# 后Bert时代模型

## 0x01. RoBERTa

RoBERTa出自 facebook 的论文 RoBERTa: A Robustly Optimized BERT Pretraining Approach

RoBERTa模型是对BERT模型的**精调**，它细致分析了BERT模型的超参数，并对模型做了微小的改动，来提升模型的表征能力。同时，使用了比BERT多1000%的数据集和算力。但是这篇文章没有多大创新，感觉更像一个Bert实验报告，导致被ICLR2020拒了...

> (We) carefully measures the impact of many key hyperparameters and training data size.... We find that BERT was significantly undertrained and propose an improved recipe for training BERT models, which we call RoBERTa. --原论文

为了改进训练过程，RoBERTa：**【移除NSP任务；引入动态掩码；更大的batchsize；更多的语料】**

（1）**移除【Next Sentence Prediction (NSP)】 任务**

实验发现，删除NSP任务并未对模型在下游任务上的表现造成影响，甚至能够有所提升。

其实，NSP任务在直觉上是没毛病的，但在各paper的实际实验中很拉跨。具体的原因见下文ALBERT。

（2）引入动态掩码（dynamic mask）

在BERT中，为了使得预训练数据更加多样，在预处理阶段随机生成了10份语料，并给这10份语料标记不同的[MASK]。但是，如果模型训练的epoch数大于10，模型还是会训练重复的语料。（假如epoch是40，模型会训练4次同样的语料）。在RoBERTa中，为了解决上述问题，使用了动态MASK方法，即在每个批次数据喂进模型的时候执行MASK操作，而不是在BERT的预处理阶段MASK，这就使得模型学习重复语料的可能性大大降低。

> The original BERT implementation performed masking once during data preprocessing, resulting in a single static mask. To avoid using the same mask for each training instance in every epoch, training data was duplicated 10 times so that each sequence is masked in 10 different ways over the 40 epochs of training. Thus, each training sequence was seen with the same mask four times during training. -- 原论文

| Masking | SQuAD 2.0 | MNLI-m | SST-2 |
|---|---|---|---|
| reference | 76.3 | 84.3 | 92.8 |
| *Our reimplementation:* | | | |
| static | 78.3 | 84.3 | 92.5 |
| dynamic | 78.7 | 84.0 | 92.9 |

Table 1: Comparison between static and dynamic masking for $\text{BERT}_{\text{BASE}}$. We report F1 for SQuAD and accuracy for MNLI-m and SST-2. Reported results are medians over 5 random initializations (seeds). Reference results are from Yang et al. (2019).

提升倒也不是很明显...

（3）发现更大的batch_size在训练过程中更有用

| 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 |

Table 3: Perplexity on held-out training data (*ppl*) and development set accuracy for base models trained over BOOKCORPUS and WIKIPEDIA with varying batch sizes (*bsz*). We tune the learning rate (*lr*) for each setting. Models make the same number of passes over the data (epochs) and have the same computational cost.

（4）**更多的语料**，更长的训练时间

更重要的是，RoBERTa 使用 160 GB 的文本进行预训练，其中包括 16 GB 的图书语料库和 BERT 中使用的英文维基百科。附加数据包括 CommonCrawl 新闻数据集（6300 万篇文章，76 GB）、Web 文本语料库（38 GB）和来自 Common Crawl 的 Stories（31 GB）。再加上运行了一天的 1024 V100 Tesla GPU，对 RoBERTa 进行了预训练。

因此，RoBERTa 在 GLUE 基准测试结果上优于 BERT 和 XLNet：

| | MNLI | QNLI | QQP | RTE | SST | MRPC | CoLA | STS | WNLI | Avg |
|---|---|---|---|---|---|---|---|---|---|---|
| *Single-task single models on dev* | | | | | | | | | | |
| $\text{BERT}_{\text{LARGE}}$ | 86.6/- | 92.3 | 91.3 | 70.4 | 93.2 | 88.0 | 60.6 | 90.0 | - | - |
| $\text{XLNet}_{\text{LARGE}}$ | 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** | - |
| *Ensembles on test (from leaderboard as of July 25, 2019)* | | | | | | | | | | |
| 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** |

