

Transformer

一、Transformer

1. Transformer 是一种新的、基于 attention 机制来实现的特征提取器，可用于代替 CNN 和 RNN 来提取序列的特征。

Transformer 首次由论文《Attention Is All You Need》提出，在该论文中 Transformer 用于 encoder - decoder 架构。事实上 Transformer 可以单独应用于 encoder 或者单独应用于 decoder。

2. Transformer 相比较 LSTM 等循环神经网络模型的优点：

- 可以直接捕获序列中的长距离依赖关系。
- 模型并行度高，使得训练时间大幅度降低。

1.1 结构

1. 论文中的 Transformer 架构包含了 encoder 和 decoder 两部分，其架构如下图所示。

- Transformer 包含一个编码器 encoder 和一个解码器 decoder。
- 编码器 encoder 包含一组 6 个相同的层 Layer，每层包含两个子层 subLayer。
 - 第一个子层是一个多头自注意力 multi-head self-attention 层，第二个子层是一个简单的全连接层。
 - 每个子层都使用残差直连，并且残差直连之后跟随一个 layer normalization:LN。

假设子层的输入为 \vec{h} ，则经过 LN 之后整体的输出为：

$$\text{LayerNorm}(\vec{h} + \text{Sublayer}(\vec{h})).$$

为了 Add 直连，论文将内部所有层的输入、输出的向量维度设置为 $d_{model} = 512$ 维。

- 解码器 decoder 也包含一组 6 个相同的层 Layer，但是每层包含三个子层 subLayer。

- 第一个子层也是一个多头自注意力 multi-head self-attention 层。

但是，在计算位置 i 的 self-attention 时屏蔽掉了位置 i 之后的序列值，这意味着：位置 i 的 attention 只能依赖于它之前的结果，不能依赖它之后的结果。

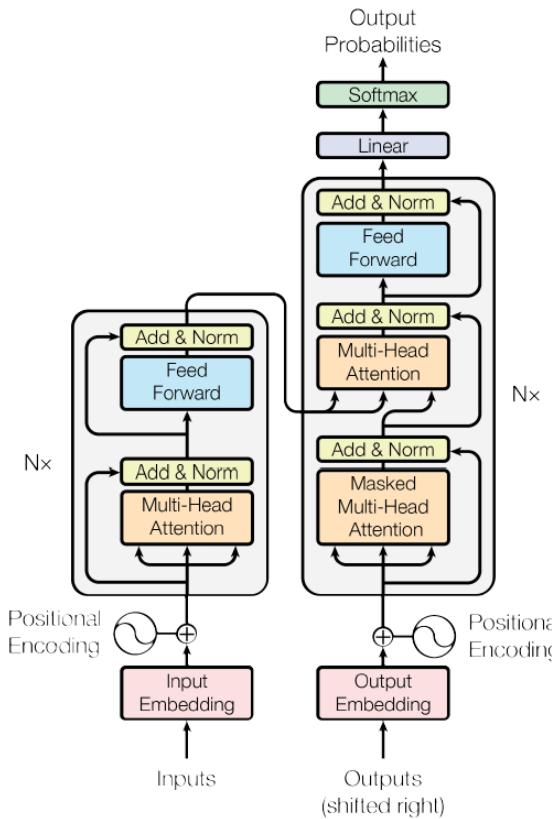
因此，这种 self-attention 也被称作 masked self-attention。

- 第二个子层是一个多头注意力 multi-head attention 层，用于捕获 decoder output 和 encoder output 之间的 attention。

第三个子层是一个简单的全连接层。

- 和 encoder 一样：

- 每个子层都使用残差直连，并且残差直连之后跟随一个 layer normalization:LN。
- decoder 所有层的输入、输出的向量维度也是 $d_{model} = 512$ 维。



1.1.1 attention

1. 编码器和解码器的 `attention` 都是采用 `scaled dot attention`。

设有 M 个 `query` 向量 $\{\vec{q}_1, \dots, \vec{q}_M\}$ 、 M 个 `key` 向量 $\{\vec{k}_1, \dots, \vec{k}_M\}$ 、 M 个 `value` 向量 $\{\vec{v}_1, \dots, \vec{v}_M\}$ ，其中 `query` 向量和 `key` 向量的维度为 d_k ，`value` 向量的维度为 d_v 。经过 `attention` 之后，位置 m 的输出为：

$$\tilde{\vec{v}}_m = \sum_{i=1}^M \alpha_{m,i} \vec{v}_i$$

其中 $\alpha_{m,i}$ 表示位置 m 与位置 i 之间的权重：

$$\text{score}_{m,i} = \frac{\vec{q}_m \cdot \vec{k}_i}{\sqrt{d_k}}, \quad \alpha_{m,i} = \frac{\exp(\text{score}_{m,i})}{\sum_{i'=1}^M \text{score}_{m,i'}}, \quad i = 1, 2, \dots, M$$

- 除以 $\sqrt{d_k}$ 是为了降低 $\text{score}_{m,i}$ 的数值，防止它落入到 `softmax` 函数的饱和区间。

因为 `softmax` 函数的饱和区梯度几乎为 0，容易发生梯度消失。

- 如果使用了 `Masked attention`，则有：

$$\text{score}_{m,i} = \begin{cases} \frac{\vec{q}_m \cdot \vec{k}_i}{\sqrt{d_k}}, & i = 1, 2, \dots, m \\ -\infty, & i = m + 1, \dots, M \end{cases}$$

- 令：

$$\mathbf{Q} = \begin{bmatrix} \vec{q}_1^T \\ \vdots \\ \vec{q}_M^T \end{bmatrix} \in \mathbb{R}^{M \times d_k}, \quad \mathbf{K} = \begin{bmatrix} \vec{k}_1^T \\ \vdots \\ \vec{k}_M^T \end{bmatrix} \in \mathbb{R}^{M \times d_k}, \quad \mathbf{V} = \begin{bmatrix} \vec{v}_1^T \\ \vdots \\ \vec{v}_M^T \end{bmatrix} \in \mathbb{R}^{M \times d_v}$$

则有：

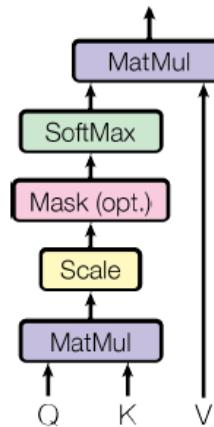
$$\mathbf{QK}^T = \begin{bmatrix} \vec{\mathbf{q}}_1 \cdot \vec{\mathbf{k}}_1 & \vec{\mathbf{q}}_1 \cdot \vec{\mathbf{k}}_2 & \cdots & \vec{\mathbf{q}}_1 \cdot \vec{\mathbf{k}}_M \\ \vec{\mathbf{q}}_2 \cdot \vec{\mathbf{k}}_1 & \vec{\mathbf{q}}_2 \cdot \vec{\mathbf{k}}_2 & \cdots & \vec{\mathbf{q}}_2 \cdot \vec{\mathbf{k}}_M \\ \vdots & \vdots & \ddots & \vdots \\ \vec{\mathbf{q}}_M \cdot \vec{\mathbf{k}}_1 & \vec{\mathbf{q}}_M \cdot \vec{\mathbf{k}}_2 & \cdots & \vec{\mathbf{q}}_M \cdot \vec{\mathbf{k}}_M \end{bmatrix}$$

令: $\mathbf{S} = \text{softmax}\left(\frac{\mathbf{QK}^T}{\sqrt{d_k}}\right) \in \mathbb{R}^{M \times M}$, 其中 softmax 沿着矩阵的行归一化。令:

$$\tilde{\mathbf{V}} = \begin{bmatrix} \tilde{\vec{\mathbf{v}}}^T_1 \\ \vdots \\ \tilde{\vec{\mathbf{v}}}^T_M \end{bmatrix} \in \mathbb{R}^{M \times d_v}$$

则有: $\tilde{\mathbf{V}} = \mathbf{SV}$ 。即:

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



2. 事实上, 一组多个 attention 的效果要优于单个 attention, 这称作 multi-head attention。

给定 query 矩阵 \mathbf{Q} 、key 矩阵 \mathbf{K} 、value 矩阵 \mathbf{V} , multi-head attention 的 head i 先通过一个线性映射然后再经过 attention, 得到 head i 的输出 $\tilde{\mathbf{V}}_i$:

$$\tilde{\mathbf{V}}_i = \text{Attention}(\mathbf{QW}_i^Q, \mathbf{KW}_i^K, \mathbf{VW}_i^V) \in \mathbb{R}^{M \times d_{v'}}$$

其中:

- $\mathbf{W}_i^Q \in \mathbb{R}^{d_k \times d_{k'}}$ 将 M 个 query 向量 $\vec{\mathbf{q}}_m$ 从 d_k 维降低到 $d_{k'}$ 维。
- $\mathbf{W}_i^K \in \mathbb{R}^{d_k \times d_{k'}}$ 将 M 个 key 向量 $\vec{\mathbf{k}}_m$ 从 d_k 维降低到 $d_{k'}$ 维。
- $\mathbf{W}_i^V \in \mathbb{R}^{d_v \times d_{v'}}$ 将 M 个 value 向量 $\vec{\mathbf{v}}_m$ 从 d_v 维降低到 $d_{v'}$ 维。

将多个 head i 的输出 $\tilde{\mathbf{V}}_i$ 进行拼接, 并再经过一个线性映射即可得到多头 attention 的结果:

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\tilde{\mathbf{V}}_1, \dots, \tilde{\mathbf{V}}_a) \mathbf{W}^O$$

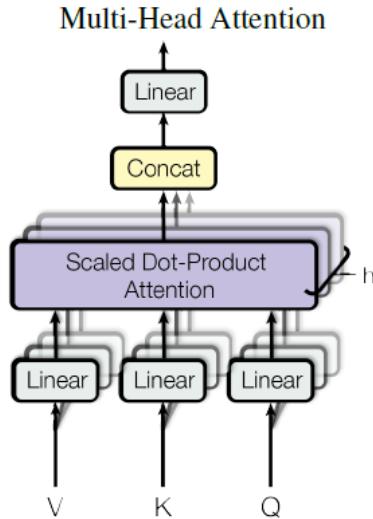
其中:

- a 为 head 的数量, $\mathbf{W}^O \in \mathbb{R}^{(ad_{v'}) \times d_v}$ 是为了确保 multi-head attention 前后的输入输出维度一致。
- concat 操作在 M 个向量上进行:

$$7_Transformer$$

$$Concat(\tilde{\mathbf{V}}_1, \dots, \tilde{\mathbf{V}}_a) = \begin{bmatrix} \tilde{\mathbf{v}}_{1,1}^T & \tilde{\mathbf{v}}_{2,1}^T & \cdots & \tilde{\mathbf{v}}_{a,1}^T \\ \tilde{\mathbf{v}}_{1,2}^T & \tilde{\mathbf{v}}_{2,2}^T & \cdots & \tilde{\mathbf{v}}_{a,2}^T \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{\mathbf{v}}_{1,M}^T & \tilde{\mathbf{v}}_{2,M}^T & \cdots & \tilde{\mathbf{v}}_{a,M}^T \end{bmatrix} \in \mathbb{R}^{M \times (ad_{v'})}$$

其中 $\tilde{\mathbf{v}}_{i,m}$ 为第 i 个 head 的第 m 个输出向量。



3. 论文中选择 $d_k = d_v = d_{model} = 512$ 。为了保证 multi-head attention 的表达空间与 single-head attention 一致，论文中选择：

$$a = 8, \quad d_{k'} = d_{v'} = \frac{d_{model}}{a} = 64$$

4. multi-head attention 将整个 attention 空间拆分成多个 attention 子空间，其表达能力更强。

从原理上看，multi-head 相当于在整体计算代价几乎保持不变的条件下，引入了更多的非线性从而增强了模型的表达能力。

5. 在论文中，有三种方式使用多头注意力机制：

- encoder-decoder attention : query 来自前一个 decoder 层的输出，keys, values 来自 encoder 的输出。

其意义是：decoder 的每个位置去查询它与 encoder 的哪些位置相关，并用 encoder 的这些位置的 value 来表示。

- encoder self-attention : query, key, value 都来自前一层 encoder 的输出。

这允许 encoder 的每个位置关注 encoder 前一层的所有位置。

- decoder masked self-attention : query, key, value 都来自前一层 decoder 的输出。

这允许 decoder 的每个位置关注 encoder 前一层的、在该位置之前的所有位置。

1.1.2 全连接层

1. encoder 和 decoder 还包含有全连接层。对 encoder/decoder 的每个 attention 输出，全连接层通过一个 ReLU 激活函数以及一个线性映射：

$$\vec{o}_m = \mathbf{W}_2 \max(0, \mathbf{W}_1 \tilde{\mathbf{v}}_m + \vec{b}_1) + \vec{b}_2$$

- 对于同一个 multi-head attention 的所有 M 个输出，采用相同的参数；对于不同的 multi-head attention 的输出，采用不同的参数。

- 输入和输出的维度保持为 $d_{model} = 512$, 但是中间向量的维度是 2048 维, 即 $\mathbf{W}_1 \in \mathbb{R}^{2048 \times 512}, \mathbf{W}_2 \in \mathbb{R}^{512 \times 2048}$ 。这是为了扩充中间层的表示能力, 从而抵抗 ReLU 带来的表达能力的下降。

1.1.3 embedding 层

1. 网络涉及三个 `embedding` 层:

- `encoder` 输入 `embedding` 层: 将 `encoder` 输入 `token` 转化为 d_{model} 维的向量。
- `decoder` 输入 `embedding` 层: 将 `decoder` 输入 `token` 转化为 d_{model} 维的向量。
- `decoder` 输出 `embedding` 层: 将 `decoder` 输出 `token` 转化为 d_{model} 维的向量。

在论文中这三个 `embedding` 矩阵是共享的, 并且论文中在 `embedding` 层将该矩阵乘以一个常量 $\sqrt{d_{model}}$ 来放大每个权重。

1.1.4 position embedding

1. 从 `attention` 的计算公式 $\tilde{\mathbf{v}}_m = \sum_{i=1}^M \alpha_{m,i} \mathbf{v}_i$ 可以看到: 调整输入的顺序对 `attention` 的结果没有任何影响。即: `attention` 的输出中不包含任何顺序信息。

事实上输入的顺序对于很多任务是非常重要的, 比如 `我喜欢喝牛奶, 而不喜欢喝咖啡` 与 `我喜欢喝咖啡, 而不喜欢喝牛奶` 的语义完全不同。

2. 论文通过将位置编码添加到 `encoder` 和 `decoder` 底部的输入 `embedding` 来解决问题。即有:

$$\begin{aligned}\tilde{\mathbf{v}}_m &= \sum_{i=1}^M \alpha_{m,i}^p \mathbf{v}_i^p \\ \mathbf{v}_i^p &= \mathbf{v}_i + \vec{\mathbf{p}}_i\end{aligned}$$

其中 $\vec{\mathbf{p}}_i \in \mathbb{R}^{d_{model} \times 1}$ 为位置 i 的 `position embedding`, $\alpha_{m,i}^p$ `attention` 权重与位置有关。

对于同一个输入序列如果打乱序列顺序, 则不同 `token` 的 `attention` 权重发生改变使得 `attention` 的结果不同。

3. 位置编码有两种选择:

- 可以作为参数来学习, 即: 将 `encoder` 的每个输入的位置 `embedding`、`decoder` 的每个输入的位置 `embedding` 作为网络的参数, 这些参数都从训练中学得。
- 也可以人工设定固定值。论文中使用:

$$\begin{aligned}\vec{\mathbf{p}}_i &= (p_{i,1}, p_{i,2}, \dots, p_{i,d_{model}})^T \\ p_{i,2j} &= \sin\left(\frac{i}{10000^{2j/d_{model}}}\right), \quad p_{i,2j+1} = \cos\left(\frac{i}{10000^{2j/d_{model}}}\right)\end{aligned}$$

其中 $i = 1, 2, 3, \dots$ 表示位置编号, $j = 0, 1, 2, \dots, d_{model}/2$ 表示向量的维度。

- 不同的维度对应不同的波长的正弦曲线, 波长从 2π 到 2000π 。
 - 选择这样的函数是因为: 不同位置之间的 `embedding` 可以简单的相互表示。即: 对于给定的偏移量 k , $\vec{\mathbf{p}}_{i+k}$ 可以表示为 $\vec{\mathbf{p}}_i$ 的线性函数。
- 这意味着模型可以捕获到位置之间的相对位置关系。

4. 论文的实验表明: 通过参数学习的 `position embedding` 的效果和采用固定的 `position embedding` 相差无几。

另外, 固定方式的 `position embedding` 可以在测试阶段处理那些超过训练序列长度的测试序列。

1.2 Transformer vs CNN vs RNN

1. 假设输入序列长度为 n , 每个元素的维度为 d : $\{\vec{x}_1, \dots, \vec{x}_n\}, \vec{x}_i \in \mathbb{R}^d$ 。输出序列长度也为 n , 每个元素的维度也是 d : $\{\vec{y}_1, \dots, \vec{y}_n\}, \vec{y}_i \in \mathbb{R}^d$ 。

可以从每层的计算复杂度、并行的操作数量、学习距离长度三个方面比较 `Transformer`、`CNN`、`RNN` 三个特征提取器。

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(\log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- 每层的计算复杂度:

- 考虑到 n 个 `key` 和 n 个 `query` 两两相乘, 因此 `self-attention` 每层计算复杂度为 $O(n^2 \times d)$
- 考虑到矩阵 (维度为 $n \times n$) 和输入向量相乘, 因此 `RNN` 每层计算复杂度为 $O(n \times d^2)$
- 对于 k 个卷积核经过 n 次一维卷积, 因此 `CNN` 每层计算复杂度为 $O(k \times n \times d^2)$ 。如果考虑深度可分离卷积, 则计算复杂度下降为 $O(k \times n \times d + n \times d^2)$ 。

因此:

- 当 $n \leq d$ 时, `self attention` 要比 `RNN` 和 `CNN` 快, 这也是大多数常见满足的条件。
- 当 $n > d$ 时, 可以使用受限 `self attention`, 即: 计算 `attention` 时仅考虑每个输出位置附近窗口的 r 个输入。这带来两个效果:
 - 每层计算复杂度降为 $O(r \times n \times d)$
 - 最长学习距离降低为 r , 因此需要执行 $O(n/r)$ 次才能覆盖到所有输入。

- 并行操作数量: 可以通过必须串行的操作数量来描述。

- 对于 `self-attention`, `CNN`, 其串行操作数量为 $O(1)$, 并行度最大。
- 对于 `RNN`, 其串行操作数量为 $O(n)$, 较难并行化。

- 最长计算路径: 覆盖所有输入的操作的数量。

- 对于 `self-attention`, 最长计算路径为 $O(1)$; 对于 `self-attention stricted`, 最长计算路径为 $O(n/r)$ 。
- 对于常规卷积, 则需要 $O(n/k)$ 个卷积才能覆盖所有的输入; 对于空洞卷积, 则需要 $O(\log_k n)$ 才能覆盖所有的输入。
- 对于 `RNN`, 最长计算路径为 $O(n)$ 。

2. 作为额外收益, `self-attention` 可以产生可解释性的模型: 通过检查模型中的注意力分布, 可以展示与句子语法和语义结构相关的信息。

1.3 实验结果

1. 训练包含两个训练数据集:

- `WMT 2014 English - German` 数据集: 包含 450 万句子对, 每个句子通过 `byte-pair encoding:BPE` 编码。
 - `BPE` 编码首先将所有文本转换为 `unicode bytes`, 然后将其中出现次数较高的、连续的一组 `bytes` 用一个 `token` 替代。
 - 源句和翻译句共享词汇表, 包含 37000 个 `token`。
- `WMT 2014 English-French` 数据集: 包含 3600 万句子对, 每个句子通过 `BPE` 编码。源句和翻译句共享词汇表, 包含 32000 个 `token`。

2. 训练在 8 NVIDIA P100 GPU 上，相近大小的句子对放到一个 batch 里，每个训练 batch 包含大约 25000 个源 token 和 25000 个翻译 token。

- 优化器采用 Adam optimizer, $\beta_1 = 0.9, \beta_2 = 0.98, \epsilon = 10^{-9}$ 。学习率为：

$$lrate = d_{model}^{-0.5} \times \min(step_num^{-0.5}, step_num \times warmup_steps^{-1.5})$$

其中 $warmup_steps = 4000$ 。

- 训练在基础版本上执行 10 万步约 12 个小时，在大型版本上（更大的参数）执行 30 万步约 3.5 天。
- 通过两种方式进行正则化：

- dropout :

- residual dropout：在每个子层添加到 ADD & LN 之前执行 dropout, $p_{drop} = 0.1$ 。
- embedding dropout：在 encoder 和 decoder 添加 position embedding 之后，立即执行 dropout, $p_{drop} = 0.1$ 。
- label smoothing：训练期间使用 $\epsilon_{ls} = 0.1$ 的 label smoothing。

虽然这会降低模型的困惑度，因为模型学到了更大的不确定性，但是这会提高 BLEU。

3. 在 WMT 2014 English-German/ WMT 2014 English-French 翻译任务上，各种模型的效果如下图：

- 训练参数：见后一个表的最后一行。
- 超参数选择：使用 beam size = 4、长度惩罚因子 $\alpha = 0.6$ 的 beam search，超参数在验证集上选择。
- 训练代价 FLOPs：通过训练时间、训练 GPU 个数、每个 GPU 的单精度浮点计算能力三者相乘得到估计值。
- 对于 base model，采用单个模型的最后 5 个 checkpoint 的平均（10 分钟生成一个 checkpoint）。

对于 big model，采用单个模型的最后 20 个 checkpoint 的平均。

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	41.29	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$3.3 \cdot 10^{18}$	
Transformer (big)	28.4	41.8		$2.3 \cdot 10^{19}$

4. 为了评估不同参数的影响，论文在 English-German 翻译任务中采用不同的模型，评估结果是 newstest2013 验证集上得到。

- beam search 参数同前，但是并没有采用 checkpoint 平均的策略。
- PPL 困惑度是在 word-piece 上得到的。考虑到 BPE 编码，这和 word 级别的困惑度有所出入。
- 为空的参数表示和 base 模型一致。
 - (A) 展示了不同 attention head 和 attention key/value dimension 的影响。为了保持模型参数不变，二者需要满足一定条件。

结果表明：单头的效果要比多头模型差，但是头数太多也会导致效果下降。

- (B) 展示了不同的 `attention key` 的影响。

结果表明：降低 d_k 会带来性能下降。因为降低 d_k 会降低模型的 `attention` 矩阵 $\mathbf{Q}\mathbf{K}^T$ 的容量，从而降低模型的整体表达能力。

- (C) 展示了不同大小模型的影响。

结果表明：模型越大，性能越好。

- (D) 展示了 `dropout` 和 `label smoothing` 的影响。

结果表明：`dropout` 和 `label smoothing` 有助于帮助缓解过拟合。

- (E) 展示了通过学习 `position embedding` (而不是固定的 `position embedding`) 的影响。

结果表明：这两种 `position embedding` 策略的效果几乎相同。

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
							0.0			5.77	24.6	
(D)							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
	(E)									4.92	25.7	
big	6	1024	4096	16			0.3		300K	4.33	26.4	213

二、Universal Transformer

1. `Transformer` 模型存在两个缺陷：非图灵完备；每个 `token` 都计算相同的次数（深度相同）。

`universal Transformer` 是 `Transformer` 的推广，它是一个时间并行的递归自注意力序列模型，解决了上述两个问题。

- 通过引入了递归，因此在内存充足的条件下，它是图灵完备的。
- 通过引入自适应处理时间 `Adaptive Computation Time(ACT)`，它使得不同 `token` 的迭代时间步不同。

2. `Universal Transformer` 结合了 `Transformer` 和 `RNN` 的优势：

- 具有 `Transformer` 计算并行性和全局接受域 `global receptive field` 的优点
- 也具有 `Transformer` 不具备的递归归纳偏差 `recurrent inductive bias`

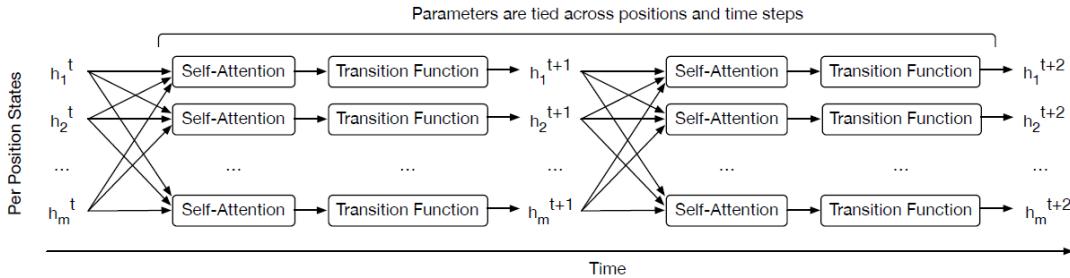
2.1 结构

1. `universal Transformer` 总体计算流程如下图所示：

- 和 `Transformer` 相同，它使用自注意力机制来组合来自不同位置的信息。
- 和 `Transformer` 不同，它在所有的时间步的所有位置上应用一个转换函数 `Transition Function`。

- 和 Transformer 不同，它的时间步不再是固定的，而是一个循环。循环的停止条件由动态停止 dynamic halting 技术决定，并且每个位置的停止时间可以不同。

图中：箭头表示计算的依赖关系， $\vec{h}_1^t, \dots, \vec{h}_m^t$ 表示第 t 个时间步各位置的向量表达。其中 $\vec{h}_1^0, \dots, \vec{h}_m^0$ 由输入 token 的 embedding 来初始化。



2. Universal Transformer 基于 BERT 结构进化而来。

- Universal Transformer 对 encoder 和 decoder 的每个 token 的表达都应用了循环操作，但是这种循环不是基于 token 的序列位置（横向），而是基于 token 的处理深度（纵向）。
 - 因此 universal Transformer 的计算限制不受序列长度的影响，而受网络深度的影响。
 - 因此 universal Transformer 具有可变深度，这与 Transformer、RNN 等具有固定深度的模型完全不同。
- 在每个时间步，对每个 token 的表达并行的执行以下操作：
 - 通过 self-attention 机制来交换序列中所有 token 的信息。
 - 在所有时间步的所有位置上应用同一个转换函数 Transition Function。
 有两种转换函数：全连接、深度可分离卷积。
- 与 Transformer 相同，Universal Transformer 也会引入残差连接、层归一化 Layer Normalization、Dropout。
- 与 Transformer 不同，Universal Transformer 的 position embedding 在每个时间步都发生变化，而不仅仅只有起始时设置一次。

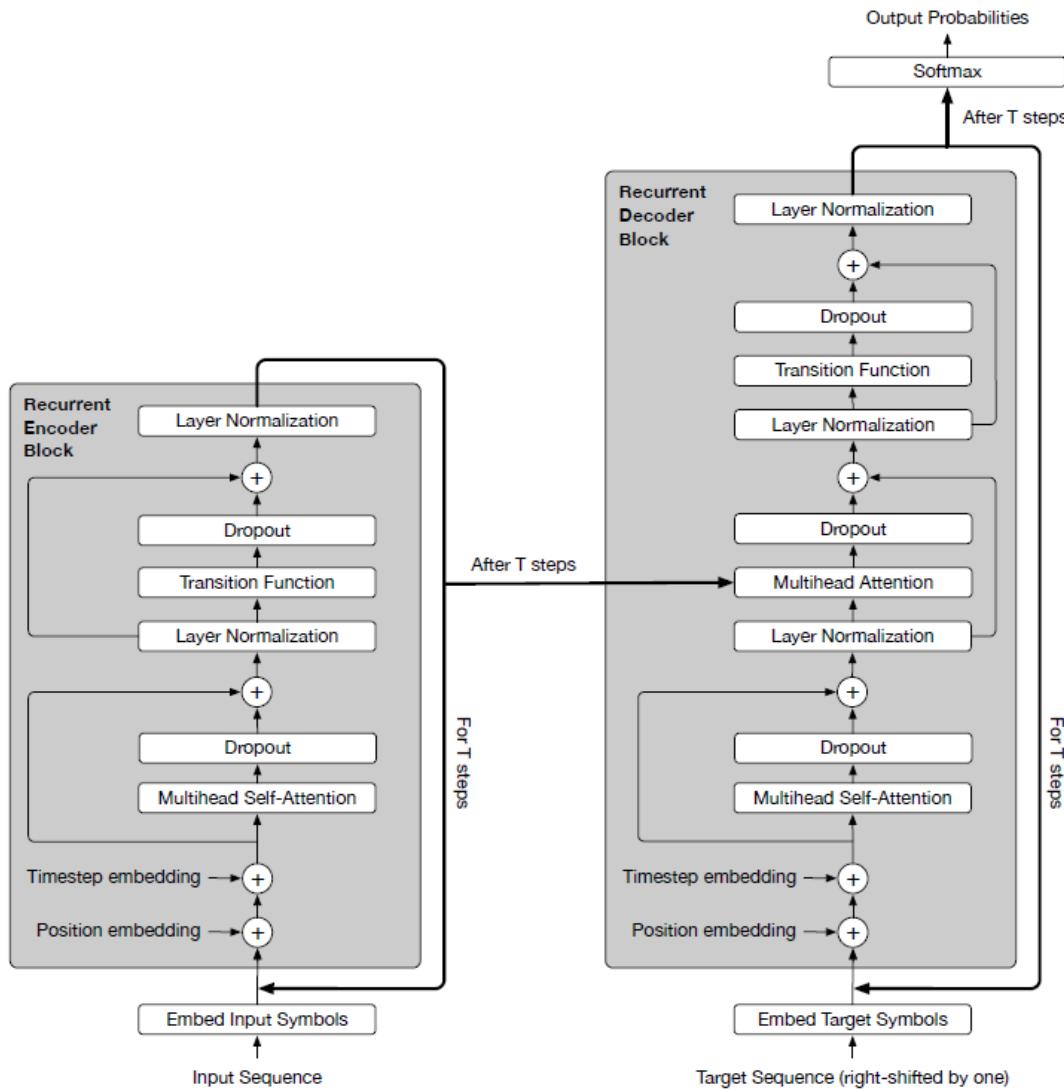
设时间步 t 时，位置 i 的 position embedding 为 \vec{p}_i^t ，则有：

$$p_{i,2j}^t = \sin\left(\frac{i}{10000^{2j/d_{model}}}\right) + \sin\left(\frac{t}{10000^{2j/d_{model}}}\right)$$

$$p_{i,2j+1}^t = \cos\left(\frac{i}{10000^{2j/d_{model}}}\right) + \cos\left(\frac{t}{10000^{2j/d_{model}}}\right)$$

其中 $j = 0, 1, 2, \dots, d_{model}/2$ 表示向量的维度， m 为序列的长度。写成矩阵的形式有：

$$\mathbf{P}^t = \begin{bmatrix} p_{1,1}^t & p_{1,2}^t & \cdots & p_{1,d_{model}}^t \\ p_{2,1}^t & p_{2,2}^t & \cdots & p_{2,d_{model}}^t \\ \vdots & \vdots & \ddots & \vdots \\ p_{m,1}^t & p_{m,2}^t & \cdots & p_{m,d_{model}}^t \end{bmatrix} \in \mathbb{R}^{m \times d_{model}}$$



3. 编码器 `encoder`：给定一个长度为 m 的输入 `token` 序列 $\{w_1, \dots, w_m\}$ ：

- 首先根据每个 `token` 的 d 维 `embedding`，得到一个初始输入矩阵 \mathbf{H}^0 ：

$$\mathbf{H}^0 = \begin{bmatrix} \vec{\mathbf{x}}_1^T \\ \vec{\mathbf{x}}_2^T \\ \vdots \\ \vec{\mathbf{x}}_m^T \end{bmatrix} \in \mathbb{R}^{m \times d}$$

其中 $\vec{\mathbf{x}}_i \in \mathbb{R}^{d \times 1}$ 为 `token` w_i 的一维 `embedding`。

- 然后在时刻 t ，根据 `self attention` 机制和 `Transition Function` 来并行计算 \mathbf{H}^t 。

- `self attention` 机制：

$$\text{Attention}(\mathbf{H}^t, \mathbf{H}^t, \mathbf{H}^t) = \text{softmax} \left(\frac{\mathbf{H}^t (\mathbf{H}^t)^T}{\sqrt{d}} \right) \mathbf{H}^t$$

$$\tilde{\mathbf{V}}_i = \text{Attention}(\mathbf{H}^t \mathbf{W}_i^Q, \mathbf{H}^t \mathbf{W}_i^K, \mathbf{H}^t \mathbf{W}_i^V) \in \mathbb{R}^{m \times (d/k)}, i = 1, 2, \dots, k$$

$$\text{MultiHead}(\mathbf{H}^t, \mathbf{H}^t, \mathbf{H}^t) = \text{Concat}(\tilde{\mathbf{V}}_1, \dots, \tilde{\mathbf{V}}_k) \mathbf{W}^O$$

其中 $\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V \in \mathbb{R}^{d \times (d/k)}$ 分别将 `query`, `key`, `value` 映射成 d/k 维度, $\mathbf{W}^O \in \mathbb{R}^{d \times d}$ 用于调整 `concat` 的输出结果。

- 整体步骤（其中 `LN` 为 `Layer Normalize`, `Trans` 为 `Transition Function`）：

- 计算 t 时刻的输入： $\mathbf{I}^t = \mathbf{H}^{t-1} + \mathbf{P}^t$ 。

- 计算 `self attention` (考虑残差连接和层归一化)：
 $\mathbf{B}^t = LN(\mathbf{I}^t + MultiHead(\mathbf{I}^t, \mathbf{I}^t, \mathbf{I}^t))$ 。
- 计算 `Transition Function` : $\mathbf{H}^t = LN(\mathbf{B}^t + Trans(\mathbf{B}^t))$ 。

