Vanilla Transformer

为了解决长文本分类的问题，即长距离依赖问题。比如Transformer-XL、XLNet等，这篇Vanilla Transformer在Transformer-XL中被用来做对比。

1. 介绍

主要解决的是字符级语言模型。在使用Transformer之前，自然语言文本的Character-level modeling通常具有如下一些 **挑战/困难**：

- （1）模型学习单词需要“from scratch”从头开始
- （2）自然文本表现出的长甚至超长距离依赖
- （3）基于字符的序列长度比词级别的序列显著增加，需要更多的计算step

作者发现

（1）transformer适合于长序列的语言建模。作者推测，transformer在这里的成功是由于它能够“快速”地在任意距离上传播信息

- （2）对basic transformer进行一些修改在这一领域（字符级语言模型）是有益的：增加3个辅助损失auxiliary losses
  - 【Multiple Positions】 at intermediate sequence positions
  - 【Intermedia Layer Losses】 from intermediate hidden representations
  - 【Multiple Targets】 at target positions multiple steps in the future

这些loss加速了收敛，并且使得训练更深的网络成为可能。

Vanilla Transformer只利用了原Transformer的decode的**部分结构**，也就是一个带有mask的attention层+一个ff层。

将 ”一个带有mask的attention层+一个ff层” 称为一个layer，那么Vanilla Transformer一共有64个这样的layer，每一个layer有2个head，model_dim=512，ff层的hidden_units=2048，sequence的长度为512。

使用mask结构是因为语言模型的定义是 $p(x_i|x_0 \times x_1 \times ..... x_{i-1})$ ，也就是根据前 i 个字符预测第 $i+1$ 个字符，如果你已经提前看到了答案（也就是第 $i+1$ 个字符甚至更后面的字符内容），那就没有预测的意义了，这里加mask与原Transformer的decode部分的带有mask的self-attention道理都是一样的。

对于Vanilla Transformer，作者认为它的网络深度太深了，如果只在第一层加入pos embedding，那么经过多层传递，这个信息很容易丢失，所以它是每层都会将上一层的输出与pos embedding加在一起作为下一层的输入，而且，pos embedding是需要学习的。所以，光pos embedding模型就要学习 N*L*dim 个参数，其中N是网络的层数（本文64层），L是上下文的长度(本文512)，dim是embedding的维度（本文=512）。

总之，从结构上来说，Vanilla Transformer没有什么太特别的地方，用的组件都是原Transformer这篇论文中用到的，甚至还精简了一些，无非就是Vanilla Transformer

的网络深度非常深。这个深度导致在训练的时候很难收敛，对于作者使用的一些小trick，这些小trick对于我们以后解决类似的问题是很有帮助的。2.Vanilla Transformer训练时作者的一些小trick.

2.Vanilla Transformer训练的trick

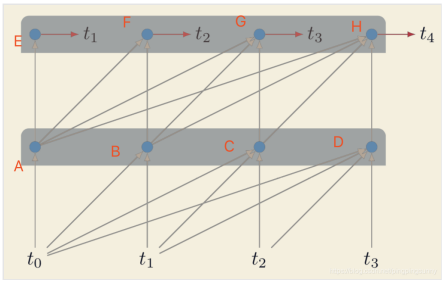
作者在论文中说当网络的深度超过10的时候，就很难让模型收敛，准确率也很低，**引入辅助的loss。**这个辅助的loss分为3类：Multiple Positions； Intermediate Layer Losses； Multiple Targets.

只以2层来展示，且每一个segment(段)的length=4,原本我们是根据 t_0 ~ t_3 的输入，在 H 节点这个位置预测 t_4 的结果， $loss$ 就是 H 节点的输入计算一个交叉熵。

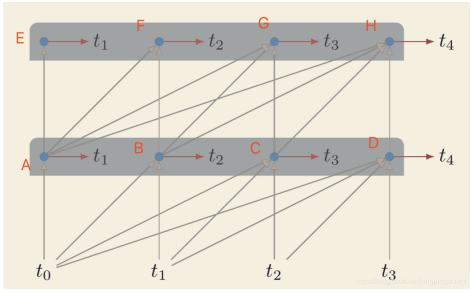
现在辅助 $loss$ 的**第一类loss**就是：对于最后一层所有的节点都计算下一步应该预测的字符，即在节点E处根据输入t0，预测输出为 t_1 ，在节点F处根据输入为 t_0 和 t_1 ，输出是 t_2 ，以此类推。然后将每一个Positions处的loss加起来。

预测时不是指预测最后一个位置，而是序列的每个位置都进行预测，例如上图中 t_1 ， t_2 ， t_3 ， t_4 4个位置都会产生预测然后都会计算loss。训练时， t_1 ~ t_4

产生的loss都不会decay，都是同等重要的。【这一类loss贯穿整个train的全部阶段，不发生衰减】



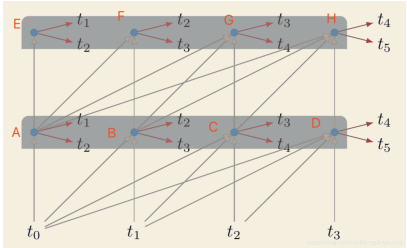
1



2

辅助loss的第二类是**除了在最后一层计算交叉熵loss之外，在中间层也要计算。**即在节点A处根据输入 t_0 ，预测输出为 t_1 ，以此类推，但中间层的loss并不贯穿整个train始终，而是随着训练进行，逐渐衰减，衰减的方式是，一共有n层网络，当训练进行到 $\frac{k}{2n}$ 时停止计算第 k 层loss。也就是说当训练进行到一半的时候，所有的 中间层 都不再贡献loss。

辅助loss的第三类是**每次预测时所预测几个字符**，在本论文中，每次预测下一步和下下步的字符结果，具体的看下面的图即可，非常清楚。但对于下下步的预测结果产生的loss是要发生衰减的，论文中该loss乘以0.5后再加入到整体的loss中。



3.Vanilla Transformer的相关结果

作者这篇论文里提到了加了3种辅助loss帮助训练，还有就是作者使用了moment一um优化器训练，使用的pos embedding也是跟之前不同的。那么这些因素到底有没有用，如果有用，哪个用处大，有多大？针对这个问题作者进行了一个比较，比较的基线是上面讲的64层模型。

辅助loss中的Multiple Positions和Intermediate Layer Losses效果是最明显的，至于使用了需要学习的pos embedding并没有太大的作用，优化器和Multiple Targets的辅助loss感觉效果都不大。

明显看出是这2条因素影响模型的结果最大

Description	bpc	Δbpc
T64 (Baseline)	1.062	-
T64 w/out Multiple Positions	2.482	1.420
T64 w/out Intermediate Layer Losses	1.158	0.096
T64 w/out Positional Embeddings	1.069	0.007
T64 w/out Multiple Targets	1.068	0.006
T64 w/ SGD Optimizer	1.065	0.003

Table 4: Evaluation of T64 on `text8` dev with context set to 512. Disabling each feature or loss lowers the quality of the model. The biggest win comes from adding multiple positions and intermediate layers losses.