全一致的。

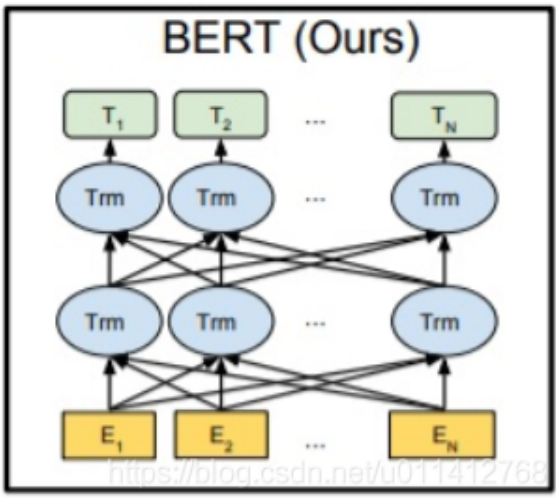


图 5: Bert 的模型结构

### 2.2.3 Bert 的输入输出表示

为了可以适用于多样化的下游任务，BERT 的输入序列既可以是一个句子，也可以是一个用特殊符号分开的句子对。BERT 中序列的一个 token 的表示，最终可以由三部分组成：

- BERT 使用词表大小为 30000 的 Wordpiece 词嵌入模型，每个 token 会有一个词嵌入表示
- 为了表示一个 token 是在句子对的句子 A 中还是句子 B 中，BERT 使用一个名为 segment embedding 的编码方式进行表示
- 为了表示一个 token 在序列中的位置，BERT 还使用了类似 Transformer 中的位置编码（position embedding）方式总的输入表示即为以上三个部分的加和，可以同时包含各部分的信息

总的输入表示即为以上三个部分的加和，可以同时包含各部分的信息

#### 2.2.4 Bert 的预训练

BERT 并不使用传统的语言模型进行预训练，而是通过在两个特定任务上进行无监督学习来完成，这两个任务分别是掩盖词预测（Masked LM，MLM）和下一句预测（Next Sentence Prediction，NSP）

#### 2.2.5 Bert 的微调

每个下游任务中，只需要设计好任务的输入输出表示，输入 BERT，然后据此对 BERT 所有参数进行微调即可。

BERT 是第一个大型的基于微调的模型，其出现简化了后续很多 NLP 任务（句子级别的或词级别的）的训练，并提升了其性能，使得自然语言处理在 BERT 出现后的几年有了一个质的飞跃。

# 3 方法

## 3.1 Bert 微调

本次情感分析的任务在本质上就是对文本进行二分类，根据Devlin et al., “Bert: Pre-training of deep bidirectional transformers for language understanding”一文提出的 Bert 模型，我们在此 Bert 预训练模型上进行微调来完成本次情感分析的下游任务。