4. 解码器和编码器的结构基本相同，有以下不同的地方：

- 与 `Transformer` 相同，`Universal Transformer` 的解码器也采用的是 `masked self-attention` 机制。
- 与 `Transformer` 不同，`Universal Transformer` 的解码器对编码器 τ 步的输出 \mathbf{H}^τ 计算注意力，其中 `query` 来自解码器、`key, value` 来自编码器。
- 设解码器在 τ 步的输出为 \mathbf{H}_{dec}^τ ，则输出 `token` 为 y_j 的概率为：

$$p(y_j | y_1, y_2, \dots, y_{j-1}, \mathbf{H}^\tau) = \text{softmax}(\mathbf{W}\mathbf{H}_{dec}^\tau)$$

其中 j 为输出的位置。

5. `Universal Transformer` 的训练采用 `teacher-forcing` 策略。

- 训练阶段：解码器的输入来自目标序列。由于采用 `masked` 机制，因此每次迭代时序列右边界右移一个位置，表示下一个预测目标。
- 测试阶段：
 - 解码器的输入来自前一个位置的输出。
 - 首先由编码器重复执行多次，得到输入序列的编码 \mathbf{H}^τ ；然后解码器重复执行多次，解码器每次执行会生成下一个 `token`。

2.2 ACT

1. 在文本处理任务中，文本中的某些单词相比较于文本中的其它单词可能更难以理解。对于这些难以理解的单词，可能需要执行更多的处理步骤。
`Adaptive Computation Time(ACT)` 技术就是通过为每个 `token` 计算个性化的迭代时间步来解决这个问题。

2. 在 `Universal Transformer` 中，`ACT` 在每个循环为每个 `token` 计算一个停止概率。

一旦该 `token` 应该停止计算，则该 `token` 的状态直接向后传递，直到所有的 `token` 都停止计算，或者达到一个最大的迭代步阈值。

2.3 实验结果

1. `BABI Question-Answering` 数据集：包含20种不同的任务，每个任务都是根据一段文本来回答问题。该数据集的目标是：通过对每个故事中的事实 `fact` 进行某种类型的推理，从而衡量语言理解能力。

实验结果如下所示，其中：

- `(12/20)` 表示 20 个任务中，失败任务（测试集错误率超过 5%）的数量。
- `10K/1K` 表示训练集的大小。
- `train single` 表示针对每个任务训练一个模型；`train joint` 表示对所有任务共同训练一个模型。
- 所有的模型都采用 10 轮不同初始化，并根据验证集选择最佳初始化对应的模型来输出。

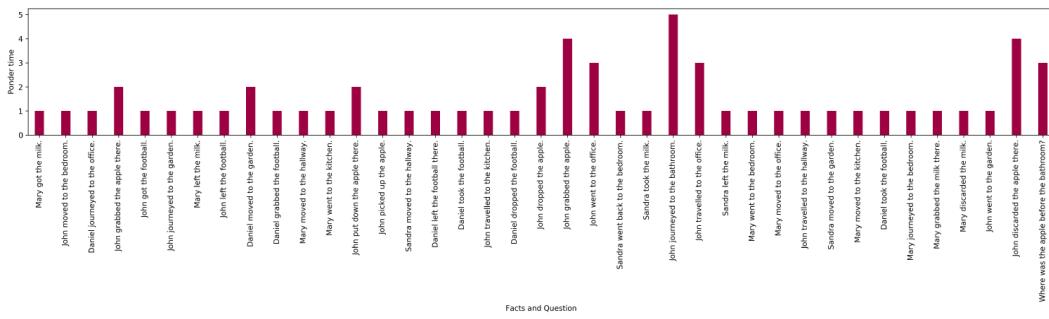
Model	10K examples		1K examples	
	train single	train joint	train single	train joint
Previous best results:				
QRNet (Seo et al., 2016)	0.3 (0/20)	-	-	-
Sparse DNC (Rae et al., 2016)	-	2.9 (1/20)	-	-
GA+MAGE (Dhingra et al., 2017)	-	-	8.7 (5/20)	-
MemN2N (Sukhbaatar et al., 2015)	-	-	-	12.4 (11/20)
Our Results:				
Transformer (Vaswani et al., 2017)	15.2 (10/20)	22.1 (12/20)	21.8 (5/20)	26.8 (14/20)
Universal Transformer (this work)	0.23 (0/20)	0.47 (0/20)	5.31 (5/20)	8.50 (8/20)
UT w/ dynamic halting (this work)	0.21 (0/20)	0.29 (0/20)	4.55 (3/20)	7.78 (5/20)

2. 针对 **BABI** 任务的可视化分析表明：

- 注意力分布开始时非常均匀。但是随后的步骤中，围绕每个答案所需的正确事实 **fact**，注意力分布逐渐变得更加清晰。
- 通过动态暂停 **ACT** 技术观察到：对于需要三个事实 **fact** 支持的问题，其平均思考时间高于只有两个事实 **fact** 支持的问题；对于需要两个事实 **fact** 支持的问题，其平均思考时间又高于只有一个事实 **fact** 支持的问题。
 - 平均思考时间：用一组测试样本每个 **token** 的平均迭代时间步来衡量。
 - 对于三个/两个/一个事实支持的问题，其平均思考时间分别为： 3.8 ± 2.2 、 3.1 ± 1.1 、 2.3 ± 0.8 。
 - 这表明：模型根据问题所需的支持事实 **fact** 的数量来调整迭代时间步。
- 通过动态暂停 **ACT** 技术观察到：对于需要一个事实 **fact** 支持的问题，其不同位置的思考时间更为均匀。对于需要两个/三个事实 **fact** 支持的问题，其不同位置的思考时间差距较大。

尤其在三个事实支持的问题中，很多位置只迭代一到两步就停止，只有少数位置迭代更多步。这意味着：大多数位置与任务无关。

下图为某个三事实支持的问题对应的 **token** 迭代时间步的分布。



3. 针对 **BABI** 任务的可视化分析表明：**Universal Transformer** 类似动态记忆网络 **dynamic memory network**，它具有一个迭代注意力的过程。

Universal Transformer 模型根据前一个迭代的结果（也称作记忆）来调节其注意力分布。

下图为注意力权重在不同时间步上的可视化，左侧不同的颜色条表示不同 **head** 的注意力权重（一共4个头）。

- 一个事实 **fact** 支持的问题：

An example from tasks 1: (requiring one supportive fact to solve)**Story:**

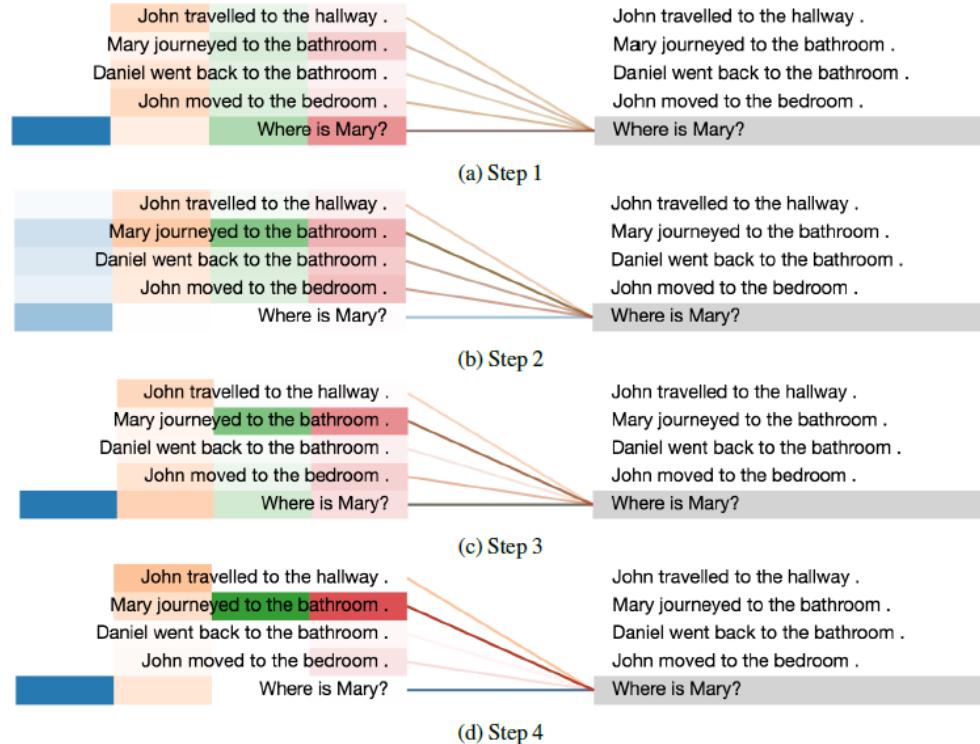
John travelled to the hallway.
 Mary journeyed to the bathroom.
 Daniel went back to the bathroom.
 John moved to the bedroom

Question:

Where is Mary?

Model's output:

bathroom



- 两个事实 `fact` 支持的问题：

An example from tasks 2: (requiring two supportive facts to solve)**Story:**

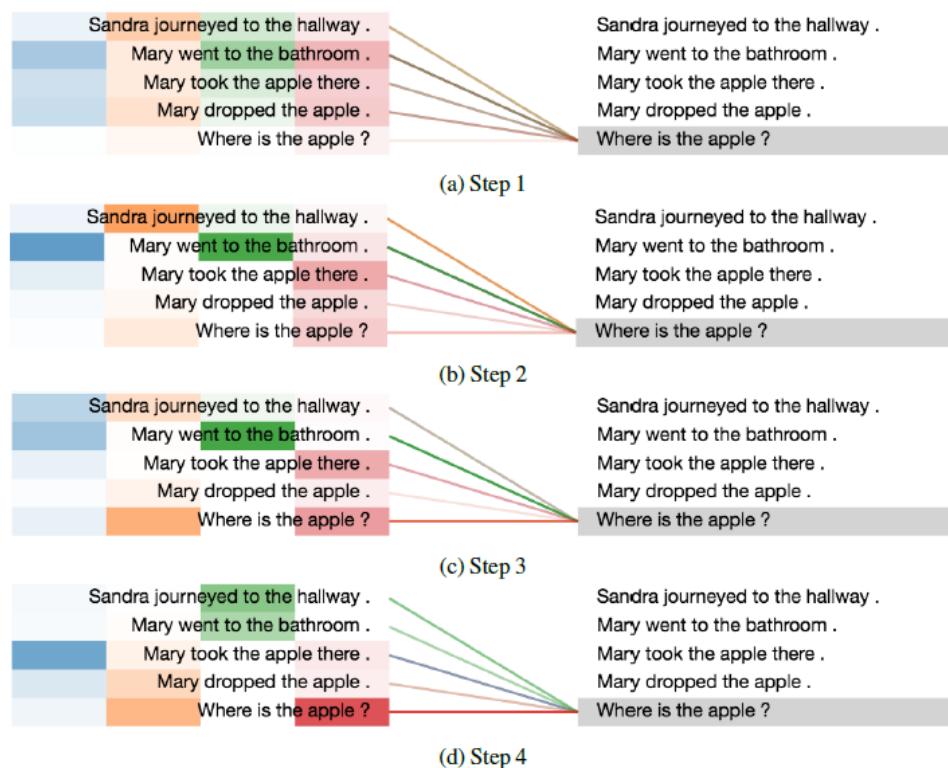
Sandra journeyed to the hallway.
 Mary went to the bathroom.
 Mary took the apple there.
 Mary dropped the apple.

Question:

Where is the apple?

Model's output:

bathroom



An example from tasks 2: (requiring two supportive facts to solve)**Story:**

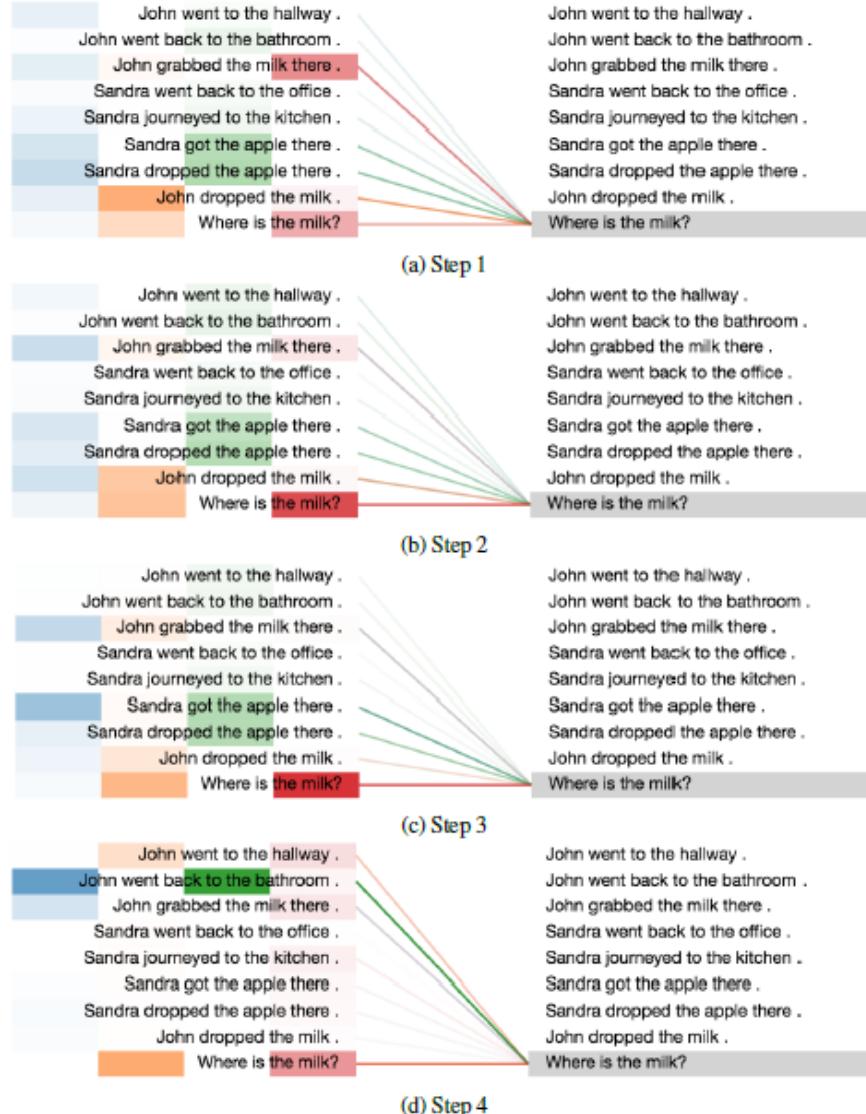
John went to the hallway.
 John went back to the bathroom.
 John grabbed the milk there.
 Sandra went back to the office.
 Sandra journeyed to the kitchen.
 Sandra got the apple there.
 Sandra dropped the apple there.
 John dropped the milk.

Question:

Where is the milk?

Model's output:

bathroom



- 三个事实 `fact` 支持的问题：

An example from tasks 3: (requiring three supportive facts to solve)**Story:**

Mary got the milk.

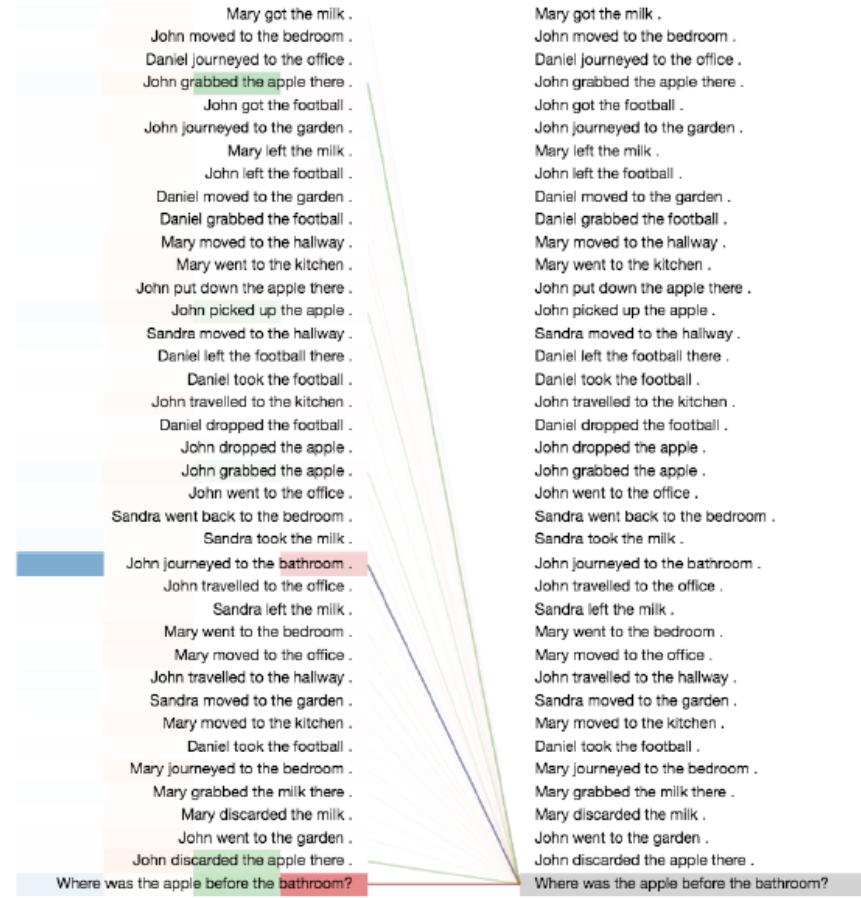
John moved to the bedroom.
Daniel journeyed to the office.
John grabbed the apple there.
John got the football.
John journeyed to the garden.
Mary left the milk.
John left the football.
Daniel moved to the garden.
Daniel grabbed the football.
Mary moved to the hallway.
Mary went to the kitchen.
John put down the apple there.
John picked up the apple.
Sandra moved to the hallway.
Daniel left the football there.
Daniel took the football.
John travelled to the kitchen.
Daniel dropped the football.
John dropped the apple.
John grabbed the apple.
John went to the office.
Sandra went back to the bedroom.
Sandra took the milk.
John journeyed to the bathroom.
John travelled to the office.
Sandra left the milk.
Mary went to the bedroom.
Mary moved to the office.
John travelled to the hallway.
Sandra moved to the garden.
Mary moved to the kitchen.
Daniel took the football.
Mary journeyed to the bedroom.
Mary grabbed the milk there.
Mary discarded the milk.
John went to the garden.
John discarded the apple there.

Question:

Where was the apple before the bathroom?

Model's output:

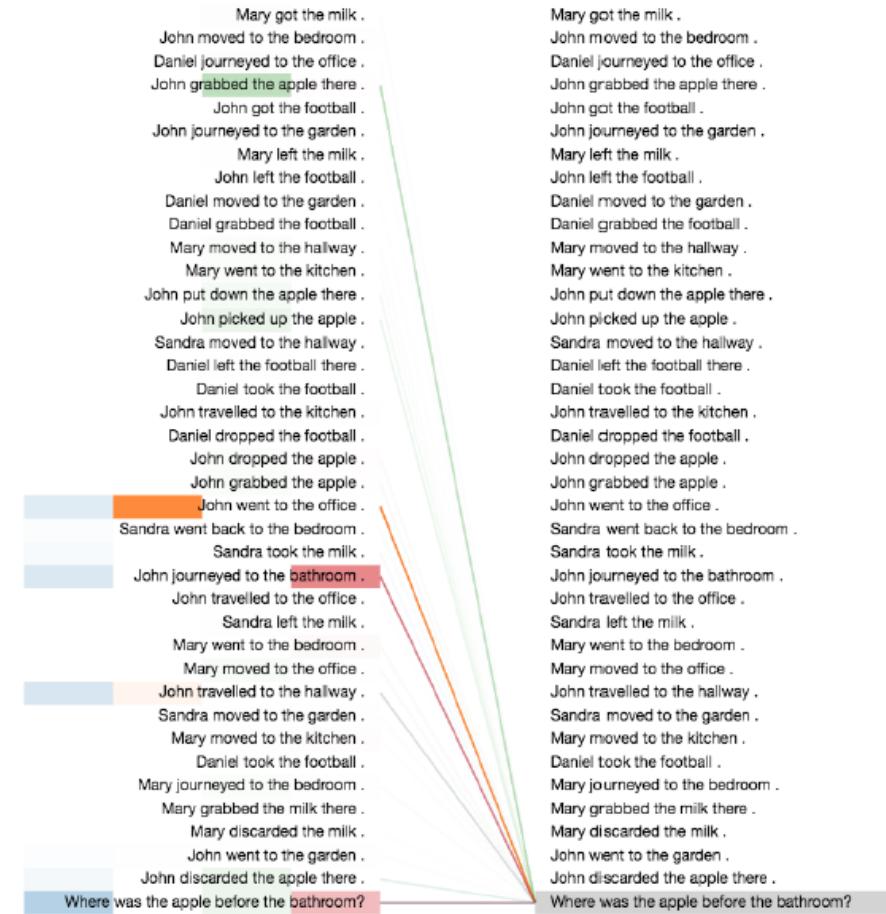
office



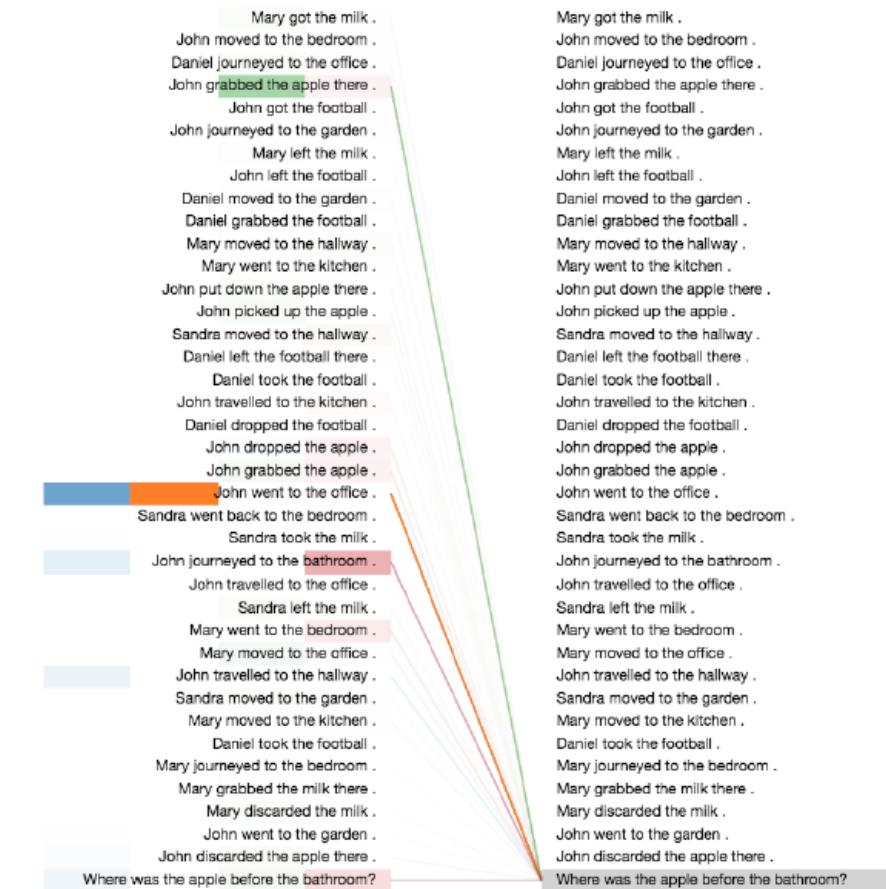
(e) Step 1



(f) Step 2



(g) Step 3

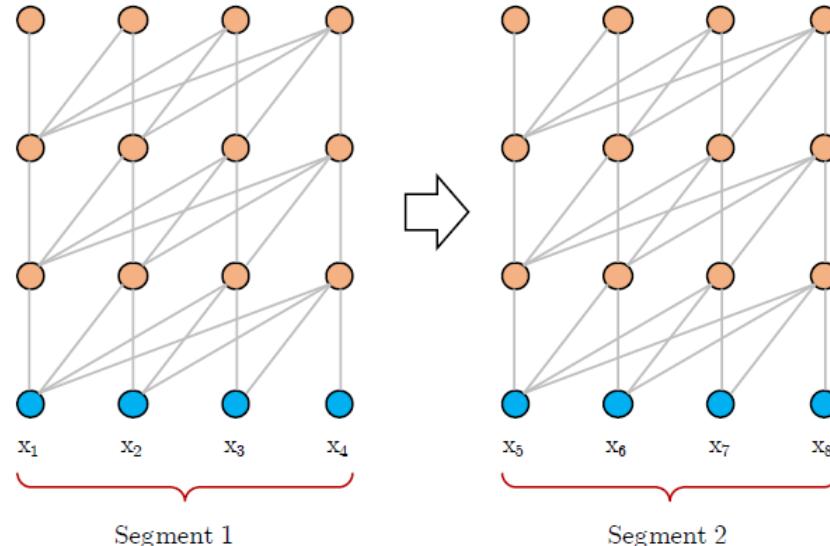


(h) Step 4

三、Transformer XL

1. `Transformer` 解决的核心问题是：如何将任意长度的文本序列编码成一个固定维度的向量。

- 假设没有无限的内存和算力，一个简单的方法是：取语料库中最长序列的长度，然后把所有较短的序列填充到最大长度作为输入。但是由于内存和算力的限制，这种方式不现实。
 - 一个可行的方案是：将整个语料库分割成固定大小的、较短的片段 `segment`，然后用每个片段来训练模型，而忽略之前所有片段的信息。
- 这种模型称作 `vanilla model`，这也是 `Transformer` 的做法。



2. `vanilla model` 没有任何跨 `segment` 的信息流，这会带来两个问题：

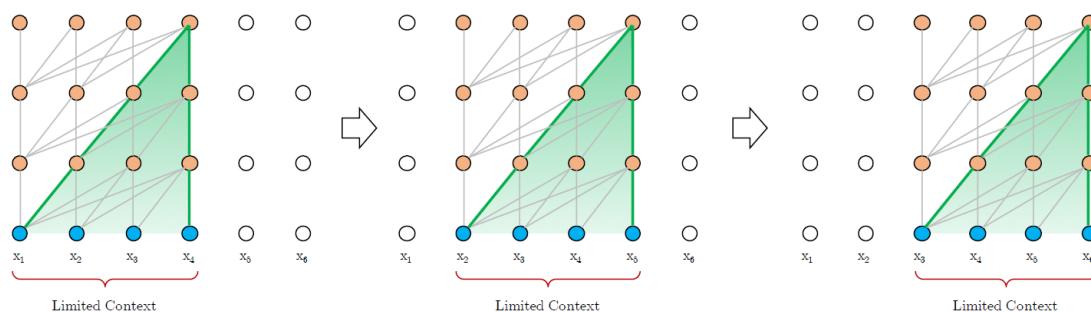
- 模型能够捕获的最大依赖的长度不超过 `segment` 的大小。
假设 `segment` 的长度为 n ，则模型无法捕获长度超过 n 的依赖性。
- 划分 `segment` 的时候未考虑句子边界或者短语的边界，这破坏了语义完整性。因此模型缺乏必要的上下文信息来预测 `segment` 的开头的前几个 `token` 和结束的尾部几个 `token`。这会导致低效的优化效率和更差的泛化能力。

这个问题称作上下文碎片化 `context fragmentation` 问题。

3. `vanilla model` 不仅在训练期间遇到问题，在推断期间也会遇到问题。

`Transformer` 在推断期间反复执行推断过程，每次推断都根据前几轮输入结果和输入来预测下一个输出结果。

- 推断期间，模型也采取与训练期相同的 `segment` 大小，因此也会遇到上下文碎片化问题。
- 每个新的 `segment` 的计算需要从头开始重新计算，计算速度较慢。



4. `Transformer XL` 通过引入递归机制和相对位置编码来解决上述问题。

- 能够学到的最长依赖性的长度：`Transformer XL` 比 `RNN` 长 80%，比 `Transformer` 长 450%。
- 推断速度：`Transformer XL` 比 `Transformer` 快 1800 多倍。

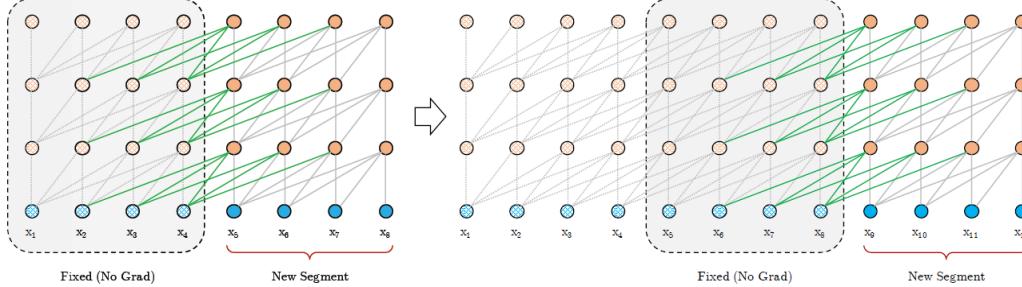
3.1 Segment-level 递归

1. 为解决固定长度上下文的局限性, Transformer XL 引入了 segment-level 递归机制。

- 训练期间: Transformer XL 缓存前一个 segment 计算得到的隐状态序列, 然后在下一个 segment 中重用。

这种额外的输入使得网络能够利用历史中的有效信息, 从而能够对长期依赖建模, 并且避免了上下文碎片化。

- 在对语料库进行分割来生成样本之后, Transformer XL 要求对样本不能进行混洗。因为一旦混洗就会破坏 segment 的先后顺序。
 - 由于需要考虑 segment 之间的先后关系, 因此训练期间要将连续的一组 segment 分别放置在连续的一组 batchment 中。
- 这样可以在尽可能满足 segment 先后关系的条件下提高数据并行度。



- 推断期间: Transformer XL 缓存前一个 segment 计算得到的隐状态序列, 使得后续需要用到这些隐状态时直接使用而不必重新计算, 加快了推断速度。

实验表明, Transformer XL 的推断速度是传统 Transformer 的 1800 倍。

Attn Len	How much Al-Rfou et al. (2018) is slower
3,800	1,874x
2,800	1,409x
1,800	773x
800	363x

2. 令 segment 长度为 n , 第 τ 个 segment 的输入为 token 序列: $\mathbf{x}_\tau = (x_{\tau,1}, x_{\tau,2}, \dots, x_{\tau,n})$; 第 $\tau+1$ 个 segment 的输入为: $\mathbf{x}_{\tau+1} = (x_{\tau+1,1}, x_{\tau+1,2}, \dots, x_{\tau+1,n})$ 。

假设网络有 L 层, 第 l 层网络每个位置的输出隐状态分别为 ($l = 1, 2, \dots, L$):

$$\begin{aligned} \text{for } \tau\text{th segment : } & \{\vec{\mathbf{h}}_{\tau,1}^{(l)}, \dots, \vec{\mathbf{h}}_{\tau,n}^{(l)}\} \\ \text{for } (\tau+1)\text{th segment : } & \{\vec{\mathbf{h}}_{\tau+1,1}^{(l)}, \dots, \vec{\mathbf{h}}_{\tau+1,n}^{(l)}\} \end{aligned}$$

其中 $\vec{\mathbf{h}} \in \mathbb{R}^{d \times 1}$ 为 d 维列向量。令:

$$\mathbf{H}_\tau^{(l)} = \begin{bmatrix} \vec{\mathbf{h}}_{\tau,1}^{(l)T} \\ \vec{\mathbf{h}}_{\tau,2}^{(l)T} \\ \vdots \\ \vec{\mathbf{h}}_{\tau,n}^{(l)T} \end{bmatrix} \in \mathbb{R}^{n \times d}, \quad \mathbf{H}_{\tau+1}^{(l)} = \begin{bmatrix} \vec{\mathbf{h}}_{\tau+1,1}^{(l)T} \\ \vec{\mathbf{h}}_{\tau+1,2}^{(l)T} \\ \vdots \\ \vec{\mathbf{h}}_{\tau+1,n}^{(l)T} \end{bmatrix} \in \mathbb{R}^{n \times d}$$

考虑 Segment-level 递归机制, 则 $\mathbf{x}_{\tau+1}$ 的第 l 层各位置的隐向量为:

- 拼接 \mathbf{x}_τ 的第 $l-1$ 层的隐向量序列 和 $\mathbf{x}_{\tau+1}$ 的第 $l-1$ 层的隐向量序列:

$$\tilde{\mathbf{H}}_{\tau+1}^{(l-1)} = \text{concat}(SG(\mathbf{H}_\tau^{(l-1)}), \mathbf{H}_{\tau+1}^{(l-1)})$$

其中 SG 表示冻结参数 (不用计算梯度), concat 表示沿位置拼接:

7_Transformer

$$\tilde{\mathbf{H}}_{\tau+1}^{(l-1)} = \begin{bmatrix} SG(\vec{\mathbf{h}}_{\tau,1}^{(l-1)T}) \\ \vdots \\ SG(\vec{\mathbf{h}}_{\tau,n}^{(l-1)T}) \\ \vec{\mathbf{h}}_{\tau+1,1}^{(l-1)T} \\ \vdots \\ \vec{\mathbf{h}}_{\tau+1,n}^{(l-1)T} \end{bmatrix}$$

- 计算 `query, key, value` 向量:

$$\mathbf{Q}_{\tau+1}^{(l)} = \tilde{\mathbf{H}}_{\tau+1}^{(l-1)} \mathbf{W}_q, \quad \mathbf{K}_{\tau+1}^{(l)} = \tilde{\mathbf{H}}_{\tau+1}^{(l-1)} \mathbf{W}_k, \quad \mathbf{V}_{\tau+1}^{(l)} = \tilde{\mathbf{H}}_{\tau+1}^{(l-1)} \mathbf{W}_v$$

其中 $\mathbf{W}_q, \mathbf{W}_k, \mathbf{W}_v \in \mathbb{R}^{d \times d}$ 分别为 `query, key, value` 转换矩阵。

