# 预训练语言模型Bert的特性与潜力

组长 李珂

## 一、摘要

基于transformer的语言模型在引入大规模预训练后在自然语言处理领域表现出来巨大的潜力。然而，目前尚不清楚为什么预训练然后微调方式可以提高跨不同任务的性能和泛化能力。有经验可知，预训练模型下游任务的能力受到预训练任务的限制，但同时预训练的开销巨大，从预训练入手有违预训练模型的初衷。那么如何更有效地利用预训练模型便成为一个很有价值的问题。本文以Bert为例，在语义等效、逻辑推理、语法对错三个数据集上进行实验，探究不同词向量池化方式获得句子向量、不同block输出的句子向量对文本的理解能力。然而本文发现，对于不同的自然语言理解任务，并没有一个统一、简单的答案，因此本文主要是进行实验性探究，给出了一些应用中可行的建议流程。

## 二、介绍

自此ChatGPT发布以来，Large language model（LLM）开始被大众作为一个强大的人工智能助手，在需要广泛领域的专家知识的一些任务中表现出了惊人的能力，其中包括对逻辑要求很高的编程和一些偏感性的创意写作等领域。它们通过直观的聊天界面实现了与人类的互动，这导致了公众的快速广泛采用。

LLM的训练过程看似简单直接，但是展现出了惊人的能力。预训练语言模型能够很容易地通过添加额外的输出层迁移到下游任务进行微调，因此预训练语言模型通常作为一个初始化点。GPT系列的单向自回归transformer解码器是在广泛的自监督数据语料库上进行预训练的，其单向注意力的特点使得非常适合文本生成。BERT系列则使用transformer编码器这种双向注意力机制，希望以此捕获上下文之间的关系，在从语言序列和语言序列的分类和问题回答等许多自然语言理解任务中超过了GPT-1的性能。这一事实表明BERT采用transformer编码器这种双向特征可能在除了文本生成之外的文本理解任务中拥有更强的能力，即能够生成信息量更充足的句子向量表示。

虽然预训练语言模型在下游任务上表现出了很强的能力，但是其高额的预训练开销，使个人开发者或小开发团体很难承担。同时预训练模型的下游任务能力又受到预训练任务的限制。对于小型开发群体来说，如何有效地利用预训练模型在区别于预训练任务的数据集上发挥性能是很有意义的。

不同于由动物神经元启发的全连接、卷积神经网络模型，基于transformer结构的语言模型并没有很明显的启发意义。在本次《深度学习》课程大作业中我们更希望通过实验证据来说明一些问题。

在BERT原始文献中，作者在文本分类上仅使用了[cls] token作为全部信息的汇总，同时也仅是使用简单的单层感知机分类器。由此我们很自然地产生两个疑问：

1. *是否有能更有效的文本嵌入表示作为文本理解？*
2. *是否有能更有效的轻量级分类器配合BERT进行微调？*

在本学期《深度学习》课程大作业中，我们对BERT的不同隐藏层输出和多种weight light降维方式获得通用句子向量 (文本嵌入)进行了实验性探究，以此方式了解BERT预训练模型对自然语言的基本理解能力，即能在多大程度上捕获了语法和语义信息。然后我们尝试在预训练模型上使用类型的分类器来探究不同分类器与BERT的配合能否产生更好的效果。

### 三、 BERT Sentence Embedding的有效性

1. transformer编码器的影响

Transformer模型结构的设计是高度对齐的，每个transformer block的输入与输出的信息尺寸都是一致的。同时因为自注意力机制的作用，每个block输出的token都是整个输入句子的全部token信息互相交互得到的语义信息。而通常深度学习模型的底层与顶层获得的信息的抽象程度是不同的，底层往往是更接近输入层的低级信息，而顶层往往是更接近信息本质的高级信息。Bert对文本的理解也随着模型加深有不同的变化[2][4]。在不同语言理解任务上不同程度的抽象信息对于文本理解的影响。