## 0x02. ALBERT

ALBERT是Google于2020年发表的论文ALBERT: A LITE BERT FOR SELF-SUPERVISED LEARNING OF LANGUAGE REPRESENTATIONS 中提出的。 **ALBERT在保持性能的基础上，大大减少了模型的参数，** 使得实用变得更加方便，是经典的BERT变体之一。

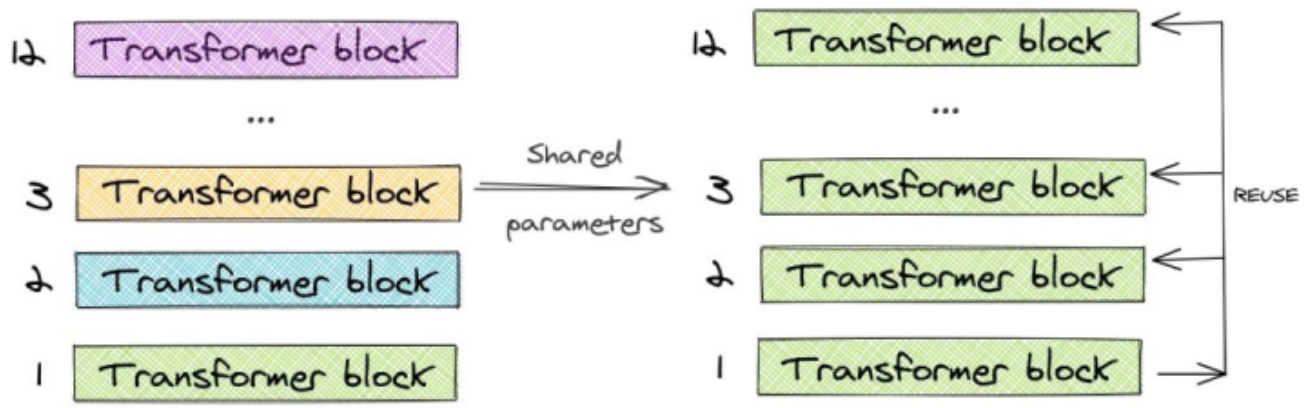
ALBERT做了如下改进：**跨层参数共享；句子顺序预测（Sentence Order Prediction）;对embedding做分解**

1.跨层参数共享

BERT-large模型有24层，而它的基础版本有12层。随着层数的增加，参数的数量呈指数增长。

| Version | Hidden units | #layers | #parameters |
|---|---|---|---|
| BERT-large | 1024 | 24-layer | 340 million |
| BERT-base | 768 | 12-layer | 110 million |

为了解决这个问题，ALBERT使用了跨层参数共享的概念。为了说明这一点，让我们看一下12层的BERT-base模型的例子。我们只学习第一个块的参数，并在剩下的11个层中重用该块，而不是为12个层中每个层都学习不同的参数。我们可以**只共享feed-forward层的参数/只共享注意力参数/共享所有的参数。**论文中的default方法是对所有参数都进行了共享。




与BERT-base的1.1亿个参数相比，相同层数和hidden size的ALBERT模型只有3100万个参数。当hidden size为128时，对精度的影响很小。精度的主要下降是由于feed-forward层的参数共享。共享注意力参数的影响是最小的。

| | Model | Parameters | SQuAD1.1 | SQuAD2.0 | MNLI | SST-2 | RACE | Avg |
|---|---|---|---|---|---|---|---|---|
| ALBERT base $E$=768 | all-shared | 31M | 88.6/81.5 | 79.2/76.6 | 82.0 | 90.6 | 63.3 | 79.8 |
| | shared-attention | 83M | 89.9/82.7 | 80.0/77.2 | 84.0 | 91.4 | 67.7 | 81.6 |
| | shared-FFN | 57M | 89.2/82.1 | 78.2/75.4 | 81.5 | 90.8 | 62.6 | 79.5 |
| | not-shared | 108M | 90.4/83.2 | 80.4/77.6 | 84.5 | 92.8 | 68.2 | 82.3 |
| ALBERT base $E$=128 | all-shared | 12M | 89.3/82.3 | 80.0/77.1 | 82.0 | 90.3 | 64.0 | 80.1 |
| | shared-attention | 64M | 89.9/82.8 | 80.7/77.9 | 83.4 | 91.9 | 67.6 | 81.7 |
| | shared-FFN | 38M | 88.9/81.6 | 78.6/75.6 | 82.3 | 91.7 | 64.4 | 80.2 |
| | not-shared | 89M | 89.9/82.8 | 80.3/77.3 | 83.2 | 91.5 | 67.9 | 81.6 |