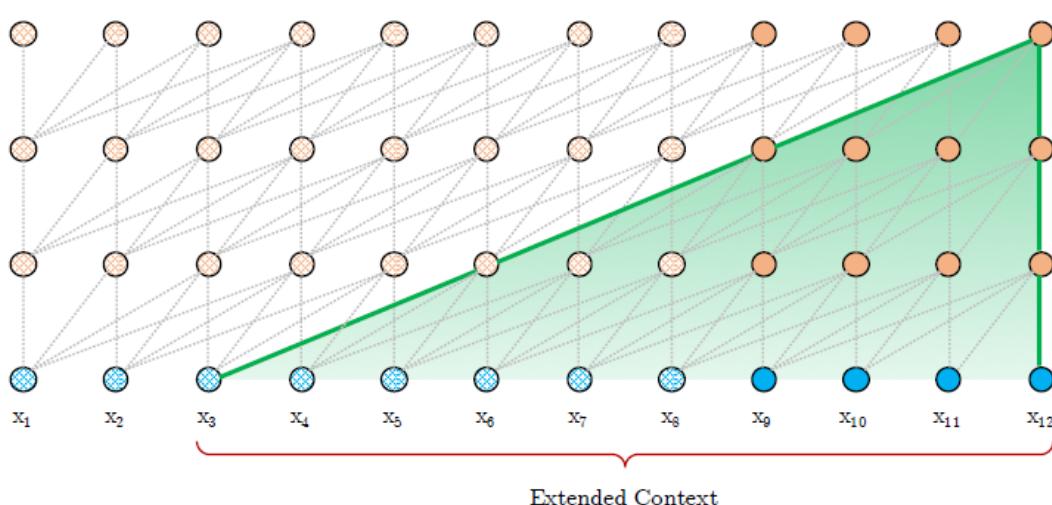
- 计算 $\mathbf{H}_{\tau+1}^{(l)}$:

$$\mathbf{H}_{\tau+1}^{(l)} = \text{Transformer-Layer}(\mathbf{Q}_{\tau+1}^{(l)}, \mathbf{K}_{\tau+1}^{(l)}, \mathbf{V}_{\tau+1}^{(l)})$$

其中 `Transformer-Layer` 为常规的 `Transformer` 层。

- 与标准 `Transformer` 不同, 这里计算 `key` 向量和 `value` 向量时使用了扩展上下文, 其中 $\tilde{\mathbf{H}}_{\tau+1}^{(l-1)}$ 缓存了前一个 `segment` 的状态。
- 这种在前后两个 `segment` 之间共享状态的方式构成了 `segment-level` 的递归, 因此上下文信息可以跨 `segment` 流动。
- $\mathbf{H}_{\tau}^{(l-1)}$ 是在 $\mathbf{H}_{\tau+1}^{(l)}$ 中被使用的, 这不仅跨了一个段, 也跨了一层。这显著的区别于其它的 `RNN` 语言模型。
- 正因为如此, `Transformer XL` 最大依赖长度扩展到了 $O(n \times L)$ 。

下图中, 每个位置的 `context` 都是它左下方的4个位置。



- `Transformer XL` 的训练方式类似于 `BPTT`, 但是与 `BPTT` 不同的是: `Transformer XL` 缓存了上一个 `segment` 的多组隐向量。

理论上不仅可以缓存前一个 `segment`, 也可以缓存前面 m 个 `segment`。

3.2 相对位置编码

- 采用 `segment-level` 递归机制之后存在一个问题: 如何保持位置信息的一致性。

令:

- `segment` 长度为 n 。

- 第 τ 个 segment 的输入为 token 序列: $\mathbf{x}_\tau = (x_{\tau,1}, x_{\tau,2}, \dots, x_{\tau,n})$, 对应的 token embedding 矩阵为 $\mathbf{E}_{\mathbf{x}_\tau} \in \mathbb{R}^{n \times d}$ 。
- 第 $\tau+1$ 个 segment 的输入为: $\mathbf{x}_{\tau+1} = (x_{\tau+1,1}, x_{\tau+1,2}, \dots, x_{\tau+1,n})$, 对应的 token embedding 矩阵为 $\mathbf{E}_{\mathbf{x}_{\tau+1}} \in \mathbb{R}^{n \times d}$ 。
- position embedding 矩阵为 $\mathbf{P}_{1:n} \in \mathbb{R}^{n \times d}$ 。

则有:

$$\begin{aligned}\mathbf{H}_{\tau+1} &= f(\mathbf{H}_\tau, \mathbf{E}_{\mathbf{x}_{\tau+1}} + \mathbf{P}_{1:n}) \\ \mathbf{H}_\tau &= f(\mathbf{H}_{\tau-1}, \mathbf{E}_{\mathbf{x}_\tau} + \mathbf{P}_{1:n})\end{aligned}$$

可见 \mathbf{x}_τ 和 $\mathbf{x}_{\tau+1}$ 都采用了同样的位置编码, 因此模型没有任何信息来区分一个 token 是位于 segment τ 还是 segment $\tau+1$ 。

2. 令:

- $\vec{\mathbf{e}}_i$ 为第 i 个位置的 token 的 embedding (对应于 token embedding 矩阵 \mathbf{E} 的第 i 行)
- $\vec{\mathbf{p}}_i$ 为第 i 个位置的 position embedding (对应于 position embedding 矩阵 \mathbf{P} 的第 i 行)
- $\vec{\mathbf{e}}_i^p = \vec{\mathbf{e}}_i + \vec{\mathbf{p}}_i$ 为第 i 个位置的 token + position embedding
- $\mathbf{W}_q^T \vec{\mathbf{e}}_i^p, \mathbf{W}_k^T \vec{\mathbf{e}}_i^p, \mathbf{W}_v^T \vec{\mathbf{e}}_i^p$ 分别为 query, key, value 向量

则 Transformer 的 attention score (不考虑 softmax 归一化, 以及除以 \sqrt{d}) 为:

$$\begin{aligned}\text{score}_{i,j} &= \mathbf{W}_q^T \vec{\mathbf{e}}_i^p \cdot \mathbf{W}_k^T \vec{\mathbf{e}}_j^p = (\vec{\mathbf{e}}_j^p)^T \mathbf{W}_k \mathbf{W}_q^T \vec{\mathbf{e}}_i^p \\ &= (\vec{\mathbf{e}}_j + \vec{\mathbf{p}}_j)^T \mathbf{W}_k \mathbf{W}_q^T (\vec{\mathbf{e}}_i + \vec{\mathbf{p}}_i) \\ &= \vec{\mathbf{e}}_j^T \mathbf{W}_k \mathbf{W}_q^T \vec{\mathbf{e}}_i + \vec{\mathbf{p}}_j^T \mathbf{W}_k \mathbf{W}_q^T \vec{\mathbf{e}}_i + \vec{\mathbf{e}}_j^T \mathbf{W}_k \mathbf{W}_q^T \vec{\mathbf{p}}_i + \vec{\mathbf{p}}_j^T \mathbf{W}_k \mathbf{W}_q^T \vec{\mathbf{p}}_i\end{aligned}$$

它表示 query i 与 key j 的相关性。

- 第一项刻画了位置 i 的 token 和位置 j 的 token 的相关性。
- 第二项刻画了位置 i 的 token 和位置 j 的 position 的相关性。
- 第三项刻画了位置 i 的 position 和位置 j 的 token 的相关性。
- 第四项刻画了位置 i 的 position 和位置 j 的 position 的相关性。

3. Transformer XL 引入相对位置编码。位置 j 相对于位置 i 的距离为 $b = j - i$, 则位置 j 相对于位置 i 的 relative position embedding 为:

$$\begin{aligned}\vec{\mathbf{r}}_b &= (r_{b,1}, r_{b,2}, \dots, r_{b,d})^T \\ r_{b,2j} &= \sin\left(\frac{b}{10000^{2j/d}}\right), \quad r_{b,2j+1} = \cos\left(\frac{b}{10000^{2j/d}}\right)\end{aligned}$$

令 n_{max} 为 Transformer XL 中使用的最大相对距离, 令相对位置编码矩阵为:

$$\mathbf{R} = \begin{bmatrix} \vec{\mathbf{r}}_1 \\ \vdots \\ \vec{\mathbf{r}}_{n_{max}} \end{bmatrix} \in \mathbb{R}^{n_{max} \times d}$$

Transformer XL 修改 attention score 为:

$$\text{score}_{i,j} = \vec{\mathbf{e}}_j^T \mathbf{W}_{kE} \mathbf{W}_q^T \vec{\mathbf{e}}_i + \vec{\mathbf{r}}_{i-j}^T \mathbf{W}_{kR} \mathbf{W}_q^T \vec{\mathbf{e}}_i + \vec{\mathbf{e}}_j^T \mathbf{W}_{kE} \vec{\mathbf{u}} + \vec{\mathbf{r}}_{i-j}^T \mathbf{W}_{kR} \vec{\mathbf{v}}$$

- 将第二、四项中的绝对位置编码 $\vec{\mathbf{p}}$ 修改为相对位置编码 $\vec{\mathbf{r}}$ 。其中的相对位置是: key 相对于 query 的位置。
- 通过参数 $\vec{\mathbf{u}} \in \mathbb{R}^d$ 来代替 $\mathbf{W}_q^T \vec{\mathbf{p}}_i$ 。这表示对于位置 j 的 key token, 同一个 query 在不同位置上无影响。因为这种影响被剥离到第二项中。
- 通过参数 $\vec{\mathbf{v}} \in \mathbb{R}^d$ 来代替 $\mathbf{W}_q^T \vec{\mathbf{p}}_i$ 。这表示对 key 相对于 value 的相对位置 $i - j$, 同一个 query 在不同位置上无影响。因为这种影响被剥离到第二项中。
- 通过 \mathbf{W}_{kE} 和 \mathbf{W}_{kR} 来生成不同的 key 向量。

修改后的 attention score 各项的意义为：

- 第一项刻画了基于内容的 attention
- 第二项刻画了内容相对于每个相对位置的 bias
- 第三项刻画了内容的全局的 bias
- 第四项刻画了位置的全局 bias

3.3 实验结果

1. Transformer XL 验证了 word-level 语言模型（以困惑度 PPL 为指标），以及 char-level 语言模型（以 bpc:Bit per Character 为指标）。其中包含以下数据集：
 - wikiText-103 数据集：最大的 word-level 语言模型 benchmark，包含 2.8万篇文章总计103M 训练 token，平均每篇文章 3.6K token。
 - 由于文章的平均 token 数量较大，因此这会考验模型的长期依赖建模能力。
 - 训练期间 attention length = 384，推断期间 attention length = 1600。
 - attention length 也等于 segment 长度

Model	#Param	PPL
Grave et al. (2016b) - LSTM	-	48.7
Bai et al. (2018) - TCN	-	45.2
Dauphin et al. (2016) - GCNN-8	-	44.9
Grave et al. (2016b) - LSTM + Neural cache	-	40.8
Dauphin et al. (2016) - GCNN-14	-	37.2
Merity et al. (2018) - QRNN	151M	33.0
Rae et al. (2018) - Hebbian + Cache	-	29.9
Ours - Transformer-XL Standard	151M	24.0
Baevski and Auli (2018) - Adaptive Input [◦]	247M	20.5
Ours - Transformer-XL Large	257M	18.3

- enwik8 数据集：包含 100M 字节的未经处理的 wiki 文本。

结果表明：12层的 Transformer XL 的表现超过了 12层的 Transformer。

Model	#Param	bpc
Ha et al. (2016) - LN HyperNetworks	27M	1.34
Chung et al. (2016) - LN HM-LSTM	35M	1.32
Zilly et al. (2016) - RHN	46M	1.27
Mujika et al. (2017) - FS-LSTM-4	47M	1.25
Krause et al. (2016) - Large mLSTM	46M	1.24
Knol (2017) - cmix v13	-	1.23
Al-Rfou et al. (2018) - 12L Transformer	44M	1.11
Ours - 12L Transformer-XL	41M	1.06
Al-Rfou et al. (2018) - 64L Transformer	235M	1.06
Ours - 18L Transformer-XL	88M	1.03
Ours - 24L Transformer-XL	277M	0.99

- text8 数据集：包含 100M 字节的、经过处理的 wiki 文本。处理方式为：大写字母改小写、移除 a~z 之外的所有字符。

Model	#Param	bpc
Cooijmans et al. (2016) - BN-LSTM	-	1.36
Chung et al. (2016) - LN HM-LSTM	35M	1.29
Zilly et al. (2016) - RHN	45M	1.27
Krause et al. (2016) - Large mLSTM	45M	1.27
Al-Rfou et al. (2018) - 12L Transformer	44M	1.18
Al-Rfou et al. (2018) - 64L Transformer	235M	1.13
Ours - 24L Transformer-XL	277M	1.08

- One Billion word 数据集：数据集的句子已经被混洗过，因此该数据集无法验证序列的长期依赖。因而该数据集用于测试模型的短期依赖。

结果表明：`Transformer XL` 对短期依赖的建模效果也非常好。

Model	#Param	PPL
Shazeer et al. (2014) - Sparse Non-Negative	33B	52.9
Chelba et al. (2013) - RNN-1024 + 9 Gram	20B	51.3
Kuchaiev and Ginsburg (2017) - G-LSTM-2	-	36.0
Dauphin et al. (2016) - GCNN-14 bottleneck	-	31.9
Jozefowicz et al. (2016) - LSTM	1.8B	30.6
Jozefowicz et al. (2016) - LSTM + CNN Input	1.04B	30.0
Shazeer et al. (2017) - Low-Budget MoE	~5B	34.1
Shazeer et al. (2017) - High-Budget MoE	~5B	28.0
Shazeer et al. (2018) - Mesh Tensorflow	4.9B	24.0
Baevski and Auli (2018) - Adaptive Input [◦]	0.46B	24.1
Baevski and Auli (2018) - Adaptive Input [◦]	1.0B	23.7
Ours - Transformer-XL Base	0.46B	23.5
Ours - Transformer-XL Large	0.8B	21.8

- Penn Treebank 数据集：包含 1M token，用于验证模型在小数据集上的表现。

结果表明：`transformer XL` 在小数据集上效果也非常好。

Model	#Param	PPL
Inan et al. (2016) - Tied Variational LSTM	24M	73.2
Zilly et al. (2016) - Variational RHN	23M	65.4
Zoph and Le (2016) - NAS Cell	25M	64.0
Merity et al. (2017) - AWD-LSTM	24M	58.8
Pham et al. (2018) - Efficient NAS	24M	58.6
Liu et al. (2018) - Differentiable NAS	23M	56.1
Yang et al. (2017) - AWD-LSTM-MoS	22M	55.97
Melis et al. (2018) - Dropout tuning	24M	55.3
Ours - Transformer-XL	24M	54.52
Merity et al. (2017) - AWD-LSTM+Finetune [†]	24M	57.3
Yang et al. (2017) - MoS+Finetune [†]	22M	54.44

2. 在 `wikitext-103` 数据集上验证 `segment-level` 递归、相对位置编码的作用，验证结果如下。

其中：

- 结果划分为三个部分，分别为 128M 参数的 `Transformer-XL`、128M 参数的 `Transformer`、151M 参数的 `Transformer-XL`。
- 最后四列的意义：
 - `PPL init` 表示推断期间使用与训练时相同的 `attention length`
 - `PPL best` 表示推断期间使用最佳长度的 `attention length`（通过超参数搜索得到）
 - `Attn Len` 给出了推断期间使用的最佳 `attention length`
 - `Full Loss/ Half Loss` 表示计算当前 `segment` 中所有位置的损失，还是计算后一半位置的损失

结果表明：

- 相对位置编码非常重要对于 `Transformer XL` 非常重要，而绝对位置编码只有在 `Half Loss` 中工作良好。
因为 `Half Loss` 只考虑当前 `segment` 后一半位置的损失，因此受到前一个 `segment` 的绝对位置编码的影响比较小。
- 随着 `attention length` 的增加，模型的困惑度下降。
- 尽管每个 `segment` 训练期间的长度是 128，但推断期间可以推广到 640。

Remark	Recurrence	Encoding	Loss	PPL init	PPL best	Attn Len
Transformer-XL (128M)	✓	Ours	Full	27.02	26.77	500
-	✓	Shaw et al. (2018)	Full	27.94	27.94	256
-	✓	Ours	Half	28.69	28.33	460
-	✗	Ours	Full	29.59	29.02	260
-	✗	Ours	Half	30.10	30.10	120
-	✗	Shaw et al. (2018)	Full	29.75	29.75	120
-	✗	Shaw et al. (2018)	Half	30.50	30.50	120
-	✗	Vaswani et al. (2017)	Half	30.97	30.97	120
Transformer (128M) [†]	✗	AI-Rfou et al. (2018)	Half	31.16	31.16	120
Transformer-XL (151M)	✓	Ours	Full	23.43	23.09	640
				23.16	23.16	450
				23.35	23.35	300

此外，由于 `Transformer XL` 采取了缓存前一个 `segment` 的状态的策略，因此会消耗更多的内存。下图给出在相同 `GPU` 内存的约束条件下的比较结果。

结果表明：即使 `Transformer XL` 由于内存约束而不得不使用一个较小的 `backprop len`，其最终效果仍然超越 `Transformer`。

Backprop Len	Recurrence	Encoding	Loss	pplx best	pplx init	Attn Len
128	✓	Ours	Full	26.77	27.02	500
128	✓	Ours	Partial	28.33	28.69	460
176	✗	Ours	Full	27.98	28.43	400
172	✗	Ours	Partial	28.83	28.83	120

3. 为了证明 `Transformer XL` 能够解决上下文碎片问题，作者在 `One billion word` 数据集上做了实验。

因为该数据集的句子经过混洗，导致不需要对长期依赖建模。因此效果的提升一定是由于模型能够解决上下文碎片，而不是由于模型捕捉到更长的上下文。

Method	PPL
Ours	25.2
With Shaw et al. (2018) encodings	25.7
Without recurrence	27.1

四、GPT

1. 目前机器学习系统通过使用大型标注数据集、高容量模型的监督学习来训练。这种方式的缺点是：对数据分布和任务目标敏感，一旦数据分布发生变化，或者任务目标发生变化，则训练好的模型就无法很好工作。

因此这种机器学习系统是数据相关的、任务相关的狭隘系统，而不是数据无关、任务无关的通用系统。

`GPT` 的目标是学得一个能够执行多种任务的、更通用的机器学习系统，同时不需要为每个任务提供标注数据集(`zero-shot learning`)，或者只需要提供很少的标注数据集(`one-shot learning`)。

4.1 GPT V1

- 通过未标记语料来训练词向量具有挑战性，有两个主要困难：
 - 无法确定合适的目标函数来获取通用词向量，使得词向量有助于迁移学习。
如：生成式语言模型的目标函数是句子生成概率，判别式上下文预测模型的目标函数是给定上下文时单词出现的概率。
- 每种方法在不同的任务中表现不同，没有哪一个方法在所有任务中均表现最好。

- 无法确定合适的迁移学习方式。

如：无监督词向量作为辅助特征引入监督学习模型，或者将无监督词向量作为监督学习模型初始化点。

- 论文《Improving Language Understanding by Generative Pre-Training》提出了GPT模型来利用未标记语料，它采用两阶段训练任务：

- 首先在大量的未标记语料上进行生成式预训练。
- 然后通过标记语料对具体任务进行针对性的微调。

其中未标记文本和标记文本可以来自不同的领域。

- GPT结合了无监督预训练模型和监督微调模型，可以认为是一种半监督学习。GPT区别于其它模型的地方在于：GPT在微调期间仍然使用无监督预训练网络，只是针对具体的监督学习任务来调整网络的输出层。

- 可以认为无监督预训练是半监督学习的一种特殊情况，其目标是为监督学习任务找到一个好的初始化点。

也可以将无监督预训练模型的隐向量作为监督学习任务的辅助特征，同时在目标任务上训练监督模型。

这种方式涉及到为每个目标任务建立监督模型，并且每个目标任务的大量新参数需要更新。

与之相比，GPT只需要在迁移学习过程中对预训练模型架构进行很小的修改就能够适应所有目标任务。

4.1.1 模型

- 预训练阶段：通过在大量未标记预料上学习网络的初始参数。

给定一个token序列 $s = \{w_1, w_2, \dots, w_\tau\}$, $w_t \in [1, V]$ 为单词在词表中的编号。GPT通过生成式语言模型来最大化概率：

$$L_1(s) = \sum_{t=1}^{\tau} \log p(w_t | w_{t-k}, \dots, w_{t-1}; \Theta)$$

其中 k 为窗口大小， Θ 为网络参数。

论文通过一个多层次 Transformer decoder 来建模，其中：

$$\begin{aligned}\vec{h}_0 &= \mathbf{W}_e^T \vec{u} + \mathbf{W}_p \\ \vec{h}_l &= \text{transformer_block}(\vec{h}_{l-1}), \forall l \in [1, n] \\ p(s) &= \text{softmax}(\mathbf{W}_o^T \vec{h}_n)\end{aligned}$$

其中 $\vec{u} = (w_1, w_2, \dots, w_\tau)^T$ 为上下文， \mathbf{W}_e 为词向量矩阵， \mathbf{W}_p 为位置向量矩阵， n 为网络层数。

原始 Transformer 采用正弦函数来产生位置向量，但这里采用模型来学习位置向量。

- 微调阶段：针对具体任务改变网络的输出层，并微调网络参数来匹配具体任务。

给定监督学习语料库 \mathbb{D} ，设样本 $s^{(i)} = (w_1^{(i)}, w_2^{(i)}, \dots, w_{n_i}^{(i)})$ 对应的标签为 $y^{(i)}$ ，其中 $(s^{(i)}, y^{(i)}) \in \mathbb{D}$ 。

将预训练模型的输出层替换为线性输出层，则有：

$$p(y | s^{(i)}) = \text{softmax}(\mathbf{W}_y^T \vec{h}_l^{(i)})$$

其中 $\vec{h}_l^{(i)}$ 为预训练模型最后一层 transformer 的输出。

因此得到监督学习任务的损失函数： $L_2(s^{(i)}, y^{(i)}) = \log p(y^{(i)} | s^{(i)})$ 。

- 论文通过实验发现：在微调任务中引入语言模型能够改进监督学习的泛化性能，同时也能够加速监督学习的收敛速度。

因此监督学习任务的损失函数为：

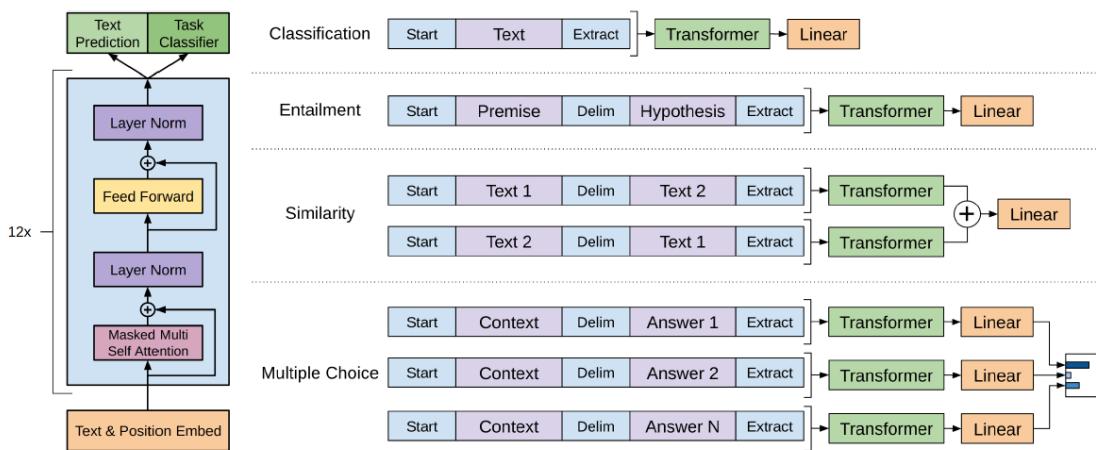
$$L_3(\mathbf{s}, y) = L2(\mathbf{s}, y) + \lambda L1(\mathbf{s})$$

其中 λ 为加权系数，是模型的一个超参数。

4. 可以看到：微调过程唯一需要学习的参数是 \mathbf{W}_y ，因此网络学习参数的数量相比于重新训练网络要小得多。
5. 对于分类任务可以简单的、直接使用微调后的模型，对于文本蕴含、知识问答等任务需要调整模型来适应这些任务。

对所有任务都执行输入转换，不同的任务转换方式不同。

- 对于所有任务，都在序列开始添加 `start` 标记，在序列结束添加 `extract` 标记。
- 文本分类任务：直接使用微调后的模型。
- 文本蕴含任务：将前提 `Premise` 和假设 `Hypothesis` 通过分隔符 `Delim` 拼接，然后使用微调后的模型。
- 文本相似度任务：考虑到 `A` 和 `B` 的相似度等于 `B` 和 `A` 的相似度，即：句子之间的相似度与句子顺序无关，因此：
 - 首先将句子 `Text1` 和句子 `Text2` 通过分隔符 `Delim` 拼接，然后将句子 `Text2` 和句子 `Text1` 通过分隔符 `Delim` 拼接。
 - 将得到的两个样本输入网络得到的 `representation` 相加，结果灌入线性输出层。
- 问答和知识推理任务：给定文档 `Doc` 和问题 `Question`，以及一组回答 `Answer1, Answer2, ..., GPT` 将其转化为一个文本分类任务。
 - 首先根据每个回答通过分隔符 `Delim` 来拼接文档和问题，得到样本 `Doc Question Delim Answer_k`。
 - 其中上下文 `Context` 是文档和问题的直接拼接（未添加分隔符）：`Context = Doc Question`。
 - 然后对每个样本提取 `representation`，灌入线性分类层（得到每个答案是否正确的概率），最后送入 `softmax` 输出层得到归一化概率。



4.1.2 实验结论

1. 模型结构：`GPT` 模型采用 12 层的、仅包含 `decoder` 的 `Transformer` 模型。`Transformer` 的结构为：
 - 12 个 `masked` 的 `self attention head`，其向量为 768 维。
 - `Feed Forward` 的输出为 3072 维。
2. `GPT` 模型在不同任务上的表现：
 - 文本蕴含任务：其中 `5x` 表示集成了 5 个模型。

自然语言推理任务 `natural language inference:NLI` 也称作文本蕴含任务 `textual entailment`，其目标是：给定一对句子，判定其关系是蕴含、矛盾、还是中立。这里评估了5个不同来源的数据集，包括图片标题 `SNLI`、新闻文章 `RTE`、维基百科文章 `QNLI`、科学考试 `SciTail`、转录语音&通俗小说&政府报告 `MNLI`。

Method	MNLI-m	MNLI-mm	SNLI	SciTail	QNLI	RTE
ESIM + ELMo [44] (5x)	-	-	89.3	-	-	-
CAFE [58] (5x)	80.2	79.0	89.3	-	-	-
Stochastic Answer Network [35] (3x)	80.6	80.1	-	-	-	-
CAFE [58]	78.7	77.9	88.5	83.3	-	-
GenSen [64]	71.4	71.3	-	-	82.3	59.2
Multi-task BiLSTM + Attn [64]	72.2	72.1	-	-	82.1	61.7
Finetuned Transformer LM (ours)	82.1	81.4	89.9	88.3	88.1	56.0

- 问答和知识推理任务：其中 `9x` 表示集成了 9 个模型。

问答和知识推理任务包含两个数据集：

- `RACE` 数据集：包含来自初中、高中考试的一些问答题。
- `Story Cloze` 数据集：该任务从两个选项中选择一个来作为故事的正确结尾。

Method	Story Cloze	RACE-m	RACE-h	RACE
val-LS-skip [55]	76.5	-	-	-
Hidden Coherence Model [7]	77.6	-	-	-
Dynamic Fusion Net [67] (9x)	-	55.6	49.4	51.2
BiAttention MRU [59] (9x)	-	60.2	50.3	53.3
Finetuned Transformer LM (ours)	86.5	62.9	57.4	59.0

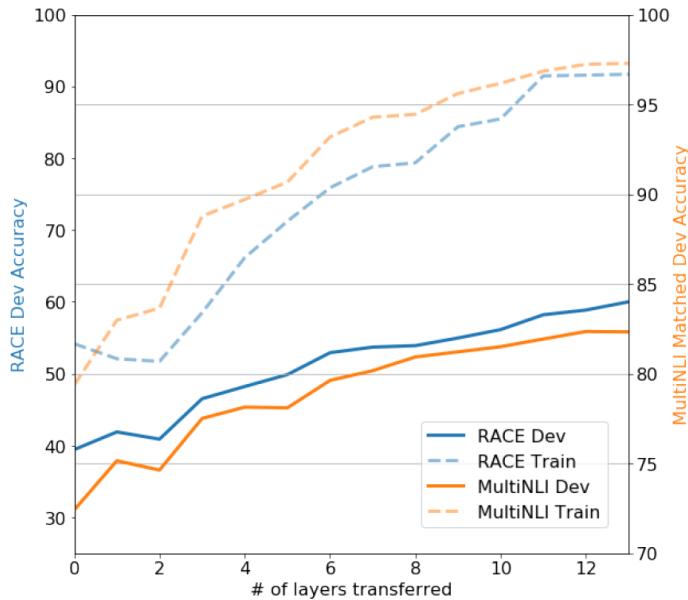
- 语义相似度任务：其中 `mc` 表示 `Mathews` 相关系数，`acc` 表示准确率，`pc` 表示皮尔逊系数。

语义相似度任务检测两个句子在语义上是否等价。这里采用了3个数据集：`Microsoft Paraphrase corpus:MRPC`、`Quora Question Pairs:QQP`、`Semantic Textual Similarity benchmark:STS-B`。

Method	Classification		Semantic Similarity		GLUE
	CoLA (mc)	SST2 (acc)	MRPC (F1)	STSB (pc)	
Sparse byte mLSTM [16]	-	93.2	-	-	-
TF-KLD [23]	-	-	86.0	-	-
ECNU (mixed ensemble) [60]	-	-	-	81.0	-
Single-task BiLSTM + ELMo + Attn [64]	35.0	90.2	80.2	55.5	66.1
Multi-task BiLSTM + ELMo + Attn [64]	18.9	91.6	83.5	72.8	63.3
Finetuned Transformer LM (ours)	45.4	91.3	82.3	82.0	70.3
					72.8

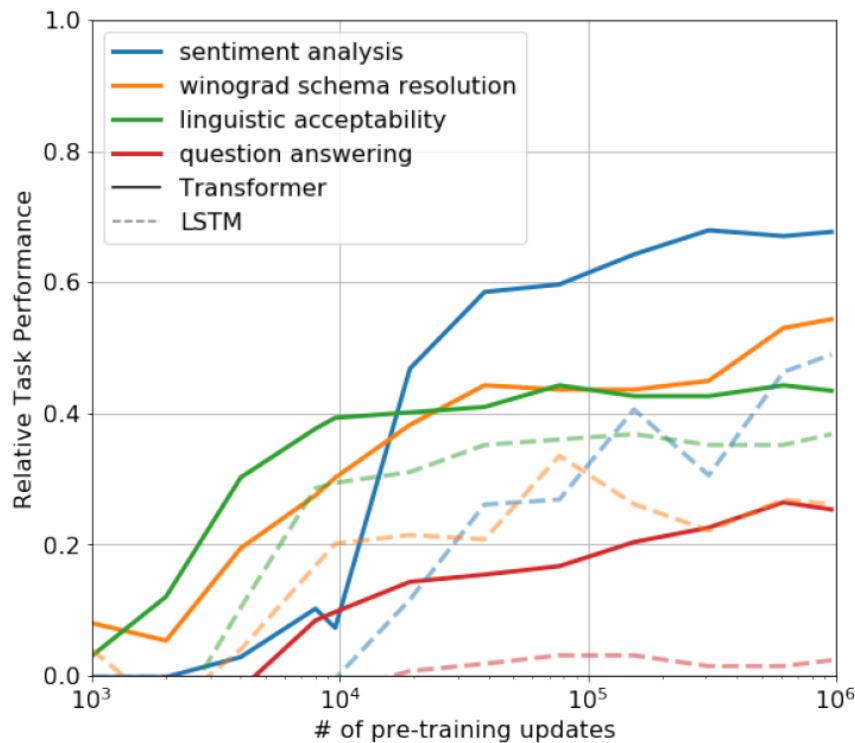
- 实验表明：迁移学习时 `GPT` 的每一层都对目标任务有帮助。下图为采用不同层数时，模型在 `RACE` 和 `Multinli` 任务上的表现。即：迁移学习时并不是使用 \vec{h}_l ，而是 $\vec{h}_k, k = 1, 2, 3, \dots, l$

。



4. 与 `LSTM` 结构相比, `Transformer` 能够捕捉到更长的语言结构。因此 `Transformer` 的迁移学习效果更好, 且表现更加稳定。

下图为直接使用预训练模型而不进行微调时, 二者在新任务上的表现对比: `LSTM` 的预测结果更差, 且波动更大。



5. 通过实验表明: `GPT` 在微调任务中引入语言模型的损失这个辅助目标可以提升监督任务的性能。

实际上辅助目标对于大数据集有效, 对于小数据集效果不佳。

下表中:

- `aux LM` 表示使用预训练模型和辅助目标
- `o pre-training` 表示不使用预训练模型, 也不使用辅助目标
- `o aux LM` 表示使用预训练模型, 但是不使用辅助目标

Method	Avg. Score	CoLA (mc)	SST2 (acc)	MRPC (F1)	STS-B (pc)	QQP (F1)	MNLI (acc)	QNLI (acc)	RTE (acc)
Transformer w/ aux LM (full)	74.7	45.4	91.3	82.3	82.0	70.3	81.8	88.1	56.0
Transformer w/o pre-training	59.9	18.9	84.0	79.4	30.9	65.5	75.7	71.2	53.8
Transformer w/o aux LM	75.0	47.9	92.0	84.9	83.2	69.8	81.1	86.9	54.4
LSTM w/ aux LM	69.1	30.3	90.5	83.2	71.8	68.1	73.7	81.1	54.6

4.2 GPT V2

1. 论文《Language Models are Unsupervised Multitask Learners》提出了GPT V2模型，其基本结构与GPT V1相同，其区别在于：

- GPT V2具有15亿参数，网络层数更深、网络容量更大。

下表中最小的模型等价于GPT V1，次小的模型等价于BERT中的最大模型。最大的模型为GPT V2。

- 网络结构微调：

- GPT V2将Layer Normalization移动到每个子块的输入之后，类似于pre-activation残差网络。
- GPT V2在最终的self-attention块结束之后添加额外的layer normalization层。