1. Pooling 方法的影响

Transformer模型结构能够输出是各个输入token对应动态token表示。在BERT中则是使用 [cls] token 作为整个句子的向量表示。但这很难说是否是最佳选择[3]，在BERT的预训练任务2中，[cls] token 作为分类器作用的token看似是一种很合理的设计。因为[cls]token既不属于句子1，也不属于句子2，用于分类似乎是最公平的，但由于预训练任务的限制，[cls] token向量表示是基于预训练任务2的理解，对于其他文本理解任务的能力如何是难说的。我们希望探索如何综合不同的token向量可以获得更好文本嵌入表示。

1. 分类器的影响

预训练语言模型希望通过自监督的方式学习到对自然语言的理解，这种理解希望被表示成向量。人们将大量算力集中预训练语言模型的预训练上，并希望能通过简单的微调获得在下游任务上更好的表现，因此往往下游任务都是使用简单的mlp网络作为分类器。但也有些人在特定的任务中尝试将预训练语言模型与其他分类器相结合获得更好的性能。因此我们将尝试不同的轻量级分类器(light weight)来探索BERT与其他分类器之间的能力。

有的研究[5]指出，复杂的分类器对文本分类任务上微调效果一般，他们的设计看起来过于复杂，因此我们尝试将Mlp与Cnn看作一种降维方式来探究其能力。

### 四、数据集

1. **句子对语义是否等效**

MRPC(The Microsoft Research Paraphrase Corpus，微软研究院释义语料库)，相似性和释义任务，是从在线新闻源中自动抽取句子对语料库，并人工注释句子对中的句子是否在语义上等效。

正例：

句子1：The largest gains were seen in prices , new orders , inventories and exports .

句子2：Sub-indexes measuring prices , new orders , inventories and exports increased .

反例：

句子1：Earnings per share from recurring operations will be 13 cents to 14 cents .

句子2：That beat the company 's April earnings forecast of 8 to 9 cents a share .

1. **句子对是否能够推断\对齐**

RTE(The Recognizing Textual Entailment datasets，识别文本蕴含数据集)，自然语言推断任务，这些数据样本都从新闻和维基百科构建而来。RTE数据集是为了判断两个句子是否能够推断或对齐，属于句子对的文本二分类任务。

正例：

句子1：The gastric bypass operation, also known as stomach stapling, has become the most common surgical procedure for treating obesity.

句子2：Obesity is medically treated.

反例：

句子1：No Weapons of Mass Destruction Found in Iraq Yet.

句子2：Weapons of Mass Destruction Found in Iraq.

1. **句子是否语法正确**

CoLA(The Corpus of Linguistic Acceptability，语言可接受性语料库)，单句子分类任务，语料来自语言理论的书籍和期刊，每个句子被标注为是否合乎语法的单词序列。本任务是一个二分类任务，标签共两个，分别是0和1，其中0表示不合乎语法，1表示合乎语法。

正例：

She is proud.

she is the mother.

反例：

Mary sent.

Yes, she used.

### 五、实验设计

我们使用huggingface发布的transoformers项目[7]中的预训练Bert-base模型上进行实验。选择GLUE基准数据集中MRPC、RTE和CoLA三项自然语言理解任务。

由于我们对仅对Bert嵌入中编码的语言信息感兴趣，我们只在每个分类任务的嵌入之上添加一个简单的两层的感知机模型来探索其对文本的理解。

然后我们将进行全参数微调，并获得了超越基线Bert的效果。

### 六、问题探究

1. **是否有能更有效的文本嵌入表示作为文本理解？**

为了考虑Bert最本源对语义的理解，希望以light weight的方式低成本地实现这一目标。设模型输出层词嵌入为一组向量 ，其中，通过不同的方式获得一组新的tokens ，最终得到句子向量表示，我们尝试如下几种方式：

1. **[cls] token**

Bert原始设计中，作者通过设计Next Sentence Prediction（NSP）任务任务来让模型产生对句子层面的理解，同时使用[cls] token来作为最终的分类向量，这种设计看似时最公平的，因为得益于Bert的双向结构[cls] token可以综合全部的上下文信息，同时其不属于任何一个句子，看似是最公平的。

1. **平均池化**

一种简单的池化方式是对每个取平均，这种方式提供简单的运算结合了全部信息，很多探索文本向量表示的工作都是基于这种方式。

1. **最大池化**

由于观察到，通常只有少数关键词有助于最终预测，沿着单词向量的每个维度取最大激活值，从每个单词嵌入维度提取最显著的特征。该策略类似于卷积神经网络中的最大池化操作，希望拥有一定的鲁棒性。