本次项目中，模型主要由 Bert 预训练模型和在 Bert 模型后添加的一个由全连接层构成的分类器网络组成。网络架构如图6所示。

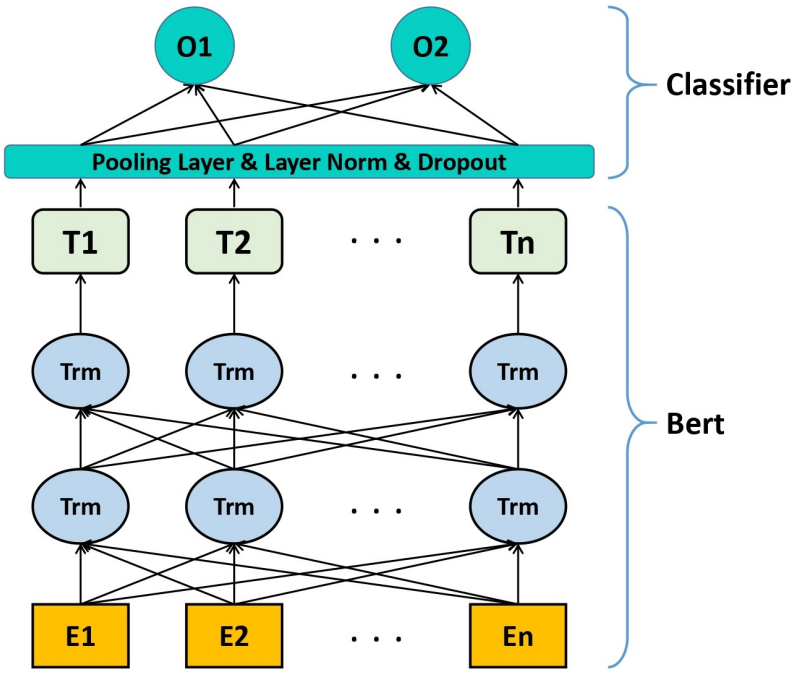


图 6: Model Architecture

## 3.2 Weight Decay(L2 正则化)

由于考虑到此次 Proj 中情感分析训练数据集较小，如果中文英文合并训练则数据集大小为 2 万条数据，如果分开训练，则各自只有 1 万条数据，这样的数据量对于 Bert 模型而言是 few-shot 的，所以需要考虑采用 weight-decay（L2 正则化）来防止在 fine-tuning 过程中模型迅速过拟合。

预训练的权重衰减 weight-decay（WD）是一种常见的正则化技术 (Krogh and Hertz, "A Simple Weight Decay Can Improve Generalization")。在每次优化迭代中，从模型参数中减去 $\lambda \mathbf{w}$，其中 是正则化的超参数，$\mathbf{w}$ 是模型参数。预先训练的权重衰减将这种方法用于微调预先训练的模型 (Chelba and Acero, "Adaptation of maximum entropy capitalizer: Little data can help a lot")，通过从目标中减去 $\lambda(\mathbf{w}-\hat{\mathbf{w}})$，其中 $\hat{\mathbf{w}}$ 是预先训练的参数。Lee, Cho, and Kang, "Mixout: Effective regularization to finetune large-scale pretrained language models"的经验表明，在 BERT 微调中，预先训练的权重衰减比传统的权重衰减效果更好，并且能够稳定微调。

在我们的代码中，我们通过 transformer 库实现的 AdamW 优化器，并将其中参数`weight_decay` 设为 0.01，进而实现 weight-decay。

### 3.3 权重初始化

用 bert 做 finetune 时，通常会直接使用 bert 的预训练模型权重，去初始化下游任务中的模型参数，这样做是为了充分利用 bert 在预训练过程中学习到的语言知识，将其能够迁移到下游任务的学习当中。以 bert-base 为例，由 12 层的 transformer block 堆叠而成。其中底部的层也就是靠近输入的层，学到的是通用语义信息，比如词性、词法等语言学知识，而靠近顶部的层也就是靠近输出的层，会倾向于学习到接近下游任务的知识，拿预训练任务来说，就是 masked word prediction、next sentence prediction 任务的知识。所以 finetune 时，可以保留底部的 bert 权重，对于顶部层的权重（1 6 layers）可以重新进行随机初始化，让这部分参数在下游任务上进行重新学习。Zhang et al., "Revisiting Few-sample BERT Fine-tuning"通过实验证明，采取重新初始化部分层参数的方法，在一部分任务上，指标获得了一些明显提升。实验结果如图7所示。