2 **句子顺序预测 (SOP)**

上文提到，RoBERTa的论文已经阐明了NSP的无效性，并且发现它对下游任务的影响是不可靠的。在取消NSP任务之后，多个任务的性能都得到了提高。为什么NSP无效呢？ALBERT论文中的解释是这样的：

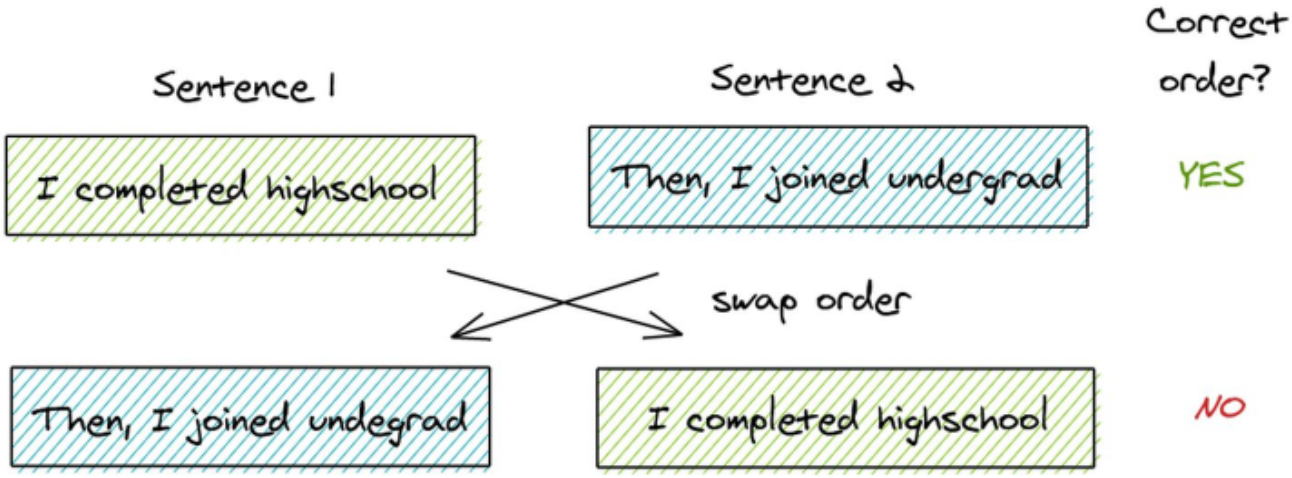
> ... the main reason behind NSP's ineffectiveness is its lack of difficulty as a task, as compared to MLM. As formulated, NSP conflates topic prediction and coherence prediction in a single task. However, topic prediction is easier to learn compared to coherence prediction.

ALBERT推测NSP是无效的，因为与掩码语言建模相比，它并不是一项困难的任务。在单个任务中，它混合了**主题预测(topic prediction) 和连贯性预测(coherence prediction)**。主题预测部分很容易学习，因为它与掩码语言建模的损失有重叠。

这个说法还是很make sense的。记得NSP的负样本是取的两篇不同文章里的句子，这两个句子可能完全不相关，那么我只要判断出它们不相关，就能确定这不是一个next sentence了。也就是说，单单只凭借topic prediction就能够完成NSP任务，而完全没有学习到coherence prediction。

因此，ALBERT提出了另一个任务"句子顺序预测"。关键思想是:

- 从同一个文档中取两个连续的段落作为一个正样本
- 交换这两个段落的顺序，并使用它作为一个负样本




SOP完全抛弃了topic prediction，而专注于coherence prediction。

SOP提高了下游任务(SQUAD 1.1, 2.0, MNLI, SST-2, RACE)的性能。

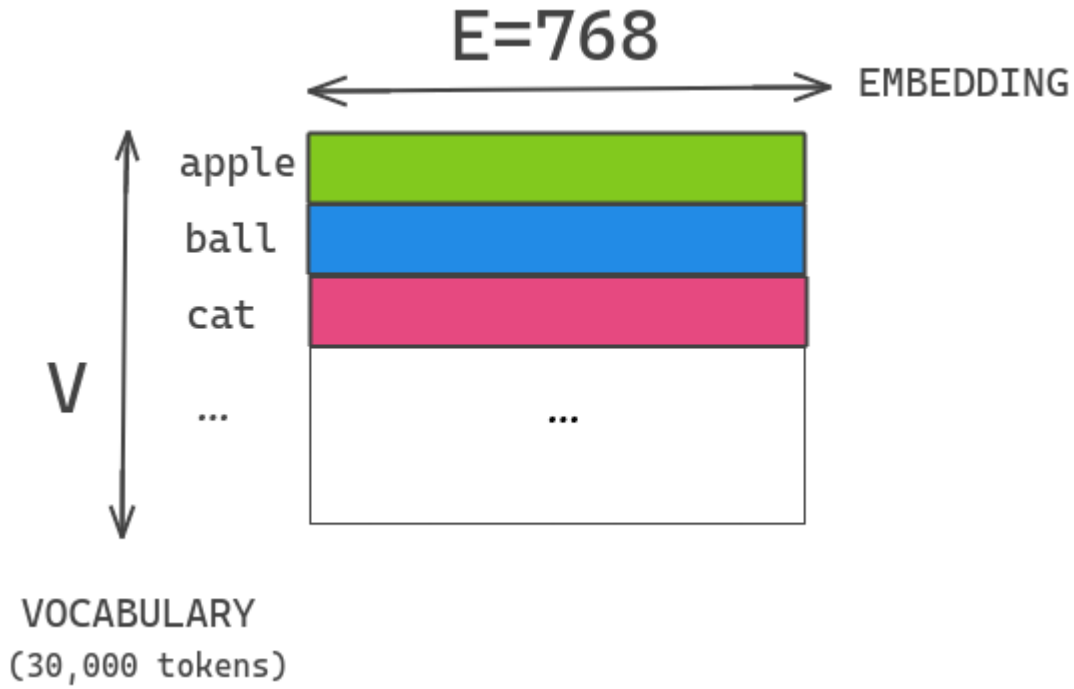
| | Intrinsic Tasks | | | Downstream Tasks | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| SP tasks | MLM | NSP | SOP | SQuAD1.1 | SQuAD2.0 | MNLI | SST-2 | RACE | Avg |
| None | 54.9 | 52.4 | 53.3 | 88.6/81.5 | 78.1/75.3 | 81.5 | 89.9 | 61.7 | 79.0 |
| NSP | 54.5 | 90.5 | 52.0 | 88.4/81.5 | 77.2/74.6 | 81.6 | **91.1** | 62.3 | 79.2 |
| SOP | 54.0 | 78.9 | 86.5 | **89.3/82.3** | **80.0/77.1** | **82.0** | 90.3 | **64.0** | **80.1** |

在这里我们可以看到，在SOP任务上，一个只经过NSP训练的模型给出的分数只比随机猜略好一点(52.0)，这也恰恰证明了上文的猜想：NSP只学习了topic prediction, 完全没有学习到coherence prediction。但是经过SOP训练的模型可以非常有效地解决NSP任务。这就证明SOP能带来更好的学习表现。

3、**嵌入参数分解(对Embedding部分进行矩阵分解)**

ALBERT中使用和BERT大小相近的30K词汇表。假如我们的embedding size和hidden size一样，都是768，那么如果我们想增加了hidden size，就也需要相应的增加embedding size，这会导致embedding table变得很大。

> In BERT, ... the WordPiece embedding size E is tied with the hidden layer size H, i.e. E = H. This decision appears suboptimal for both modeling and practical reasons. -- ALBERT论文




ALBERT通过将大的词汇表embedding矩阵分解成两个小矩阵来解决这个问题。这将**隐藏层的大小与词汇表嵌入的大小分开。**

从模型的角度来讲，因为WordPiece embedding只是要学习一些上下文无关的表示(context-independent representations), 而hidden layer是要学习上下文相关的表示(context-dependent representations). 而BERT类模型的强大之处就在于它能够建模**上下文相关**的表示。所以，理应有 H >> E.