1. **最小池化**

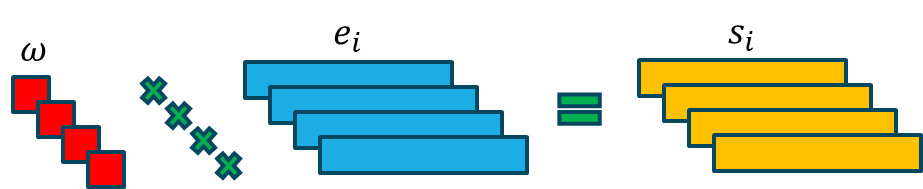
稀疏表达在神经网络中有许多优点，比如可解释性、易于降维等，最重要的是可以使得神经网络提取的特征在高维空间上稀疏从而使得其变得线性可分。深度神经网络中，例如12层的transformer的Bert提取的特征也具有高维稀疏性，提供最小池化使其更加稀疏。

1. **可学习加权池化**

语言模型区别于视觉任务的一个很重要的特点是其输入特征具有时序性，transformer中self-attention可以使每个token综合全部上下文的时序特性，但由于预训练中NSP任务是基于[cls] token训练的，那么其输出tokens中就可能有不同的重要程度。我们**假设**BERT由于预训练任务限制，其最后输出的tokens是有位置性的，通过定义一组可学习参数，让模型在训练过程中自己找到token之间的位置重要程度。

定义一组可学习参数， ,得到一组新最后句子向量表示为： , 为非线性映射，这里取tanh。

初始化，当的元素值一般1e-2量级，在时，在，多个经过tanh时映射区间接近于线性，因此**初始效果**应该接近于取平均池化，我们希望在训练数据中捕捉token位置重要程度的理解。此种方法可以看作学习输出tokens的位置重要性的注意力。



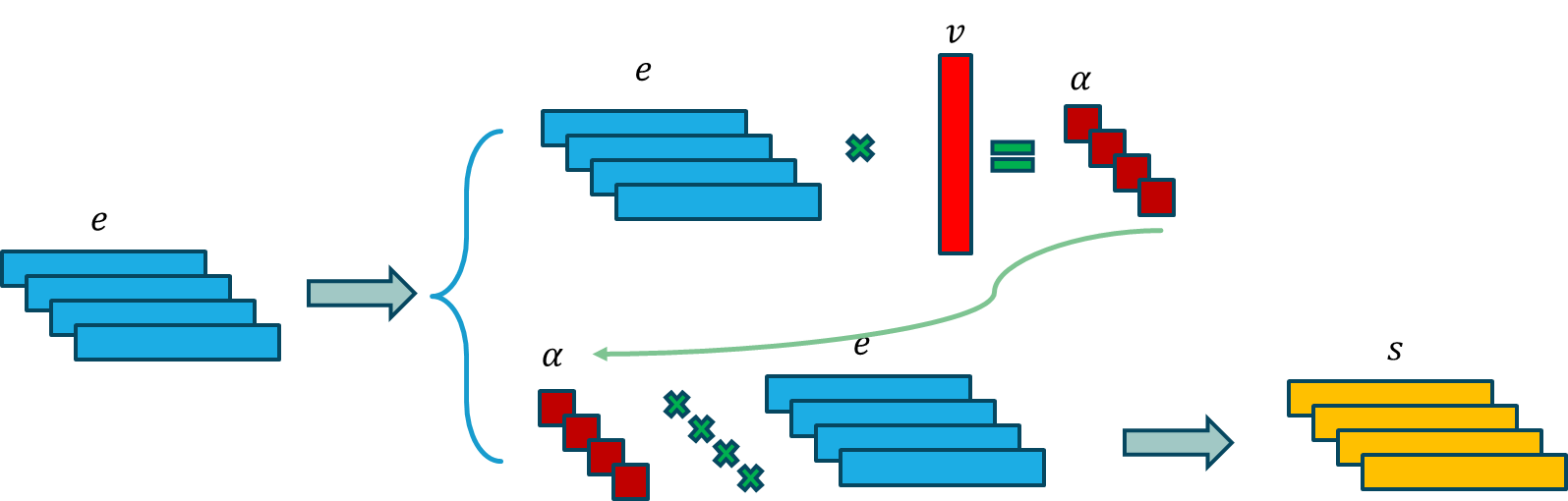
1. **注意力加权池化**

Transformer输出的token是基于self-attention获得综合全部上下文信息的高维词嵌入，这个词嵌入是在高维空间上特征向量，借鉴self-attention我们定义一种注意力池化方式：