| Dataset | RTE | | MRPC | | STS-B | | CoLA | |
|---|---|---|---|---|---|---|---|---|
| | 3 Epochs | Longer | 3 Epochs | Longer | 3 Epochs | Longer | 3 Epochs | Longer |
| Standard | $69.5 \pm 2.5$ | $72.3 \pm 1.9$ | $90.8 \pm 1.3$ | $90.5 \pm 1.5$ | $89.0 \pm 0.6$ | $89.6 \pm 0.3$ | $63.0 \pm 1.5$ | $62.4 \pm 1.7$ |
| Re-init | $72.6 \pm 1.6$ | $73.1 \pm 1.3$ | $91.4 \pm 0.8$ | $91.0 \pm 0.4$ | $89.4 \pm 0.2$ | $89.9 \pm 0.1$ | $63.9 \pm 1.9$ | $61.9 \pm 2.3$ |
| Dataset | RTE (1k) | | MRPC (1k) | | STS-B (1k) | | CoLA (1k) | |
| | 3 Epochs | Longer | 3 Epochs | Longer | 3 Epochs | Longer | 3 Epochs | Longer |
| Standard | $62.5 \pm 2.8$ | $65.2 \pm 2.1$ | $80.5 \pm 3.3$ | $83.8 \pm 2.1$ | $84.7 \pm 1.4$ | $88.0 \pm 0.4$ | $45.9 \pm 1.6$ | $48.8 \pm 1.4$ |
| Re-init | $65.6 \pm 2.0$ | $65.8 \pm 1.7$ | $84.6 \pm 1.6$ | $86.0 \pm 1.2$ | $87.2 \pm 0.4$ | $88.4 \pm 0.2$ | $47.6 \pm 1.8$ | $48.4 \pm 2.1$ |
| Dataset | SST (1k) | | QNLI (1k) | | QQP (1k) | | MNLI (1k) | |
| | 3 Epochs | Longer | 3 Epochs | Longer | 3 Epochs | Longer | 3 Epochs | Longer |
| Standard | $89.7 \pm 1.5$ | $90.9 \pm 0.5$ | $78.6 \pm 2.0$ | $81.4 \pm 0.9$ | $74.0 \pm 2.7$ | $77.4 \pm 0.8$ | $52.2 \pm 4.2$ | $67.5 \pm 1.1$ |
| Re-init | $90.8 \pm 0.4$ | $91.2 \pm 0.5$ | $81.9 \pm 0.5$ | $82.1 \pm 0.3$ | $77.2 \pm 0.7$ | $77.6 \pm 0.6$ | $66.4 \pm 0.6$ | $68.8 \pm 0.5$ |

图 7: Re-Initialize

### 3.4 Warmup and lr-decay

Warmup 是在 Bert 训练中常用的技巧，就是模型前期的学习率先线性上升，随着迭代轮次增多，在逐步下降，后期采取较小的学习率进行梯度下降。Xiong et al., "On Layer Normalization in the Transformer Architecture"对此进行一些分析，总的来说就是作者发现 Transformer 在训练的初始阶段，输出层附近的期望梯度非常大，warmup 可以避免前向 FC 层的不稳定的剧烈改变，所以没有 warm-up 的话模型优化过程就会非常不稳定。特别是深网络，batch_size 较大的时候，这个影响会比较明显。

在代码的具体实现上，通过调用`scheduler = transformers.optimization.get_polynomial_decay_schedule_with_warmup()`实现一个自带 warmup 并以多项式速率递减的学习率 scheduler。

### 3.5 Pooling layer

在利用 Bert 预训练模型处理下游任务时，需要对 Bert 的输出进行相应的处理，在本次项目中，我们在 Bert 输出与分类器之间实现了不同的 pooling layer 来处理 Bert 模型的输出:

- **cls:** 直接获取 Bert 输出的 cls
- **pooler:** 直接获取 Bert 输出的 pooler_output
- **last-avg:** 对 Bert 最后一层的输出做 average pooling
- **first-last-avg:** 对 Bert 第一层与最后一层的输出分别做 average pooling，然后将两个输出拼接再做 average pooling。
- **mean-max-avg:** 对 Bert 最后一层的输出分别做 average pooling 与 max pooling，将两者拼接并通过一个全连接层映射回原有维度。mean-max-avg 具体计算方式如下:

$$X_{hidden} : [batch\_size, seq\_len, embedding\_dim] \tag{3}$$

$$mean\_pooled = mean\left(X_{hidden}, dimension = seq\_len\right) \tag{4}$$

$$max\_pooled = max\left(X_{hidden}, dimension = seq\_len\right) \tag{5}$$

$$mean\_max\_pooled = concatenate(mean\_pooled, max\_pooled, dimension = embedding\_dim) \tag{6}$$

