# 复现深度自适应Transformer

## 问题的提出

近年来，基于自注意力机制的深度学习模型在机器翻译上的表现超过了以往算法，已经成为目前机器翻译的主流方法之一。但是自注意力机制以及深层次的网络往往也带来巨大的参数量和计算代价。虽然大型模型在复杂的输入上能获得更好的表现，但在小规模模型在简单的输入上也可以获得很好的效果。因此如何根据输入的复杂程度动态地调整模型的计算量，成为当前的一大研究热点。条件计算（Conditional Computation）正是解决该问题的方法之一。Transformer不考虑输入问题是简单的还是复杂，对每个输入序列执行固定层数的计算。高昂的计算成本使得模型在训练和使用上都变得更加困难，限制了模型的应用场景。并且浅层模型可能在简单的输入上表现也很好。模型所表现出来的深度低效行为，使得如何让模型根据输入的复杂程度，动态地调整所使用的计算量，以适应所执行任务的复杂性，成为未来的一大研究热点。

## 论文介绍

Elbayad等人在《Depth-Adaptive Transformer》一文中提出了深度自适应的方法，在标准Transformer上进行添加随时结构化预测机制以及深度估计模块。在随时结构化预测机制中，分别使用了单个分类器参数和多个分类器参数两种方法，使用对齐训练方法对分类器参数进行优化。在深度估计模块中，使用基于最大可能性和基于最大正确性两种深度估计方法，完成在sequence粒度上模型的深度动态调整。在此基础上设计相关实验，对比不同的训练策略和深度估计方法下模型的翻译效果和计算效率。

2.1 随时结构化预测

为了满足不同的输入可以根据其复杂程度选择不同的计算量，需要使用一个可以在不同的计算层次上输出预测结果的模型，这在计算机视觉领域被称为模型的随时预测。在标准的Transformer模型中，编码器和解码器网络都包含个块，每个编码器块和解码器块都由几个子块组成，每个子块之间可以通过残差进行连接。标准Transformer只有一个单一的输出分类器链接到解码器网络的顶部。我们需要能够在网络的不同层输出预测。为了实现这一点，在标准Transformer的基础上进行了如图1的修改，将多个分类器链接到个解码器块后面，个解码器块输出的中间的隐藏层分别输入到不同的分类器中得到不同层的预测结果，或者将个解码器块都链接到同一个分类器中。这样的好处可以减少参数量，但是会将分类器暴露在不同的解码器块的输出中。

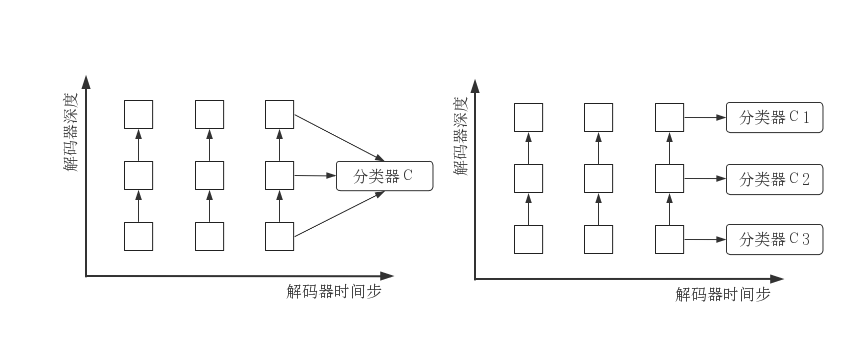


图1 单分类器和多分类器网络结构图

2.2 对齐训练

条件计算使模型能够使用任意个输出分类器进行预测并退出，而不是仅仅是最后一个。模型可以在不同的解码时间步上选择不同层的输出分类器，这会导致在序列长度上可能的输出分类器组合数的呈指数级增长，使得如何训练这些分类器成为一个问题。对齐训练同时优化了所有的分类器，并假设自注意力所需的所有先前的隐藏状态都是可用的。对齐训练在计算自注意力时假设所有隐藏状态都是可用的，并且使用个损失项来进行优化，每一个损失项代表了每一层输出分类器的损失：

