bert变种

1.RoBERTa

> 原论文链接: https://arxiv.org/pdf/1907.11692.pdf

RoBERTa 的全称是 Robustly optimized BERT approach。

RoBERTa 是在 bert 的基础上做了一些改进，这些改进并不是设计什么新颖的结构，而是尽量使模型得到更充分的预训练，释放 bert 模型的潜力。改进共有五个方面：

1. 更大的 Batch Size 与更大规模的数据集

RoBERTa 在训练过程中采用了 **显著增大的 batch size** （例如从 BERT 的 256 提升到 8192），并且使用了 **更大、更多样化的预训练语料库**，包括 BookCorpus、Wikipedia、Common Crawl 等多个来源的数据。这种做法使得模型能够在更多样本上进行学习，提高了泛化能力。此外，更大的 batch size 也有助于提升训练的稳定性与效率，使模型更容易收敛到一个更优的状态。

2. 统一使用更长的序列长度（Sequence Length）

在 BERT 的训练中，为了提高训练效率，通常会先用较短的句子（如平均长度为 128 tokens）进行初期训练，然后再逐步过渡到最大长度（512 tokens）。这种方式虽然节省了计算资源，但可能影响模型对长文本的理解能力。

相比之下，RoBERTa 在预训练阶段始终使用接近最大长度（512 tokens）的输入序列。也就是说，每条训练样本都尽量填充到接近 512 个 token 的长度，而不是掺杂大量短句。这种做法有助于模型更好地学习长距离依赖关系，并提升其在处理复杂、长文本任务时的表现。

3. 移除下一句预测任务（NSP, Next Sentence Prediction）

原始 BERT 在预训练阶段引入了一个辅助任务：**下一句预测（NSP）**，用于判断两个句子是否连续出现。然而，RoBERTa 的实验结果表明，这个任务对于模型性能的提升作用有限，甚至可能带来负面影响。因此，RoBERTa 完全去除了 NSP 任务，仅保留了原始 BERT 中的 **Masked Language Modeling（MLM）** 作为唯一的预训练目标。

实验结果显示，去除 NSP 后，模型在多项自然语言理解任务上的表现反而有所提升，说明 MLM 已经足够让模型学习句子之间的相关性。

4. 将静态 Mask 改为动态 Mask

在 BERT 中，token 的 mask 是在预处理阶段就固定下来的，也就是所谓的“静态 mask”。这意味着每个训练样本在整个训练周期中被 mask 的位置是不变的。

而 RoBERTa 引入了“动态 mask”机制，在每次训练迭代中都会随机选择不同的 token 进行 mask。这相当于每次训练看到的是略有不同的训练目标，增加了训练的多样性，有助于提升模型的鲁棒性和泛化能力。动态 mask 机制模拟了真实推理场景下的不确定性，增强了模型的适应性。

5. 采用 GPT-2 的 BPE 分词策略

RoBERTa 在分词（tokenization）阶段采用了与 GPT-2 相同的 **Byte-Pair Encoding（BPE）** 策略，而不是 BERT 所使用的 WordPiece 分词方法。BPE 是一种基于子词单元（subword units）的编码方式，相比传统的词汇级分词，它可以更灵活地处理未登录词（OOV），并减少词汇表大小。此外，BPE 能够更好地捕捉不同语言结构之间的共性，从而提升模型的语言建模能力。

2.ALBERT

> 原文链接：https://openreview.net/pdf?id=H1eA7AEtvS

ALBERT 的全称为 A Lite BERT。所以从名字可以看出，这个模型的目的是想搞一个比 Bert 更小、更轻量级的模型。这个模型相比于 Bert 在三个方面做了修改。

2.1 对 Embedding 层参数进行因式分解

在 BERT 模型中，**词嵌入层（Embedding Layer）的向量维度 E 与 Transformer 层中隐藏状态的向量维度 H 是相同的。**然而，本文作者认为这两者没有必要保持一致，理由如下：

首先，从模型各层所学习的信息来看，不同层级通常捕捉到的是不同类型的语言特征。例如，在 ELMo 模型中已有研究表明：靠近输入的低层网络主要学习语法层面的信息，而高层网络则更多地捕捉语义层面的内容。