从实用的角度来讲，这允许我们**在不显著增加词汇表embedding的参数大小的情况下增加隐藏的大小。**

我们将one-hot encoding向量投影到 E=100 的低维嵌入空间，然后将这个嵌入空间投影到隐含层空间H=768。其实这也可以理解为：使用E = 100的embedding table，得到每个token的embedding之后再经过一层全连接转化为768维。这样，模型参数量从原来的 $O(VH)$ 降低为现在的 $O(VE+EH)$

## 3. XLNet

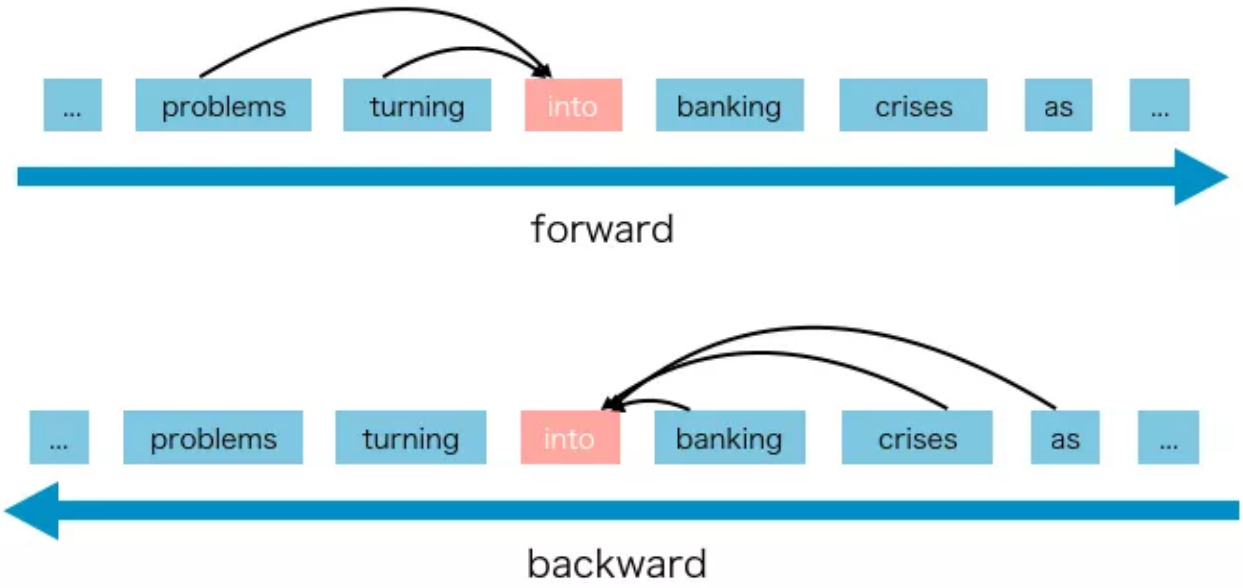
BERT最大的创新就是提出了Masked Language Model作为预训练任务，解决了GPT不能双向编码、ELMo不能深度双向编码的问题。之后很多任务都不再需要复杂的网络结构，也不需要大量的标注数据，业界学术界都基于BERT做了很多事情。

2019年谷歌又发表了模型 XLNet：XLNet: Generalized Autoregressive Pretraining for Language Understanding，找到并解决了BERT的缺点，刷爆了BERT之前的成绩（当然数据、算力相比去年都增加了很多）

3.1 什么是XLNet?