### 3.6 Dropout

本次项目中使用的 Bert 模型的参数太多，但是训练样本太少，训练出来的模型非常容易产生过拟合的现象。在训练神经网络的时候经常会遇到过拟合的问题，过拟合具体表现在：模型在训练数据上损失函数较小，预测准确率较高；但是在测试数据上损失函数比较大，预测准确率较低。在 2012 年，Hinton et al., "Improving neural networks by preventing co-adaptation of feature detectors"了提出 Dropout（暂退法）。当一个复杂的前馈神经网络被训练在小的数据集时，容易造成过拟合。为了防止过拟合，可以通过阻止特征检测器的共同作用来提高神经网络的性能。

Srivastava et al., "Dropout: a simple way to prevent neural networks from overfitting"提出在训练过程中，他们建议在计算后续层之前向网络的每一层注入噪声。因为当训练一个有多层的深层网络时，注入噪声只会在输入-输出映射上增强平滑性。在标准暂退法正则化中，通过按保留（未丢弃）的节点的分数进行规范化来消除每一层的偏差。换言之，每个中间活性值 $h$ 以 * 暂退概率 *$p$ 由随机变量 $h'$ 替换，如下所示:

$$h' = \begin{cases} 0 & p \\ \frac{h}{1-p} & \end{cases}$$

根据此模型的设计，其期望值保持不变，即 $E[h'] = h$。Dropout 示意图8如图所示:

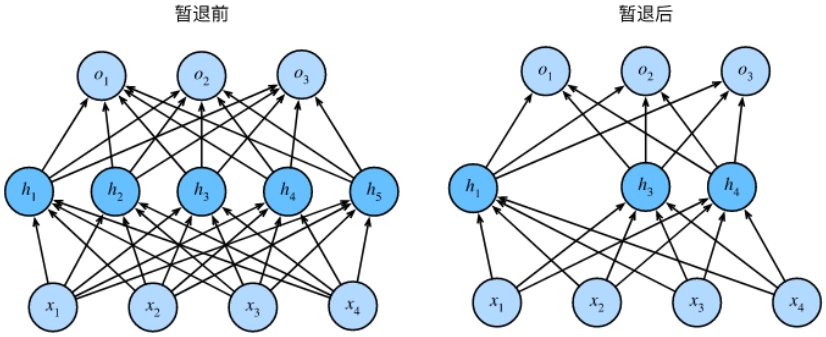


图 8: Drop out

# 4 实验过程和结果

本次实验中，在使用了上一节所说明的各类方法的基础上，我们小组所使用的模型可以分为四个版本，分别如下：

- V1：中英文混合训练，使用 bert-base-multilingual-cased 模型
- V2：在 V1 的模型基础上，修改了一下网络结构，在 Bert 模型的输出后增加了一个 RNN
- V3：中英文分开训练，分别使用 bert-base-cased 和 bert-base-chinese 模型进行英文和中文语料的训练
- V4：在 V3 的基础上，将所使用的 bert 模型由 base 模型改为 large 模型，分别是 bert-large-cased 和 bert-large-chinese

## 4.1 V1

在 V1 中，我们采用 bert-base-multilingual-cased 模型，并将其输出放入一个线性分类器中得到预测结果，并与实际 label 进行比较。在该版本中，我们针对超参数的设置进行了一系列的实验。

### 4.1.1 Dropout rate 的影响

在本实验中，选用不同的 batchsize，改变 dropout rate，固定其他超参数如下，测试模型在验证集上的准确率。

- epochs = 10
- lr = 1e-05
- model = bert-base-multilingual-cased
- pooling = cls
- reinit_layers = 4
- reinit_pooler = False
- weight_decay = True

实验结果如图 9 所示。

可以看到，改变不同的 batchsize，ACU 随着 dropout rate 的增大有先提高后降低的趋势，在 dropout rate=0.2 时的 ACU 率显著较低，而 dropout rate=0.5 时的 ACU 略低于为 0.4 的时候，在 0.4 时的效果相对最好。