在本文中，作者提出：**Embedding 层输出的向量是尚未引入上下文信息（context-free）的基础表示，而 Transformer 层输出的向量则是经过自注意力机制处理后、融合了上下文信息的高阶表示。**因此，为了更好地表达复杂的上下文依赖关系，Transformer 层所需的表示能力更强，其向量维度 H 应该显著大于 Embedding 层的维度 E。

其次，**Embedding 层的参数在整个模型参数中占比相当大。**以 BERT-Base 为例，其词汇表大小 $V = 30,522$，隐藏维度 $H = 768$，Embedding 层参数数量为：
$V \times H = 30{,}522 \times 768 \approx 23\text{M}$，占整个模型参数的一半以上。此外，由于 Embedding 层的更新依赖于词频分布，而在训练过程中，高频词的 embedding 向量更新频繁，低频词则更新稀疏，导致 Embedding 层整体的更新效率较低。 因此，减少 Embedding 层的参数量不仅有助于降低模型整体复杂度，也有利于提升训练效率。

基于上述分析，作者提出对 Embedding 层进行参数因式分解。具体来说：

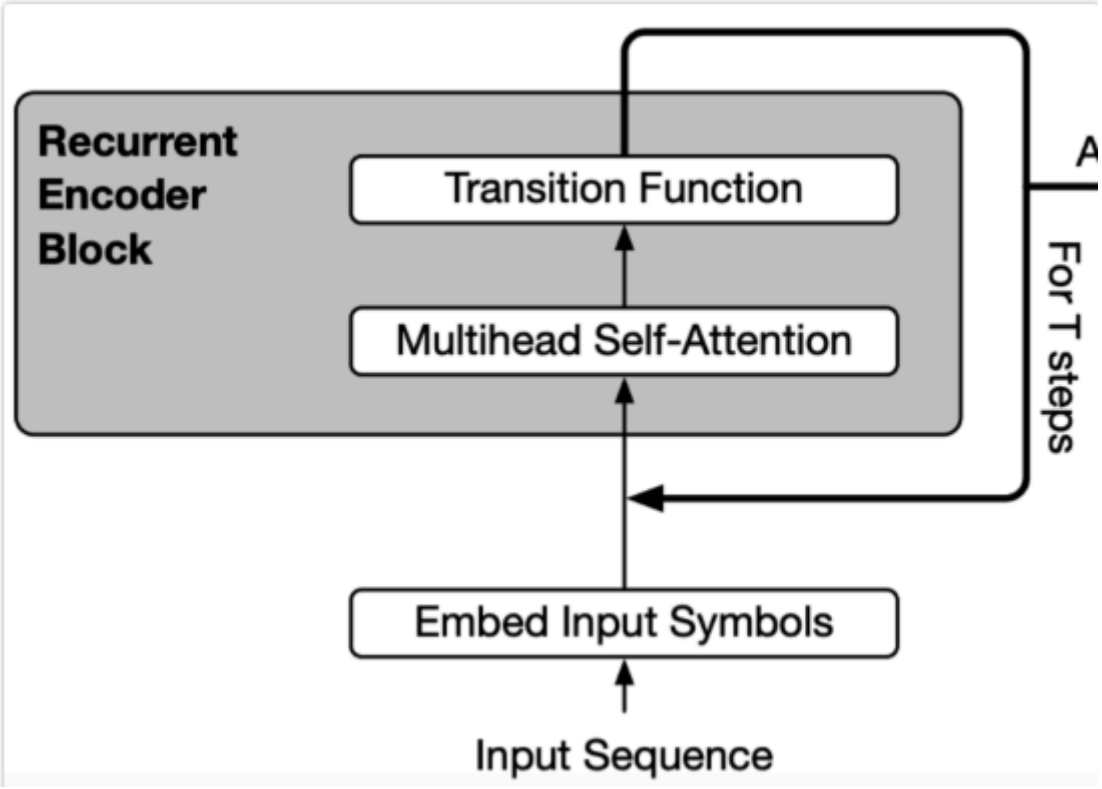
- 将 Embedding 层的维度设为 E，其中 $E < H$；
- Embedding 权重矩阵的维度为 $V \times E$；
- 在得到词嵌入向量后，再通过一个线性变换将其投影到 Transformer 层的隐空间中，该变换矩阵的维度为 $E \times H$。