- 权重初始化方式微调：GPT V2将残差层的权重初始化缩放到GPT 1的 $\frac{1}{\sqrt{N}}$ ，其中N为残差层的数量。
- GPT V2扩大词表到50247个词汇。
- GPT V2将上下文大小从512 token扩大到1024 token。
- GPT V2使用更大的batch size(512)。

Parameters	Layers	d_{model}
117M	12	768
345M	24	1024
762M	36	1280
1542M	48	1600

2. GPT V2通过实验结果说明了语言模型可以基于zero-shot来执行下游任务，而无需修改任何参数或者网络架构。

4.2.1 训练数据集

1. 为了构建尽可能大和多样性的训练数据集，从而收集尽可能自然的语言分布，GPT V2采用爬取网页文本的方式。

但是这种方式爬取的大多数网页的内容难以理解，因此严重降低了文本的质量。GPT V2的解决方案是：仅仅爬取由人类编辑、筛选的网页。

2. GPT V2构建训练数据集的步骤：

- 首先从Reddit抓取出站计数超过3的所有超链接，最终爬取到4500万条超链接。
- 然后爬取这些超链接的HTML文本，并进行过滤。
 - 剔除2017年12月之后创建的超链接。这是为了划定时间范围，从而给定一个不可变的文档集合。
 - 删除所有维基百科文档，因为它是很多任务的公共资源，可能会对后续的迁移学习任务评估有影响。
 - 去重。

最终得到800多万文档，超过40GB的文本。该数据集称作webText数据集。

4.2.2 预处理

1. 通用语言模型应该能计算任何字符串的概率。但是，由于当前的大规模语言模型包含一些预处理步骤，如：大写字符转小写、词干化 tokenization、表外标记 oov token，这些步骤限制了字符串的模型空间。

一种解决方案是将字符串转换为 UTF-8 字节序列。但是实验表明：在大型数据集上，bytes-level 字节粒度的语言模型不如 word-level 单词级别的自然语言模型。

2. GPT V2 采用 Byte pair Encoding:BPE 方案，该方案结合了 word-level 的效果优势和 bytes-level 的通用性，是二者之间的折中方案。

- BPE 对高频的符号采用 word-level 的输入，对低频的符号采用 char-level 的输入。
- 事实上 BPE 并不是采用字节，而是采用 Unicode 码点。

考虑到码点的数量较大（超过 13万个），而常见的单词词表大小在32000到 64000，因此也可以采用 bytes-level 的 BPE，它仅仅对词表引入 256个额外的 token。

3. GPT V2 不需要执行词干化，也不需要执行其它的有损预处理（如 oov 标记）。

但是，对于标点符号需要特殊处理。由于英文是空格来分词，因此对于 dog? 这类带标点符号的单词需要将其拆分为 dog 和标点符号两个 token。这是因为 GPT V2 没有词干化，所以 dog? 并未归一化到单词 dog。

4.2.3 实验结论

1. GPT V2 在8个数据集上与其它模型进行比较，结果在其中 7 个数据集上 GPT V2 取得了最佳效果。

- 实验表明：数据集越小，GPT V2 的优势越大。
- 对于 One Billion Word Benchmark 数据集，GPT V2 差得多。原因可能有两个：
 - One Billion Word benchmark 本身就是一个超大数据集，在它之上训练的专用模型效果已经非常好。
 - One Billion Word benchmark 数据集执行了某些破坏性的预处理：进行了句子级别的混洗。这会破坏所有的长范围的结构。
- 评估指标：PPL 困惑度、ACC 准确率、BPC、BPC。
- 评估数据集：
 - LAMBADA 数据集：预测句子的最后一个单词。它测试了模型对文本长期依赖关系建模的能力。
 - CBT 数据集：Children's Book Test 数据集，预测省略单词的10个中哪个可能是正确的。它考察模型在不同类别的单词（命名实体、名字、动词）上的表现。

CBT-CN 为普通名词数据集，CBT-NE 为命名实体数据集。

Language Models are Unsupervised Multitask Learners

	LAMBADA (PPL)	LAMBADA (ACC)	CBT-CN (ACC)	CBT-NE (ACC)	WikiText2 (PPL)	PTB (PPL)	enwik8 (BPC)	text8 (BPC)	WikiText103 (PPL)	IBW (PPL)
SOTA	99.8	59.23	85.7	82.3	39.14	46.54	0.99	1.08	18.3	21.8
117M	35.13	45.99	87.65	83.4	29.41	65.85	1.16	1.17	37.50	75.20
345M	15.60	55.48	92.35	87.1	22.76	47.33	1.01	1.06	26.37	55.72
762M	10.87	60.12	93.45	88.0	19.93	40.31	0.97	1.02	22.05	44.575
1542M	8.63	63.24	93.30	89.05	18.34	35.76	0.93	0.98	17.48	42.16

2. 在语言模型中困惑度 perplexity 是一个比较重要的指标，它刻画了一个生成一个给定句子的概率。

困惑度常用于评估语言模型的好坏：给测试集的句子赋予较高概率值的语言模型较好，否则语言模型较差。

对于给定句子 $s = \{w_1, w_2, \dots, w_n\}$ ，其定义为：

$$\text{Perplexity} = p(\{w_1, w_2, \dots, w_n\})^{-1/n} = \sqrt[n]{\frac{1}{p(\{w_1, w_2, \dots, w_n\})}}$$

句子概率越大则困惑度越小，因此模型也就越好。

3. 计算机视觉领域最近的研究表明：普通的图像数据集包含了大量的近似重复图像，如：[CIFAR-10](#) 数据集在训练集和测试集之间有 3.3% 的重复。这会导致模型的泛化能力被高估。

随着数据集大小的增长，这个问题可能会越来越严重：训练集越大，测试集中的样本越可能重复出现在训练集中。因此 [GPT V2](#) 的优秀表现有可能是由于测试集的测试样本曾经出现在训练集中，导致模型“记住”了这些样本。

通过两个证据可以说明 [GPT V2](#) 的优秀表现是因为其本身泛化能力较强，而不是因为“记忆”效果。

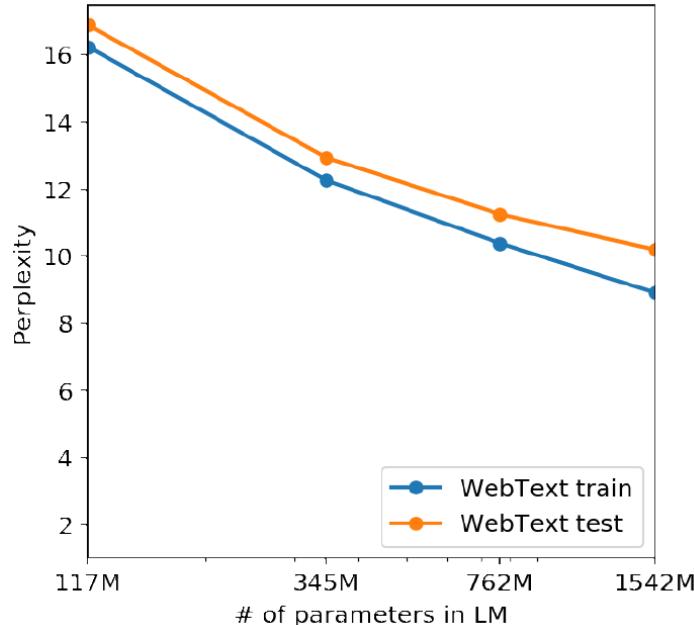
- 通过 [Bloom](#) 过滤器来检查不同数据集中，测试集和训练集的重复度。

数据表明：[webText train](#) 的测试集、训练集的重复度较低。

	PTB	WikiText-2	enwik8	text8	Wikitext-103	1BW
Dataset train	2.67%	0.66%	7.50%	2.34%	9.09%	13.19%
WebText train	0.88%	1.63%	6.31%	3.94%	2.42%	3.75%

- 通过考察模型容量的性能表明：随着模型容量的增大，模型在训练集和测试集上的表现都在提升。

这说明更大的模型可以得到更好的效果，模型目前处于欠拟合（而不是过拟合）阶段。



五、BERT

- 将预训练模型用于下游任务有两种策略：

- 基于微调的策略。如 [GPT](#)，通过简单微调预训练模型的参数来训练下游任务。

该策略在预训练期间通过单向语言模型来学习通用语言 [representation](#)，而单向语言模型严重限制了预训练模型的表达能力。

例如，在 [token](#) 级别的任务（如：词性标注任务），结合两个方向的上下文对模型性能非常重要。

- 基于特征的策略。如 `ELMO`，将预训练模型的 `representation` 作为下游任务模型的额外特征。

该策略虽然是双向语言模型，但是该模型是浅层的。

与它们不同，`BERT: Bidirectional Encoder Representations from Transformers` 是一个同时利用了左右双向上下文的、深度的预训练模型，它在11项 `nlp` 任务中取得最领先的结果。

2. 基于特征的方法具有一定优势：

- 并不是所有的 `NLP` 任务都可以很容易地用 `Transformer encoder` 架构来表示，因此需要添加特定于任务的模型架构。
- 如果通过训练数据预先计算一次昂贵的数据表示，然后在该表示的基础上用廉价的模型来完成许多任务，这会带来很大的计算优势。

3. 单向语言模型可以是从左到右 `Left to Right: LTR` 或者从右到左 `Right to Left :RTL`。

`BERT` 也可以像 `ELMO` 一样训练独立的 `LTR` 和 `RTL` 模型后拼接在一起，但是这么做有两个问题：

- 其训练代价是单个双向模型的两倍。
- 对于 `Question - Answer` 之类的问题是反直觉的，因为 `RTL` 模型需要根据答案来反推问题。
- `BERT` 可以自由的组合左侧上下文和右侧上下文。

5.1 预训练

- `BERT` 预训练模型包含两个预训练任务：预测被屏蔽的单词、预测下一个句子。
- `BERT` 的预训练语料库必须使用 `document-level` 的语料库，而不是经过混洗的 `sentence-level` 的语料库。因为混洗句子会破坏句子预测预训练任务。
 - 这里的“句子”不一定是真实的句子，而是一段话或者几段话，代表了一个 `token` 序列。
 - `BERT` 预训练时，每个“句子”的 `token` 长度小于等于 512。
- `BERT` 的训练语料库经过了 `WordPiece` 词干化。如：

1	原始: He is an engineer
2	WordPiece词干化: He is an engine ##er

在“预测被屏蔽的单词”任务中，无需预测单词片段 `##er`。

- `BERT` 预训练采用 `gelu` 激活函数，训练一百万步，`bath size = 256`。

5.1.1 MLM

- 受完形填空任务启发，`BERT` 通过提出一个新的预训练目标来解决前面提到的单向限制：掩码语言模型 `masked language model:MLM`。
 - 从直觉上，深度双向模型要比深度单向模型、单层双向模型表达能力更强。
 - 标准的条件语言模型只能从左到右或者从右到左训练，因为双向条件作用允许每个单词在多层上下文中间接“看到自己”。
 - `MLM` 模型从输入中随机屏蔽一些 `token`，目标是基于上下文预测被屏蔽单词。方法是：将被屏蔽的单词替换为 `[MASK]` 标记，然后被屏蔽的单词作为真实 `label`。
 - 与单向语言模型不同，`MLM` 结合了左右两侧上下文。
 - 为了训练 `MLM`，模型随机屏蔽一定百分比（论文中为 15%）的 `token`，然后仅预测那些被屏蔽的 `token`。
- 这种方式有两个缺陷：

- 预训练和微调之间的不匹配。因为在微调阶段，模型永远不会看到 [MASK] 标记。

为了缓解这种状况，MLM 在预训练时并不总是用真的 [MASK] 标记，而是从输入中随机选择 15% 的 token：80% 替换为 [MASK] 标记，10% 替换为一个随机单词，10% 保持原样。

MLM 并不知道哪些词被替换，因此它总是努力的学习每个单词的正确表达。

```

1 | my dog is hairy -> my dog is [MASK] // 80% 概率
2 | my dog is hairy -> my dog is test // 10% 概率
3 | my dog is hairy -> my dog is hairy // 10% 概率

```

- 每个 batch 预测的 token 只有 15%，这意味着模型需要更多的训练步才可能收敛。

实验证明 MLM 的收敛速度确实比单向语言模型稍慢，但是相对于 MLM 的泛化能力的提升，这种代价是值得的。

5.1.2 NSP

- 许多重要的下游任务，如：知识问答和自然语言推理，都是基于理解两个句子之间的关系，而这种关系不是由语言模型直接捕获的。

为了训练理解句子关系的模型，BERT 训练一个二元化的句子预测任务，称作 Next Sentence Prediction: NSP：

- 每个训练样本由一对句子 A 和 B 组成：50% 的样本中 B 是紧跟在 A 之后的句子，50% 的样本中二者是随机挑选的。
- 模型需要预测的是 B 是否是 A 的下一个句子。

如：

```

1 | Input1 = [CLS] the man went to [MASK] store [SEP] he bought a gallon
   | [MASK] milk [SEP]
2 | Label1 = IsNext
3 | Input2 = [CLS] the man [MASK] to the store [SEP] penguin [MASK] are
   | flight ##less birds [SEP]
4 | Label2 = NotNext

```

5.2 模型结构

- BERT 采用双向 self-attention 的 Transformer 来构建。

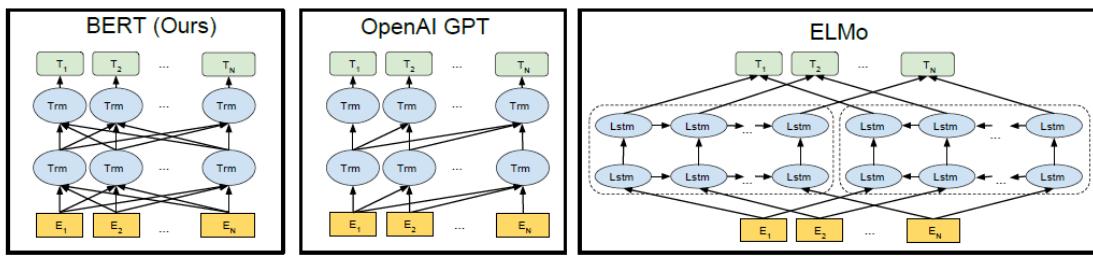
双向 self-attention 的 Transformer 也称作 Transformer encoder，而单向 self-attention 的 Transformer 被称作 Transformer decoder。

- BERT 有两种尺寸：令 BERT 层数为 L ，每层每个隐向量的维度均为 H ，self-attention 的 head 数为 A 。BERT 将所有前馈层和 filter 层的维度设为 $4H$ 。

- BERT_{BASE}： $L = 12, H = 768, A = 12$ ，模型参数数量为 110 M。

其模型大小与 GPT v1 大小几乎相同。选择这组参数的目的是为了与 GPT V1 进行更好的比较（openAI GPT V1 模型的参数也是 $L=12, H=768, A=12$ ）。

- BERT_{LARGE}： $L = 24, H = 1024, A = 16$ ，模型参数数量为 340M。

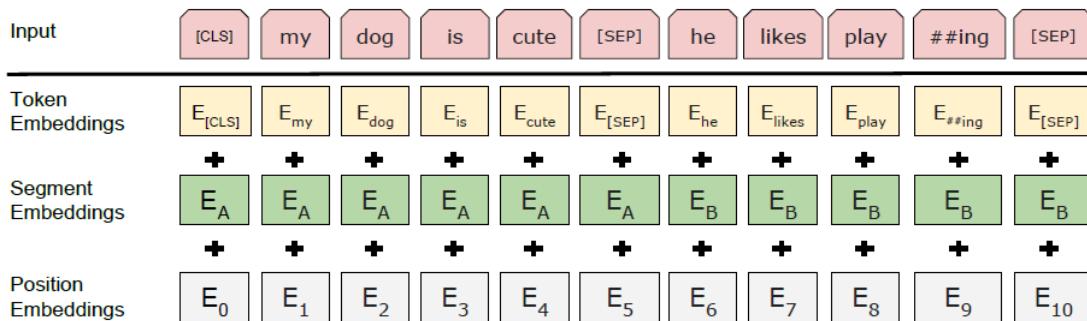


3. BERT 的模型输入能够表达单个句子或者一对句子。

- 每个 `token` 的输入 `representation` 由三个部分相加而成: `token embedding`、`segment embedding`、`position embedding`。
- 每个序列的第一个 `token` 是一个特殊的 `[CLS]`。网络最后一层对应于该位置的一个隐向量作为整个序列的 `representation` 来用于分类任务。
对于非分类任务，该隐向量被忽略。
- 如果是一对句子，则在句子之间插入特殊的 `token`: `[SEP]`。然后对句子的前后关系学习一个 `segment embedding`:
 - 前一个句子的每个 `token` 学习和使用 `A embedding`，代表前后关系的“前关系”的表达。
 - 后一个句子的每个 `token` 学习和使用 `B embedding`，代表前后关系的“后关系”的表达。

通过这种方式，模型得以学习和区分两个句子的前后关系。

- 对于单个句子，模型仅学习和使用 `A embedding`。
- `position embedding` 是模型学习和使用的 `input` 每个绝对位置的表达。
- `token embedding` 是模型学习和使用的每个 `token` 的表达。

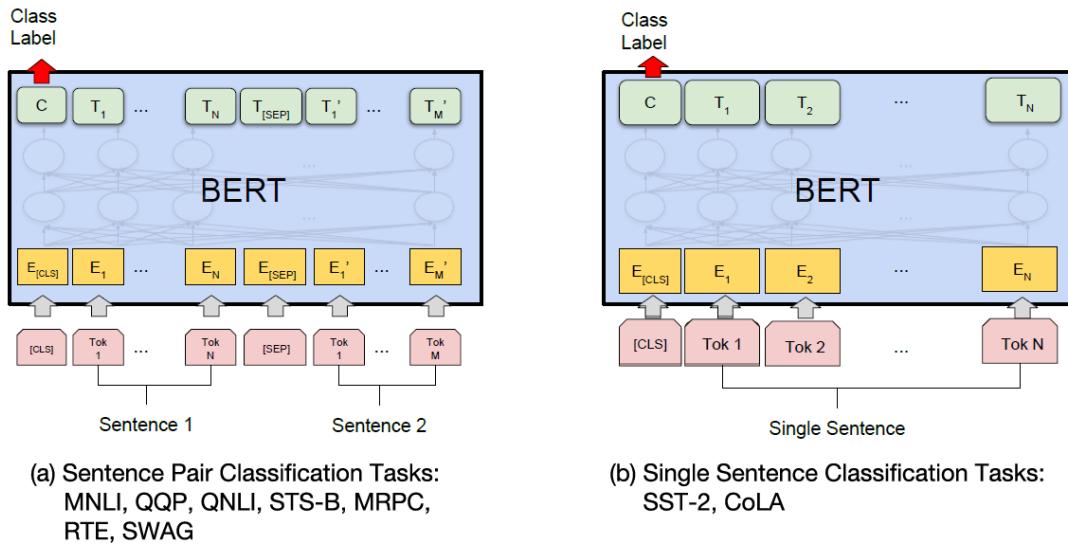


5.3 微调

1. 对于句子级别的分类任务，微调方式为：

- 取得预训练模型对应于 `[CLS]` 位置的最后一层隐向量 (一个 H 维的向量) 作为整个句子的 `representation`。
如果是句子对，则在两个句子之间插入分隔符 `[SEP]`。
- 增加一个 `softmax` 输出层，将该隐向量作为输出层的输入，输出为样本对应于各分类类别 的概率。

此时新增 `softmax` 输出层仅仅需要训练额外的权重矩阵 $\mathbf{W} \in \mathbb{R}^{K \times H}$ ，其中 K 为分类类别的数量。



2. 文本区间预测任务：给定一个问题和一段话，任务目标是预测正确答案在该段话中的区间位置。

假设问题为 Q ，包含 N 个 token；段落为 A ，包含 M 个 token。BERT 在微调期间：

- 首先将问题和段落通过分隔符 [SEP] 拼接在一起： $Q [SEP] A$ 。
- 然后模型需要预测段落 A 的第 i 个 token 分别为答案开始、结束的位置。

设起始向量 $\vec{S} \in \mathbb{R}^H$ 、终止向量 $\vec{E} \in \mathbb{R}^H$ ，这两个向量是模型需要学习的参数。设段落 A 的第 i 个 token 的输出向量为 $\vec{T}_i \in \mathbb{R}^H$ ，则该单词为答案开始、结束的概率分别为：

$$P_i^{start} = \frac{\exp(\vec{S} \cdot \vec{T}_i)}{\sum_{j=1}^M \exp(\vec{S} \cdot \vec{T}_j)}, \quad P_i^{end} = \frac{\exp(\vec{E} \cdot \vec{T}_i)}{\sum_{j=1}^M \exp(\vec{E} \cdot \vec{T}_j)}$$

预测的区间开始和结束为：

$$\text{start} = \arg \max_{i=1,2,\dots,M} P_i^{start}, \quad \text{end} = \arg \max_{i=1,2,\dots,M} P_i^{end}$$

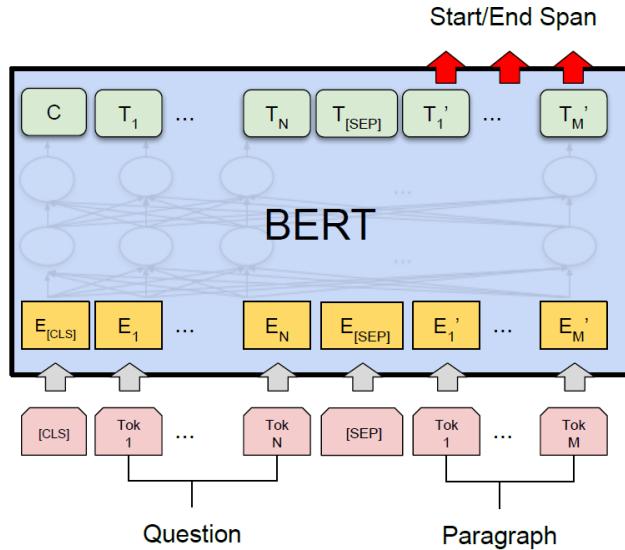
- 训练期间，训练目标为最大化开始 + 结束的最大似然：

$$\max \sum_{(x_s, b_s, e_s) \in \mathcal{D}} \left(\log P_{b_s}^{start} + \log P_{e_s}^{end} \right)$$

其中：

- (x_s, b_s, e_s) 表示训练集中的第 s 个样本的序列、答案在段落中的开始位置、答案在段落中的结束位置
- $\log P_{b_s}^{start}$ 表示第 s 个样本预测到段落位置 b_s 为答案开始位置的概率
- $\log P_{e_s}^{end}$ 表示第 s 个样本预测到段落位置 e_s 为答案结束位置的概率
- 由于模型并未对预测的结束位置和开始位置施加约束，因此推断期间可能发生结束位置在前、开始位置在后的违反常理的情形。

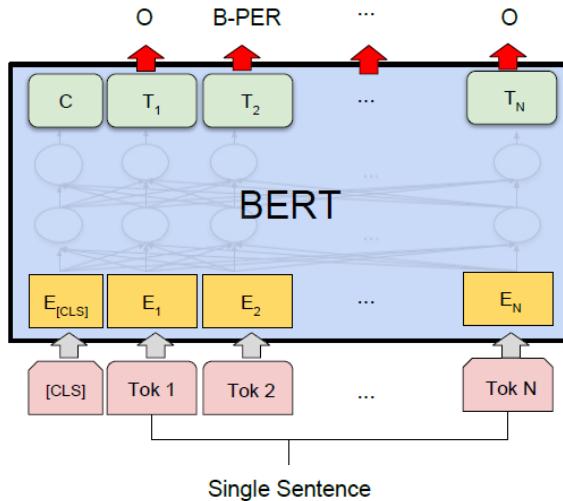
因此 BERT 在推断期间增加强制约束：结束在开始之后，即 $\text{end} \geq \text{start}$ 。



3. 命名实体识别 NER 任务是为了评估模型的 token tagging 能力。

BERT 微调阶段增加一个 softmax 输出层，然后获取最后一层每个位置 i 的隐向量 $\vec{T}_i \in \mathbb{R}^H$ ，并将该隐向量作为输出层的输入，输出为 token i 属于各 NER label 的概率。

注意：这里隐含假设每个 token 的 NER label 与周围其它单词无关。



4. 在微调阶段，除了 batch size、学习率、训练 epoch 不同之外，其它训练参数与预训练阶段的训练参数相同。

- 微调阶段通常很快，因此建议对超参数进行彻底搜索并选择在验证集上表现最好的模型。
- 论文发现：数据集越小，模型性能对超参数的选择越敏感。
大数据集（超过10万的标记样本）对超参数的敏感性要低的多。

5.4 性能

1. BERT 中的一些设计被有意选择为尽可能接近 GPT，从而方便二者的比较。二者除了在结构上不同以外，在预训练阶段还有以下不同：

- GPT 在 BooksCorpus (800M 单词) 上预训练，BERT 在 BooksCorpus 和 Wikipedia (2500M 单词) 上预训练。
- GPT 预训练100万步，32000 token/batch；BERT 训练 100 万步，128000 token/batch (batch size=256 sentence, 一个序列包含 512 token)。
- GPT 仅仅在微调阶段引入句子分隔符 [SEP] 和分类符 [CLS]，而 BERT 在预训练阶段就引入 [SEP], [CLS] 以及句子词序 A/B 从而学习它们的 embedding。

- GPT 对所有的微调任务都使用 $5e-5$ 的学习率，而 BERT 根据不同任务的验证集来选择合适的学习率。

5.4.1 文本分类任务

- GLUE 数据集：通用语言理解评估 General Language Understanding Evaluation:GLUE 基准是一组各种 NLP 文本任务。
 - 大多数 GLUE 数据集已经存在多年，而 GLUE 将它们收集在一起的目的是：
 - 为这些数据集建立统一的训练集、验证集、测试集拆分标准。
 - 用一个标准的 evaluation 服务器来缓解 evaluate 不一致，以及测试集过拟合的问题。
 - GLUE 不给测试集打标签，用户必须将测试结果上传到 GLUE 服务器进行评估（但是提交次数有限）。
 - GLUE 基准包含以下数据集：
 - Multi-Genre Natural Language Inference:MNLI：大规模的、众包的蕴含分类任务。
给定一对句子，其目标是预测第二句相对于第一句是蕴含句 entailment、矛盾句、还是中性句。
 - Quora Question Pairs:QQP：一个二元分类任务。
给定一对 Quora 上的两个问题，其目标是预测它们在语义上是否等价。
 - Question Natural Language Inference:QNLI：斯坦福 Question Answering 数据集的一个转换为分类任务的版本。
正类样本由问题和正确答案组成，负类样本由问题和非正确答案组成，其中非正确答案来自于同一段文本。
 - Stanford Sentiment Treebank:SST-2：一个二元单句分类任务。
其数据集从电影评论中提取的句子组成，由人类对其进行二元情感标注。
 - Corpus of Linguistic Acceptability: CoLA：一个二元单句分类任务。
其目标是预测一个英语句子是否在语法上可接受的。
 - Semantic Textual Similarity Benchmark : STS-B：一个句子相似度多元分类任务。
其数据集从新闻标题和其它数据源提取的句子对，通过人工来标记一对句子在语义上的相似得分（1分到5分）。
 - Microsoft Research Paraphrase Corpus:MRPC：一个句子相似度二元分类任务。
从在线新闻数据源抽取的句子对，由人工标记一对句子是否语义上相等。
 - Recognizing Textual Entailment: RTE：类似 MNLI 的二元蕴含关系任务，但是 RTE 数据集规模更小。
 - Winograd NLI:WNLI：一个小型的自然语言推理数据集。
GLUE 网页指出该数据集有问题：所有提交上来的结果都不如 baseline（仅仅将结果预测为训练集中出现次数最多的类别）。因此 BERT 并未评估该数据集。
- BERT 在所有 GLUE 任务上选择 batch size=32, epoch = 3，学习率根据验证集的效果在 $5e-5, 4e-5, 3e-5, 2e-5$ 中选择最好的那个。

注意：BERT_{LARGE} 在小数据集上的微调效果可能不稳定（发生过拟合），此时会进行随机重启，最终选择使得验证集最好的那个随机重启的模型。

System	MNLI-(m/mm)	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average
	392k	363k	108k	67k	8.5k	5.7k	3.5k	2.5k	-
Pre-OpenAI SOTA	80.6/80.1	66.1	82.3	93.2	35.0	81.0	86.0	61.7	74.0
BiLSTM+ELMo+Attn	76.4/76.1	64.8	79.9	90.4	36.0	73.3	84.9	56.8	71.0
OpenAI GPT	82.1/81.4	70.3	88.1	91.3	45.4	80.0	82.3	56.0	75.2
BERT _{BASE}	84.6/83.4	71.2	90.1	93.5	52.1	85.8	88.9	66.4	79.6
BERT _{LARGE}	86.7/85.9	72.1	91.1	94.9	60.5	86.5	89.3	70.1	81.9

3. **situations with Adversarial Generations dataset: SWAG** 数据集：包含11.3万组句子对，用于评估常识推理。

- SWAG 数据集的每个样本是视频字幕数据集中的一句句子，任务是在四个选项中选择最合理的延续。
- SWAG 任务的微调类似于 GLUE：
 - 首先将问题和四个选项给自拼接成一个序列，将每个序列的 [CLS] 对应位置的表达 $\vec{C} \in \mathbb{R}^H$ 作为该序列的整体表达。
 - 然后引入向量 $\vec{V} \in \mathbb{R}^H$ ，它是模型待学习的参数。将它和第 i 个序列的表达 \vec{C}_i 相乘即可得到评分： $\text{score}_i = \vec{V} \cdot \vec{C}_i$ 。
 - 最后对评分归一化即可得到每个选项为正确答案的概率：

$$P_i = \frac{\exp(\text{score}_i)}{\sum_{j=1}^4 \exp(\text{score}_j)}$$

4. BERT 在 SWAG 任务上选择 batch size=16, epoch = 3, 学习率为 2e-5。

其中 Human performance 是人类在 100 个问题上回答的结果。

System	Dev	Test
ESIM+GloVe	51.9	52.7
ESIM+ELMo	59.1	59.2
BERT _{BASE}	81.6	-
BERT _{LARGE}	86.6	86.3
Human (expert) [†]	-	85.0
Human (5 annotations) [†]	-	88.0

5.4.2 文本区间预测任务

1. **Standford Question Answering Dataset:SQuAD** 数据集：包含10万个众包的 question-answer 样本。

- 每个样本包含一个问题以及 wiki 中的、包含答案的一段话，标签信息是人工标注的、从该段话中提取的答案。
- 任务目标是：给定一个问题和 wiki 中的一段话，预测答案在段落中的位置区间。因此该任务是区间预测任务。
- 考虑到数据的预处理过程（wordPiece 词干化），最终在评估结果时需要将预处理后的答案区间转换为预处理前的答案区间。

2. BERT 在 SQuAD 任务上选择 batch size = 32, epoch = 3, 学习率为 5e-5。

其中：

- BERT Single 采用单个 BERT 模型；BERT Ensemble 使用了 7 个模型的融合，这些模型采用不同的预训练 checkpoints 和不同的微调 seeds。
- + TriviaQA 表示使用 TriviaQA 数据集来扩充训练数据。

System	Dev		Test	
	EM	F1	EM	F1
Leaderboard (Oct 8th, 2018)				
Human	-	-	82.3	91.2
#1 Ensemble - nlnet	-	-	86.0	91.7
#2 Ensemble - QANet	-	-	84.5	90.5
#1 Single - nlnet	-	-	83.5	90.1
#2 Single - QANet	-	-	82.5	89.3
Published				
BiDAF+ELMo (Single)	-	85.8	-	-
R.M. Reader (Single)	78.9	86.3	79.5	86.6
R.M. Reader (Ensemble)	81.2	87.9	82.3	88.5
Ours				
BERT _{BASE} (Single)	80.8	88.5	-	-
BERT _{LARGE} (Single)	84.1	90.9	-	-
BERT _{LARGE} (Ensemble)	85.8	91.8	-	-
BERT _{LARGE} (Sgl.+TriviaQA)	84.2	91.1	85.1	91.8
BERT _{LARGE} (Ens.+TriviaQA)	86.2	92.2	87.4	93.2

5.4.3 命名实体识别任务

1. CoNLL 2003 Named Entity Recognition(NER) dataset 数据集：包含20万个单词的命名实体识别数据集。

单词被人工标注为： Person, Organization, Location, Miscellaneous 或者 Other (表示非命名实体)。

考虑到数据的预处理过程 (wordPiece 词干化) , 以 ## 开头的单词片段不需要进行 NER 识别。

2. BERT 在 CONLL-NER 任务上的超参数通过验证集来选取，最终给出的 Dev F1, Test F1 是使用最好的超参数、并且在5个随机重启的模型上的平均结果。

System	Dev F1	Test F1
ELMo+BiLSTM+CRF	95.7	92.2
CVT+Multi (Clark et al., 2018)	-	92.6
BERT _{BASE}	96.4	92.4
BERT _{LARGE}	96.6	92.8

5.4.4 探索实验

1. 对 MLM 和 NSP 的效果实验探索结果表明：

- 单向语言模型在 token-level 和 span-level 任务上表现很差。因为 token -level 的隐向量无法感知到右侧的上下文信息。
- NSP 对于句子级别任务的效果提升非常显著。

这些模型和 BERT_{BASE} 使用相同的训练语料、微调方式 和 Transformer 超参数。其中：

- No NSP : 使用了 MLM 但是未使用 NSP。
- LTR & No NSP : 使用了单向 LTR 的语言模型，而不是 MLM，也未使用 NSP。其结构类似 GPT 1.0。