定义一个可学习的注意力键，，于每个进行注意力计算，即，, 为非线性映射，取softmax或者tanh。

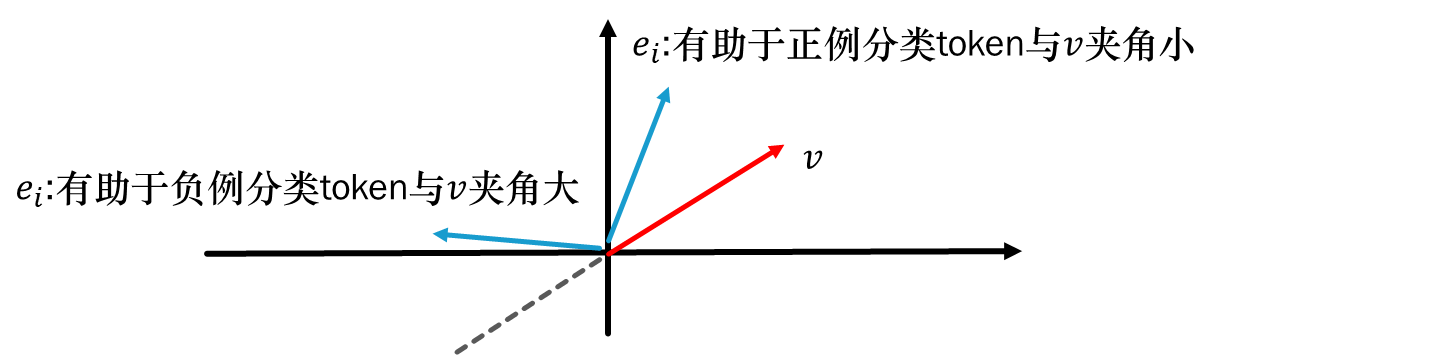
当选择softmax时，，即进行求和为1的加权，当选择tanh时，的激活范围在，自由度高，使其表达能力变强。

由此权重计算 ,得到一组新token向量，最终句子向量表示：。

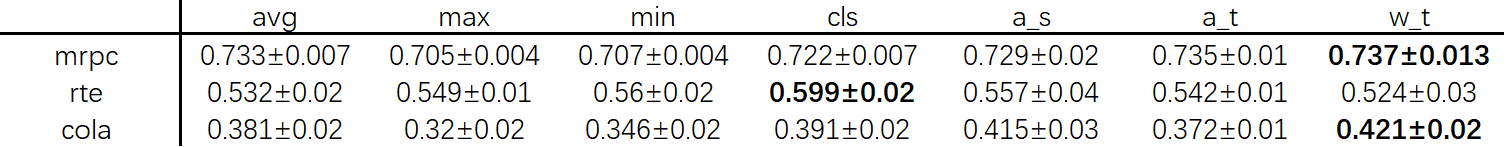


这种方式实际上是对token的高维空间关系的解耦，在训练集上获得一个注意力键，使其与有助于正例样本分类的token之间的余弦相似度最大，而与有助于反例样本分类的token之间的余弦相似度最小，从而达到对token注意力。

针对于我们选择的下游任务均为文本2分类任务。通过这种方式得到注意力向量，正例样本产生有助于分类与的夹角趋向于0时，重要程度最大，与负例样本产生有助于分类与的夹角趋向于180°时，重要程度最小。