这样改进后的 Embedding 层总参数量为：$V \times E + E \times H$。以 BERT-Base 模型为例，若设定 $E = 128$，则 Embedding 层的参数量为：
$V \times E + E \times H = 30{,}522 \times 128 + 128 \times 768 = 3{,}906{,}816 + 98{,}304 = 4{,}005{,}120 \approx 4\text{M}$。
相比之下，原始 BERT 的 Embedding 参数量为：$V \times H = 30{,}522 \times 768 \approx 23\text{M}$。可以看到，Embedding 层的参数量减少了约 **83%**，大大降低了模型的整体参数规模和训练开销。

通过对 Embedding 层参数进行因式分解，将词嵌入维度 E 设置为低于 Transformer 层的隐藏维度 H，并在中间加入一个小型的线性映射层，不仅可以有效减少 Embedding 层的参数量，还能提升训练效率和模型的表达能力。这种设计思路体现了对模型结构中各组件功能差异的深入理解，是一种实用且高效的优化策略。

2.2 跨层参数共享

这部分的做法很容易理解，就是**所有的 transformer 层共享相同的参数，也就是说实际上只有一个 transformer 层的权重**，然后会多次经过这个 transformer 层。比如 bert-base 有 12 层 transformer，改成 ALBERT就是数据会经过同一个 transformer 层 12 次，如下图：



2.3 将 NSP 任务换成了 SOP 任务

SOP 的全称为 sentence order prediction。在该文章之前已经有些文章发现，bert 的论文中的 NSP 任务没有什么作用。该论文任务 NSP 任务之所以没有作用，是因为其太简单了，所以在其基础上设计了一个难度更高的新任务，也就是 SOP 任务。**SOP 任务就是预测两句话有没有被交换过顺序。**

3. SpanBERT：对BERT的改进

SpanBERT 是对原始 BERT 模型的一种改进版本。它在训练目标和预训练任务上进行了三项关键性的修改，使得模型在理解连续文本片段（span）方面表现得更好。这三项改进分别是：

1. **将 Token-level Mask 改为 Span-level Mask**
2. **引入 Span Boundary Objective (SBO) 损失函数**
3. **去除了 Next Sentence Prediction (NSP) 任务**

3.1 将 Token-level Mask 改为 Span-level Mask

在原始的 BERT 中，mask 的方式是随机选择一些 token 进行 mask。这种方式虽然有效，但并没有考虑语言中词语或短语通常是以“连续片段”形式出现的特点。为了更好地建模连续语义单元，**SpanBERT 提出了使用 Span-level Masking（跨度掩码）的方法。**
具体来说：

- 不再是随机地选择单个 token 进行 mask；
- 而是每次从句子中选择一个连续的 span（比如一个词组、短语等），并对这个 span 内的所有 token 进行 mask。

实现细节如下：

- 首先，从一个**几何分布**中采样出一个 span 的长度 $L \sim \mathrm{Geometric}(p)$；
- 然后，在句子中**相等概率随机地**选择一个起始位置 s；
- 最终得到被 mask 的 span：$(x_s, x_{s+1}, \ldots, x_{s+L-1})$。

这样的做法更符合自然语言的结构，也更有助于模型学习到上下文中连续语义单元的整体表示。

3.2 引入 Span Boundary Objective (SBO)

很多下游任务（如共指消解、命名实体识别等）会用到“span 的边界信息”来代表整个 span 的含义。受此启发，SpanBERT 引入了一个新的训练目标：**利用 span 的边界 token 来预测被 mask 掉的内部 token。**

我们假设有一个输入序列：$X = (x_1, x_2, \ldots, x_n)$。其中某个 span 被 mask 掉了，记作：$(x_s, x_{s+1}, \ldots, x_e)$,这里的 s 和 e 分别是被 mask 的起始和结束位置。SpanBERT 使用这两个边界 token —— 即前一个位置 x_{s-1} 和后一个位置 x_{e+1}，来重建中间被 mask 掉的内容。对于每一个被 mask 的位置 p_i（即在 s 到 e 之间的某个位置），模型通过以下函数进行预测：

$$y_i = f(x_{s-1}, x_{e+1}, p_i)$$

其中，函数 f 的具体实现是一个两层全连接神经网络，中间使用 GeLU 激活函数。

这种设计让模型不仅关注局部上下文，还能更好地捕捉 span 整体的信息，从而提升其在需要理解连续语义单元的任务上的表现。

3.3 去除 NSP（Next Sentence Prediction）

在原始 BERT 中，NSP 是一个用于判断两个句子是否连续的二分类任务。然而，后续研究表明，NSP 对模型的帮助有限，甚至可能影响模型对语言本身的建模能力。

因此，SpanBERT 直接去掉了 NSP 任务，只保留 MLM（Masked Language Model）任务，并结合新提出的 SBO 任务进行训练。这种简化不仅提高了训练效率，也增强了模型的语言理解能力。