XLNet是一种广义的**自回归(auto-regressive)**预训练方法。

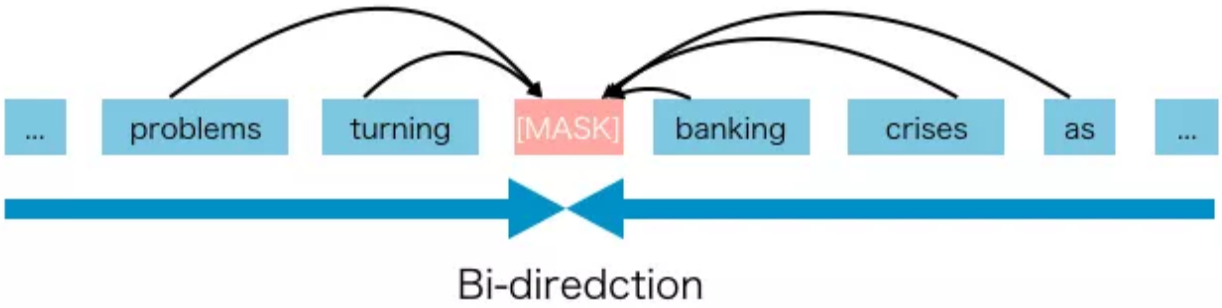
Auto-regressive语言模型是利用上下文单词预测下一个单词的一种模型。但是在这里，上下文单词被限制在两个方向，要么向前，要么向后。




GPT和Elmo都是Auto-regressive。Auto-regressive语言模型的优点是擅长NLP生成任务。因为在生成上下文时，通常是正向的。

但是AR语言模型有一些缺点，它只能使用前向上下文或后向上下文，这意味着它不能同时使用前向上下文和后向上下文。

与AR语言模型不同，BERT被归类为自动编码器(Auto-Encoder)语言模型。AE语言模型的目的是从损坏的输入中重建原始数据。




"损坏的输入"意味着我们使用在训练前阶段将原始的token [into]替换为 [MASK] 。我们的目标是预测出[into]。AE语言模型的优点是它可以在向前和向后两个方向上看到上下文。

但是AE语言模型也有其不足之处。它在预训练中使用了[MASK]，但是这种人为的符号在fine-tuning的时候在实际数据中时没有的，导致了**预训练与fine-tuning的不一致。**[MASK]的另一个缺点是它假设所预测的(mask掉的)token是相互独立的。例如，我们有一句话"It shows that the housing crisis was turned into a banking crisis"。我们盖住了“banking”和“crisis”。注意这里，我们知道，盖住的“banking”与“crisis”之间隐含着相互关联。但AE模型是利用那些没有盖住的tokens试图预测“banking”，并独立利用那些没有盖住的tokens预测“crisis”。它忽视了“banking”与“crisis”之间的关系。换句话说，它假设预测的(屏蔽的)tokens是相互独立的。但是我们知道模型应该学习(屏蔽的)tokens之间的这种相关性来预测其中的一个token。

作者想要强调的是，XLNet提出了一种新的方法，让AR语言模型从双向的上下文中学习，**避免了AE语言模型中mask方法带来的弊端。**

3.2 XLNet如何工作?

AR语言模型只能使用前向或后向的上下文，如何让它学习双向上下文呢？语言模型由预训练阶段和调优阶段两个阶段组成。XLNet专注于预训练阶段。在预训练阶段，它提出了一个新的目标，称为重排列语言建模。我们可以从这个名字知道基本的思想，它使用**重排列**。