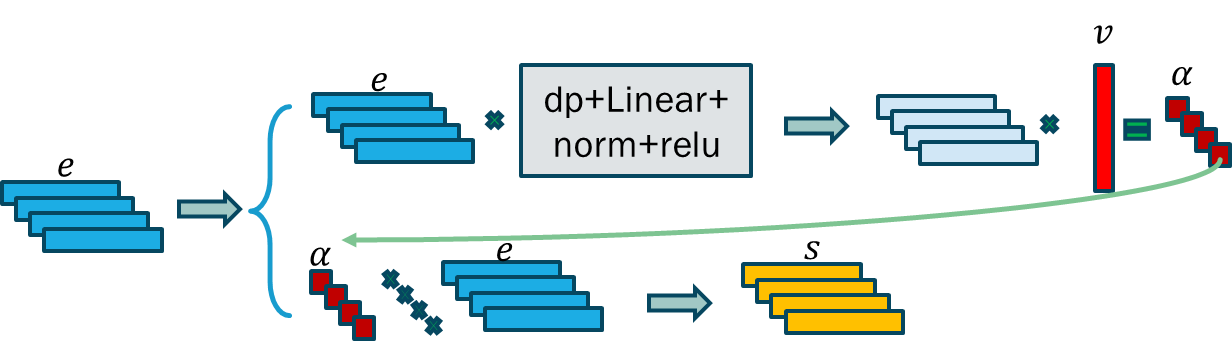
实验效果来看，可学习加权池化与注意力池化通过仅增加了768个参数却在mrpc和cola数据集上表现出了比[cls]、平均池化更具有竞争力的结果，分别提升了0.4%和4%的提升。但是在推理数据集rte上，[cls] token却表现出惊人的效果，我们推测是因为句子对推理任务与其预训练任务NSP有很高的相似性，这进一步体现了预训练的重要性。

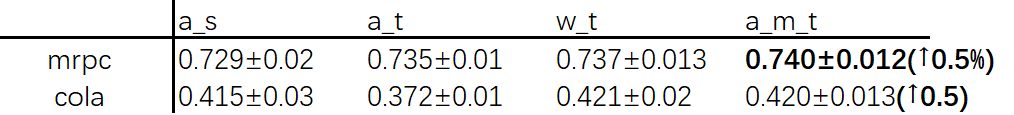


**图表 1**不同pooling方法在mrpc,rte,cola数据集上效果。其中mrpc,rte展示正确率，cola展示马修斯相关系数。上述数据从左到右依次是平均池化、最大池化、最小池化、[cls] token、注意力加权池化(softmax)、注意力加权池化(tanh)、可学习加权池化(tanh)。

可学习加权池化的初始化点是平均池化，其效果显著似乎验证了token之间绝对位置之间的信息有效性。但是在rte数据集上[cls] token表现出了惊人效果，但是可学习加权池化却并没有如我们想的一样可以达到：[cls] token位置的权重远大于其他位置，从而接近[cls] token的效果，这似乎意味着前述绝对位置信息并不是我们想的简单，这也许是非线性映射tanh造成的影响。

注意力加权池化在三个数据集上均表现出了不错的效果，其没有超过可学习加权池化的原因之一可能是原始Bert最终的输出token是**正例多簇**和**反例多簇**的情况，即仅使用一个注意力键难以表征多簇，因此我们尝试添加非线性映射来将多簇映射成单簇。如图标2所示，确实有一定提升。





图表 2添加feature mapping后注意力加权池化在mrpc和cola数据集上效果。其中mrpc展示正确率，cola展示马修斯相关系数。上述数据从左到右依次是注意力加权池化(softmax)、注意力池化(tanh)、可学习加权池化(tanh)、注意力映射池化(tanh)。括号中为相对原始的提升。