4.XLNet

XLNet是由卡内基梅隆大学和Google大脑联合提出的一种算法，其沿用了自回归的语言模型，并利用排列语言模型合并了bert的优点，同时集成transformer-xl对于长句子的处理，达到了SOTA的效果。

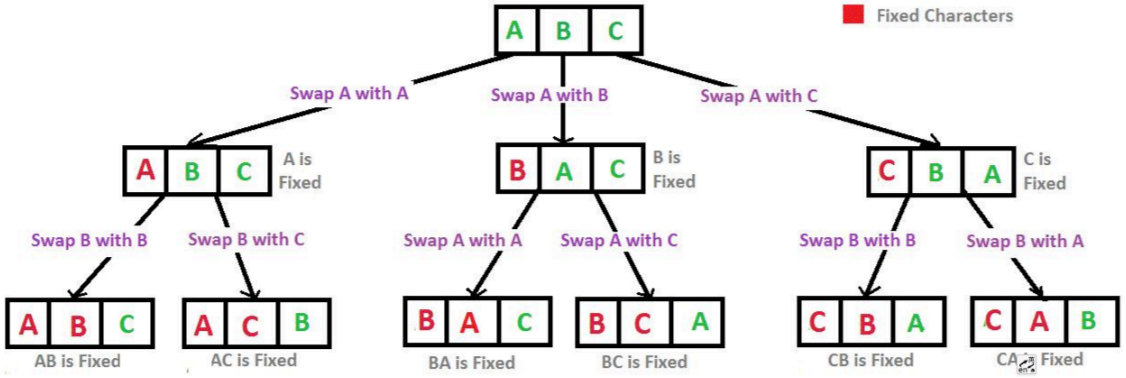
4.1 AR和AE

- AR：Autoregressive Language Modeling，自回归语言模型是一种**从左到右或从右到左** 预测下一个词的语言模型。它的目标是根据前面的词来预测当前词。
- AE： Autoencoding Language Modeling，自编码语言模型的核心思想是随机 mask 掉一些词，然后让模型根据上下文去还原这些被 mask 的词** 。

XLNet 的出发点就是：能否融合AR LM 和 AE LM 两者的优点。具体来说就是，站在 AR 的角度，如何引入和双向语言模型等价的效果.