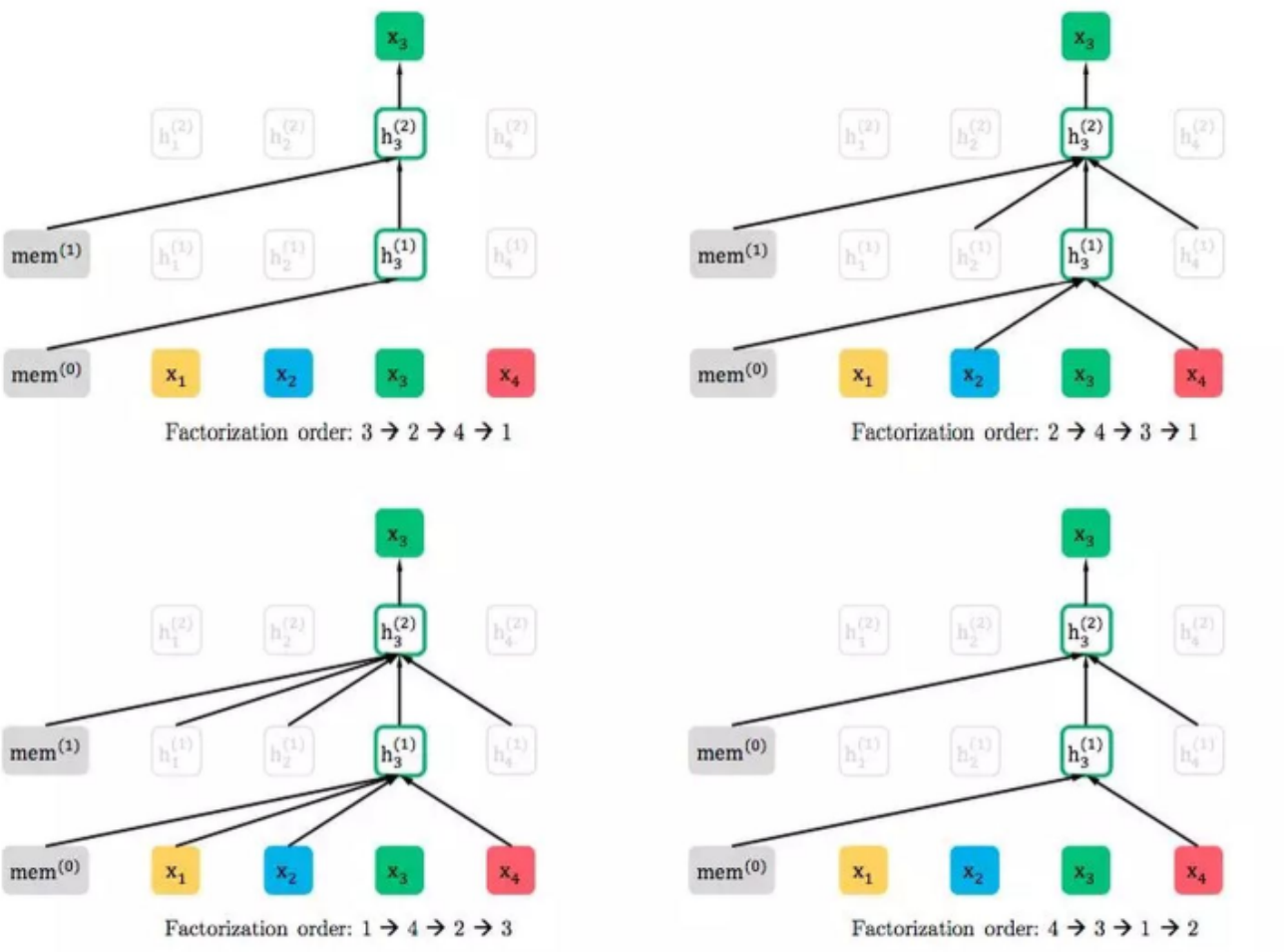
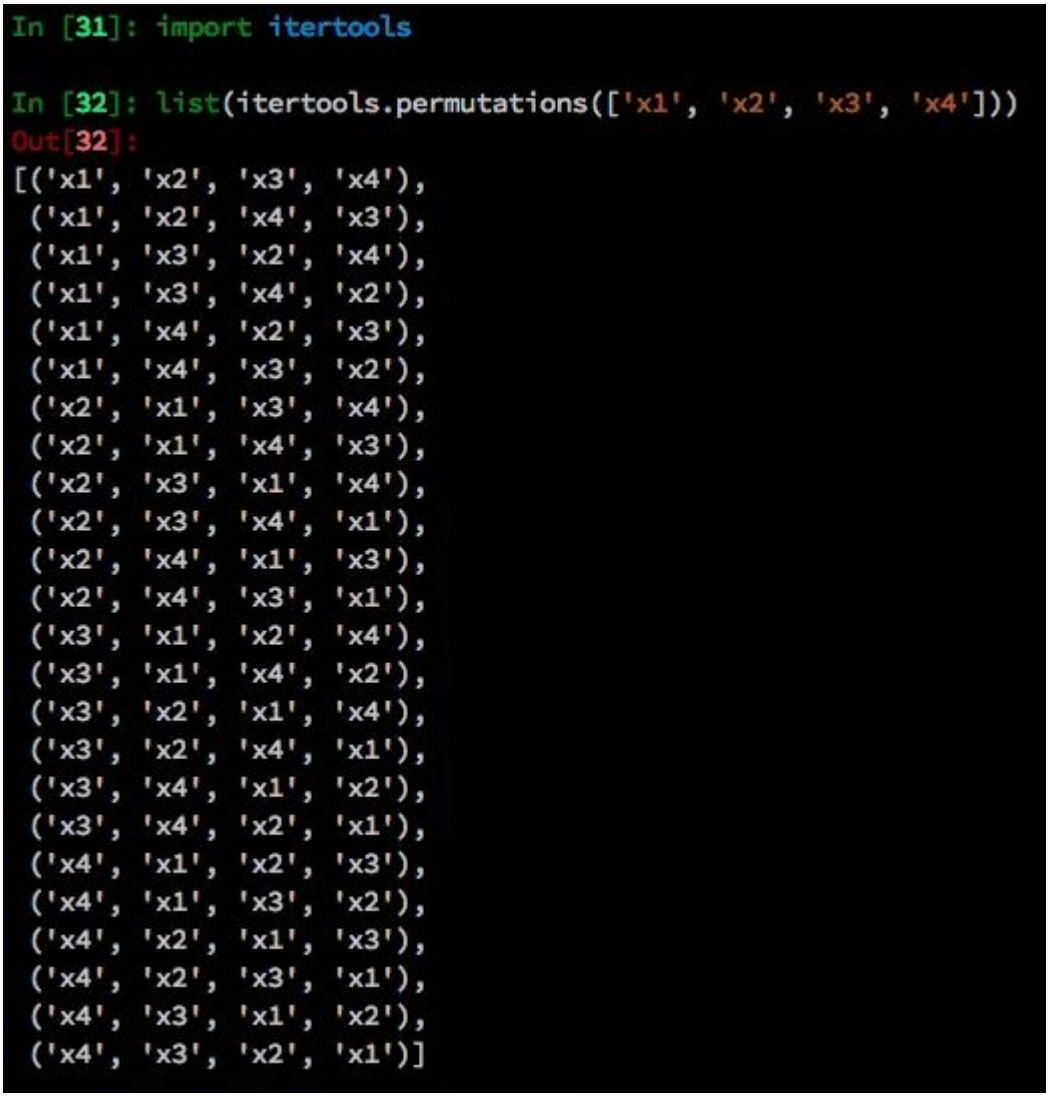