同时在微调阶段也使用 LTR 而不是双向的。因为实验发现：单向的训练双向的微调导致效果较差。

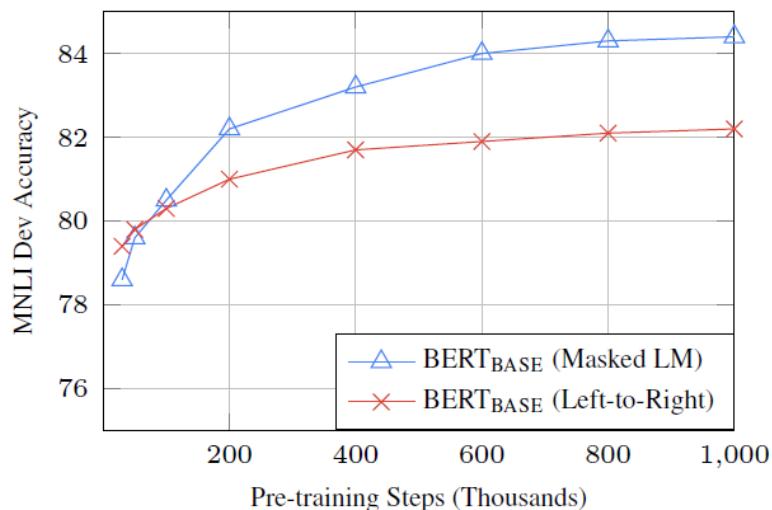
- + BiLSTM : 在 LTR + No NSP 的微调阶段在模型头部添加一个随机初始化的 BiLSTM 。

Tasks	Dev Set				
	MNLI-m (Acc)	QNLI (Acc)	MRPC (Acc)	SST-2 (Acc)	SQuAD (F1)
BERT _{BASE}	84.4	88.4	86.7	92.7	88.5
No NSP	83.9	84.9	86.5	92.6	87.9
LTR & No NSP	82.1	84.3	77.5	92.1	77.8
+ BiLSTM	82.1	84.1	75.7	91.6	84.9

2. 对预训练步数的实验探索结果表明：

- BERT 需要大量的训练步从而产生一个更好的效果。
- 双向语言模型（MLM）比单向语言模型（LTR）收敛速度稍微慢一点，但是其微调能力几乎一开始就强于单向语言模型模型。

其中每个训练步的模型是从训练过程中生成的 checkpoint 提取出来的。



3. 对模型尺寸的实验探索表明：BERT 模型越大，效果越好。

其中：#L 表示层数，#H 表示隐向量的维度，#A 表示 attention heads，LM(ppl) 表示 MLM 困惑度。

结果是微调阶段经过5次随机重启的验证集平均准确率。

#L	#H	#A	LM (ppl)	Hyperparams			Dev Set Accuracy		
				MNLI-m	MRPC	SST-2	MNLI-m	MRPC	SST-2
3	768	12	5.84	77.9	79.8	88.4			
6	768	3	5.24	80.6	82.2	90.7			
6	768	12	4.68	81.9	84.8	91.3			
12	768	12	3.99	84.4	86.7	92.9			
12	1024	16	3.54	85.7	86.9	93.3			
24	1024	16	3.23	86.6	87.8	93.7			

4. 对 BERT 应用于基于特征的策略而不是基于微调的策略表明：BERT 在基于微调和基于特征的方式上都能够取得很好的效果。

这里基于特征的方式为：将 BERT 的一个或多个层的隐向量采取类似于 ELMO 的方式灌入到一个两层 768 维度的 BiLSTM。

Layers	Dev F1
Finetune All	96.4
First Layer (Embeddings)	91.0
Second-to-Last Hidden	95.6
Last Hidden	94.9
Sum Last Four Hidden	95.9
Concat Last Four Hidden	96.1
Sum All 12 Layers	95.5

六、ERNIE

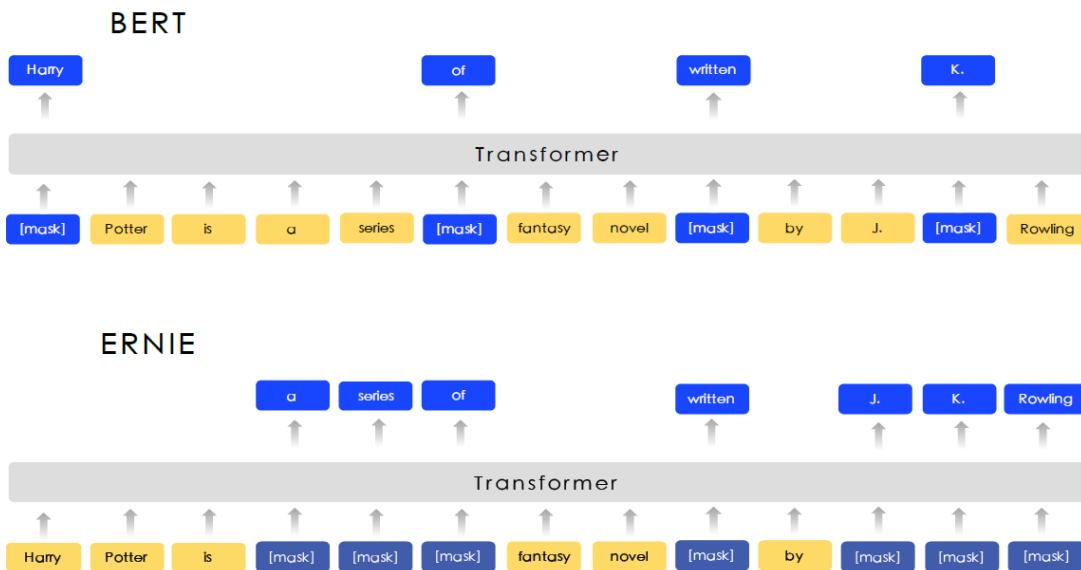
6.1 ERNIE 1.0

1. BERT 的 MLM 任务在执行 mask 的时候，未能考虑先验知识。如：假设要预测 `__ is a series of fantasy novels written by J. K. Rowling`。

如果有 Harry Potter 和 J. K. Rowling 关系的先验知识，则无需借助很长的上下文就可以很容易的预测出缺失的单词为 Harry Potter。

2. ERNIE: Enhanced Representation through Knowledge Integration 改变了 BERT 的 mask 策略，通过引入 entity-level mask 和 phrase-level mask 来引入先验知识。

但是 ERNIE 并没有直接添加先验知识，而是隐式的学习实体间的关系、实体的属性等知识。



6.1.1 Entity-level Mask & Phrase-level Mask

1. 与直接嵌入先验知识不同，ERNIE 将短语级 phrase-level 和实体级 entity-level 知识通过多阶段学习集成到语言表示中。

- ERNIE 将一个短语 phrase 或者一个实体 entity 作为一个单元，每个单元通常由几个 token 组成。在训练期间，同一个单元中的所有 token 都会被屏蔽，而不是只有一个 token 被屏蔽。

通过这种方式，ERNIE 在训练过程中隐式的学习短语或实体的先验知识。

- 对于英语，论文使用单词分析和分块工具来获取短语的边界；对于中文，论文使用分词工具来获取词的边界。
- 命名实体 entity 包括人名、地名、组织名、产品名等。
- ERNIE 通过三阶段学习来学得短语或实体的先验知识：
 - 第一阶段 Basic-level masking：使用基本的掩码策略，做法与 BERT 完全相同。

这个阶段是在基本语言单元的随机 `mask` 上训练，因此很难对高级语义知识建模。

对于英文，基本语言单元是词 `word`；对于中文，基本语言单元是汉字 `char`。

- 第二阶段 `Phrase-level masking`：使用基本语言单元作为训练输入，但是使用 `phrase-level` 的掩码策略。

这个阶段模型屏蔽和预测同一个短语的所有基本语言单元。

- 第三阶段 `Entity-level masking`：使用基本语言单元作为训练输入，但是使用 `entity-level` 的掩码策略。

这个阶段模型屏蔽和预测同一个命名实体的所有基本语言单元。

Sentence	Harry	Potter	is	a	series	of	fantasy	novels	written	by	British	author	J.	K.	Rowling
Basic-level Masking	[mask]	Potter	is	a	series	[mask]	fantasy	novels	[mask]	by	British	author	J.	[mask]	Rowling
Entity-level Masking	Harry	Potter	is	a	series	[mask]	fantasy	novels	[mask]	by	British	author	[mask]	[mask]	[mask]
Phrase-level Masking	Harry	Potter	is	[mask]	[mask]	[mask]	fantasy	novels	[mask]	by	British	author	[mask]	[mask]	[mask]

- ERNIE 编码器和 BERT 相同，但是对于中文语料，ERNIE 把 CJK 编码范围内的字符都添加空格来分隔，然后使用 `WordPiece` 来对中文语句词干化。

6.1.2 预处理

- ERNIE 使用异构语料库进行预训练，其中包括：

- 中文维基百科（2100万句子）
- 百度百科（5100万句子），包含书面语编写的百科文章
- 百度新闻（4700万句子），包含电影名、演员名，足球队名等等的最新信息
- 百度贴吧（5400万句子），每个帖子都被视为一个对话 `thread`

另外，ERNIE 执行汉字繁体到简体的转换、英文字符大写到小写的转换。最终模型的词汇表 17964 个 `unicode` 字符。

6.1.3 对话语言模型

- 对话数据对于语义表示很重要，因为相同回答对应的问题通常是相似的。因此 ERNIE 采用对话语言模型 `Dialogue Language Model:DLM` 任务对 `query - response` 对话进行建模，通过引入 `dialogue embedding` 来识别对话中的角色。

- `Q` 表示 `query` 的 `embedding`，`R` 表示 `response` 的 `embedding`。
- ERNIE 不仅可以表示单轮对话 `QR`，还可以表示多轮对话，如 `QRQ, QRR, QQR` 等等。



- 和 `MLM` 任务类似，`DLM` 任务随机 `mask` 掉 `query` 或者 `response` 中的 `token`，然后根据 `query` 和 `response` 来预测缺失的单词。

通过这种方式，`DLM` 帮助 ERNIE 去学习对话中的隐含关系，从而增强了模型学习语义表达的能力。

- 也可以将 `query` 或者 `response` 用一个随机挑选的句子替代，然后预测该对话是真的还是假的。
- `DLM` 任务的结构和 `MLM` 任务的结构是兼容的，因此可以直接替代 `MLM` 任务。

6.1.4 实验

- 为了与 `BERT-base` 比较，`ERNIE` 选择了同样的模型尺寸：`Transformer-Layer` 共 12 层，网络中隐向量均为 768 维，网络的 `multi-head` 为 12 头。
- `ERNIE` 在 5 项中文 `NLP` 任务中的表现如下图所示。这 5 项任务包含：自然语言推理 `natural language inference:NLI`，语义相似度 `semantic similarity`，命名实体识别 `named entity recognition:NER`，情感分析 `sentiment analysis:SA`，问答 `question answering:QA`。其中：
 - 自然语言推理数据集采用 `Cross-lingual Natural Language Inference :XNLI` 数据集：包含一个众包的 `MultinLI` 数据集，每一对被人工标记了蕴含关系，并被翻译成包括汉语在内的 14 种语言。
标签包含：矛盾 `contradiction`、中立 `neutral`、蕴含 `entailment`。
 - 语义相似度数据集采用 `Large-scale Chinese Question Matching Corpus:LCQMC` 数据集：每一对句子被人工标记一个二元标签，表示语义是否相似。
 - 命名实体识别数据集采用 `MSRA-NER` 数据集：由微软亚研院发布，包含几类实体：人名、地名、组织名称等。
 - 情感分析采用 `ChnSentiCorp` 数据集：包含了酒店、书籍、电脑等多个领域的评论及其人工标注的二元情感分类。
 - QA 采用 `NLPCC-DBQA` 数据集：包含了一些问题及对应的人工筛选出来的答案。

Task	Metrics	Bert		ERNIE	
		dev	test	dev	test
XNLI	accuracy	78.1	77.2	79.9 (+1.8)	78.4 (+1.2)
LCQMC	accuracy	88.8	87.0	89.7 (+0.9)	87.4 (+0.4)
MSRA-NER	F1	94.0	92.6	95.0 (+1.0)	93.8 (+1.2)
ChnSentiCorp	accuracy	94.6	94.3	95.2 (+0.6)	95.4 (+1.1)
nlpcc-dbqa	mrr	94.7	94.6	95.0 (+0.3)	95.1 (+0.5)
	F1	80.7	80.8	82.3 (+1.6)	82.7 (+1.9)

- `ERNIE` 对各种 `level` 的 `mask` 的实验结果如下图所示，其中 `10% of all` 表示使用 `1/10` 的语料库进行训练。

结果表明：

- 引入 `phrase-level mask` 能提高模型性能，引入 `entity-level mask` 可以进一步提高模型性能。
- 使用更大的预训练集，效果会更好。

pre-train dataset size	mask strategy	dev Accuracy	test Accuracy
10% of all	word-level(chinese character)	77.7%	76.8%
10% of all	word-level&phrase-level	78.3%	77.3%
10% of all	word-level&phrase-level&entity-level	78.7%	77.6%
all	word-level&phrase-level&entity-level	79.9 %	78.4%

- `ERNIE` 引入 `DLM` 任务的实验结果如下图所示，其中：

- `10% of all` 表示使用 `1/10` 的语料库来训练，但是这 `10%` 采用了不同的挑选策略：
 - `Baike(100%)`：这 `10%` 全部由百度百科语料组成。

- `Baike(84%) / news(16%)`：这 10% 由 84% 的百度百科语料、16% 的百度新闻语料组成。
- `Baike(71.2%)/news(13%)/forum Dialogue(15.7%)`：这 10% 由 71.2% 的百度百科、13% 的百度新闻、15.7% 的百度贴吧对话语料组成。
- 所有结果是在 `XNLI` 任务上执行5次随机重启的 `fine-tuning` 的均值。

corpus proportion(10% of all training data)	dev Accuracy	test Accuracy
Baike(100%)	76.5%	75.9%
Baike(84%) / news(16%)	77.0%	75.8%
Baike(71.2%)/ news(13%)/ forum Dialogue(15.7%)	77.7%	76.8%

5. 完形填空实验：将命名实体从段落中移除，然后模型需要预测被移除的实体。`ERNIE` 在完形填空实验的结果如下图所示。

- 例 1 中，`BERT` 试图复制上下文中出现的名称，而 `ERNIE` 记得文章中提到的关于实体关系的知识。
- 例 2 和 5 中，`BERT` 能根据上下文成功地学习模式，从而可以正确预测命名实体类型，但是预测了错误的实体。
- 例 3、4、6 中，`BERT` 用几个与句子相关的字符填充 `mask`，完全未能学习到缺失的实体对应的概念。
- 除了例 4 之外，`ERNIE` 都预测到了正确的实体。

对于例 4，虽然 `ERNIE` 预测错了实体，但是它给出的结果具有相同的语义类型。

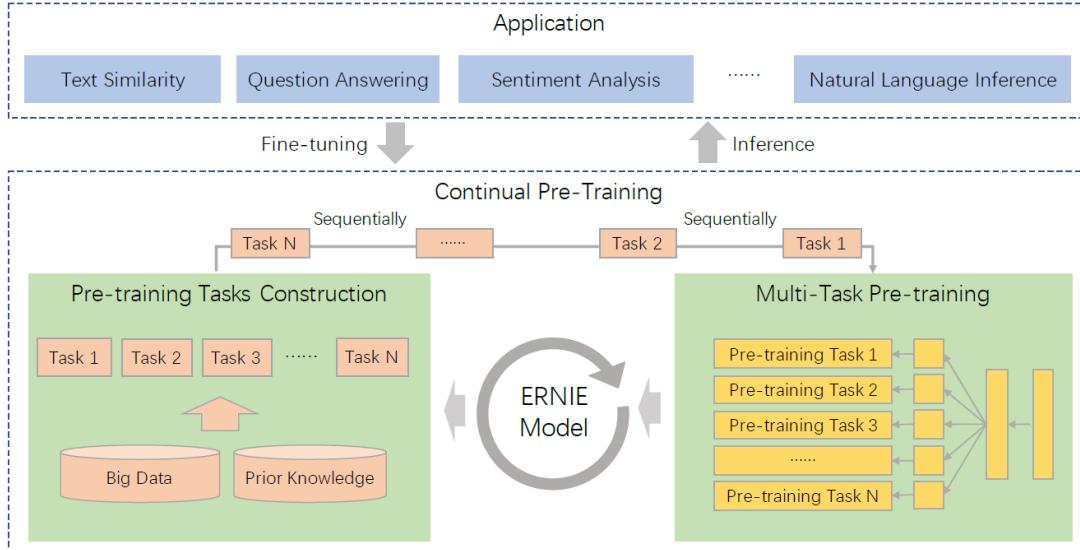
这些案例表明：`ERNIE` 在基于上下文的知识推理中表现得更好。

No	Text	Predict by ERNIE	Predict by BERT	Answer
1	2006年9月，_____与张柏芝结婚，两人婚后育有两儿子——大儿子Lucas谢振轩，小儿子Quintus谢振南； In September 2006, _____ married Cecilia Cheung. They had two sons, the older one is Zhenxuan Xie and the younger one is Zhennan Xie.	谢霆锋	Tingfeng Xie	Tingfeng Xie
2	戊戌变法，又称百日维新，是_____、梁启超等维新派人士通过光绪帝进行的一场资产阶级改良。 The Reform Movement of 1898, also known as the Hundred-Day Reform, was bourgeois reform carried out by the reformists such as _____ and Qichao Liang through Emperor Guangxu.	康有为	Shichang Sun	Youwei Kang
3	高血糖则是由于_____分泌缺陷或其生物作用受损，或两者兼有引起。糖尿病时长期存在的高血糖，导致各种组织，特别是眼、肾、心脏、血管、神经的慢性损害、功能障碍。 Hyperglycemia is caused by defective _____ secretion or impaired biological function, or both. Long-term hyperglycemia in diabetes leads to chronic damage and dysfunction of various tissues, especially eyes, kidneys, heart, blood vessels and nerves.	胰岛素	(Not a word in Chinese)	Insulin
4	澳大利亚是一个高度发达的资本主义国家，首都为_____。作为南半球经济最发达的国家和全球第12大经济体、全球第四大农产品出口国，其也是多种矿产出口量全球第一的国家。 Australia is a highly developed capitalist country with _____ as its capital. As the most developed country in the Southern Hemisphere, the 12th largest economy in the world and the fourth largest exporter of agricultural products in the world, it is also the world's largest exporter of various minerals.	墨尔本	(Not a city name)	Canberra (the capital of Australia)
5	_____是中国神魔小说的经典之作，达到了古代长篇浪漫主义小说的巅峰，与《三国演义》《水浒传》《红楼梦》并称为中国古典四大名著。 _____ is classic novel of Chinese gods and demons, which reaching the peak of ancient Romantic novels. It is also known as the four classical works of China with Romance of the Three Kingdoms, Water Margin and Dream of Red Mansions.	西游记	The Journey to the West	《西游记》
6	相对论是关于时空和引力的理论，主要由_____创立。 Relativity is a theory about space-time and gravity, which was founded by _____.	爱因斯坦	(Not a word in Chinese)	Einstein

6.2 ERNIE 2.0

1. `ERNIE 2.0` 提出了一个持续学习框架 `framework`，它通过持续的多任务学习来学习预训练模型。
 - 与其它预训练模型相同，它也是采取预训练阶段和微调阶段的两阶段模式。
 - 与其它预训练模型不同，其预训练阶段的预训练任务可以持续的、增量的构建，从而帮助模型有效的学习词法 `lexical`、句法 `syntactic` 和语义 `semantic` 表示。

ERNIE 2.0 : A Continual Pre-training framework for Language Understanding



2. ERNIE 2 受到人类学习过程的启发：人类能够不断积累那些通过学习获得的经验，然后基于过去的经验可以拓展出新的技能。

同样的，ERNIE 2 预期模型通过多个任务训练之后，在学习新任务时能够记住以前学得的知识。

3. 常见的预训练模型基于单词的共现或者句子的共现来训练模型，而 ERNIE 2 基于词法、句法、语义上的信息来训练模型。

- 命名实体可能包含一些概念，可用于捕获词法信息
- 句子之间的顺序、句子之间的相似度包含了一些语言结构，可用于捕获句法信息
- 文档之间的语义相似性、句子之间的对话关系包含了一些语言的语义，可用于捕获语义信息。

6.2.1 模型结构

1. 持续预训练阶段包含两个步骤：

- 不断构建包含大量数据和先验知识的无监督预训练任务，其中包括：word-aware 单词相关的任务、structure-aware 句子结构相关的任务、semantic-aware 句子语义相关的任务。

这些任务仅仅依赖于无需人工标注的自监督 self-supervised 语料，或者弱监督 weak-supervised 语料。

- 通过多任务学习不断更新 ERNIE 模型。

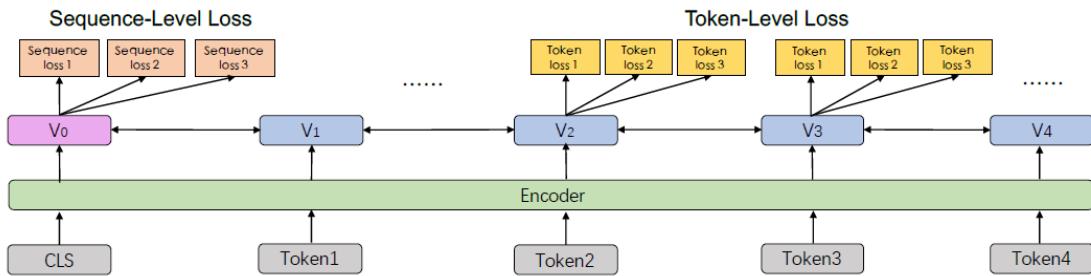
- 首先通过一个简单的预训练任务来训练模型。
- 然后每次添加一个新的预训练任务时，用前一个模型的参数来初始化当前模型。

每当引入一个新的预训练任务时，它就会和先前的任务一起训练，从而确保模型不会忘记它之前所学到的知识。这样 ERNIE 就能够不断学习和积累在学习过程中获得的知识。

2. 持续预训练阶段的训练框架包含一系列用于编码上下文信息的 encoder 层，这可以通过 RNN/LSTM/Transformer 来实现。encoder 层的参数在所有预训练任务中共享和更新。

每个预训练任务都有自己的损失函数。其中损失函数有两种类型：

- sequence-level 损失函数：每个序列一个 loss，如句子情感分析任务。
- token-level 损失函数：每个 token 一个 loss，如机器阅读理解任务。

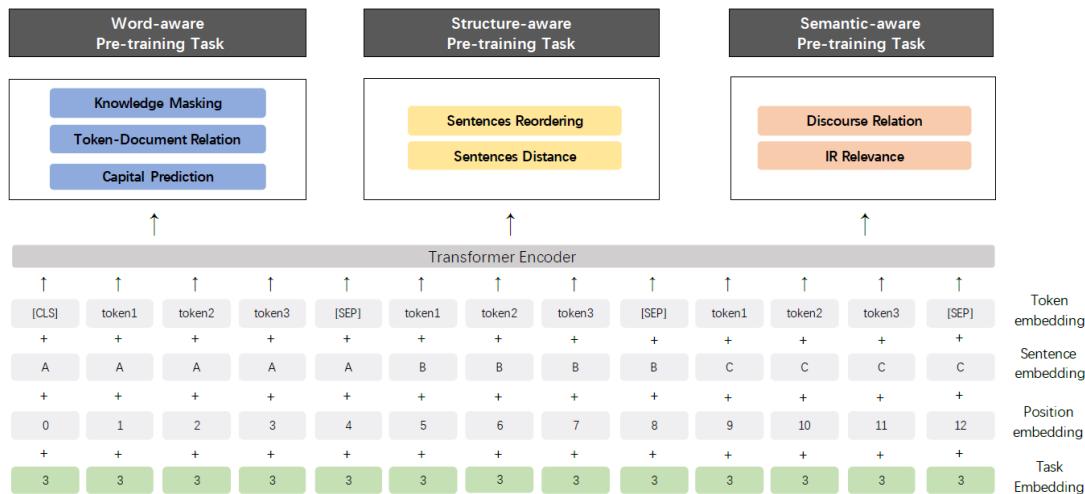


3. 微调阶段：使用每个下游任务的具体监督语料来进行微调，最终每个下游任务都有自己的微调模型。

4. ERNIE 2 给不同的预训练任务分配 task id，对每个 task 训练一个唯一的 task embedding。

因此 ERNIE 2 的输入 embedding 包含了 token embedding, segment embedding, position embedding, task embedding 这四种 embedding。

而在微调阶段，因为下游任务通常与预训练任务不同，此时选择任意一个 task id 的 task embedding 来初始化即可。



5. 预训练任务：

- word-aware 单词相关的任务：捕捉词法信息。
 - Knowledge Masking Task**：在 ERNIE 1.0 中使用的预训练任务，通过知识集成来提高模型表达能力。
该任务通过引入短语 mask 和命名实体 mask 来预测整个短语或者命名实体，从而帮助模型学习到局部上下文（短语、命名实体内部字符）和全局上下文（短语之间、命名实体之间）的依赖信息。
 - Capitalization Prediction Task**：预测一个单词是大写单词还是小写单词。
大写单词通常具有特定的语义价值。因此通过该任务的学习，模型有利于命名实体识别任务。
 - Token-Document Relation Prediction Task**：预测一个段落中的 token 是否出现在原始文档的其它段落中。
通常如果一个单词出现在文档的很多地方，则该单词是常用词或者文档的关键词。因此通过该任务，模型能够在一定程度上捕获文档关键词。
- structure-aware 句子结构相关的任务：捕获句法信息。
 - Sentence Reordering Task**：学习同一篇文档句子之间的顺序关系。
给定的段落被随机切分成 1 到 m 个部分，然后被随机混洗。模型需要预测其正确的句子顺序。

该问题视为一个 K 分类问题，其中 $K = \sum_{j=1}^m j!$ 。

- Sentence Distance Task：学习文档级别的句子之间的距离关系。

给定两个句子，如果它们在同一个文档中且相邻则标记为 0；如果在同一个文档中且不相邻则标记为 1；如果不在同一个文档中则标记为 2。

- semantic-aware 句子语义相关的任务：捕获句子语义信息。

- Discourse Relation Task：

- IR Relevance Task：学习搜索中的用户 query 和文档标题之间的关系。

给定用户 query 和文档标题：

- 如果用户查询后点击文档标题则标记为 0，表示强相关
- 如果文档出现在 query 的搜索结果里，但是用户未点击文档标题则标记为 1，表示弱相关
- 如果文档未能出现在 query 的搜索结果中则标记为 2，表示不相关

6.2.2 实验

1. 预训练数据：

- 英文训练语料库：除了 BERT 用到的英文维基百科、BookCorpus 数据集，ERNIE 2 还使用了 Reddit, Discovery 数据集。
- 中文训练语料库：采用百度搜索引擎收集到的各种数据，包括：百度百科、新闻数据、对话数据、检索日志、对话关系。

Corpus Type	English(#tokens)	Chinese(#tokens)
Encyclopedia	2021M	7378M
BookCorpus	805M	-
News	-	1478M
Dialog	4908M	522M
IR Relevance Data	-	4500M
Discourse Relation Data	171M	1110M

Table 1: The size of pre-training datasets.

2. 预训练配置：

- 为了和 BERT 比较，模型尺寸几乎相同。
 - base model 尺寸为：12 层 Transformer、12 头、隐向量 768 维
 - large model 尺寸为：24 层 Transformer、16 头、隐向量 1024 维
- 采用 Adam 优化器，参数为： $\beta_1 = 0.9, \beta_2 = 0.98$ 。
- 学习率为：基于 noam 衰减，其中初始学习率：英文模型 $5e-5$ ，中文模型 $1.28e-4$ 。并且前 4000 步采取 warmup 机制。
- batch size 包含 393216 个 token。
- 通过采用 float16 精度的操作来加快训练和降低内存使用。
- ERNIE 2.0 的 base model 在 48 块 Nvidia V100 GPU 显卡上训练，large model 在 64 块 Nvidia V100 GPU 显卡上训练。

3. 英文微调任务：在 GLUE 上微调，其中包括以下任务及其标记数据：

- Corpus of Linguistic Acceptability: CoLA：预测一个句子是否符合语法规范。
标记数据由来自 23 个语言学领域的 10657 个句子组成，人工标注其是否语法上可接受。
- Stanford Sentiment Treebank:SST-2：预测评论的情感。
标记数据由 9645 条电影评论组成，人工标注情感标签。
- Multi-genre Natural Language Inference:MNLI：预测句子之间的蕴含关系。

标记数据由众包的 43.3 万句子对组成，人工标注了句子之间的蕴含关系。

- Recognizing Textual Entailment:RTE：类似 MNLI，但是标记数据的规模更小。
- Winograd Natural Language Inference:WNLI：预测两段文本之间的共指信息。
- Quora Question Pairs：预测两个句子是否语义相同。
标记数据从 Quora 问答社区提取的40万句子对组成，人工标注了句子之间是否语义相同。
- Microsoft Research Paraphrase Corpus:MRPC：预测两个句子是否语义相同。
标记数据由互联网新闻中提取的 5800对句子组成，人工标注句子之间是否语义相同。
- Semantic Textual Similarity Benchmark:STS-B：预测两个句子之间语义是否相似。
标记数据包含从图片标题、新闻标题、用户论坛的文本，用于评估语义相似性。
- Question Natural Language Inference：预测一对给定文本之间的关系是否是问题和对应的答案。
- AX：一个辅助的手工制作的诊断测试工具，用于对模型分析。

数据集的大小如下图所示，其中 #Train, #Dev, #Test 分别给出了训练集、验证集、测试集大小，#Label 给出了类别集合的大小。

Corpus	Task	#Train	#Dev	#Test	#Label	Metrics
CoLA	Acceptability	8.5k	1k	1k	2	Matthews corr
SST-2	Sentiment	67k	872	1.8k	2	Accuracy
MNLI	NLI	393k	20k	20k	3	Accuracy
RTE	NLI	2.5k	276	3k	2	Accuracy
WNLI	NLI	634	71	146	2	Accuracy
QQP	Paraphrase	364k	40k	391k	2	Accuracy/F1
MRPC	Paraphrase	3.7k	408	1.7k	2	Accuracy/F1
STS-B	Similarity	7k	1.5k	1.4k	1	Pearson/Spearman corr
QNLI	QA/NLI	108k	5.7k	5.7k	2	Accuracy
AX	NLI	-	-	1.1k	3	Matthews corr

微调超参数参数：

Task	BASE			LARGE		
	Epoch	Learning Rate	Batch Size	Epoch	Learning Rate	Batch Size
CoLA	3	3e-5	64	5	3e-5	32
SST-2	4	2e-5	256	4	2e-5	64
MRPC	4	3e-5	32	4	3e-5	16
STS-B	3	5e-5	128	3	5e-5	128
QQP	3	3e-5	256	3	5e-5	256
MNLI	3	3e-5	512	3	3e-5	256
QNLI	4	2e-5	256	4	2e-5	256
RTE	4	2e-5	4	5	3e-5	16
WNLI	4	2e-5	8	4	2e-5	8

微调结果：

Task(Metrics)	BASE model		LARGE model				
	Test		Dev		Test		
	BERT	ERNIE 2.0	BERT	XLNet	ERNIE 2.0	BERT	
CoLA (Matthew Corr.)	52.1	55.2	60.6	63.6	65.4	60.5	63.5
SST-2 (Accuracy)	93.5	95.0	93.2	95.6	96.0	94.9	95.6
MRPC (Accuracy/F1)	84.8/88.9	86.1/89.9	88.0/-	89.2/-	89.7/-	85.4/89.3	87.4/90.2
STS-B (Pearson Corr./Spearman Corr.)	87.1/85.8	87.6/86.5	90.0/-	91.8/-	92.3/-	87.6/86.5	91.2/90.6
QQP (Accuracy/F1)	89.2/71.2	89.8/73.2	91.3/-	91.8/-	92.5/-	89.3/72.1	90.1/73.8
MNLI-m/mm (Accuracy)	84.6/83.4	86.1/85.5	86.6/-	89.8/-	89.1/-	86.7/85.9	88.7/88.8
QNLI (Accuracy)	90.5	92.9	92.3	93.9	94.3	92.7	94.6
RTE (Accuracy)	66.4	74.8	70.4	83.8	85.2	70.1	80.2
WNLI (Accuracy)	65.1	65.1	-	-	-	65.1	67.8
AX(Matthew Corr.)	34.2	37.4	-	-	-	39.6	48.0
Score	78.3	80.6	-	-	-	80.5	83.6

4. 中文微调任务：包括机器阅读理解、命名实体识别、自然语言推理、语义相似度、情感分析和问答等 9 个中文 NLP 任务。

- 机器阅读理解 Machine Reading Comprehension:MRC：典型的文档级建模任务，用于从给定文本中提取出连续的片段来回答问题。

数据集：

- **Chinese Machine Reading Comprehension 2018 : CMRC 2018**：由中国信息处理学会、**IFLYTEK** 和哈工大发布的机器阅读理解数据集。
- **Delta Reading Comprehension Dataset: DRCD**：由**Delta** 研究所发布的繁体中文机器阅读理解数据集，通过工具将它转换为简体中文。
- **DuReader**：百度在 2018年 **ACL** 发布的大规模中文机器阅读理解、问答数据集，所有问题是来自真实的、匿名的用户的 **query** 中得到的，问题的回答是人工生成的。
- 命名实体识别 **Named Entity Recognition:NER**：单词级别的建模任务，识别出文本中的命名实体。

数据集：微软亚洲研究院发布的 **MSRA-NER** 数据集

- 自然语言推理 **Natural Language Inference:NLI**：句子级别的建模任务，判定两个给定句子之间的蕴含关系。

数据集：**XNLI** 数据集。

- 情感分析 **Sentiment Analysis:SA**：句子级别的建模任务，用于判定给定的句子情感是积极的还是消极的。

数据集：**ChnSentiCorp** 数据集。该数据集包含酒店、书籍和 **PC** 等多个领域的评论。

- 语义相似度 **Semantic Similarity:ss**：句子级别的建模任务，用于判断给定两个句子是否语义相似。

数据集：

- **LCQMC**：哈工大 2018年发布的语义相似数据集
- **BQ Corpus**：由哈工大和 **Webank** 在 2018年 **EMNLP** 上发布的语义相似数据集
- 问答 **Question Answering:QA**：句子级别的建模任务，用于给每个问题挑选一个正确答案。