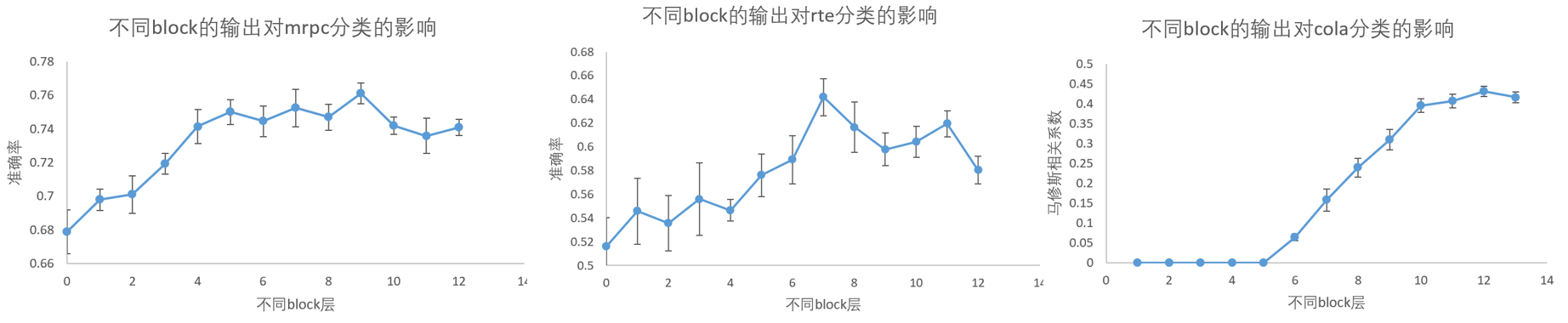
1. **不同transformer block的输出token对文本理解程度有什么不同？**

神经网络可以看作一种通用的特征提取器，通过最小化优化目标，使得数据表示转化为一种神经网络输出表示，比如auto-encoder进行图像数据压缩，word2vec获得词向量、deep SVDD获得更灵活的特征映射进行单分类等等。很多算法都在尝试通过神经网络端到端的优化来获得其中间隐藏层输出作为一种信息表示形式。

在通常的神经网络认知中，我们任务神经网络从浅层到深层，特征的抽象程度通常会逐渐增加。具体来说，神经网络的各个层可以被看作是特征提取器。在网络的较浅层，神经元通常对输入数据的低级特征有较好的响应，比如卷积层对边缘、颜色等提取。随着层次的加深和非线性的作用叠加，神经元逐渐组合这些低级特征，形成对更抽象、更复杂特征的响应，神经网络提取的特征开始变得有统一。在网络靠近输出的部分的特征图通常是高级特征，这些特征通常能够表示数据集中一些本质的信息，对于任务的解决具有更高层次的表征能力。

在传统的神经网络认知框架中，通常认为随着信息从网络的输入层传播到输出层，特征的抽象程度会逐渐增加。然而，与此观点不同的是，神经网络并未表现出特征通用性随着接近输出层而增强的趋势。相反，神经网络的输入层特征更倾向于涉及原始数据的低级抽象，而输出层的特征信息则更注重优化特定任务。Transformer架构通过一层层堆叠block来进行更深的特征提取，我们可以认为每个block的输出为抽象的特征。

为了深入研究预训练模型BERT在文本理解方面的能力。在本节中，我们将考察不同的block块输出，以探讨是否存在相较于输出层更为本质的文本理解特征。



**图表 3** 不同block的输出对mrpc、rte、cola数据集分类的影响。Y轴为分类准确率，X轴为不同的block输出，1为embedding层，12为bert的输出层。

通过图表3可知，我们得到如下**结论**：

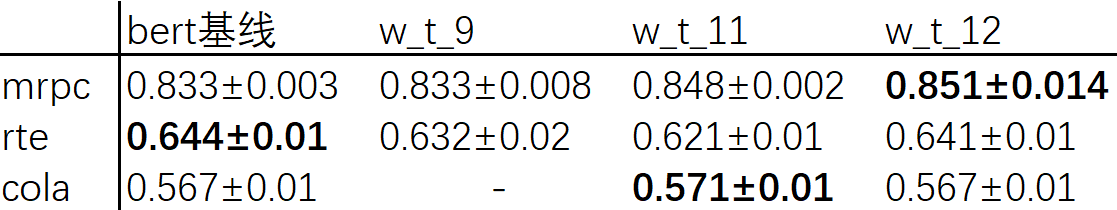
1. 从第一个block(embedding layer)开始，并没有上下文信息，随着嵌入深入网络，它们在每一层都能获得越来越多的上下文信息。
2. 到开始接近最后一层时，block输出的特征向量的效果开始变差，模型对语义的理解开始偏向于预训练任务的方向。
3. 选择倒数2~4个block输出的特征是一个更好的选择。
4. **以上讨论总结出的最有效Bert子结构是否也是最佳微调子结构？**