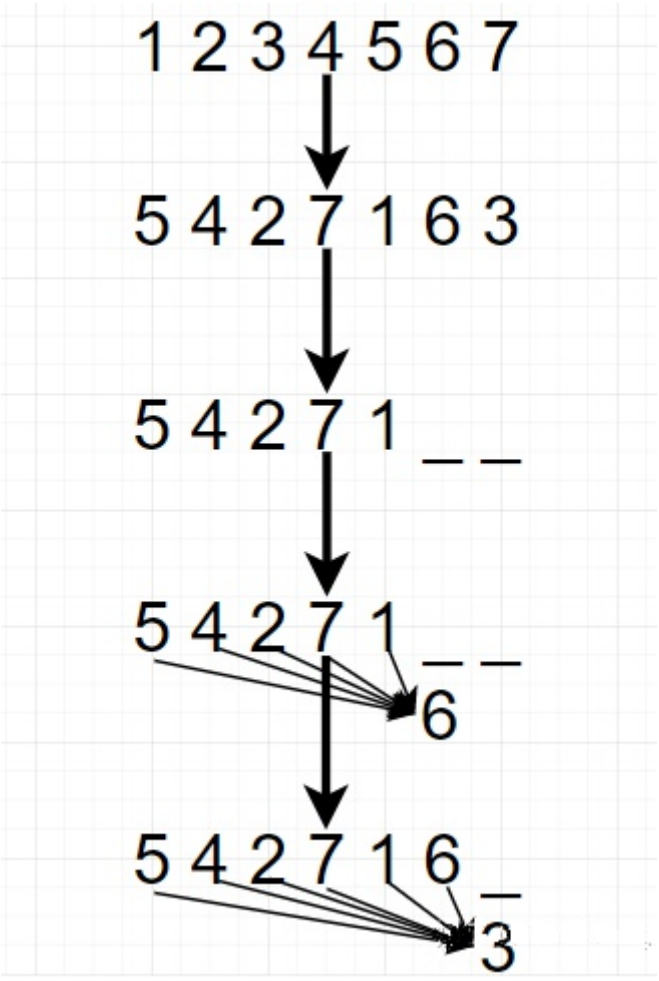
4.2 排列语言模型（Permutation Language Model）

作者发现，只要在 AR中再加入一个步骤，就能够完美地将AR与AE的优点统一起来，那就是提出 **Permutation Language Model**（PLM）。传统的 AR 模型只能从左到右预测，所以只能利用左侧上下文；但如果我们对输入 token 的顺序进行**排列组合** ，就能让模型看到不同位置的信息，从而模拟出“双向”的效果。

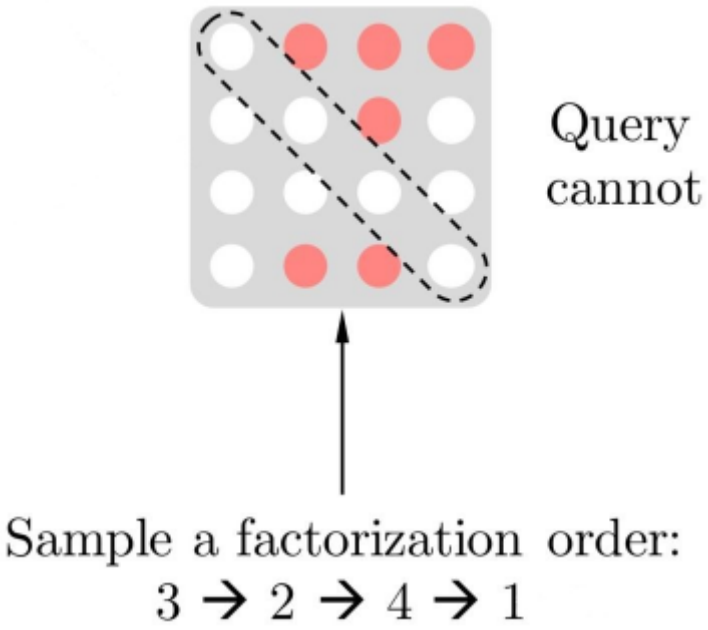


Recursion Tree for Permutations of String "ABC"

具体实现方式是，通过**随机取一句话的一种排列，然后将末尾一定量的词给“遮掩”**（和 BERT 里的直接替换 “[MASK]” 有些不同）掉，最后**用 AR 的方式来按照这种排列依次预测被“遮掩”掉的词。**



我们可以发现通过随机取排列（Permutation）中的一种，就能非常巧妙地通过 **AR 的单向方式来习得双向信息**了。论文中 Permutation 具体的实现方式是通过直接对 Transformer 的 **Attention Mask** 进行操作。



为了确保模型按照排列的顺序预测，我们需要对 Transformer 的注意力机制施加掩码。具体来说：对于每个 token，它只能看到排列中位于它前面的 token；排列中位于它后面的 token 被遮盖掉（即注意力权重为 0）。

比如说序号依次为 1234 的句子，先随机取一种排列3241。于是根据这个排列就做出类似上图的 Attention Mask。先看第1行，因为在新的排列方式中 1 在最后一个，根据从左到右 AR 方式，1 就能看到 234 全部，于是第一行的 234 位置是红色的（没有遮盖掉，会用到），以此类推。第2行，因为 2 在新排列是第二个，只能看到 3，于是 3 位置是红色。第 3 行，因为 3 在第一个，看不到其他位置，所以全部遮盖掉...