数据集：**NLPCC-DBQA**，该数据集 2016年在 **NLPCC** 上发布。

数据集的大小如下图所示，其中 **#Train, #Dev, #Test** 分别给出了训练集、验证集、测试集大小，**#Label** 给出了类别集合的大小。

Corpus	Task	#Train	#Dev	#Test	#Label	Metrics
CMRC 2018	MRC	10K	3.2K	-	-	EM/F1
DRCD	MRC	27K	3.5K	3.5K	-	EM/F1
DuReader	MRC	271.5K	10K	-	-	EM/F1
MSRA-NER	NER	21K	2.3k	4.6K	7	F1
XNLI	NLI	392K	2.5K	2.5K	3	Accuracy
ChnSentiCorp	SA	9.6K	1.2K	1.2K	2	Accuracy
LCQMC	SS	240K	8.8K	12.5K	2	Accuracy
BQ Corpus	SS	100K	10K	10K	2	Accuracy
NLPCC-DBQA	QA	182K	41K	82K	2	mrr/F1

中文微调超参数：

Task	BASE			LARGE		
	Epoch	Learning Rate	Batch Size	Epoch	Learning Rate	Batch Size
CMRC 2018	2	3e-5	64	2	3e-5	64
DRCD	2	5e-5	64	2	3e-5	64
DuReader	2	5e-5	64	2	2e-5	64
MSRA-NER	6	5e-5	16	6	1e-5	16
XNLI	3	1e-4	65536(#tokens)	3	4e-5	65536(#tokens)
ChnSentiCorp	10	5e-5	24	10	1e-5	24
LCQMC	3	2e-5	32	3	5e-6	32
BQ Corpus	3	3e-5	64	3	1.5e-5	64
NLPCC-DBQA	3	2e-5	64	3	1e-5	64

中文微调结果：

Task	Metrics	BERT _{BASE}		ERNIE 1.0 _{BASE}		ERNIE 2.0 _{BASE}		ERNIE 2.0 _{LARGE}	
		Dev	Test	Dev	Test	Dev	Test	Dev	Test
CMRC 2018	EM/F1	66.3/85.9	-	65.1/85.1	-	69.1/88.6	-	71.5/89.9	-
DRCD	EM/F1	85.7/91.6	84.9/90.9	84.6/90.9	84.0/90.5	88.5/93.8	88.0/93.4	89.7/94.7	89.0/94.2
DuReader	EM/F1	59.5/73.1	-	57.9/72.1	-	61.3/74.9	-	64.2/77.3	-
MSRA-NER	F1	94.0	92.6	95.0	93.8	95.2	93.8	96.3	95.0
XNLI	Accuracy	78.1	77.2	79.9	78.4	81.2	79.7	82.6	81.0
ChnSentiCorp	Accuracy	94.6	94.3	95.2	95.4	95.7	95.5	96.1	95.8
LCQMC	Accuracy	88.8	87.0	89.7	87.4	90.9	87.9	90.9	87.9
BQ Corpus	Accuracy	85.9	84.8	86.1	84.8	86.4	85.0	86.5	85.2
NLPCC-DBQA	MRR/F1	94.7/80.7	94.6/80.8	95.0/82.3	95.1/82.7	95.7/84.7	95.7/85.3	95.9/85.3	95.8/85.8

七、XLNet

7.1 自回归语言模型 vs 自编码语言模型

1. 无监督预训练有两种策略：

- 自回归语言模型 autoregressive language model : AR LM : 通过自回归模型来预测一个序列的生成概率。

令序列为 $\mathbf{w} = (w_1, w_2, \dots, w_n)$, 则模型计算概率

$$p(\mathbf{w}; \Theta) = \prod_{i=1}^n p(w_i | w_1, \dots, w_{i-1}; \Theta), \text{ 或者反向概率}$$

$$p(\mathbf{w}; \Theta) = \prod_{i=n}^1 p(w_i | w_{i+1}, \dots, w_n; \Theta).$$

令 $\vec{\mathbf{h}}_\Theta(w_1, \dots, w_{i-1})$ 是通过 RNN 或者 Transformer 提取的单向上下文 representation 向量, $\vec{\mathbf{e}}(w)$ 是 w 的 embedding 向量, 则有:

$$p(w_i | w_1, \dots, w_{i-1}; \Theta) = \frac{\exp(\vec{\mathbf{h}}_\Theta(w_1, \dots, w_{i-1}) \cdot \vec{\mathbf{e}}(w_i))}{\sum_{w'} \exp(\vec{\mathbf{h}}_\Theta(w_1, \dots, w_{i-1}) \cdot \vec{\mathbf{e}}(w'))}$$

$$\log p(\mathbf{w}; \Theta) = \sum_{i=1}^n \log \frac{\exp(\vec{\mathbf{h}}_\Theta(w_1, \dots, w_{i-1}) \cdot \vec{\mathbf{e}}(w_i))}{\sum_{w'} \exp(\vec{\mathbf{h}}_\Theta(w_1, \dots, w_{i-1}) \cdot \vec{\mathbf{e}}(w'))}$$

给定预训练数据集 \mathbb{D} , 模型基于最大对数似然的准则来最优化目标函数:

$$\max_{\Theta} \sum_{\mathbf{w} \in \mathbb{D}} \log p(\mathbf{w}; \Theta)$$

即：预训练数据集 \mathbb{D} 的生成概率最大。

- 自编码语言模型 autoencoding language model : AE LM : 通过自编码模型来从损坏的输入中重建原始数据。

令原始序列为 $\mathbf{w} = (w_1, w_2, \dots, w_n)$, 添加多个噪音之后的序列为:

$$\hat{\mathbf{w}} = (w_1, \dots, w_{i-1}, [\text{MASK}], w_{i+1}, \dots, [MASK], \dots, w_n)$$

令这些被 mask 的 token 为: $\bar{\mathbf{w}} = (w_i, \dots)$, 则模型计算概率:

$$p(\bar{\mathbf{w}} | \hat{\mathbf{w}}; \Theta) \simeq \prod_{i=1}^n \mathbb{I}_i \times p(w_i | \hat{\mathbf{w}}; \Theta)$$

其中 \mathbb{I}_i 是一个示性函数, 当第 i 个位置被 mask 时取值为 1; 否则取值为 0。

令 $H_\Theta(\cdot)$ 为一个 Transformer 函数, 它将长度为 n 的序列 $\hat{\mathbf{w}}$ 映射成一个长度为 n 的 representation 向量序列: $\{\vec{\mathbf{h}}_\Theta^1(\hat{\mathbf{w}}), \dots, \vec{\mathbf{h}}_\Theta^n(\hat{\mathbf{w}})\}$, 其中第 i 个位置的 representation 向量为 $\vec{\mathbf{h}}_\Theta^i(\hat{\mathbf{w}})$ 。令 $\vec{\mathbf{e}}(w)$ 是 w 的 embedding 向量, 则有:

$$\log p(w_i | \hat{\mathbf{w}}; \Theta) = \log \frac{\exp(\vec{\mathbf{h}}_\Theta^i(\hat{\mathbf{w}}) \cdot \vec{\mathbf{e}}(w_i))}{\sum_{w'} \exp(\vec{\mathbf{h}}_\Theta^i(\hat{\mathbf{w}}) \cdot \vec{\mathbf{e}}(w'))}$$

$$\log p(\bar{\mathbf{w}} | \hat{\mathbf{w}}; \Theta) \simeq \sum_{i=1}^n \mathbb{I}_i \log \frac{\exp(\vec{\mathbf{h}}_\Theta^i(\hat{\mathbf{w}}) \cdot \vec{\mathbf{e}}(w_i))}{\sum_{w'} \exp(\vec{\mathbf{h}}_\Theta^i(\hat{\mathbf{w}}) \cdot \vec{\mathbf{e}}(w'))}$$

给定预训练数据集 \mathbb{D} , 首先通过预处理生成带噪音的数据集 $\tilde{\mathbb{D}}$, 然后模型最大化目标函数:

$$\max_{\Theta} \sum_{\hat{\mathbf{w}} \in \mathbb{D}} \log p(\bar{\mathbf{w}} | \hat{\mathbf{w}}; \Theta)$$

2. 自回归语言模型和自编码语言模型的区别:

- 自回归语言模型是生成式模型, 自编码语言模型是判别式模型。
- 自回归语言模型是单向的 (前向或者后向), 自编码语言模型是双向的, 可以使用双向上下文

3. 自回归语言模型和自编码语言模型的缺点:

- 自回归语言模型是单向模型, 它无法对双向上下文进行建模。而下游任务通常需要双向上下文的信息, 这使得模型的能力不满足下游任务的需求。
- 自编码语言模型:
 - 在预训练期间引入噪声 [MASK], 而在下游任务中该噪声并不存在, 这使得预训练和微调之间产生差距。
 - 根据概率分解 $p(\bar{\mathbf{w}} | \hat{\mathbf{w}}; \Theta) \simeq \prod_{i=1}^n \mathbb{I}_i \times p(w_i | \hat{\mathbf{w}}; \Theta)$ 可以看到: 自编码语言模型假设序列中各位置处的 [MASK] 彼此独立, 即: 预测目标独立性假设。

事实上, 这和实际情况不符合。如: 这里有很多水果, 比如苹果、香蕉、葡萄 经过引入噪音之后: 这里有很多水果, 比如 [MASK] [MASK]、香蕉、葡萄。实际上这两个 [MASK] 之间并不是相互独立。

4. XLNet 是一种广义自回归语言模型, 它结合了自回归语言模型和自编码语言模型的优点, 同时避免了它们的局限性。

- 对于每个 token $w \in \mathbf{w}$, 通过概率分解的排列顺序全组合的形式, w 可以 "见到" \mathbf{w} 中的所有其它 token, 因此可以捕获到双向上下文。
即: 在某些排列顺序下, w 可以见到它的左侧上下文; 在另一些排列顺序下, w 可以见到它右侧上下文。
- XLNet 没有引入噪音 [MASK], 因此预训练阶段和微调阶段不存在差异。
- XLNet 基于乘积规则来分解整个序列的联合概率, 消除了自编码语言模型中的预测目标独立性假设。

7.2 Permutation Language Model

1. XLNet 是一个基于全组合的语言模型 permutation language modeling。
2. 设一个长度为 3 的文本序列 $\mathbf{w} = (w_1, w_2, w_3)$, 一共 $3! = 6$ 种顺序来实现一个有效的自回归概率分解:

$$\begin{aligned} p(\mathbf{w}) &= p(w_1) \times p(w_2 | w_1) \times p(w_3 | w_1, w_2) \\ &= p(w_1) \times p(w_3 | w_1) \times p(w_2 | w_1, w_3) \\ &= p(w_2) \times p(w_3 | w_2) \times p(w_1 | w_2, w_3) \\ &= p(w_2) \times p(w_1 | w_2) \times p(w_3 | w_1, w_2) \\ &= p(w_3) \times p(w_1 | w_3) \times p(w_2 | w_1, w_3) \\ &= p(w_3) \times p(w_2 | w_3) \times p(w_1 | w_2, w_3) \end{aligned}$$

这等价于先将序号 (1, 2, 3) 进行全组合, 得到 $3! = 6$ 种顺序:

$$(1, 2, 3), (1, 3, 2), (2, 3, 1), (2, 1, 3), (3, 1, 2), (3, 2, 1)$$

然后对每一种顺序，按照从左到右执行自回归概率分解。

- 对于一个长度为 T 的文本序列 $\mathbf{w} = (w_1, \dots, w_T)$ ，则有 $T!$ 种顺序来实现一个有效的自回归概率分解。如果在这些概率分解之间共享模型参数，则模型能够捕捉到所有位置的信息，即：模型能够访问双向上下文。
- 这种分解方式也考虑了 `token` 之间的顺序。

令 W_1, W_2, W_3 分别代表第一、二、三个位置的随机变量，则有：

$$\begin{aligned} p(\mathbf{w}) &= p(W_1 = w_1, W_2 = w_2, W_3 = w_3) = \\ &p(W_1 = w_1) \times p(W_2 = w_2 | W_1 = w_1) \times p(W_3 = w_3 | W_1 = w_1, W_2 = w_2) \\ &\quad \vdots \\ &= p(W_3 = w_3) \times p(W_2 = w_2 | W_3 = w_3) \times p(W_1 = w_1 | W_2 = w_2, W_3 = w_3) \end{aligned}$$

其物理意义为：序列 (w_1, w_2, w_3) 可以有如下的生成方式：

- 首先挑选第一个位置的 `token` w_1 ，然后已知第一个位置的 `token` 条件下挑选第二个位置的 `token` w_2 ，最后在已知第一个、第二个位置的 `token` 条件下挑选第三个位置的 `token` w_3
-
- 首先挑选第三个位置的 `token` w_3 ，然后已知第三个位置的 `token` 条件下挑选第二个位置的 `token` w_2 ，最后在已知第三个、第二个位置的 `token` 条件下挑选第一个位置的 `token` w_1

3. 对于一个长度为3的文本序列 $\mathbf{w} = (w_1, w_2, w_3)$ ，传统的自回归语言模型的目标函数是生成概率的对数似然：

$$L = \log p(\mathbf{w}) = \log p(w_1) + \log p(w_2 | w_1) + \log p(w_3 | w_1, w_2)$$

这仅仅对应于上述 6 种概率分解中的第一个分解。

`XLNet` 与此不同，它考虑所有的 6 种概率分解，其目标函数为：

$$\begin{aligned} L &= \log p(w_1) + \log p(w_2 | w_1) + \log p(w_3 | w_1, w_2) \\ &+ \log p(w_1) + \log p(w_3 | w_1) + \log p(w_2 | w_1, w_3) \\ &+ \log p(w_2) + \log p(w_3 | w_2) + \log p(w_1 | w_2, w_3) \\ &+ \log p(w_2) + \log p(w_1 | w_2) + \log p(w_3 | w_1, w_2) \\ &+ \log p(w_3) + \log p(w_1 | w_3) + \log p(w_2 | w_1, w_3) \\ &+ \log p(w_3) + \log p(w_2 | w_3) + \log p(w_1 | w_2, w_3) \end{aligned}$$

4. 令 \mathbb{Z}_T 表示位置编号 $(1, 2, \dots, T)$ 的所有可能排列的组合，令 \mathbf{z} 为 \mathbb{Z}_T 的某个排列，如：
 $\mathbf{z} = (1, 3, 5, \dots, 2, 4, 6, \dots)$ 。

令 z_t 为该排列的第 t 个位置编号，如： $z_5 = 9$ ；令 $\mathbf{z}_{<t}$ 为该排列中前 $t - 1$ 个位置编号，如 $\mathbf{z}_{<5} = (1, 3, 5, 7)$ 。

令 $\mathbf{w}_{\mathbf{z}_{<t}}$ 为 $\mathbf{w} = (w_1, w_2, \dots, w_T)$ 中的编号位于 $\mathbf{z}_{<t}$ 的 `token` 组成的序列。

`XLNet` 考虑所有的概率分解，因此其目标函数为：

$$L = \sum_{\mathbf{z} \in \mathbb{Z}_T} \sum_{t=1}^T \log p_{\Theta}(w_{z_t} | \mathbf{w}_{\mathbf{z}_{<t}})$$

假设每一种排列是等概率的，均为 $P_T = \frac{1}{T!}$ 。则有：

$$L = T! \times \sum_{\mathbf{z} \in \mathbb{Z}_T} P_T \sum_{t=1}^T \log p_{\Theta}(w_{z_t} | \mathbf{w}_{\mathbf{z}_{<t}}) = T! \times \mathbb{E}_{\mathbf{z} \in \mathbb{Z}_T} \left[\sum_{t=1}^T \log p_{\Theta}(w_{z_t} | \mathbf{w}_{\mathbf{z}_{<t}}) \right]$$

最终 `XLNet` 的目标函数为：

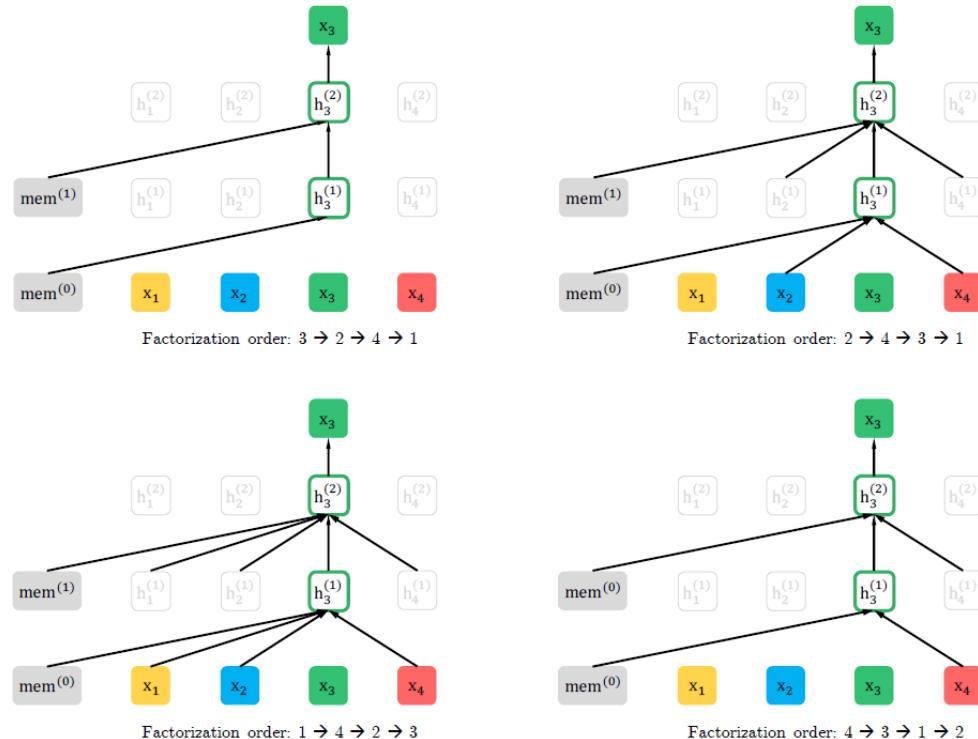
$$L = \mathbb{E}_{\mathbf{z} \in \mathbb{Z}_T} \left[\sum_{t=1}^T \log p_{\Theta}(w_{z_t} | \mathbf{w}_{\mathbf{z}_{<t}}) \right]$$

- 采用期望的形式是因为：如果考虑所有的概率分解，则需要计算 $T!$ 种组合的概率。当序列长度为 $T = 16$ 时 组合数量 $16!$ 超过 20 万亿次，计算复杂度太大导致无法计算。
所以采用期望的形式，然后利用采样来计算期望，可以使得计算复杂度大大降低。
因此对于给定的输入序列 $\mathbf{w} = (w_1, \dots, w_T)$ ，训练期间每次采样一个顺序 \mathbf{z} ，然后根据该顺序进行概率分解。
- 参数 Θ 在所有的排列顺序中共享。

5. XLNet 仅仅调整了联合概率 $p(\mathbf{w})$ 的分解顺序，并没有调整序列 \mathbf{w} 本身的 token 顺序。因此在 XLNet 中保留序列 \mathbf{w} 的原始顺序，并使用原始顺序的 position embedding。

- 联合概率 $p(\mathbf{w})$ 的分解顺序的调整依赖于 Transformer 的 mask 机制。

下图为输入序列 $\mathbf{x} = (x_1, x_2, x_3, x_4)$ 中，计算 x_3 的几种排列顺序。（右下角的图形应该为 $\text{mem}^{(0)}$ 和 $\text{mem}^{(1)}$ ）。



- 因为在微调阶段遇到的文本序列都是原始顺序的，所以这种方式保证了预训练阶段和微调阶段在输入文本序列的顺序上是一致的：都是原始顺序。

7.3 Two-Stream Self-Attention

1. 虽然 permutation language model 有诸多优点，但是如果直接采用标准的 Transformer 结构来实现该模型，则会出现问题。

考虑输入序列 $\mathbf{w} = (w_1, w_2, \dots, w_5)$ ，考虑位置编号 $(1, 2, \dots, 5)$ 的两种排列顺序：

$$\mathbf{z}^{(1)} = (2, 5, 1, 4, 3), \quad \mathbf{z}^{(2)} = (2, 5, 4, 3, 1)$$

当 $t = 3$ 时，有：

$$z_3^{(1)} = 1, \quad \mathbf{z}_{<3}^{(1)} = (2, 5); \quad z_3^{(2)} = 4, \quad \mathbf{z}_{<3}^{(2)} = (2, 5)$$

对于第一种排列，条件概率分布 $p_{\Theta}(w_{z_3^{(1)}} = w | \mathbf{w}_{\mathbf{z}^{(1)} < 3}) = p_{\Theta}(w_1 = w | w_2, w_5)$ ，其物理意义为：在序列第二个 token 为 w_2 、第五个 token 为 w_5 的条件下，第一个 token 为 w 的概率分布。

对于第二种排列，条件概率分布 $p_{\Theta}(w_{z_3^{(2)}} = w | \mathbf{w}_{\mathbf{z}^{(2)} < 3}) = p_{\Theta}(w_4 = w | w_2, w_5)$ ，其物理意义为：在序列第二个 token 为 w_2 、第五个 token 为 w_5 的条件下，第四个 token 为 w 的概率分布。

对于标准 Transformer，有：

$$p_{\Theta}(w | w_2, w_5) = \frac{\exp(\vec{h}_{\Theta}(w_2, w_5) \cdot \vec{e}(w))}{\sum_{w'} \exp(\vec{h}_{\Theta}(w_2, w_5) \cdot \vec{e}(w'))}$$

其中 $\vec{h}_{\Theta}(w_2, w_5)$ 表示 Transformer 从输入序列的第二个 token、第五个 token (w_2, w_5) 中抽取的 representation 向量； $\vec{e}(w)$ 表示 token w 的 embedding。

因此当已知第二个 token w_2 和第五个 token w_5 ，无论预测目标位置是第一个位置（对应于 w_1 ）还是第四个位置（对应于 w_4 ），其概率分布都相同。实际上，对于不同的目标位置，其概率分布应该是不同的。

该问题产生的本质原因是： $p_{\Theta}(w | w_2, w_5)$ 未能考虑预测目标 w 的位置，当 w 代表第一个 token，与 w 代表第四个 token，其分布应该不同。

2. 解决该问题的方式是：对预测目标 token 引入位置信息。

重新定义条件概率分布：

$$p_{\Theta}(w_{z_t} | \mathbf{w}_{z < t}) = \frac{\exp(\vec{g}_{\Theta}(\mathbf{w}_{z < t}, z_t) \cdot \vec{e}(w_{z_t}))}{\sum_{w'} \exp(\vec{g}_{\Theta}(\mathbf{w}_{z < t}, z_t) \cdot \vec{e}(w'))}$$

其中 $\vec{g}_{\Theta}(\mathbf{w}_{z < t}, z_t)$ 定义了序列的一个新的 representation，它考虑了目标位置 z_t 。

如上述例子中：

- 排列顺序 $\mathbf{z}^{(1)}$ 中，预测 $w_{z_3^{(1)}}$ 的条件概率分布为：

$$p_{\Theta}(w_1 | w_2, w_5) = \frac{\exp(\vec{g}_{\Theta}(w_2, w_5, 1) \cdot \vec{e}(w_1))}{\sum_{w'} \exp(\vec{g}_{\Theta}(w_2, w_5, 1) \cdot \vec{e}(w'))}$$

- 排列顺序 $\mathbf{z}^{(2)}$ 中，预测 $w_{z_3^{(2)}}$ 的条件概率分布为：

$$p_{\Theta}(w_4 | w_2, w_5) = \frac{\exp(\vec{g}_{\Theta}(w_2, w_5, 4) \cdot \vec{e}(w_4))}{\sum_{w'} \exp(\vec{g}_{\Theta}(w_2, w_5, 4) \cdot \vec{e}(w'))}$$

由于 $\vec{g}_{\Theta}(w_2, w_5, 1) \neq \vec{g}_{\Theta}(w_2, w_5, 4)$ ，因此不同的目标位置，其概率分布会不同。

3. 问题是：如何定义一个合理的 $\vec{g}_{\Theta}(\mathbf{w}_{z < t}, z_t)$ 函数。

- 当基于上下文 $\mathbf{w}_{z < t}$ 来预测位置 z_t 的单词时，要求最多能够使用位置 z_t ，而不能使用位置 z_t 的内容 w_{z_t} 。

即：对于序列 \mathbf{w} ， $\vec{g}_{\Theta}(\mathbf{w}_{z < t}, z_t)$ 一定不能对 w_{z_t} 进行编码。否则，当利用该编码来预测 w_{z_t} 时损失函数一定为零，因为这相当于给定单词 w_{z_t} 来预测它本身。

- 当预测 $j > t$ 之后的 token w_{z_j} 时， $\vec{g}_{\Theta}(\mathbf{w}_{z < t}, z_t)$ 必须对 w_{z_t} 进行编码，从而为预测 w_{z_j} 提供上下文。

即：对于预测位置 z_{t+1}, z_{t+2}, \dots 的 token， $\vec{g}_{\Theta}(\mathbf{w}_{z < t}, z_t)$ 必须对 w_{z_t} 进行编码。

总而言之，是否对 w_{z_t} 进行编码依赖于当前预测的目标单词的位置 z_j 。但是 Transformer 中，每个 token 的 embedding 信息和位置的 embedding 信息在一开始就融合在一起。

解决这一对冲突的方法时 two-stream self attention 机制：

- content stream：提供了内容表达 content representation $\mathbf{H}_{\Theta}(\mathbf{w}_{z \leq t})$ ，记作 \mathbf{H}_{z_t} ，它类似于标准 Transformer 中的隐状态向量，编码了上下文和 w_{z_t} 本身。

$$\mathbf{H}_{z_t} = \begin{bmatrix} \vec{h}_{z_1}^T \\ \vec{h}_{z_2}^T \\ \vdots \\ \vec{h}_{z_t}^T \end{bmatrix} \in \mathbb{R}^{t \times d}$$

其中 $\vec{h}_{z_i}, i = 1, 2, \dots, t$ 为 w_{z_i} 的内容隐状态向量，它是 d 维列向量。

- query stream：提供了查询表达 content representation $\mathbf{G}_\Theta(\mathbf{w}_{z < t}, z_t)$ ，记作 \mathbf{G}_{z_t} ，它仅编码了 w_{z_t} 的上下文及其位置 z_t ，并未编码 w_{z_t} 本身。

$$\mathbf{G}_{z_t} = \begin{bmatrix} \vec{\mathbf{g}}_{z_1}^T \\ \vec{\mathbf{g}}_{z_2}^T \\ \vdots \\ \vec{\mathbf{g}}_{z_{t-1}}^T \end{bmatrix} \in \mathbb{R}^{(t-1) \times d}$$

其中 $\vec{\mathbf{g}}_{z_i}, i = 1, 2, \dots, t-1$ 为 w_{z_i} 的 query 隐状态向量，它是 d 维列向量。

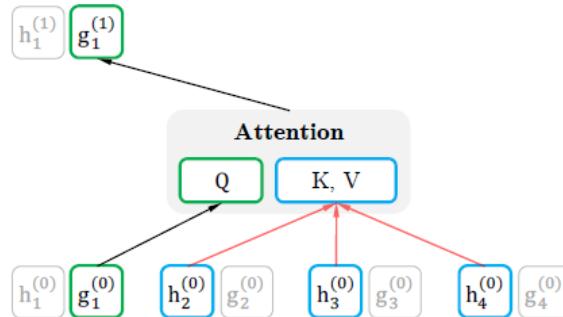
4. 双路 self-Attention 的更新规则：

- 第一层的 query 状态向量通过一个参数 $\vec{\mathbf{w}}$ 来初始化： $\vec{\mathbf{g}}_i^{(0)} = \vec{\mathbf{w}}$ ，其中 $\vec{\mathbf{w}}$ 是待学习的参数。
- 第一层的 content 状态向量通过 word embedding 初始化： $\vec{\mathbf{h}}_i^{(0)} = \vec{\mathbf{e}}(w_i)$ 。
- 对每一个 self-attention 层， $m = 1, 2, \dots$ ，这两路 self-attention 通过共享参数来更新：

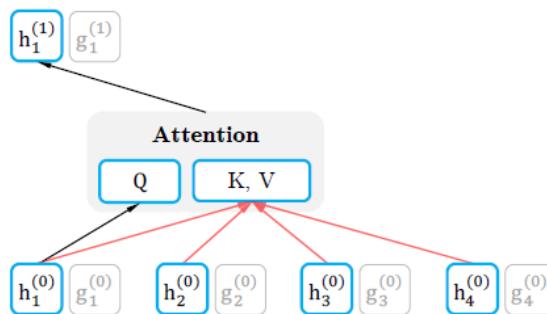
$$\begin{aligned} \vec{\mathbf{g}}_{z_t}^{(m)} &\leftarrow \text{Attention}(Q = \vec{\mathbf{g}}_{z_t}^{(m-1)}, KV = \mathbf{H}_{z_{t-1}}^{(m-1)}; \Theta) \\ \vec{\mathbf{h}}_{z_t}^{(m)} &\leftarrow \text{Attention}(Q = \vec{\mathbf{h}}_{z_t}^{(m-1)}, KV = \mathbf{H}_{z_t}^{(m-1)}; \Theta) \end{aligned}$$

其中 Attention 的计算规则与 BERT 的 Self-Attention 一致。

- query-stream 状态更新如下图所示：位置 z_t 的 query 状态更新只可能用到 $\{z_1, z_2, \dots, z_{t-1}\}$ 位置的内容信息（不包括它自身）。

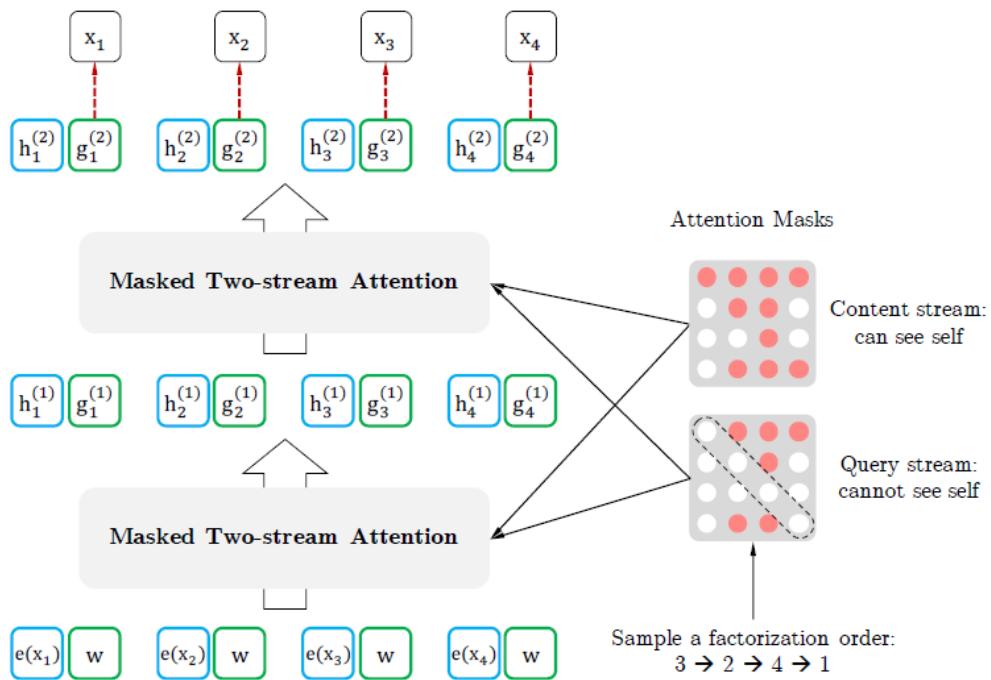


- content-stream 状态更新如下图所示：位置 z_t 的 content 状态更新用到 $\{z_1, z_2, \dots, z_t\}$ 位置的内容信息（包括它自身）。



- 位于最后一层的、位置 z_t 的 query 状态向量就是我们想要的 $\vec{\mathbf{g}}_\Theta(\mathbf{w}_{z < t}, z_t)$ 函数。
 - 通过前面定义的 Attention 机制，它包含了 $\{z_1, z_2, \dots, z_{t-1}\}$ 位置的内容信息（不包括它自身）。
 - 通过指定位置 z_t 处的 query 状态向量（而不是指定其它位置），它包含了位置 z_t 的位置信息。
- 整体更新示意图如下所示：

采样顺序为 $3 \rightarrow 2 \rightarrow 4 \rightarrow 1$ ，Attention Masks 第 k 行表示 w_k 的上下文。



5. 微调阶段：仅仅使用 `content stream self-attention`，并采用 `Transformer-XL` 的推断机制。

7.4 Partial Prediction

1. `permutation language model` 会带来优化困难：由于位置排列的各种组合导致收敛速度很慢。

为解决该问题，模型仅预测位置排列 \mathbf{z} 最后面的几个位置对应的 `token`。

- 首先根据拆分点 c 拆分 \mathbf{z} 为： $\mathbf{z}_{\leq c}$ （不预测）、 $\mathbf{z}_{>c}$ （预测）。
- 然后调整模型的优化目标为：

$$\max_{\Theta} \mathbb{E}_{\mathbf{z} \in \mathcal{Z}_T} \log p_{\Theta}(\mathbf{w}_{\mathbf{z}_{>c}} \mid \mathbf{w}_{\mathbf{z}_{\leq c}}) = \max_{\Theta} \mathbb{E}_{\mathbf{z} \in \mathcal{Z}_T} \left[\sum_{t=c+1}^T \log p_{\Theta}(w_{z_t} \mid \mathbf{w}_{\mathbf{z}_{<t}}) \right]$$

这称作 `Partial Prediction` 部分预测。

对于拆分点之前的 `token`，无需计算它们的 `query representation`，这会大大节省内存，降低计算代价。

2. 部分预测选择后半部分来预测的原因是：

- 后半部分的 `token` 具有较长的上下文，使得上下文信息更丰富，从而为模型提供了更丰富的输入特征，更有利与模型对上下文的特征抽取。
- 推断阶段，真实的序列一般都具有较长的上下文。根据第一点的结论，模型更有利与对这些序列的特征抽取。