以上讨论均是基于冻结Bert参数，得到tokens 向量表示，讨论Bert对语言的理解能力。

经过实验，我们发现在mrpc、cola数据集上使用可学习加权池化效果最佳，同时其分别在第9、11层的输出处表现出最佳效果。因此我们想要知道是否在以上模型子结构上进行全参数微调是否是一个超越原始Bert的一个更好的结构、出发点。

即：冻结Bert时最佳的句子向量时的模型结构是否是一个好的群参数微调起始点？

因此我们产生过hi全参数微调结合可最佳隐藏层输出tokens、学习权重池化参与微调。



**图表 4** Bert全参数微调。Bert基线复现于项目原始超参，但是seed随机取的所以可能略有差别。w\_t\_9分别代表：可学习加权池化-tanh-第9个隐藏层。其他类似。

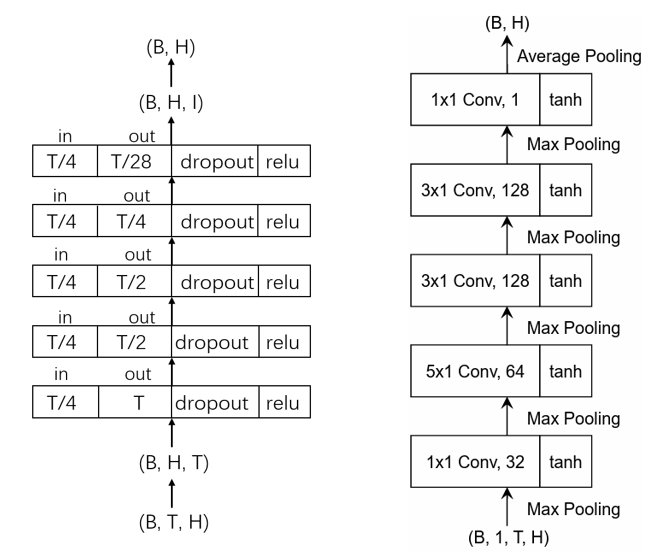
上述实验结果发现，我们在mrpc、cola数据集上比基线提升了大约2%、0.4%，但是在rte数据集上并没有显著效果，但微调修正了预训练任务引起的句子表征效果。且不同层的输出进行分类的效果并没有达到六(二)实验的预期，可能是经过微调后最后一输出层离反向传播起点最近，导致梯度更新的效果更明显能够弥补预训练任务带来的差异。

综上所述，我们发现预训练模型微调能够很大程度弥补预训练任务带来的限制。

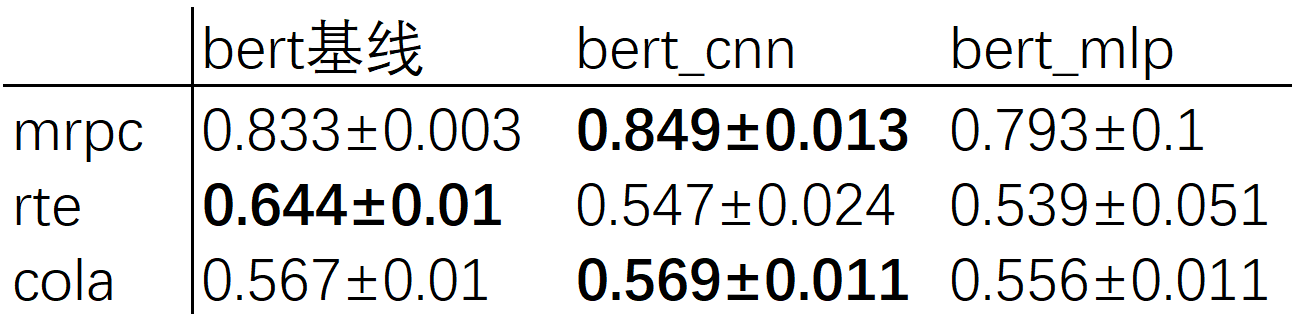
1. **“臃肿”的分类器能否用在预训练语言模型微调上？**

上述讨论我们基于“light weight”的方式对输出层全部token进行池化，随后使用简单的两层感知机进行分类。那么是否能够设计一个不限制参数量的分类器，同时综合全部tokens信息进行分类？