对此的理解是，如前文所言，我们在模型中增加 Dropout 层，是为了在每个 batch 的训练过程中，随机地暂时丢弃部分参数，可以有效防止模型对于训练数据的过拟合。考虑到 Bert 是一个参数量达到 1 亿的大模型，而训练的数据集又比较小，若不增加 Dropout 则很容易发生过拟合现象，导致在验证集和测试集的表现不佳。但同时，Dropout 的设置也不能过高，过高的话相当于减少了模型的参数量，且每个 batch 都只能对少部分参数进行训练，会容易导致训练不充分和模型效果不佳。

因此，通过实验，我们发现在该任务该模型，使用该训练集的基础上，选用 dropout rate=0.4 可以取得较好的结果。

| 改变batch_size大小，观察dropout对准确率ACU的影响 | | |
|---|---|---|
| batch_size | dropout | ACU |
| 12 | 0.2 | 84.58083832 |
| | 0.4 | 85.10479042 |
| | 0.5 | 85.05489022 |
| 24 | 0.2 | 84.76316598 |
| | 0.4 | 84.95508982 |
| | 0.5 | 84.60525037 |
| 36 | 0.2 | 83.47420635 |
| | 0.4 | 84.23412698 |
| | 0.5 | 83.84623016 |
| 48 | 0.2 | 84.18025031 |
| | 0.4 | 84.43191392 |
| | 0.5 | 84.67071123 |

图 9: Dropout rate 对准确率 ACU 的影响

#### 4.1.2 Pooling 方式的影响

在本实验中，选用不同的 pooling 方式，固定其他超参数如下，测试模型在验证集上的准确率。

- batch_size = 48
- dropout = 0.4
- epochs = 10
- lr = 1e-05
- model = bert-base-multilingual-cased
- reinit_layers = 4
- reinit_pooler = False
- weight_decay = True

实验结果如图 10 所示。

| pooling方式对准确率ACU的影响 | |
|---|---|
| pooling | ACU |
| cls | 84.23191392 |
| first-last-avg | 84.24145299 |
| last-avg | 85.15941697 |
| mean-max-avg | 85.29075092 |
| pooler | 83.63858364 |

图 10: Pooling layers 对准确率 ACU 的影响

根据实验结果可以看到，在该任务该模型，使用该训练集的基础上，使用 mean-max-avg 的 pooling 方式可以取得较好的结果。

### 4.1.3 batchsize 的影响

在上一个 pooling 实验的基础上，本实验选用不同的 batchsize，固定其他超参数如下，测试模型在验证集上的准确率，探究 batchsize 的对 ACU 的影响。

- dropout = 0.4
- epochs = 10
- lr = 1e-05
- model = bert-base-multilingual-cased
- pooling = mean-max-avg
- reinit_layers = 4
- reinit_pooler = False
- weight_decay = True

实验结果如图 11 所示。

| 使用mean-max-avg进一步实验，进行batchsize的实验（dropout 0.4） | |
|---|---|
| batchsize | ACU |
| 12 | 83.93213573 |
| 24 | 84.62498081 |
| 36 | 85.06944444 |
| 48 | 85.29075092 |

图 11: Batchsize 对准确率 ACU 的影响

根据实验结果可以看到，在该任务该模型，使用该训练集的基础上，使用 mean-max-avg 的 Pooling 方式时，batch-size 越大得到的模型的性能越好，在验证集上的准确率更高。

对此的理解是，batchsize 越大，loss 传递的方向就越稳定，参数的调整会更加的稳定，得到的结果相对会更好。但是同时，batchsize 越大对于显卡的显存大小要求就越高，因此这里也存在一个平衡的问题。

### 4.1.4 Reinit layers 数目的影响

本实验选用不同的 Reinit layers 数目，对预训练的 Bert 模型的后几层权重参数进行初始化，固定其他超参数如下，测试模型在验证集上的准确率，探究 Reinit layers 数目对 ACU 的影响。

- batchsize = 36
- dropout = 0.4
- epochs = 10
- lr = 1e-05
- model = bert-base-multilingual-cased