3. 拆分点 c 的选择由超参数 K 来选择。定义：

$$\frac{1}{K} = \frac{T - c}{T}$$

即：拆分点之后的 `token` 数量比上总 `token` 数量等于 $\frac{1}{K}$ 。 K 越大，则需要预测的 `token` 数量越少。

7.5 引入 Transformer XL

1. 考虑到 `XLNet` 是自回归语言框架，天然的匹配 `Transformer-XL` 思想，因此引入 `Transformer XL`。这也是 `XLNet` 的名字的由来。

具体而言，`XLNet` 引入了相对位置编码和 `segment-level` 递归机制。

2. XLNet 引入相对位置编码比较简单，它直接采用了 Transformer-XL 中的做法。
3. XLNet 引入 segment-level 递归机制比较复杂，因为 XLNet 中需要考虑位置的重排。

令一个很长的文本序列中前后相邻的两个 segment 为：

$$\tilde{\mathbf{w}} = \mathbf{s}_{1:T}, \quad \mathbf{w} = \mathbf{s}_{T+1:2T}$$

令位置编号 $[1, \dots, T]$ 重排后的结果为 $\tilde{\mathbf{z}}$ ， $[T+1, \dots, 2T]$ 重排后的结果为 \mathbf{z} 。

- 计算 content stream：
 - 首先基于 $\tilde{\mathbf{z}}$ 来处理前一个 segment，，并缓存每一层的隐向量 $\tilde{\mathbf{H}}^{(l)}$, $l = 1, 2, \dots, L$ 。
 - 然后基于 \mathbf{z} 来处理后一个 segment，并计算每一层的隐向量：
- $$\tilde{\mathbf{h}}_{z_t}^{(l)} \leftarrow \text{Attention}(Q = \tilde{\mathbf{h}}_{z_t}^{(l-1)}, KV = [\tilde{\mathbf{H}}^{(l-1)}, \tilde{\mathbf{h}}_{\mathbf{z} \leq t}^{(l-1)}]; \Theta)$$
- 其中 [] 表示沿着序列维度拼接。
 - 其中相对位置编码仅仅依赖于原始的序列位置，而不是重排后的顺序。因此 $\tilde{\mathbf{H}}^{(l-1)}$ 中各状态向量的位置对应于原始的排列顺序。
- 因此一旦获取了 $\tilde{\mathbf{H}}^{(l-1)}$ ，上述 Attention 的计算就独立于 $\tilde{\mathbf{z}}$ 。这使得缓存 $\tilde{\mathbf{H}}^{(l-1)}$ 时，不必关心前一个 segment 的混排顺序。
- 计算 query stream：和 content stream 处理方式相同。

7.6 多输入

1. 许多下游任务有多个输入部分，如 QA 任务的输入包含两部分：一个 question、以及包含 answer 的一段话。
类似 BERT，XLNET 将这两部分输入 A 和 B 拼接成一个整体：[A, SEP, B, SEP, CLS]。其中 SEP 作为这两部分的分隔符，CLS 作为整体的标记符。
在 segment-level 递归阶段，每个部分只使用对应上下文的状态缓存。
2. 与 BERT 采用 segment 绝对编码不同，XLNet 基于 Transformer-XL 的思想，采用相对 segment 编码 relative segment encoding:RES。

注：这里的 segment 含义和 segment-level 中的不同。这里指的是输入样本的不同部分，而后者指的是长文本序列拆分成多个输入片段。

给定序列的两个位置 i, j ，如果二者在同一个部分，则使用相同的 segment 编码 $\mathbf{s}_{i,j} = \mathbf{s}_+$ ；否则使用不同的 segment 编码 $\mathbf{s}_{i,j} = \mathbf{s}_-$ 。即：只考虑两个位置是否在同一个输入部分，而不考虑每个位置来自于哪个输入部分。

query 对于 segment 部分的 attention score 为：

$$\alpha_{i,j}^{seg} = (\vec{\mathbf{q}}_i + \vec{\mathbf{b}}) \cdot \vec{\mathbf{s}}_{i,j}$$

其中：

- $\vec{\mathbf{q}}_i$ 为 query 向量
- $\vec{\mathbf{b}}$ 是一个待学习的参数，表示偏置向量
- $\alpha_{i,j}^{seg}$ 是为标准的 attention score 的一部分

3. 相对 segment 编码的优点：
 - 泛化效果较好。
 - 支持预训练期间两部分输入、但微调期间多部分输入的场景。

绝对 segment 编码不支持这种场景，因为预训练期间它只对绝对位置 seg 1, seg 2 编码，而微调期间的 seg3, ... 编码是未知的。

7.7 模型比较

7.7.1 BERT vs XLNet

1. BERT 和 XLNet 都只是预测序列中的部分 token。
 - BERT 只能预测部分 token，因为如果预测序列中的所有 token，则它必须把所有 token 给 mask。此时的输入全都是 mask，无法学到任何东西。
 - XLNet 通过部分预测来降低计算代价，提高收敛速度。
2. BERT 和 XLNet 的部分预测都要求上下文足够长，这是为了降低优化难度，提高收敛速度。
3. 独立性假设使得 BERT 无法对 mask token 的关系建模。

对于句子 New York is a city，假设 BERT 和 XLNet 对 New York 建模：
 $\log p(\text{New York} \mid \text{is a city})$ 。

- 对于 BERT，其目标函数为：

$$L = \log p(\text{New} \mid \text{is a city}) + \log p(\text{York} \mid \text{is a city})$$

- 假设 XLNet 的混排顺序为：a is city New York。则对应的目标函数为：

$$L = \log p(X_1 = \text{New} \mid X_3 = \text{is}, X_4 = \text{a}, X_5 = \text{city}) + \log p(X_2 = \text{York} \mid X_1 = \text{New}, X_3 = \text{is}, X_4 = \text{a}, X_5 = \text{city})$$

因此 XLNet 能够捕捉到 (New, York) 之间的依赖性，而 BERT 无法捕捉。

虽然 BERT 能够捕捉到 (New, city)、(York, city) 这类依赖性，但是很显然 XLNet 也可以。

4. 给定一个序列 $w = (w_1, \dots, w_T)$ ，定义序列 w 的一组目标 token 的集合为 \mathbf{T} ，对应的非目标 token 的集合为 $\mathbf{N} = \mathbf{w} - \mathbf{T}$ 。BERT 和 XLNet 都是最大化 $\log p(\mathbf{T} \mid \mathbf{N})$ ：

- BERT 的目标函数：

$$L = \sum_{w \in \mathbf{T}} \log p(w \mid \mathbf{N})$$

- XLNet 的目标函数：

$$L = \sum_{w \in \mathbf{T}} \log p(w \mid \mathbf{N} \cup \mathbf{T}_{<w})$$

其中 $\mathbf{T}_{<w}$ 表示位置重排之后，排列在 w 位置之前的那些单词。

可以看到：对于 XLNet，目标单词 $w \in \mathbf{T}$ 不仅可以捕捉到非目标 token 集合 \mathbf{N} 中的 token 的相关性，还可以捕捉到目标 token 集合 \mathbf{T} 中的部分 token 的相关性。

7.7.2 AR VS XLNet

1. 传统的自回归语言模型只已知前文来预测后文，而无法根据后文来预测前文。这种局限性在实际任务中可能会遇到。

如：给一段话 Thom Yorke is the singer of Radiohead，给一个问题 who is the singer of Radiohead。

由于 Thom Yorke 出现在序列的前面，因此在传统自回归语言模型中它不依赖于 Radiohead。因此难以获得正确答案。

2. 由于 XLNet 引入了位置全排列，因此它可以对任意顺序建模，即不仅可以对 $p(\text{Radiohead} \mid \text{Thom Yorke})$ 建模，也可以对 $p(\text{Thom Yorke} \mid \text{Radiohead})$ 建模。

7.8 实验

7.8.1 数据集和超参数

1. `XLNet-Large` 与 `BERT-Large` 拥有相同的结构超参数，因此具有差不多的模型大小。
`XLNet-Base` 与 `BERT--Base` 拥有相同的结构超参数，因此也具有差不多的模型大小。
2. `XLNet-Large` 预训练语料库：与 `BERT` 一致，`XLNet` 使用 `BooksCorpus` 和英文维基百科作为预训练语料库，一共 `13 GB` 的纯文本。
除此之外，`XLNet` 还包括 `Giga5` (`16GB` 纯文本)、`Clueweb 2012-B`、`Common Crawl` 语料库来预训练。
 - `XLNet` 使用启发式过滤策略来去掉 `Clueweb 2012-B` 和 `Common Crawl` 中的太短或者低质量的文章，最终分别包含 `19GB` 纯文本和 `78 GB` 纯文本。
 - 在使用 `SentencePiece` 词干化之后，最终维基百科、`BooksCorpus, Giga5, Clueweb, Common Crawl` 分别获取了 `2.78B, 1.09B, 4.75B, 4.30B, 19.97B` 的 `subword pieces`，一共 `32.89B`。
3. `XLNet-Base` 预训练语料库：仅仅在 `BookCorpus` 和维基百科上预训练。
4. 训练参数：
 - 序列长度 `512`, `memory length` (用于 `segment-level` 递归的缓存状态向量) `384`。
 - `XLNet-Large` 在 `512` 块 `TPU v3` 芯片上训练 `50万步`，采用 `Adam optimizer` 和线性学习率衰减，`batch size` 为 `2048`，一共大约 `2.5天`。
实验观察表明：训练结束时模型仍然欠拟合，继续训练虽然能够降低预训练损失函数，但是无法改善下游任务的表现。这表明模型仍然没有足够的能力来充分利用大规模的数据。
但是在论文并没有预训练一个更大的模型，因为代价太大且对下游任务的微调效果可能提升有限。
 - `XLNet-Large` 中，部分预测的常数 `K=6`。
 - 由于引入 `segment-level` 递归机制，论文使用了双向数据输入 `pipeline`，其中前向和后向各占比处理大小的一半。
5. `span-based prediction`：
 - 首先采样一个长度 `B ∈ [1, 2, …, 5]`。
 - 然后随机选择一个连续的长度为 `B` 的区间的 `token` 作为预测目标
 - 最后选择包含该区间的 `K × B` 长度的一段 `token` 作为一个序列样本。

7.8.2 实验结果

1. `RACE` 数据集：包含了12到18岁中国初中、高中英语考试中的近10万个问题，答案由人工专家生成。
涉及到挑战性的推理问题，所以它是最困难的阅读理解数据集之一。另外，该数据集每篇文章平均超过300个段落，远远超过其它流行的阅读理解数据集(如 `SQuAD`)。因此该数据集对于长文本理解来说是个具有挑战性的 `baseline`。

`XLNet` 在该数据集上的微调结果如下图所示，其中：

- 微调阶段的序列长度为 `640`
- 所有 `BERT` 和 `XLNet` 都是在 `24层` 网络下实现，其模型大小几乎相同。
- `*` 表示使用了多个训练模型的集成（取均值）；`Middle` 和 `High` 表示初中和高中水平的难度。

`XLNet` 在该数据集上相比 `BERT` 提升巨大的原因可能有两个：

- XLNet 的自回归语言模型解决了 BERT 的两个缺点：预测目标独立性假设、预训练阶段和微调阶段不一致。
- XLNet 引入 Transformer-XL 提升了长文本建模能力。

RACE	Accuracy	Middle	High
GPT [25]	59.0	62.9	57.4
BERT [22]	72.0	76.6	70.1
BERT+OCN* [28]	73.5	78.4	71.5
BERT+DCMN* [39]	74.1	79.5	71.8
XLNet	81.75	85.45	80.21

2. SQuAD 数据集：一个大型阅读理解数据集，包含 SQuAD 1.1 和 SQuAD 2.0。其中 SQuAD 1.1 中的问题总是有答案，而 SQuAD 2.0 中的问题可能没有答案。

- 为了微调 SQuAD 2.0，论文联合使用了两个损失函数：
 - 一个逻辑回归的损失函数，用于预测该问题是否有答案。这是一个文本分类问题。
 - 一个标准的 span extraction loss 用于预测答案的位置。
- XLNet 在该数据集上的微调结果如下图所示：
 - * 表示集成模型（多个模型的平均输出）
 - 前两行直接比较 BERT 和 XLNET，没有采用数据集增强，直接在验证集上的比较。
 - 验证集采用 SQuAD 1.1。由于 SQuAD 1.1 始终有答案，因此评估时模型只考虑 span extraction loss 这一路的输出。
 - 后四行采用了数据集增强，通过 NewsQA 来增强数据。因为排行榜排名靠前的模型都采取了数据集增强。
 - 结果在排行榜的测试集上评估。

SQuAD1.1	EM	F1	SQuAD2.0	EM	F1
<i>Dev set results without data augmentation</i>					
BERT [10]	84.1	90.9	BERT† [10]	78.98	81.77
XLNet	88.95	94.52	XLNet	86.12	88.79
<i>Test set results on leaderboard, with data augmentation (as of June 19, 2019)</i>					
Human [27]	82.30	91.22	BERT+N-Gram+Self-Training [10]	85.15	87.72
ATB	86.94	92.64	SG-Net	85.23	87.93
BERT* [10]	87.43	93.16	BERT+DAE+AoA	85.88	88.62
XLNet	89.90	95.08	XLNet	86.35	89.13

3. 文本分类任务：XLNet 在 IMDB, Yelp-2, Yelp-5, DBpedia, AG, Amazon-2, Amazon-5 等数据集上的表现如下。

Model	IMDB	Yelp-2	Yelp-5	DBpedia	AG	Amazon-2	Amazon-5
CNN [14]	-	2.90	32.39	0.84	6.57	3.79	36.24
DPCNN [14]	-	2.64	30.58	0.88	6.87	3.32	34.81
Mixed VAT [30, 20]	4.32	-	-	0.70	4.95	-	-
ULMFiT [13]	4.6	2.16	29.98	0.80	5.01	-	-
BERT [35]	4.51	1.89	29.32	0.64	-	2.63	34.17
XLNet	3.79	1.55	27.80	0.62	4.49	2.40	32.26

4. GLUE 数据集：9 个 NLP 任务的集合。XLNet 的表现如下图所示。

- 前三行表示单任务单模型的表现：针对每个任务，使用该任务的数据微调一个模型。
 - 最后四行表示多任务集成模型的表现：
 - 多任务：先在多个任务的数据集上联合训练模型，并在其它任务的数据集上微调网络。
 - 其中 XLNet 采用 MNLI, SST-2, GNLI, QQP 这四个最大的数据集来联合训练。

- 集成模型：用不同的初始化条件训练多个模型，并用这些模型的平均输出作为最终输出。
- BERT 和 XLNet 采用 24 层网络，其模型大小相差无几。

Model	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	WNLI
<i>Single-task single models on dev</i>									
BERT [2]	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-
XLNet	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-
<i>Single-task single models on test</i>									
BERT [10]	86.7/85.9	91.1	89.3	70.1	94.9	89.3	60.5	87.6	65.1
<i>Multi-task ensembles on test (from leaderboard as of June 19, 2019)</i>									
Snorkel* [29]	87.6/87.2	93.9	89.9	80.9	96.2	91.5	63.8	90.1	65.1
ALICE*	88.2/87.9	95.7	90.7	83.5	95.2	92.6	68.6	91.1	80.8
MT-DNN* [18]	87.9/87.4	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0
XLNet*	90.2/89.7[†]	98.6[†]	90.3 [†]	86.3	96.8[†]	93.0	67.8	91.6	90.4

5. ClueWeb09-B 数据集：包含了 TREC 2009–2012 Web Tracks，一共5千万 doc 及其 query，用于评估的文档排序 doc ranking 能力。该任务是用标准检索方法对检索到的前 100 份文档进行 rerank。

由于文档排序主要涉及到文档的低层表示而不是高层语义，因此该数据集用于评估词向量质量。

论文使用预训练的 XLNet 来提取 doc 和 query 的词向量，无需微调阶段，结果如下图所示。

结果表明：XLNet 比 BERT 学到了更好的低级词向量。

Model	NDCG@20	ERR@20
DRMM [12]	24.3	13.8
KNRM [8]	26.9	14.9
Conv [8]	28.7	18.1
BERT [†]	30.53	18.67
XLNet	31.10	20.28

6. 消融实验主要验证了以下几个目的：

- 相比较于 BERT 的自编码语言模型，XLNet 的 Permutation Language Model 是否有效。
- 是否有必要引入 Transformer-XL 及其 segment-level 递归。
- 一些实现细节：span-based prediction，双向输入 pipeline，next-sentence-prediction 的必要性。

所有模型都在维基百科和 BooksCorpus 数据集上预训练，模型采用 12 层 Base 模型。实验结果如下图所示。结果表明：

- 从第一/二行的比较来看：Transformer-XL 在长文本上表现优异。
- 从第四/五行的比较来看：如果没有内存缓存机制则性能下降，对于长文本任务尤为明显。
- 从第四/六/七行的比较来看：如果放弃 span-based prediction 和双向输入 pipeline，则效果也会下降。
- 论文意外发现：NSP 任务不一定能改善性能；相反，除了 RACE 任务之外它会损害性能。

#	Model	RACE	SQuAD2.0	MNLI	SST-2
		F1	EM	m/mm	
1	BERT-Base	64.3	76.30	73.66	84.34/84.65
2	DAE + Transformer-XL	65.03	79.56	76.80	84.88/84.45
3	XLNet-Base ($K = 7$)	66.05	81.33	78.46	85.84/85.43
4	XLNet-Base ($K = 6$)	66.66	80.98	78.18	85.63/85.12
5	- memory	65.55	80.15	77.27	85.32/85.05
6	- span-based pred	65.95	80.61	77.91	85.49/85.02
7	- bidirectional data	66.34	80.65	77.87	85.31/84.99
8	+ next-sent pred	66.76	79.83	76.94	85.32/85.09

八、MT-DNN

- 神经网络的监督学习需要大量的标记数据，通常这些标记数据难以获取。为了解决这个困难，有两种策略：
 - 预训练学习 **Pretrain Learning**：通过从大量未标记的数据中学习 **representation**，从而缓解目标任务的标记数据太少的问题。
 - 多任务学习 **Multi-Task Learning**：通过从多个任务及其标记数据中学习 **representation**，从而缓解目标任务的标记数据太少的问题。

另外，多任务学习可以通过针对这些任务的训练而缓解模型对具体某个任务的过拟合，从而产生了一种正则化效果。这使得模型学到的 **representation** 通用性更强，可以跨任务使用。

预训练学习和多任务学习二者可以互补，二者结合可以进一步改善文本的 **representation**。

Multi-Task Deep Neural Network:MT-DNN 就是结合了 **BERT** 预训练和多任务学习这两种策略来学习更泛化的 **representation**。
- MT-DNN** 采用了 **BERT** 的基本结构，底层网络在所有任务之间共享，顶层结构根据不同任务而不同。
 - 与 **BERT** 相似，**MT-DNN** 可以通过微调来匹配特定的任务。
 - 与 **BERT** 不同，**MT-DNN** 除了预训练模型之外，还通过多任务学习来学习 **representation**。
- MT-DNN** 结合了四种不同类型的 **Nature Language Understanding:NLU** 任务，包括：单句分类、成对文本分类、文本相似性和相关性排序。
 - 单句分类：给定一个简单句子，模型预测它属于哪个类别。如：
 - CoLA** 任务：模型预测英文句子是否语法上合理。
 - SST-2** 任务：模型预测电影评论中提取的句子的情感是积极的还是消极的。
 - 成对文本分类：给定一对句子，模型预测它们的关系。如：
 - RTE** 和 **MNLI** 语言推理任务：模型预测一个句子相对于另一个句子是蕴含关系 **entailment**、矛盾关系 **contradiction** 还是中性关系 **neutral**。
 - QQP** 和 **MRPC** 任务：模型预测一对句子在语义上是否等价。
 - 文本相似性打分：给定一对句子，模型预测它们的相似性得分。这是一个回归任务。如：**STS-B** 任务。
 - 相关性排序：给定一个 **query** 和一组候选答案，模型根据相关性对候选答案排序。如：
 - QNLI** 任务：模型评估一个句子是否包含给定问题的正确答案。

虽然它在 **GLUE** 中被视为二分类任务，但是在这里我们将其视作成对的排序任务。实验结果表明：这种方式比二分类的准确性有显著提高。

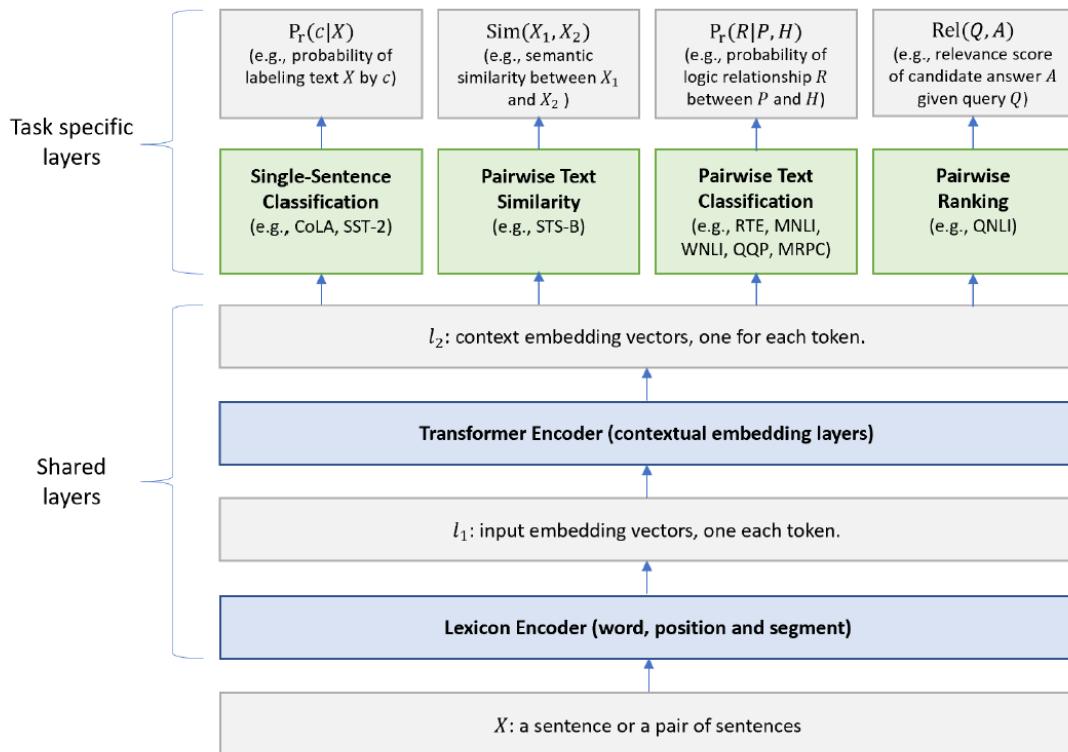
8.1 模型

8.1.1 模型结构

1. MT-DNN 模型的结构如下图所示。

- 输入 X ：一个 token 序列。模型首先学习该序列的 embedding 表示得到 l_1 ；然后通过 Transformer encoder 提取特征得到 l_2 。
- Lexicon Encoder：学习一个 token 序列的 embedding 表示，每个 token 学到一个 embedding 向量。
 - 每个 embedding 向量由 word embedding、segment embedding、position embedding 三部分组成。
 - token 序列的第一个 token 始终是 [CLS] 标记。
 - 如果 X 是由一对句子组成，则通过 [SEP] 标记分隔。如：

$$\{[CLS], X_1^1, \dots, X_{m_1}^1, [SEP], X_1^2, \dots, X_{m_2}^2\}$$
 其中 X_i^1 为第一个句子第 i 个 token， X_i^2 为第二个句子第 i 个 token。
- Transformer Encoder：通过多层双向 Transformer Encoder 来抽取特征。
- 网络高层：包含单句分类、成对文本相似性、成对文本分类、相关性排序等具体任务相关的网络结构。



2. 与 BERT 不同，MT-DNN 学习 representation 不仅通过预训练来学习，也通过多任务训练来学习。

8.1.2 task-specific 结构

1. 单句分类任务的网络结构：假设 \bar{x} 是 l_2 中对应于 [CLS] 的向量，它被视为输入 X 的表达。输入 X 为类别 c 的概率为：

$$P_r(c | X) = \text{softmax}(\mathbf{W}_{task}^T \cdot \bar{x})$$

其中 \mathbf{W}_{task} 为该任务对应的参数。

2. 成对文本相似性任务的网络结构：假设 \vec{x} 是 l_2 中对应于 [CLS] 的向量，它可以视作句子对 x_1, x_2 的表达。则该文本对的相似性为：

$$\text{Sim}(X_1, X_2) = \vec{w}_{task} \cdot \vec{x}$$

其中 \vec{w}_{task} 为该任务对应的参数。

3. 成对文本分类任务的网络结构：

在自然语言推理 NLI 任务中，前提 premise 由 m 个 token 组成 $P = (p_1, p_2, \dots, p_m)$ ，猜想 hypothesis 由 n 个 token 组成 $H = (h_1, h_2, \dots, h_n)$ 。任务目标是预测 premise 和 hypothesis 之间的关系。

论文采用随机回答网络 stochastic answer network:SAN 的实现方式：

- 首先将 premise 中的 token 的表示向量（从 l_2 中获取）拼接成矩阵 $\mathbf{M}^p \in \mathbb{R}^{m \times d}$ ，以及将 hypothesis 的表示向量（从 l_2 中获取）拼接成矩阵 $\mathbf{M}^h \in \mathbb{R}^{n \times d}$ 。

$$\mathbf{M}^p = \begin{bmatrix} \vec{\mathbf{h}}_1^p \\ \vdots \\ \vec{\mathbf{h}}_m^p \end{bmatrix}, \quad \mathbf{M}^h = \begin{bmatrix} \vec{\mathbf{h}}_1^h \\ \vdots \\ \vec{\mathbf{h}}_n^h \end{bmatrix}$$

其中 $\vec{\mathbf{h}}_i^p$ 表示 premise 中的 token p_i 的 representation， $\vec{\mathbf{h}}_i^h$ 表示 hypothesis 中的 token h_i 的 representation。

- 初始化 d 维状态向量 \vec{s} ：

$$\vec{s}^{(0)} = \sum_{j=1}^n \alpha_j \vec{\mathbf{h}}_j^h$$

$$\alpha_j = \frac{\exp(\vec{w}_1 \cdot \vec{\mathbf{h}}_j^h)}{\sum_{i=1}^n \exp(\vec{w}_1 \cdot \vec{\mathbf{h}}_i^h)}$$

其中 d 维向量 \vec{w}_1 为待学习的参数。

- 迭代 K 次， K 为超参数。迭代过程 $k = 1, 2, \dots, K$ ：

- 根据状态 $\vec{s}^{(k-1)}$ 和 \mathbf{M}^p 计算 $\vec{x}^{(k)}$ ：

$$\vec{x}^{(k)} = \sum_{j=1}^m \beta_j \vec{\mathbf{h}}_j^p$$

$$\beta_j = \text{softmax}(\mathbf{M}^p \mathbf{W}_2 \vec{s}^{(k-1)})$$

其中 $\mathbf{W}_2 \in \mathbb{R}^{d \times d}$ 为待学习的参数。

- 计算状态 $\vec{s}^{(k)}$ ：

$$\vec{s}^{(k)} = \text{GRU}(\vec{s}^{(k-1)}, \vec{x}^{(k)})$$

- 在每个迭代步，输出类别概率：

$$P_r^{(k)} = \text{softmax}(\mathbf{W}_3 [\vec{s}^{(k)}; \vec{x}^{(k)}; |\vec{s}^{(k)} - \vec{x}^{(k)}|; \vec{s}^{(k)} \cdot \vec{x}^{(k)}])$$

其中 $[;]$ 表示向量的拼接。

- 使用 K 步的平均概率输出作为预测概率：

$$P_r = \frac{\sum_{k=0}^{K-1} P_r^{(k)}}{K}$$

4. 相关性排序任务的网络结构：假设 \vec{x} 是 l_2 中对应于 [CLS] 的向量，它可以视作问答对 Q, [SEP], A 的表达。则相关性得分为：

$$score = g(\vec{w} \cdot \vec{x})$$

给定一个 `question`, 对它的所有候选 `answer` 计算相关性得分, 并按照得分排序即可得到排序结果。

8.1.3 模型训练

1. `MT-DNN` 的训练过程分为两步: 预训练和多任务训练。

- 预训练步骤参考 `BERT`: `lexicon encoder` 和 `Transformer encoder` 的参数通过两个无监督学习任务 `MLM` 和 `NSP` 来训练。
- 多任务训练步骤: 使用 `mini-batch SGD` 优化算法来训练。

在每个 `epoch` 选择 b_t 大小的 `mini-batch`, 然后根据任务 t 相关的目标函数来更新参数。这种方式等价于优化所有任务的目标函数。

- 对于分类任务, 使用交叉熵的目标函数:

$$L = - \sum_c \mathbb{I}(X, c) \log(P_r(c | X))$$

其中 $\mathbb{I}(X, c)$ 是一个示性函数, 如果样本 X 属于类别 c 则函数值为 1; 否则函数值为 0

◦

- 对于文本相似性任务, 使用平方误差损失函数:

$$L = (y - \text{Sim}(X_1, X_2))^2$$

- 对于相关性排序任务, 给定问题 `question` Q , 假设候选回答集合 \mathbb{A} 包含一个正确答案 A^+ 和其它 $|\mathbb{A}| - 1$ 个错误答案, 则优化目标为:

$$L = - \sum_{(Q, A^+)} P_r(A^+ | Q)$$

$$P_r(A^+ | Q) = \frac{\exp(\gamma \times \text{score}(Q, A^+))}{\sum_{A' \in \mathbb{A}} \exp(\gamma \times \text{score}(Q, A'))}$$

其中:

- `score` 为相关性得分: $\text{score}(Q, A) = g(\vec{w} \cdot \vec{x})$, \vec{w} 为待学习的参数, \vec{x} 为 `Q [SEP] A` 的表达。
- γ 为一个超参数。

2. `MT-DNN` 训练算法:

- 随机初始化模型参数 Θ
- 预训练共享层 (如 `lexicon encoder` 和 `Transformer encoder`)
- 设定最大 `epoch` 数 $epoch_{\max}$
- 准备 T 个任务的数据: 对每个任务的数据集 $t = 1, 2, \dots, T$, 将其拆分成 `mini-batch` 的形式 \mathbb{D}_t
- `epoch` 迭代: $epc = 1, 2, \dots, epoch_{\max}$:
 - 合并所有数据集: $\mathbb{D} = \mathbb{D}_1 \cup \mathbb{D}_2 \dots \cup \mathbb{D}_T$
 - 随机混洗数据集 \mathbb{D}
 - 对 \mathbb{D} 中每个 `mini-batch` b_t (代表了某个任务 t 的 `mini-batch`) :
 - 根据具体任务的损失函数计算 $L(\Theta)$
 - 计算梯度: $\nabla_{\Theta} L$
 - 更新模型: $\Theta = \Theta - \epsilon \nabla_{\Theta} L$

8.2 实验

8.2.1 数据集和参数

1. 数据集：

- General Language Understanding Evaluation : GLUE 数据集：是 9 项自然语言理解任务的集合，包括问题回答、情感分析、文本相似性、文本蕴含等任务。其目的是为了评估模型的泛化能力和鲁棒性。
- Stanford Natural Language Inference:SLI 数据集：包含 57 万人工标注的句子对，每对句子包含一个前提 premise 和假设 hypotheses。

前提 premise 是从 Flickr30 语料库提取，而假设 hypotheses 是人工标注的。该数据集是自然语言推理领域使用最广泛的数据集。

- SciTail 数据集：从 Science Question Answering:SciQ 数据集中抽取的，评估给定前提 premise 是否包含给定假设 hypotheses。

与前面提到的其它蕴含 entailment 数据集不同，SciTail 的假设是从科学问题中创建的，而相应的答案候选和前提是大型预料库中检索的相关网络句子。前提和假设的词汇相似性非常高，因此该任务很有挑战。

Corpus	Task	#Train	#Dev	#Test	#Label	Metrics
Single-Sentence Classification (GLUE)						
CoLA	Acceptability	8.5k	1k	1k	2	Matthews corr
SST-2	Sentiment	67k	872	1.8k	2	Accuracy
Pairwise Text Classification (GLUE)						
MNLI	NLI	393k	20k	20k	3	Accuracy
RTE	NLI	2.5k	276	3k	2	Accuracy
WNLI	NLI	634	71	146	2	Accuracy
QQP	Paraphrase	364k	40k	391k	2	Accuracy/F1
MRPC	Paraphrase	3.7k	408	1.7k	2	Accuracy/F1
Text Similarity (GLUE)						
STS-B	Similarity	7k	1.5k	1.4k	1	Pearson/Spearman corr
Relevance Ranking (GLUE)						
QNLI	QA/NLI	108k	5.7k	5.7k	2	Accuracy
Pairwise Text Classification						
SNLI	NLI	549k	9.8k	9.8k	3	Accuracy
SciTail	NLI	23.5k	1.3k	2.1k	2	Accuracy

2. 预处理：所有训练文本都使用 wordpieces 词干化，同时每个序列的长度不超过 512 个 token

。

3. 参数：

- 采用 Adamax 优化器，学习率 $5e-5$ 。同时使用线性学习率衰减，以及参数为 0.1 的 warm-up 机制。
- 使用参数为 0.1 的 dropout，但是对于 MNLI 参数为 0.3，对 CoLA 参数为 0.05。
- 最大 epoch 为 5，batch size 为 32。
- 为了防止梯度爆炸，执行梯度裁剪策略，使得梯度范数不超过 1。

8.2.2 实验结果

1. MT-DNN 在 GLUE 任务上的结果如下图所示，所有结果来自 2019 年 2 月 25 号的 leaderboard：

<https://gluebenchmark.com/leaderboard>。

- 每个数据集下面的数字给出了训练集的样本数。
- 粗体给出了最好的结果 SOA；蓝色粗体给出了超越人类的结果。
- 模型说明：
 - BERT_{LARGE} 模型：由 BERT 作者提供的模型。本论文在每个 GLUE 任务上使用对应的任务标记数据来微调该模型。