4.3 Two-Stream Self-Attention

在 XLNet 中，为了更好地实现排列语言模型（Permutation Language Model）和自回归预测的目标，作者引入了一种特殊的注意力机制，叫做 **Two-Stream Self-Attention（双流自注意力）**。它的核心目的是解决一个关键问题：**在预测某个位置的词时，既要利用上下文信息，又不能提前看到当前位置本身的词内容。**

我们知道，在传统的 BERT 中，输入是将 token 的内容 embedding 和位置 embedding 直接相加得到的。这种做法虽然有效，但在 XLNet 的排列语言模型中会带来“信息泄露”的风险。因为如果当前位置的内容信息已经包含在输入中，那么模型可能就会直接复制这个信息到输出，而不是真正去学习上下文关系。**XLNet 在 Two-Stream Self-Attention 中，先通过只包含位置信息、不含当前 token 内容的 Query Stream g 来计算注意力分数；然后再使用包含当前位置内容信息的 Content Stream h 来获取实际的内容表示。**

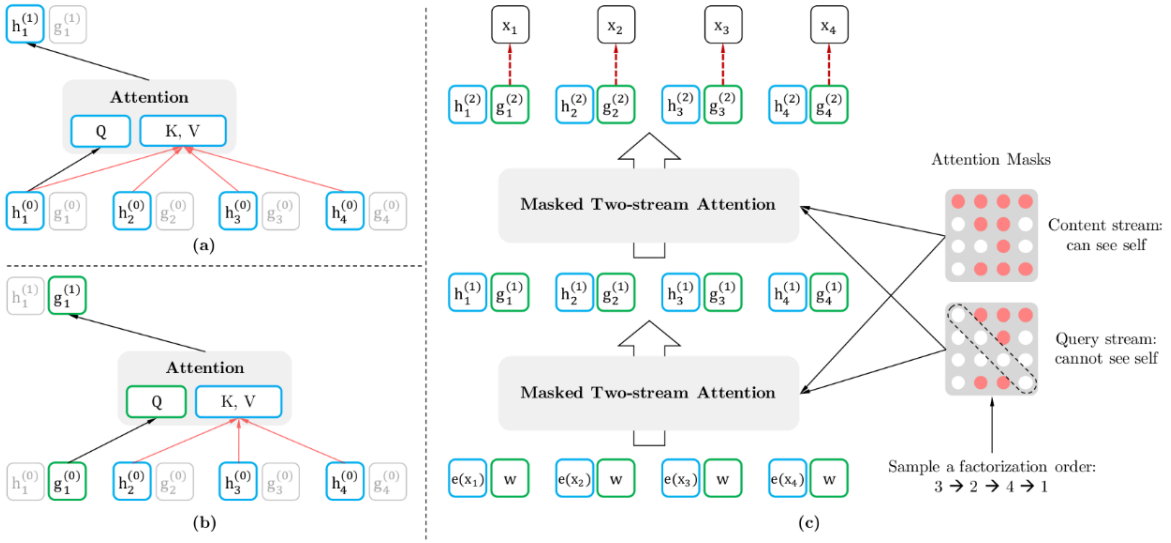
为了解决这个问题，XLNet 提出了两个不同的隐状态流（hidden state streams）来分别处理位置信息和内容信息：

1. **Content Stream（内容流）：记作 h** 这个流包含了当前位置的完整信息，包括内容和位置。它在整个序列中传播，并作为 Key（键）和 Value（值）参与注意力计算。可以把它看作是一个“全局的知识库”，记录了每个位置的真实内容信息。
2. **Query Stream（查询流）：记作 g** 这个流只包含位置信息，而不包含当前位置的词内容。它用于生成当前 token 的预测结果。也就是说，在预测第 i 个位置的词时，我们使用的是 g_i，这样就不会提前看到该位置的内容，从而避免了信息泄露。

这两个流在每一层 Transformer 中都会被更新。具体来说：

- 在计算注意力时，g_i 作为 Query（查询），从其他位置的 h_j 中提取相关信息；

- 同时，h_i 也会通过自身的注意力机制进行更新，继续保留该位置的内容信息。



上图中我们需要理解两点：

1.首先，在模型最底层输入中，Content Stream（蓝色部分）的输入是每个 token 的词向量 e^{x_i}，这是非常直观的设计：不同的词对应不同的 embedding 向量。而 Query Stream（绿色部分）的输入却都是同一个向量 w，这看起来有些奇怪。**实际上，Query Stream 的作用是用于“查询上下文信息”，而不是表示当前位置的具体内容。因此，它的输入只需要包含位置信息，而不应包含当前 token 的内容。**为了弥补这种统一输入带来的位置信息缺失，XLNet 引入了 Relative Positional Encoding（相对位置编码）。通过这种方式，即使所有 Query Stream 的初始输入都是相同的 w，模型依然能够感知到不同 token 之间的相对位置关系，从而正确建模上下文。