- pooling = mean-max-avg
- reinit_pooler = False
- weight_decay = True

实验结果如图 12 所示。

| reinit layer数目对准确率ACU的影响 | |
|---|---|
| reinit layer | ACU |
| 4 | 84.20138889 |
| 3 | 83.68055556 |
| 2 | 84.37586349 |
| 1 | 85.14384921 |
| 0 | 85.32799851 |

图 12: Reinit layers 数目对准确率 ACU 的影响

根据参考论文Zhang et al., "Revisiting Few-sample BERT Fine-tuning"，我们可以知道 Bert 中底部的层也就是靠近输入的层，学到的是通用语义信息，比如词性、词法等语言学知识，而靠近顶部的层也就是靠近输出的层，会倾向于学习到接近下游任务的知识。由于在本次实验中，在 fine-tuning 的过程中使用的数据集只有 2w 条数据，这对于 Bert 这样的大模型来说不足以使其中靠近输出的层充分学习到下游任务的特征即文本的情感，进而影响到模型推断的准确率，所以在本次项目中对于助教给定的数据集进行 fine-tuning 的过程中，虽然权重初始化可以防止模型快速过拟合，但是也会影响模型最后输出的准确率，而且 dropout 层的存在已经在一定程度上能够防止模型的过拟合，所以权衡利弊之后，我们决定不对模型进行权重初始化。

#### 4.1.5 V1 总结

进行了以上的一系列试验后，我们得到了一组在该任务该模型上表现比较好的超参数，如下：

- batchsize = 36 或 48
- dropout = 0.4
- epochs = 10
- lr = 1e-05
- model = bert-base-multilingual-cased
- pooling = mean-max-avg 或 last-avg
- reinit_layers = 0
- reinit_pooler = False
- weight_decay = True

在该版本的模型中，经过多次训练得到的表现最佳的模型，其正确率为 86.03%。

## 4.2 V2

在 V2 中，我们在 V1 的模型基础上，在 Bert 的输出和分类器之间加了一个 RNN，选用的是一个双向的 GRU，目的是为了更多地学习到句子序列层面的信息，并希望能借此提高模型对句子的整体理解能力，有助于提高其的预测能力，提高准确率。

但实际测试中，我们发现加上了该 RNN 后，模型的整体表现并没有提高，反而存在一定程度的下滑（验证集上的准确率均值在 84.8% 左右）。对此我们的分析是，Bert 模型本身的 All-Attention 机制就已经使得模型可以看到序列全局的信息了，在不加入 RNN 的时候也能关注到序列层面的信息。而加上 RNN 后效果略微变差，一方面可能是实验的误差，数据集太小以及切分训练集和测试集时随机取样的方式都可能导致每次实验的结果间存在合理的波动；二是加上 RNN 之后模型的参数量进一步增大，而学习率等的超参数又没有进行相应的调整，可能加速了模型的过拟合导致其在验证集上的效果不佳。由于效果不佳，因此在后续的版本中我们不再加上 RNN。

## 4.3 V3

在 V3 中，我们考虑到前面两个版本都是采用中英文混合训练的方式进行的，而 Bert-base-multilingual-cased 模型是在一百多种语料上进行的预训练，多种语言间的信息难免相互干扰，因此尝试在中英文语料上分两个不同的模型进行训练。其中，英文语料的训练采用 bert-base-cased，中文语料的训练采用 bert-base-chinese 模型（都是由 hugging face 官方提供）。

我们选用在 3.1.5 中得到的表现较好的超参数，在两个模型上分别进行了实验，结果如图 13 和图 14 所示。

| 英文：bert-base-cased | | |
|---|---|---|
| pooling | batchsize | ACU |
| last-avg | 36 | 92.15686275 |
| last-avg | 48 | 92.61083744 |
| mean-max-avg | 36 | 92.49452655 |
| mean-max-avg | 48 | 92.65289449 |

图 13: Bert-base-cased 模型在英文数据集上的表现

| 中文：bert-base-chinese | | |
|---|---|---|
| pooling | batchsize | ACU |
| last-avg | 36 | 83.3531746 |
| last-avg | 48 | 82.29166667 |
| mean-max-avg | 36 | 81.42857143 |
| mean-max-avg | 48 | 82.91170635 |