- MT-DNN：使用 BERT_{LARG} 来初始化共享层，然后基于所有的 GLUE 任务使用多任务学习来继续训练模型，最后基于每个 GLUE 任务的标记数据来微调模型。
- MT-DNN no fine tune：使用 BERT_{LARG} 来初始化共享层，然后基于所有的 GLUE 任务使用多任务学习来继续训练模型，但是最后并没有使用基于每个 GLUE 任务的标记数据来微调模型。

该任务是为了评估：既然 MT-DNN 在多任务学习阶段已经使用了对应任务的标记数据，那么继续微调模型有没有必要？

- 实验结论：

- 由于 MT-DNN 使用 BERT LARGE 来初始化模型，因此它超越 BERT LARGE 的原因就归因于多任务学习来继续优化共享层的参数。
- 多任务学习对于非常小规模的 in-domain 训练数据集很有效。实验结果表明：训练数据集越小，MT-DNN 超越 BERT 得越多。
- 除了在 CoLA 任务上，在其它任务上 MT-DNN no fine tune 仍然战胜了 BERT LARGE。

具体原因是 CoLA 任务非常具有挑战性，因为它的 in-domain 数据集非常小，且其任务定义和数据集内容与 GLUE 其它任务都不相同。这就使得它很难从其它任务中获取可迁移的知识。

因此，多任务学习对于该任务表现欠拟合，所以微调阶段对于该任务的性能提升是非常有必要的。

Model	CoLA 8.5k	SST-2 67k	MRPC 3.7k	STS-B 7k	QQP 364k	MNLI-m/mm 393k	QNLI 108k	RTE 2.5k	WNLI 634	AX	Score
BiLSTM+ELMo+Attn ¹	36.0	90.4	84.9/77.9	75.1/73.3	64.8/84.7	76.4/76.1	-	56.8	65.1	26.5	70.5
Singletask Pretrain Transformer ²	45.4	91.3	82.3/75.7	82.0/80.0	70.3/88.5	82.1/81.4	-	56.0	53.4	29.8	72.8
GPT on STILTs ³	47.2	93.1	87.7/83.7	85.3/84.8	70.1/88.1	80.8/80.6	-	69.1	65.1	29.4	76.9
BERT _{LARGE} ⁴	60.5	94.9	89.3/85.4	87.6/86.5	72.1/89.3	86.7/85.9	92.7	70.1	65.1	39.6	80.5
MT-DNN _{no-fine-tune}	58.9	94.6	90.1/86.4	89.5/88.8	72.7/89.6	86.5/85.8	93.1	79.1	65.1	39.4	81.7
MT-DNN	62.5	95.6	91.1/88.2	89.5/88.8	72.7/89.6	86.7/86.0	93.1	81.4	65.1	40.3	82.7
Human Performance	66.4	97.8	86.3/80.8	92.7/92.6	59.5/80.4	92.0/92.8	91.2	93.6	95.9	-	87.1

2. MT-DNN 的优势不仅在于其泛化能力强，还在于其构建模型的灵活性：允许我们整合在具体任务研究的一些先进模型和方法，有效地利用了已有的研究成果。

两个典型的例子是使用 SAN 问答模型来处理成对文本分类，使用 pairwise ranking 损失函数来处理 QNLI 任务。有鉴于此，论文提出了 ST-DNN 变种模型。

ST-DNN 表示 single-task DNN，它使用 MT-DNN 相同的模型结构，但是它的共享层是预训练 BERT 而没有多任务学习。

在预训练 ST-DNN 之后，使用 GLUE 每个具体任务的数据来微调 ST-DNN。此时 ST-DNN 与 BERT 的唯一区别在于模型的顶层输出不同：ST-DNN 是具体于每个任务的输出网络。

ST-DNN 的实验结果如下图所示。实验结论：

- 在 MNLI, QQP 等任务上，ST-DNN 战胜了 BERT，这表明 SAN 问答模型的泛化能力更强。
- 在 QNLI 任务上，ST-DNN 使用 pairwise ranking loss，而 BERT 使用交叉熵。结果显示 ST-DNN 仍然战胜了 BERT。

Model	MNLI-m/mm	QQP	RTE	QNLI (v1/v2)	MRPC	CoLa	SST-2	STS-B
BERT _{LARGE}	86.3/86.2	91.1/88.0	71.1	90.5/92.4	89.5/85.8	61.8	93.5	89.6/89.3
ST-DNN	86.6/86.3	91.3/88.4	72.0	96.1/-	89.7/86.4	-	-	-
MT-DNN	87.1/86.7	91.9/89.2	83.4	97.4/92.9	91.0/87.5	63.5	94.3	90.7/90.6

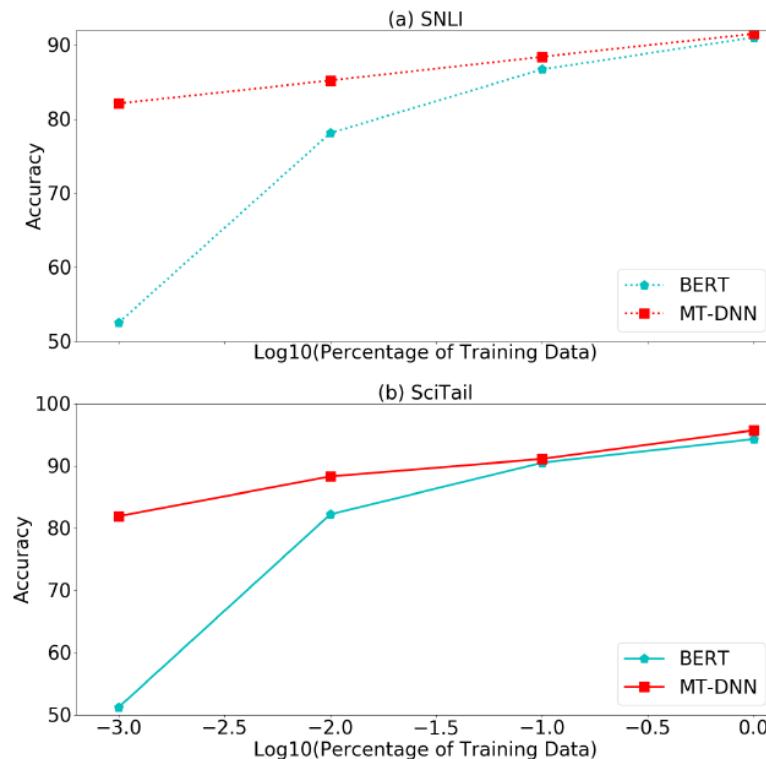
3. 构建真实应用系统最重要的标准之一是：快速适应新的任务和领域。因为新的领域和任务中获取标记数据非常昂贵，所以通常在新领域和任务中我们只有非常小的训练数据集，甚至没有训练数据。

根据该评价准则，论文在 SNLI 和 SciTail 上执行领域自适应实验从而评估 BERT 和 MT-DNN 快速适应新任务和领域的能力。对每个任务 (SNLI/SciTail) 构建一个任务具体的模型，然后通过该任务的训练集来训练模型、验证集来评估模型。

BERT 和 MT-DNN 都是基于 BERT BASE 架构，所有结果都是随机初始化5次并对这些模型的预测结果取平均。

- 采用这些任务的默认训练集、验证集、测试集，但是对训练集随机采样 0.1%、1%、10%、100%，从而验证小训练集的效果。实验结果如下图所示。

结果表明：训练数据越少，MT-DNN 战胜 BERT 得越多。因此 MT-DNN 相比 BERT，其领域适配 domain adaption 的能力越强。



Model	0.1%	1%	10%	100%
SNLI Dataset (Dev Accuracy%)				
#Training Data	549	5,493	54,936	549,367
BERT	52.5	78.1	86.7	91.0
MT-DNN	82.1	85.2	88.4	91.5
SciTail Dataset (Dev Accuracy%)				
#Training Data	23	235	2,359	23,596
BERT	51.2	82.2	90.5	94.3
MT-DNN	81.9	88.3	91.1	95.7

- 将 MT-DNN 在 SNLI 和 SciTail 的所有训练数据上训练，结果和之前的 SOA 模型比较，比较结果如下（早期的 SOA 用 * 表明）。

结果表明：MT-DNN Large 取得了最新的 SOA。

Model	Dev	Test
SNLI Dataset (Accuracy%)		
GPT (Radford et al., 2018)	-	89.9
Kim et al. (2018)*	-	90.1
BERT _{BASE}	91.0	90.8
MT-DNN _{BASE}	91.5	91.1
BERT _{LARGE}	91.7	91.0
MT-DNN _{LARGE}	92.2	91.6
SciTail Dataset (Accuracy%)		
GPT (Radford et al., 2018)*	-	88.3
BERT _{BASE}	94.3	92.0
MT-DNN _{BASE}	95.7	94.1
BERT _{LARGE}	95.7	94.4
MT-DNN _{LARGE}	96.3	95.0

九、BERT 扩展

9.1 BERT-wwm-ext

1. 原始版本的 BERT 采用了 wordPiece tokenize 来预处理，即把每个单词拆解一些 wordpiece token。

- wordpiece token 是由 wordpiece model:WMP 生成的，最初是为了解决谷歌语音识别系统遇到的日语/韩语分割问题。该模型是数据驱动，并且确保任何可能的字符序列的确定性分割。

如：文本 Jet makers feud over seat width with big orders at stake 被分割成的 wordpiece token 为：

```
1 | _J et _makers _fe ud _over _seat _width _with _big _orders _at
   | _stake
```

其中_ 表示单词的开始标记。

- 这种预处理为 BERT 带来一个严重的问题：有可能仅仅对一个单词的某个部分 wordpiece token 执行了 mask。此时 MLM 模型预测的只是单词的一部分，相当于模型对一个错误的目标进行预测。这非常不利于模型的学习。

有鉴于此，谷歌后续发布了 BERT 的 whole word Masking:wwm 版本 BERT-wwm。在该版本中，一个单词要么没有被 mask、要么该单词所有的 wordpiece token 都被 mask。

事实上，百度也提出了类似思想的 ERNIE 模型。

2. BERT-wwm-ext 是 BERT-wwm 的中文版本，该模型对中文的整个汉语单词而不是单个汉字进行 mask，在最新的中文维基百科上训练。

- 仅仅利用中文维基百科训练，没有利用任何额外的数据，也未执行繁体-简体转换。因为该模型是为简体中文、繁体中文提供一个通用的 NLP 模型。
- 其目标是：在不改变现有 BERT 的网络结构、超参数的情况下，为中文 BERT 提供简单的、可扩展性的方法从而提高模型的能力。
- 论文的主要贡献：
 - 预训练好了一个中文 BERT-wwm 模型供人们使用。
 - 在 BERT/BERT-wwm/ERNIE 上做了一些实验，验证该方法的有效性。
 - 给出了中文 NLP 预训练的一些技巧。

3. 数据预处理：

- 下载最新的 wikipedia dump 并用 `wikiExtractor.py` 预处理，最终解压成 1307 个文件。其中包含简体中文、繁体中文，无需执行繁体到简体的转换。
- 清除 `html` 标记并拆分文档，最终得到 1360 万行文本。
- 通过哈工大的 LTP 分词工具来分词
- 使用 `create_pretraining_data.py` 将文本转换成预训练样本，从而作为 `BERT` 的输入。
一共生成两组训练集，其中一组样本的最大长度为 128 个 `token`，另一组样本的最大长度为 512 个 `token`。
- 采用 `BERT-WWM` 相同的网络结构和超参数（如 `mask` 比例）。

预处理的例子如下所示：

[Original Sentence]
使用语言模型来预测下一个词的 probability。
[Original Sentence with CWS]
使用 语 模 型 来 预 测 下 一 个 词 的 prob ab ility。
[Original BERT Input]
使 用 语 言 [M A S K] 型 来 [M A S K] 测 下 一 个 词 的 pro [M A S K]##l i t y
[Whold Word Masking Input]
使 用 语 言 [M A S K][M A S K] 来 [M A S K][M A S K][M A S K]

4. 可以将 `BERT-wmm-ext` 视为 `BERT` 的一个添加了单词边界的补丁版，而不是一个新的模型。
 - 预训练阶段：直接参考 `BERT`。
 - 微调阶段：直接使用 `BERT-wmm-ext` 替代原来的中文 `BERT` 预训练模型，甚至不需要修改配置文件。
5. 预训练参数：
 - 使用 `LAMB` 优化器替代 `BERT` 的 `Adam` 权重衰减优化器。
 - 首先在最大长度 128 的样本上训练，`batch-size = 2560`，初始化学习率 `1e-4`，`warm-up` 比例 10%，训练 10 万步。
然后在最大长度 512 的样本上训练，`batch-size = 384`，初始化学习率 `1e-4`，`warm-up` 比例 10%，也训练 10 万步。

预训练语料及其参数如下表所示：

	BERT	BERT-wwm	ERNIE
Pre-Train Data	Wikipedia	Wikipedia	Wikipedia +Baike+Tieba, etc.
Sentence #	24M		173M
Vocabulary #	21,128		18,000 (17,964)
Hidden Activation	GeLU		ReLU
Hidden Size/Layers		768 & 12	
Attention Head #		12	

6. 中文微调任务：涵盖了从 `sentence-level` 到 `doc-level`、各种文本长度的任务。
 - 机器阅读理解 `Machine Reading Comprehension:MRC`：从给定文本中提取出连续的片段来回答问题。
数据集：
 - CMRC 2018：由中国信息处理学会、IFLYTEK 和哈工大发布的机器阅读理解数据集。
 - DRCD：由 Delta 研究所发布的繁体中文机器阅读理解数据集，通过工具将它转换为简体中文。
 - CJRC：来自中国法律判决文件的数据集，类似 CoQA。其中包括 `yes/no` 二元答案问题、`span-extraction` 问题、以及没有答案的问题。

- 命名实体识别 `Named Entity Recognition: NER`：识别出文本中的命名实体。其中命名实体的标签包括：`O, B-PER, I-PER, B-ORG, I-ORG, B-LOC, I-LOC`。

使用 `seqeval16` 来评估模型在 `Precision, Recall, F-score` 方面的得分。

数据集：

- `People Daily`：人民日报网站收集的新闻数据，包含官方正式新闻。
- `MSRA-NER`：微软亚洲研究院发布的命名实体识别数据集。

- 自然语言推理 `Natural Language Inference:NLI`：判定两个给定句子之间的蕴含关系。

数据集：`XNLI`。

- 情感分析 `Sentiment Analysis:SA`：判定给定的句子情感是积极的还是消极的。

数据集：

- `ChnSentiCorp` 数据集。该数据集包含酒店、书籍和 `PC` 等多个领域的评论。
- `Sina Weibo` 数据集：包含12万个带有正面、负面情感标签的新浪微博短文本。

- 句子语义匹配 `Sentence Pair Matching:SPM`：判断给定两个句子是否语义相似。

数据集：

- `LCQMC`：哈工大 2018年发布的语义相似数据集
- `BQ Corpus`：由哈工大和 `WeBank` 在 2018年 `EMNLP` 上发布的语义相似数据集

- 文档分类 `Document Classification:DC`：判断一篇文章属于哪个类别。

数据集：`THUCNews`：包含不同主题（金融、体育、科技...）的新浪新闻，是 `THUCLC` 的一部分。

为公平起见，对同一个任务的不同模型 `BERT/BERT-wwm-ext/ERNIE` 采用相同超参数（如下图所示），其中 \dagger 表示该数据集在 `BERT` 中也被评估， \ddagger 表示该数据集在 `ERNIE` 中也被评估。

Dataset	Task	MaxLen	Batch	Epoch	Train #	Dev #	Test #	Domain
CMRC 2018	MRC	512	64	2	10K	3.2K	4.9K	Wikipedia
DRCD	MRC	512	64	2	27K	3.5K	3.5K	Wikipedia
CJRC	MRC	512	64	2	10K	3.2K	3.2K	law
People Daily	NER	256	64	3	51K	4.6K	-	news
MSRA-NER \dagger	NER	256	64	5	45K	-	3.4K	news
XNLI $\dagger\ddagger$	NLI	128	64	2	392K	2.5K	2.5K	various
ChnSentiCorp \dagger	SC	256	64	3	9.6K	1.2K	1.2K	various
Sina Weibo	SC	128	64	3	100K	10K	10K	microblogs
LCQMC \ddagger	SPM	128	64	3	240K	8.8K	12.5K	Zhidao
BQ Corpus	SPM	128	64	3	100K	10K	10K	QA
THUCNews	DC	512	64	3	50K	5K	10K	news

为确保结果可靠性，每个任务进行十次相同的实验并给出最高分和平均分（括号里给出的得分）。其中最佳初始化学习率通过验证集来挑选。

- 机器阅读理解 `MRC` 结果：

- 最佳初始化学习率：

	BERT	BERT-wwm	ERNIE
1 CMRC 数据集：	3e-5	3e-5	8e-5
2 DRCD 数据集：	3e-5	3e-5	8e-5
3 CJRC 数据集：	4e-5	4e-5	8e-5

- `ERNIE` 在 `DRCD` 没有表现出竞争力，说明它不适合处理繁体中文文本。

检查 `ERNIE` 的词汇表之后发现繁体中文字符被删除了，因此导致它在繁体中文任务上性能较差。

- 在 CJRC 中, BERT-wwm 比 BERT 和 ERNIE 有所改善但是不是很显著, 有两个原因:
 - CJRC 中的文本是以专业的方式撰写。这说明专业领域的任务需要进一步的专业适配。
 - 在专业领域, 中文分词器的分词准确率也有所下降。这导致依赖于中文分词的 ERNIE/BERT-wwm 性能下降。

CMRC 2018	Dev		Test		Challenge	
	EM	F1	EM	F1	EM	F1
BERT	65.5 (64.4)	84.5 (84.0)	70.0 (68.7)	87.0 (86.3)	18.6 (17.0)	43.3 (41.3)
ERNIE	65.4 (64.3)	84.7 (84.2)	69.4 (68.2)	86.6 (86.1)	19.6 (17.0)	44.3 (42.8)
BERT-wwm	66.3 (65.0)	85.6 (84.7)	70.5 (69.1)	87.4 (86.7)	21.0 (19.3)	47.0 (43.9)

DRCD	Dev		Test	
	EM	F1	EM	F1
BERT	83.1 (82.7)	89.9 (89.6)	82.2 (81.6)	89.2 (88.8)
ERNIE	73.2 (73.0)	83.9 (83.8)	71.9 (71.4)	82.5 (82.3)
BERT-wwm	84.3 (83.4)	90.5 (90.2)	82.8 (81.8)	89.7 (89.0)

CJRC	Dev		Test	
	EM	F1	EM	F1
BERT	54.6 (54.0)	75.4 (74.5)	55.1 (54.1)	75.2 (74.3)
ERNIE	54.3 (53.9)	75.3 (74.6)	55.0 (53.9)	75.0 (73.9)
BERT-wwm	54.7 (54.0)	75.2 (74.8)	55.1 (54.1)	75.4 (74.4)

- 命名实体识别 NER 结果:

- 最佳初始化学习率:

1	BERT	BERT-wwm	ERNIE
2	人民日报 数据集:	3e-5	3e-5
3	MSR-NER 数据集:	3e-5	4e-5

- 平均而言 ERNIE 性能最好, 因为 ERNIE 采用了更多的训练数据。

NER	People Daily			MSRA-NER		
	P	R	F	P	R	F
BERT	95.3 (95.0)	95.1 (94.8)	95.2 (94.9)	95.4 (94.8)	95.3 (95.0)	95.3 (94.9)
ERNIE	95.8 (94.7)	95.6 (94.3)	95.7 (94.5)	95.3 (94.9)	95.7 (95.4)	95.4 (95.1)
BERT-wwm	95.4 (95.1)	95.3 (95.0)	95.3 (95.1)	95.4 (95.1)	95.6 (95.3)	95.4 (95.1)

- 自然语言推理 NLI 结果:

最佳初始化学习率:

1	BERT	BERT-wwm	ERNIE
2	XNLI 数据集:	3e-5	3e-5

XNLI	Dev		Test	
	BERT	ERNIE	BERT	ERNIE
BERT	77.8 (77.4)	77.8 (77.5)		
ERNIE	79.7 (79.4)	78.6 (78.2)		
BERT-wwm	79.0 (78.4)	78.2 (78.0)		

- 情感分析 SA 结果:

最佳初始化学习率:

1		BERT	BERT-wwm	ERNIE
2	ChnSentiCorp 数据集:	2e-5	2e-5	5e-5
3	新浪微博 数据集:	2e-5	3e-5	3e-5

Sentiment Classification	ChnSentiCorp		Sina Weibo (100k)	
	Dev	Test	Dev	Test
BERT	94.7 (94.3)	95.0 (94.7)	97.49 (97.38)	97.37 (97.32)
ERNIE	95.4 (94.8)	95.4 (95.3)	97.54 (97.41)	97.37 (97.29)
BERT-wwm	95.1 (94.5)	95.4 (95.0)	97.49 (97.40)	97.37 (97.35)

- 句子语义匹配 SPM 结果:

最佳初始化学习率:

1		BERT	BERT-wwm	ERNIE
2	LCQMC 数据集:	2e-5	2e-5	5e-5
3	BQ 数据集:	2e-5	3e-5	3e-5

Sentence Pair Matching	LCQMC		BQ Corpus	
	Dev	Test	Dev	Test
BERT	89.4 (88.4)	86.9 (86.4)	86.0 (85.5)	84.8 (84.6)
ERNIE	89.8 (89.6)	87.2 (87.0)	86.3 (85.5)	85.0 (84.6)
BERT-wwm	89.4 (89.2)	87.0 (86.8)	86.1 (85.6)	85.2 (84.9)

- 文档分类 DC 结果:

最佳初始化学习率:

1		BERT	BERT-wwm	ERNIE
2	LCQMC 数据集:	2e-5	2e-5	5e-5

THUCNews	Dev		Test	
	Dev	Test	Dev	Test
BERT	97.7 (97.4)	97.8 (97.6)		
ERNIE	97.6 (97.3)	97.5 (97.3)		
BERT-wwm	98.0 (97.6)	97.8 (97.6)		

7. 中文 NLP 任务经验技巧:

- 初始学习率是最重要的超参数。应该永远对其择优，从而获得能力更强的模型。
- 由于 BERT 和 BERT-wwm 在维基百科上训练，因此它们在正式文本上表现较好；而 ERNIE 在包括网络文本在内的更大数据的训练，因此对微博等非正式文本更有效果。
- 处理繁体中文时，选择 BERT 或者 bert-wwm。

9.2 RoBERTa

1. 预训练模型能够显著的提升任务效果，但是不同预训练模型的比较非常困难。

- 首先，每个预训练模型的训练成本很高，无法一一训练并进行比较。
- 其次，不同预训练模型通常是在不同规模大小的数据集上训练的，难以评估效果的好坏是预训练模型引起的还是预训练数据引起的。
- 最后，超参数的选择对预训练模型的表现影响很大。同一个预训练模型的不同超参数，其比较结果会有很大不同。

2. RoBERTa 主要工作是复现 BERT，然后对 BERT 的模型架构、训练目标、训练细节（如数据集大小、训练时间）的重要性进行探索，从而提出了改进方案，这个改进方案称为 RoBERTa。

主要修改：

- 更大的 batch size、更多的数据、更长的预训练时间
- 移除 NSP 任务
- 使用更长的输入序列
- 使用动态 mask

3. RoBERTa 采用 160 G 训练文本，远超 BERT 的 16G 文本，其中包括：

- BOOKCORPUS 和英文维基百科：原始 BERT 的训练集，大小 16GB。
- CC-NEWS：包含2016年9月到2019年2月爬取的6300万篇英文新闻，大小 76 GB（经过过滤之后）。
- OPENWEBTEXT：从 Reddit 上共享的 URL（至少3个点赞）中提取的网页内容，大小 38 GB。
- STORIES：CommonCrawl 数据集的一个子集，包含 Winograd 模式的故事风格，大小 31GB。

9.2.1 实验探索

1. 静态 mask VS 动态 mask：

- 原始静态 mask：在预处理的时候对整个数据集执行一次 mask，后续的每个训练步都采用相同的 mask。这是原始 BERT 的做法。
- 修改版静态 mask：在预处理的时候将数据集拷贝 10 次，每次拷贝采用不同的 mask。同时每个拷贝被训练 4 个 epoch。
这等价于原始的数据集采用 10 种静态 mask 来训练 40 个 epoch。
- 动态 mask：并没有在预处理的时候执行 mask，而是在每次向模型提供输入时动态生成 mask。

不同模式的实验效果如下表所示。其中 reference 为 BERT 用到的原始静态 mask，static 为修改版的静态 mask。

	Masking	SQuAD 2.0	MNLI-m	SST-2
reference	76.3	84.3	92.8	
<i>Our reimplementation:</i>				
static	78.3	84.3	92.5	
dynamic	78.7	84.0	92.9	

2. 使用 NSP VS 抛弃 NSP：

- SEGMENT-PAIR + NSP：输入包含两部分，每个部分是来自同一文档或者不同文档的 segment（segment 是连续的多个句子），这两个 segment 的 token 总数少于 512。
预训练包含 MLM 任务和 NSP 任务。
这是原始 BERT 的做法。
- SENTENCE-PAIR + NSP：输入也是包含两部分，每个部分是来自同一个文档或者不同文档的单个句子，这两个句子的 token 总数少于 512。预训练包含 MLM 任务和 NSP 任务。
- FULL-SENTENCES：输入只有一部分（而不是两部分），来自同一个文档或者不同文档的连续句子，token 总数不超过 512。如果跨文档，则在文档末尾添加文档边界 token。
预训练不包含 NSP 任务。
- DOC-SENTENCES：输入只有一部分（而不是两部分），来自同一个文档的连续句子，token 总数不超过 512。
预训练不包含 NSP 任务。

不同方式的效果如下图所示。结果表明：

- 使用 `SENTENCE-PAIR + NSP` 有损于下游任务。猜测的原因是：单个句子对于长期依赖性建模效果不好。
- 移除 `NSP` 任务相比原始 `BERT` 效果更好。猜测的原因是：原始 `BERT` 实现采用的是 `SEGMENT-PAIR` 输入，但是移除了 `NSP` 任务。
- 在移除 `NSP` 的两种策略中，`DOC-SENTENCES` 效果比 `FULL-SENTENCES` 更好。
但是 `DOC-SENTENCES` 策略中，位于文档末尾的样本可能小于 512 个 `token`。为了保证每个 `batch` 的 `token` 总数维持在一个较高水平，需要动态调整 `batch-size`。

Model	SQuAD 1.1/2.0	MNLI-m	SST-2	RACE
<i>Our reimplementation (with NSP loss):</i>				
SEGMENT-PAIR	90.4/78.7	84.0	92.9	64.2
SENTENCE-PAIR	88.7/76.2	82.9	92.1	63.0
<i>Our reimplementation (without NSP loss):</i>				
FULL-SENTENCES	90.4/79.1	84.7	92.5	64.8
DOC-SENTENCES	90.6/79.7	84.7	92.7	65.6
BERT _{BASE}	88.5/76.3	84.3	92.8	64.3
XLNet _{BASE} (K = 7)	-/81.3	85.8	92.7	66.1
XLNet _{BASE} (K = 6)	-/81.0	85.6	93.4	66.7

3. 更大的 `batch-size` :

在 `Neural Machine Translation:NMT` 任务研究中发现：当学习率适当增加、`batch-size` 非常大时，可以提高训练速度、提升模型能力。

经过实验观察发现，大的 `batch-size` 对于 `BERT` 也有类似帮助。

注：在相同的训练代价下，其中 `训练代价 = steps × batch_size`。

bsz	steps	lr	ppl	MNLI-m	SST-2
256	1M	1e-4	3.99	84.7	92.7
2K	125K	7e-4	3.68	85.2	92.9
8K	31K	1e-3	3.77	84.6	92.8

4. BPE 文本编码：Byte-Pair Encoding:BPE 是字符级和单词级表示的混合，它不是完整的单词而是 `sub-word` 单元。

`BPE` 通过训练语料库的统计分析得到的，其词表大小通常在 1 万到 10 万之间。

有两种 `BPE` 实现方式：

- 基于 `char-level`：原始 `BERT` 的方式，它通过对输入文本进行启发式的词干化之后处理得到。
- 基于 `bytes-level`：与 `char-level` 的区别在于 `bytes-level` 使用 `bytes` 而不是 `unicode` 字符作为 `sub-word` 的基本单位，因此可以编码任何输入文本而不会引入 `UNKNOWN` 标记。

当采用 `bytes-level` 的 `BPE` 之后，词表大小从 3 万（原始 `BERT` 的 `char-level`）增加到 5 万。这分别为 `BERT BASE` 和 `BERT LARGE` 增加了 1500 万和 2000 万额外的参数。

有研究表明，这样的做法会导致轻微的性能下降。但是 `SoBERTa` 的作者相信：这种统一编码的优势会超过性能的轻微下降。

5. 汇总这些改进点即可得到一个更稳定 `Rousty`、更好 `optimized` 的 `BERT` 实现 `approch`，称之为 `RoBERTa`。其中包括：动态掩码、抛弃 `NSP` 的 `FULL-SENTENCES`、更大的 `batch-size`、`bytes-level BPE`。

除此之外还包括一些细节，包括：更大的预训练数据、更多的训练步数。

Model	data	bsz	steps	SQuAD (v1.1/2.0)	MNLI-m	SST-2
RoBERTa						
with BOOKS + WIKI	16GB	8K	100K	93.6/87.3	89.0	95.3
+ additional data (§3.2)	160GB	8K	100K	94.0/87.7	89.3	95.6
+ pretrain longer	160GB	8K	300K	94.4/88.7	90.0	96.1
+ pretrain even longer	160GB	8K	500K	94.6/89.4	90.2	96.4
BERT _{LARGE}						
with BOOKS + WIKI	13GB	256	1M	90.9/81.8	86.6	93.7
XLNet _{LARGE}						
with BOOKS + WIKI	13GB	256	1M	94.0/87.8	88.4	94.4
+ additional data	126GB	2K	500K	94.5/88.8	89.8	95.6

9.2.2 实验结果

1. 在微调实验中，BERT, ROBERT, XLNET 都采用了 BERT LARGE 的网络参数：

$L = 24, H = 1024, A = 16$ ，参数总量大约 3.55 亿。

2. 微调任务：

- General Language Understanding Evaluation: GLUE：是一组各种 NLP 文本任务。
- Stanford Question Answering Dataset: SQuAD：一个大型阅读理解数据集，包含 SQuAD 1.1 和 SQuAD 2.0。其中 SQuAD 1.1 中的问题总是有答案，而 SQuAD 2.0 中的问题可能没有答案。
- Reading Comprehension from Examinations: RACE：包含了12到18岁中国初中、高中英语考试中的近10万个问题，答案由人工专家生成。

涉及到挑战性的推理问题，所以它是最困难的阅读理解数据集之一。另外，该数据集每篇文章平均超过300个段落，远远超过其它流行的阅读理解数据集(如 SQuAD)。因此该数据集对于长文本理解来说是个具有挑战性的 baseline 。

3. GLUE 微调结果：

目前有两种微调策略：

- single-task：基于每个 GLUE 任务的标记数据来训练该任务。
- multi-task：基于多个 GLUE 任务的标记数据来多任务训练，然后再微调具体的任务。

RoBERTa 执行的是 single-task 微调策略，其结果如下图所示。其中：

- RoBERTa 的 ensembles 基于 single-task 得到的多个模型的集成。这种方式在 9 个任务中的 4 个做到了 SOA 。
- 论文作者预期：一旦使用多任务微调策略，则 RoBERTa 的成绩会更好。
- QNLI 通常都被视为一个二分类问题，但是近期的研究将其视为 ranking 问题。RoBERTa 也延续了 ranking 的思路。
- 论文使用 SuperGLUE 的 WNLI 来替代 NLI 。
- single-task 在 dev 上，RoBERTa 在所有 9 个任务都取得了 SOA 。

	MNLI	QNLI	QQP	RTE	SST	MRPC	CoLA	STS	WNLI	Avg
<i>Single-task single models on dev</i>										
BERT _{LARGE}	86.6/-	92.3	91.3	70.4	93.2	88.0	60.6	90.0	-	-
XLNet _{LARGE}	89.8/-	93.9	91.8	83.8	95.6	89.2	63.6	91.8	-	-
RoBERTa	90.2/90.2	94.7	92.2	86.6	96.4	90.9	68.0	92.4	91.3	-
<i>Ensembles on test (from leaderboard as of July 25, 2019)</i>										
ALICE	88.2/87.9	95.7	90.7	83.5	95.2	92.6	68.6	91.1	80.8	86.3
MT-DNN	87.9/87.4	96.0	89.9	86.3	96.5	92.7	68.4	91.1	89.0	87.6
XLNet	90.2/89.8	98.6	90.3	86.3	96.8	93.0	67.8	91.6	90.4	88.4
RoBERTa	90.8/90.2	98.9	90.2	88.2	96.7	92.3	67.8	92.2	89.0	88.5

4. SQuAD 微调结果：

BERT 和 XLNet 都使用额外的高质量标记数据来扩充训练集，而 RoBERTa 仅使用指定的标记数据。

结果如下图所示，其中 † 表示依赖于额外的标记数据。

Model	SQuAD 1.1		SQuAD 2.0	
	EM	F1	EM	F1
<i>Single models on dev, w/o data augmentation</i>				
BERT _{LARGE}	84.1	90.9	79.0	81.8
XLNet _{LARGE}	89.0	94.5	86.1	88.8
RoBERTa	88.9	94.6	86.5	89.4
<i>Single models on test (as of July 25, 2019)</i>				
XLNet _{LARGE}			86.3†	89.1†
RoBERTa			86.8	89.8
XLNet + SG-Net Verifier			87.0†	89.9†

5. Race 微调结果：

Model	Accuracy	Middle	High
<i>Single models on test (as of July 25, 2019)</i>			
BERT _{LARGE}	72.0	76.6	70.1
XLNet _{LARGE}	81.7	85.4	80.2
RoBERTa	83.2	86.5	81.3