2.在图示中的 Query Stream Attention 部分，为了便于理解，似乎只将“非当前 token”的 h 作为 Key 和 Value 使用。但实际上，在完整实现中，**所有的 h 都会被用作 Key 和 Value**，然后通过 Attention Mask 来控制哪些位置是可以访问的。也就是说，模型会先让 Query Stream 基于位置信息 w 和相对位置编码去关注所有位置的 h，再通过 Attention Mask 屏蔽掉那些在当前排列顺序下不可见的位置。这样设计的好处是：

- 模型可以在完整的上下文中进行注意力计算；
- 同时又能保证预测某个 token 时，只能看到排列中比它早的位置，避免信息泄露。

通过这两个设计，Two-Stream Self-Attention 成功实现了对内容和位置信息的分离处理，同时借助 Relative Positional Encoding 和 Attention Mask 完整地建模了上下文依赖关系，使得 XLNet 能够灵活适应各种排列顺序，并学习到双向上下文信息，而无需像 BERT 那样依赖 `[MASK]` 标记。

4.4 Partial Prediction

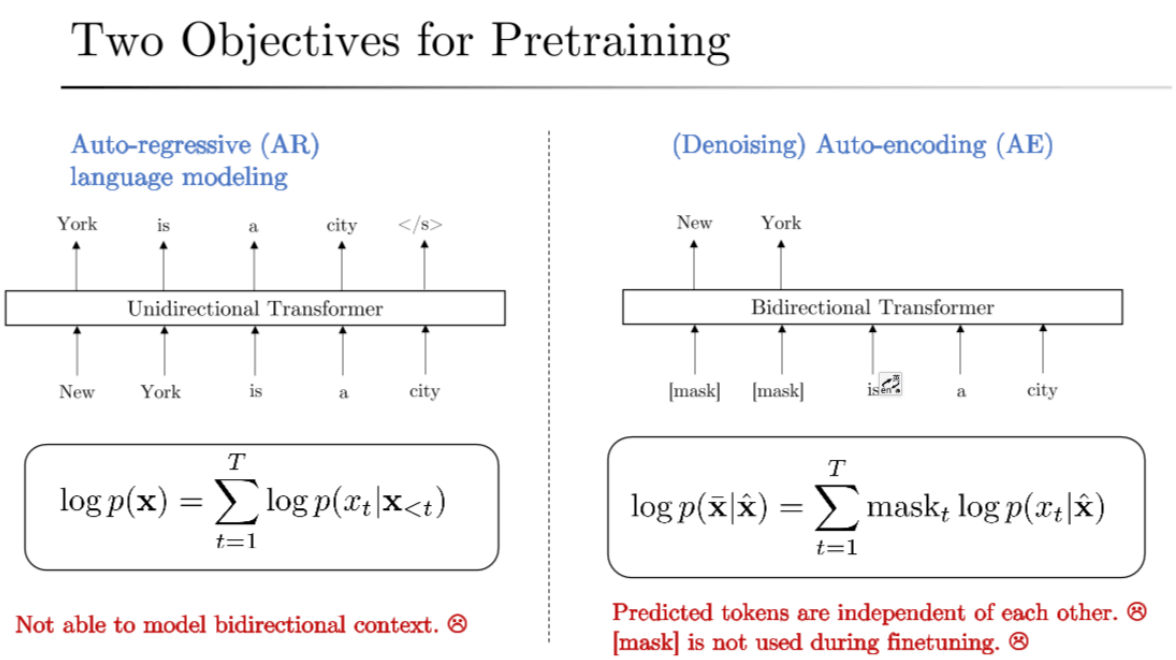
XLNet还使用了部分预测（Partial Prediction）的方法。因为LM是从第一个Token预测到最后一个Token，在预测的起始阶段，上文信息很少而不足以支持Token的预测，这样可能会对分布产生误导，从而使得模型收敛变慢。为此，XLNet只预测后面一部分的Token，而把前面的所有Token都当作上下文。

具体来说，对长度为T的句子，我们选取一个超参数K，使得后面 `1/K` 的Token用来预测，前面 `1-1/K` 的Token用作上下文。注意，`K` 越大，上下文越多，模型预测就越精确。