图 14: Bert-base-chinese 模型在中文数据集上的表现

由于中英文数据集的大小是完全一样的，都是 10000 条数据，英文模型上的最佳表现为 92.65%，中文模型上的最佳表现为 83.35%，$(92.65\% + 83.35\%)/2 = 88.0\% > 86.03\%$，因此可以简单地认为分模型训练的效果要好于中英文混合训练单个模型。

### 4.4 V4

在 V4 中，我们在 V3 的基础上，将 Bert 的 base 模型都换为 large 模型，依旧是在中文和英文两种预料上分开训练两个模型，其中，英文语料的训练采用 bert-large-cased（由 hugging face 官方提供），中文语料的训练采用 bert-large-chinese 模型（hugging face 模型库中由个人训练后提供）。

我们选用在 3.1.5 中得到的表现较好的超参数，在两个模型上分别进行了实验，结果如图 15 和图 16 所示。需要注意的是，使用 large 模型进行训练时，batchsize 需要适当地减小一些，因为 large 模型的参数量约 3 亿，对于显存的要求会更高，batchsize 过大的话会导致显存不足。

| 英文：bert-base-cased | | |
|---|---|---|
| pooling | batchsize | ACU |
| last-avg | 12 | 93.61277445 |
| last-avg | 20 | 94.2 |
| mean-max-avg | 12 | 95.13972056 |
| mean-max-avg | 20 | 94.09117647 |

图 15: Bert-large-cased 模型在英文数据集上的表现

| 中文：bert-base-chinese | | |
|---|---|---|
| pooling | batchsize | ACU |
| last-avg | 12 | 85.77844311 |
| last-avg | 20 | 86.18235294 |
| mean-max-avg | 12 | 86.42714571 |
| mean-max-avg | 20 | 86.62352941 |

图 16: Bert-large-chinese 模型在中文数据集上的表现

可以看到，英文 large 模型上的最佳表现为 95.14%，中文 large 模型上的最佳表现为 86.62%，都要显著优于 base 模型。

### 4.5 总结

各个版本的最佳测试正确率结果如图 17 所示。

| 不同版本的准确率对比 | | |
|---|---|---|
| version | ACU | |
| | 中文 | 英文 |
| 1 | 86.03 | |
| 2 | 84.8 | |
| 3 | 83.35 | 92.65 |
| 4 | 86.62 | 95.14 |

图 17: 不同模型的性能对比

使用两个小的中英文 xml 测试文件进行测试，得到的 xml 输出文件截图如图 18 和图 19 所示。

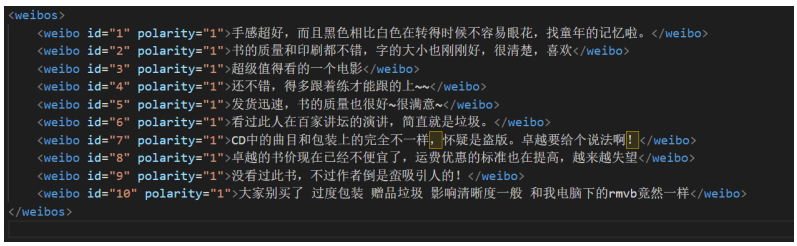


图 18: 中文输出文件

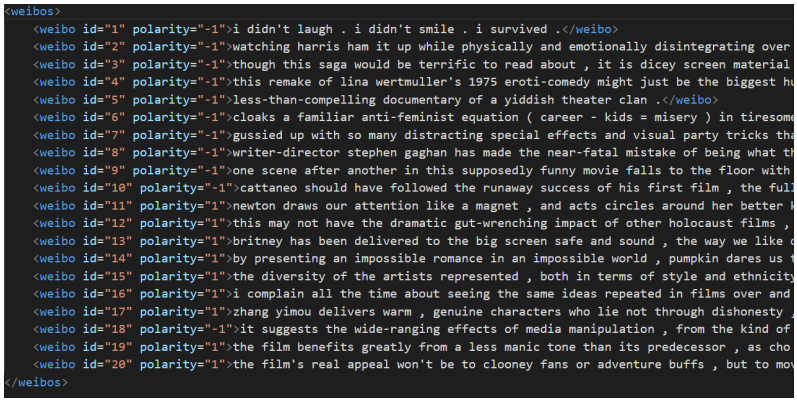


图 19: 英文输出文件

结果分析:

- 中文模型的效果明显差于英文模型的效果，我们认为这是由于我们在中文模型的训练过程前，缺少对语料的分词过程，直接以字符形式作为输入，可能丢失了一部分信息；此外，数据集的质量区别也可能是造成这种显著差异的原因之一。
- 双语料混合训练单个模型的效果确实不如中英语料分开分别训练模型的效果。
- 大模型的效果是非常显著的，large 模型相比 base 模型，其注意力头数目、隐层维度大小和网络层数都要更多，参数量大约是其的三倍，表现能力要更强，也能获取文本更深层的信息，能显著提高模型的预测准确率。
- 超参数的设置在不同的任务和数据集上的差异是比较明显的，如对于 Bert 的最后几层参数进行重新初始化的方式，在其他任务上可能可以起到防止过拟合，提高模型性能的作用，但是在本任务中，经过实验，我们发现不进行初始化的模型其性能表现反而是最好的，充分说明超参数的设置需要因任务而异，通过实验得到较好的设置方式。

# 5 展望

本次大作业受限于时间和资源，还存在一定的不足，在以下几个方面还可以作一定的尝试，有望取得更佳的结果。

- 对于我们小组采用的 Bert 模型来说，本次实验的数据集太小，可以考虑再加入一些合适的数据集作为训练语料。
- 改用新的损失函数，如更适合用于二分类任务的 BCEWithLogitsLoss 损失函数等。
- 可以在 Bert 的输出端和分类器间多尝试增加一些网络结构，如 LSTM。
- 可以引入新的分类器用来替换掉当前一个简单的线性全连接层，如 attention 机制，句法树加 GNN 等。
- 可以尝试减小学习率，增大 epoch 数来进行更精细的训练。本次大作业过程中我们已经有相关尝试，但是单次训练时间过长导致尝试次数比较少，未能得到较为理想的结果。

# References

Krogh, Anders and John Hertz. "A Simple Weight Decay Can Improve Generalization". In: *Advances in Neural Information Processing Systems*. Ed. by J. Moody, S. Hanson, and R.P. Lippmann. Vol. 4. Morgan-Kaufmann, 1991. URL: https://proceedings.neurips.cc/paper/1991/file/8eefcfdf5990e441f0fb6f3fad709e21-Paper.pdf.

Chelba, Ciprian and Alex Acero. "Adaptation of maximum entropy capitalizer: Little data can help a lot". In: *Computer Speech & Language* 20.4 (2006), pp. 382–399.

Hinton, Geoffrey E. et al. "Improving neural networks by preventing co-adaptation of feature detectors". In: *CoRR* abs/1207.0580 (2012). arXiv: 1207.0580. URL: http://arxiv.org/abs/1207.0580.

Srivastava, Nitish et al. "Dropout: a simple way to prevent neural networks from overfitting". In: *The journal of machine learning research* 15.1 (2014), pp. 1929–1958.

Vaswani, Ashish et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Devlin, Jacob et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

Lee, Cheolhyoung, Kyunghyun Cho, and Wanmo Kang. "Mixout: Effective regularization to finetune large-scale pretrained language models". In: *arXiv preprint arXiv:1909.11299* (2019).

Xiong, Ruibin et al. "On Layer Normalization in the Transformer Architecture". In: *CoRR* abs/2002.04745 (2020). arXiv: 2002.04745. URL: https://arxiv.org/abs/2002.04745.

Zhang, Tianyi et al. "Revisiting Few-sample BERT Fine-tuning". In: *CoRR* abs/2006.05987 (2020). arXiv: 2006.05987. URL: https://arxiv.org/abs/2006.05987.