|  |  |  |
| --- | --- | --- |
|  |  |  |

对于一个给定的样本，首先将第层中每一个解码时间步中输出分类器得到的目标token的log分布相加，得到第层的分类器的损失，再将每一个层的损失相加得到最后的复合损失，同时优化了所有的输出分类器：

|  |  |  |
| --- | --- | --- |
|  |  |  |

2.3 自适应深度估计

为了实现模型对不同复杂程度的输入使用不同的计算量进行处理，需要一种机制来预测模型将在哪一个解码器块停止并输出下一个token，或者在模型的哪一个层退出就可以实现在速度和效果之间很好的权衡。sequence-specific的深度估计方法使用相同的块解码一个序列中所有的token。用参数分布来建模时间步长的退出分布，其中是计算块的概率，通过计算和目标分布的交叉熵来进行优化参数，并最后逼近我们所需要的块的目标分布：

|  |  |  |
| --- | --- | --- |
|  |  |  |

退出损失反向传播到编码器解码器参数。同时优化解码损失和退出损失，两个损失通过一个超参数进行平衡，以确保模型保持良好的生成精度。

在下面将描述每种深度估计方法如何建模退出分布，以及如何推断目标分布，在本研究中使用sequence-specific深度估计方法以及基于最大正确性和最大可能性两个oracle构建目标分布。对于sequence-specific的深度，退出分布和预测分布与时间步长无关，所以我们去掉下标。

|  |  |  |
| --- | --- | --- |
|  |  |  |

其中，和是停止机制的权重和偏移量。使用两个oracle来确定应该选择个方块中的哪一个。第一种是基于最大可能性，第二种是基于最大正确性。

基于最大可能性（likelihood-based）：这个oracle是基于每个块后的整个序列的可能性，我们用以序列可能性最高的退出层号为中心，使用狄拉克函数将其转化为分布的形式。并且在最后添加了一个正则化项，当模型选择较高的层数时损失的惩罚也越大，这样可以鼓励模型选择较低层的退出，让达到良好的计算效率和效果之间的平衡：

|  |  |  |
| --- | --- | --- |
|  |  |  |

基于最大正确性（correctness-based）：最大正确性oracle选择最大的正确预测token数目的最低的块。对于每个块，我们计算序列上正确预测的token的数量，并选择具有正确token数量最多的块。同样添加一个正则化项控制了效率和准确性之间的权衡：

|  |  |  |
| --- | --- | --- |
|  |  |  |

## 代码实现

在复现过程中主要使用的工具包为Fairseq。Fairseq是一个用Python编写的Facebook人工智能研究序列到序列工具包，集成了常见的处理文本的一些网络模型，比如使用self-attention的Transformer。它允许研究人员和开发人员为翻译、摘要、语言建模和其他文本生成任务训练自定义模型。该工具包基于PyTorch，支持跨多个GPU和机器的分布式培训，还支持在现代GPU上的快速混合精度训练和推理。

3.1 模型

模型主要使用Fairseq工具包中的Model模块，在原本transformer.py文件的基础上进行修改，并且在工具包中注册自己的架构architecture。分布在其中的Transformer Encoder类以及TransformerDecoder类中进行修改。首先注册自己的模型架构depth\_adaptive\_iwslt\_de\_en，在其中完成模型维数、多头注意力数、模型层数、隐藏层维度、relu、dropout等参数设置。

在TransformerEncoder类中添加深度估计模块的定义，将所有token的值相加后送入该网络self.predict\_n()中，并且修改TransformerEncoder的返回字典，在其中添加深度估计单元的输出'predict\_n'。在TransformerDecoder类中，首先在option.py文件中添加判断使用单个分类器还是多个分类器的参数args.classifier。若使用多个分类器 ，则先需要建立参数列表torch.nn.ParameterList()。在forward()中，为了得到所有的中间的隐藏层的状态，首先建立inner\_states列表和norm\_hidden\_x列表用于保存每一个层的输出。之后，根据使用相同分类器还是多个分类器进行计算，若使用相同分类器，则将inner\_states每一个层的输出，若使用不同分类器，则需要和self.embed\_out参数列表中的对应的分类器参数进行计算。