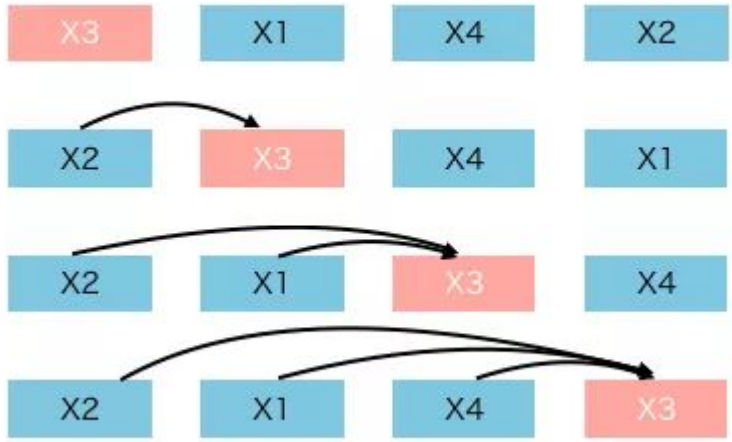

Figure 1: Illustration of the permutation language modeling objective for predicting $x_3$ given the same input sequence $\mathbf{x}$ but with different factorization orders.

这里我们用一个例子来解释。序列顺序是[x1, x2, x3, x4]。该序列的所有排列如下。




对于这4个tokens (N)的句子，有24个(N!)个排列。假设我们想要预测x3。24个排列中有4种模式，x3在第1位，第2位，第3位，第4位。

```
[x3, xx, xx, xx]
[xx, x3, xx, xx]
[xx, xx, x3, xx]
[xx, xx, xx, x3]
```




在这里，我们将x3的位置设为第t位，它前面的t-1个tokens用来预测x3。

x3之前的单词包含序列中所有可能的单词和长度。直观地，模型将学习从两边的所有位置收集信息。

具体实现要比上面的解释复杂得多，这里就不讨论了。但是你应该对XLNet有最基本和最重要的了解。