4.5 Transformer-XL

对于过长序列，如果分段来进行处理，往往会遗漏信息，且效果会下降，那么xlnet借鉴了Transformer-XL的思想，设置一个保留上一个片段的信息，在训练时进行更新。

5.AR和AE



5.1 自回归语言模型（AutoRegressive Language Model）

自回归语言模型（AR LM） 是指根据前面（或后面）已经出现的 token 来预测当前时刻的 token。典型的代表有 **ELMO** 和 **GPT**。其中，**GPT 是标准的单向自回归语言模型**，而 **ELMO 虽然看起来是双向的**（即同时利用了上文和下文），但本质上仍然是自回归语言模型。它的实现方式是分别训练两个方向的语言模型（从左到右和从右到左），然后将两个方向 LSTM 的隐状态拼接在一起。这种方式虽然能捕捉上下文信息，但融合方式相对简单，效果有限。

给定一个文本序列：

$$x = [x_1, x_2, \ldots, x_T]$$

语言模型的目标是最大化训练数据上的似然函数：

$$\max_{\theta} \log p_\theta(x) = \sum_{t=1}^{T} \log p_\theta(x_t \mid x_{<t}) = \sum_{t=1}^{T} \log \frac{\exp(h_\theta(x_{1:t-1})^T e(x_t))}{\sum_{x'} \exp(h_\theta(x_{1:t-1})^T e(x'))}$$

其中：

- $x_{<t}$ 或写作 $x_{1:t-1}$ 表示第 t 个位置之前的所有 token；
- $h_\theta(x_{1:t-1})$ 是 RNN 或 Transformer 编码得到的隐状态；
- $e(x)$ 是词 x 的 embedding 向量。

自回归语言模型的优缺点：

1.优点：

- 非常适合生成类任务（如文本摘要、机器翻译等），因为这些任务在生成过程中也是从左到右逐字生成的；
- 模型结构天然匹配生成过程，不需要额外设计。

2.缺点：

- **无法同时利用上下文的信息**（只能看到前面或者后面的内容）；
- 即使像 ELMO 使用了双向 AR 模型拼接的方式，由于融合方法较为简单，效果提升也有限；
- 相比 BERT 这类双向建模的模型，在理解任务上表现稍弱。

5.2 自编码语言模型（AutoEncoder Language Model）

BERT 是典型的自编码语言模型（AE LM）。它通过随机将输入序列中约 15% 的 token 替换为 `[MASK]`，形成带噪声的输入 $\hat{x}$。然后模型尝试根据上下文恢复被 mask 掉的原始 token。

假设被 mask 的原始 token 为 $\bar{x}$，BERT 的目标是最大化以下似然函数：

$$\max_{\theta} \log p_\theta(\bar{x} \mid \hat{x}) \approx \sum_{t=1}^{T} m_t \log p_\theta(x_t \mid \hat{x}) = \sum_{t=1}^{T} m_t \log \frac{\exp(H_\theta(\hat{x})_t^T e(x_t))}{\sum_{x'} \exp(H_\theta(\hat{x})_t^T e(x'))}$$

其中：

- $m_t = 1$ 表示第 t 个位置是一个被 mask 的 token；
- $H_\theta(\hat{x}) = [H_\theta(\hat{x})_1, H_\theta(\hat{x})_2, \ldots, H_\theta(\hat{x})_T]$ 是 Transformer 输出的隐状态序列；
- 注意：与 AR 模型不同的是，BERT 中的 Transformer 可以同时看到整个句子的所有 token。

自编码语言模型的优缺点

1.优点：

- 能自然地建模双向语言模型，同时看到目标 token 的上下文；
- 在很多理解类任务（如分类、问答、命名实体识别）中表现优异。

2. 缺点：

- **预训练阶段引入了 `[MASK]` 标记**，而在实际 fine-tuning 阶段并不会出现这个标记，导致训练与推理阶段不一致；
- 不太适合生成类任务，因为这类任务需要逐步生成 token，而 AE 模式并不直接建模这种过程。

总结对比

类型	模型代表	是否双向	是否适合生成任务	是否存在训练与推理不一致
自回归语言模型（AR LM）	GPT、ELMO	否（或伪双向）	强项	不存在
自编码语言模型（AE LM）	BERT	是	较弱	存在（因 `[MASK]`）