3.2 损失计算

模型的训练和损失计算由train、trainer、task、criterion以及optimizer几个部分共同完成。其中train为主调模块，负责构建模型，构建任务，构建criterion，处理训练数据，载入参数，控制训练何时停止，保存模型等。trainer负责具体的一个训练步train\_step，从train获取样本和模型后，调用task和criterion计算损失，保存训练记录，并且通过optimizer完成向后传播，更新参数。task存储字典，并为加载迭代数据集，初始化模型和criterion和为计算损失提供帮助。optimizer负责根据梯度更新模型参数。

3.3 推理

模型的大批量推理翻译使用generate.py和sequence\_generator.py。在generate.py中，首先完成构建任务，构建generator，载入模型和参数，载入字典，通过task的get\_batch\_iterator()读取字典的数据构建batch迭代器，构建scorer来计算翻译的BLEU值，设置TimeMeter()用于记录推理所花费的时间。此外，为了对模型的平均退出层数进行统计，设置avg\_token字典用于记录翻译的样本数和退出层数的总和。由于推理的过程不需要计算模型的梯度，因此设置torch.no\_grad()函数，调用generator进行实际的推理。

在SequenceGenerator的generate()函数中首先使用EnsembleModel()对传入的model进行重载，然后从样本sample中构建编码器网络的输入encoder\_input。sample是个字典，sample['net\_input']是其中的一个字典，包含了3个键，分别是src\_tokens，src\_lengths，prev\_output\_tokens。在字典sample['net\_input']中遍历，键值分别赋值给k和v，取除了prev\_output\_tokens的全部键值对。之后通过ne函数去除src\_lengths的终止的标签eos和padding。

1. 测试

4.1 实验设置

在模型的训练和测试使用的数据集为机器翻译的基准数据集IWSLT’14 German to English (De-En)。其中字典大小为10152，训练集大小为160K，验证集大小为7K，测试集的大小为6K，平均句子长度为23个tokens。

模型的架构上使用=6个层，前馈网络FFN的中间层为1024维，多头注意力使用4个头，attention dropout为0.1，relu dropout为0.1，编码器的embed维度为512维，解码器的embed维度为256维。

4.2 训练

在模型的训练过程中，需要对模型编解码参数以及深度估计模块进行优化，并且模型的各个模块之间的参数相互依赖，因此，当分类器参数欠拟合时，深度估计模块即使损失很小，但实际与我们真实的目标也还是差距很大。所以在本研究中首先只对模型的编解码参数进行训练，不训练模型的深度估计损失。分别使用对齐训练的单分类器模型和多分类器训练5万次更新。在模型的编解码参数和分类器进行训练完毕之后，在使得超参数等于0.5，并且将深度估计loss变化到和解码损失同一数量级中，再次对深度估计单元进行训练，再次训练次数为20000次。模型的损失变化如下图：