考虑多层感知机与卷积神经网络两种结构进行讨论，



图表 5 左侧为多层感知机设置，右侧为卷积神经网络。输入B为batchsize，T为句子长度,H为token维度。



**图表 6** Bert\_cnn与Bert\_mlp分别对应前述卷积分类器与多层感知机。

一般来说Bert产生的高维特征没有卷积计算模式中二维局部先验，但是卷积这种权值共享的计算方式，看似在高维稀疏表示中是一种很好的非线性映射，能够获得比全连接层(存在dropout)这种密集计算更有效的激活。

实验结果显示，Bert\_Cnn取得比Bert基线略好的效果，但没有超过图表4中效果。因此在小数据集微调上，简单的tokens池化就起到了很不错的效果。

由于预训练语言模型微调的不稳定性问题[6]，我们还未有能力把握，也因此没有做出更多的结算结果。

### 7、总结与讨论

本文中，我们在句子推断、语义等效、语法正误三种数据集上使用parameter free和weight light的方式对Bert-base输出的tokens进行池化来获得句子向量表示，以探究不同池化方式的效果。然后又进一步探究transformer这种堆叠block的结构中，每个block输出的tokens对文本理解的程度。最后尝试复杂的Mlp分类器和卷积分类器在Bert微调获得句子向量的效果。我们通过实验得到如下结果：

1. 句子向量表示的能力受到预训练任务的限制。如[cls] token进行分类在多种不同任务上往往不是最佳方式，但在类似于预训练任务的下游任务上将会取得极佳的效果。
2. 本文推荐在下游应用时，先尝试[cls] token进行分类，然后依次尝试可学习加权池化、注意力加权池化、平均池化的方式。
3. 从第一个block(embedding layer)开始，并没有上下文信息，随着嵌入深入网络，它们在每一层都能获得越来越多的上下文信息。
4. 到开始接近最后一层时，block输出的特征向量的效果开始变差，模型对语义的理解开始偏向于预训练任务的方向。但是微调可以弥补这一问题，因此仍然建议使用最后一层的输出。
5. 复杂的分类器有一定效果，但是简单的weight light的池化就可以取得更好的效果。

预训练语言模型打破了数据限制，同时利用transformer这种可无限堆叠的结构取得了令人瞩目的效果。因此，预训练模型的下游应用受到开发者们的重视。本文希望通过探讨以上三个问题来给出一个具有参考价值调策略。

### 八、实验附录



图表 7 mrpc数据集结果



图表 8 rte数据集结果



图表 9 cola数据集结果



图表 10 微调效果



图表 11 cnn分类器与mlp。即激活函数、归一化层、dropout、、epoch调参

1. Shen, Dinghan, et al. "Baseline needs more love: On simple word-embedding-based models and associated pooling mechanisms." *arXiv preprint arXiv:1805.09843* (2018).
2. Ma, Xiaofei, et al. "Universal text representation from bert: An empirical study." *arXiv preprint arXiv:1910.07973* (2019).
3. Zhou, Yichu, and Vivek Srikumar. "A closer look at how fine-tuning changes BERT." *arXiv preprint arXiv:2106.14282* (2021).
4. Rogers, Anna, Olga Kovaleva, and Anna Rumshisky. "A primer in BERTology: What we know about how BERT works." *Transactions of the Association for Computational Linguistics* 8 (2021): 842-866.
5. Mohammadi, Samin, and Mathieu Chapon. "Investigating the performance of fine-tuned text classification models based-on BERT." *2020 IEEE 22nd International Conference on High Performance Computing and Communications; IEEE 18th International Conference on Smart City; IEEE 6th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 2020.
6. Mosbach, Marius, Maksym Andriushchenko, and Dietrich Klakow. "On the stability of fine-tuning bert: Misconceptions, explanations, and strong baselines." *arXiv preprint arXiv:2006.04884* (2020).
7. https://github.com/huggingface/transformers/tree/main/examples/pytorch/text-classification