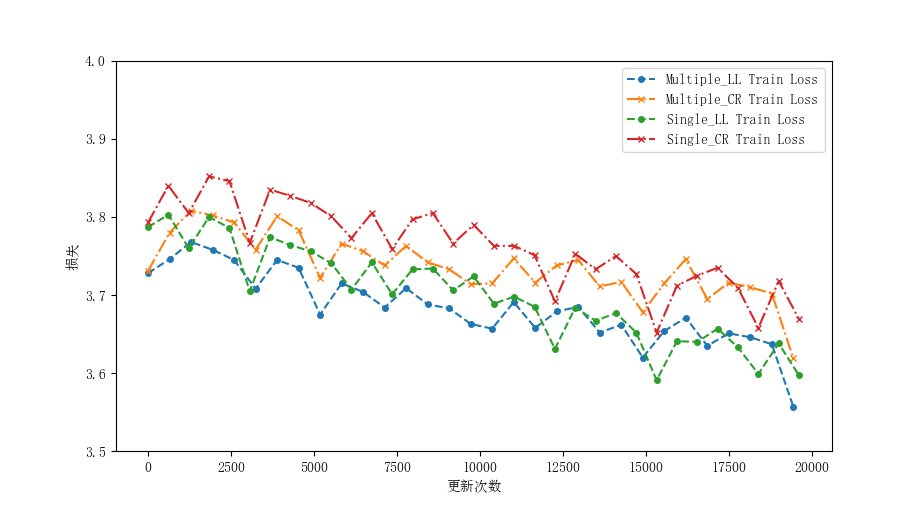
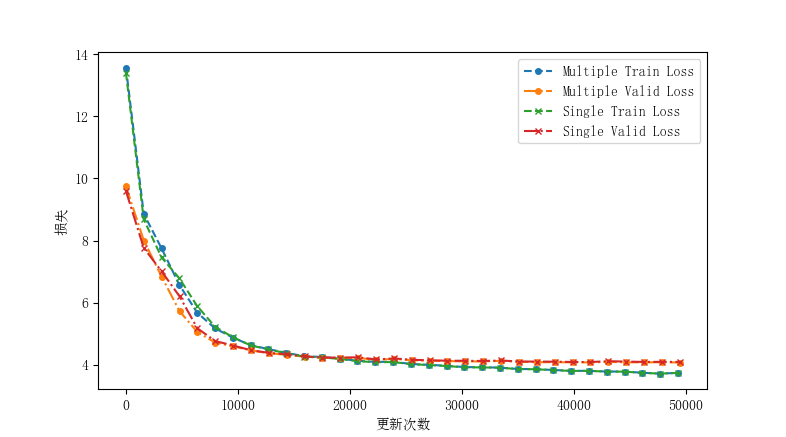


图2 对齐训练损失变化图

4.3 实验结果

为了考察使用对齐训练的方法对模型效果的影响，首先我们将标准Transformer模型作为baseline，与使用对齐训练的单分类器模型和多分类器模型Align\_进行对比，对比三者在不同的层上的BLEU值。

表1模型在不同的层上的BLEU

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | n = 1 | n = 2 | n = 3 | n = 4 | n = 5 | n = 6 | 平均 |
| Baseline | 34.2 | 35.3 | 35.6 | 35.7 | 35.6 | 35.9 | 35.4 |
| Align Single | 33.2 | 34.2 | 34.6 | 34.6 | 34.8 | 34.8 | 34.4 |
| Align Multiple | 33.2 | 34.5 | 34.7 | 35.0 | 35.1 | 35.1 | 34.6 |

从表中我们可以观察到，在三个不同的模型中，随着层数的增加，模型的BLEU值也相应增加，说明了在模型的层数堆叠是有效的。在浅层的模型中BLEU值也是不低的，这也在一定程度上支持了我们做深度自适应的可行性。

接下来，在BLEU和计算效率方面比较不同的oracle和分类器数量的模型。我们将计算效率衡量为每个输出token的平均退出层数AE。具体如下表所示：

表2深度自适应模型的BLEU和AE

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | BLEU | Baseline对比 | AE | Baseline对比 |
| Single\_LL | 34.70 | ↓1.97% | 4.95 | ↑17.50% |
| Single\_CR | 34.27 | ↓3.19% | 2.30 | ↑61.66% |
| Multiple\_LL | 34.54 | ↓2.42% | 2.52 | ↑58.00% |
| Multiple\_CR | 34.16 | ↓3.50% | 1.22 | ↑79.67% |

从表中我们可以得知，四个不同的深度自适应模型虽然相比于baseline有轻微的下降，但是计算效率都有显著的提升，其中多分类器最大正确性模型Multiple\_CR的提升效果最为明显，接近80%，但是同时它也是BLEU值下降最大的。从四个不同的模型中的可以看到，计算效率的提升和模型的翻译效果呈现反比，计算效率的提升越大相应的翻译效果也下降得越厉害，这也符合我们的预期。