大语言模型：通往通用人工智能之路

在开始讲解大语言模型前，我们还需要补充一点基础的机器学习的相关知识。

若干重要的机器学习概念

机器学习非常擅长以下三个任务：

- 回归(regression)
- 分类(classification)
- 聚类(clustering)

回归

简单的说，回归就是处理**连续数据**，如**时间序列数据**时使用的技术。
例子：过去几天的股价数据，如下表所示：

日期	股价
昨天	￥1000
2天前	￥1100
3天前	￥1070

从这样的数据中学习它的趋势，求出“明天的股价会变成多少？”的方法就是**回归**。

分类

检查邮件的内容，然后判断它是不是垃圾邮件，就是一个典型的分类问题。

例子：根据邮件内容，以及这封邮件是否属于垃圾邮件这些数据来进行学习。

邮件内容	是否为垃圾邮件
辛苦啦！下个周日我们去玩吧.....	No
加我为好友吧。这里有我的照片哟！http://...	Yes
恭喜您赢得夏威夷旅游大奖...	Yes

在开始学习之前，我们必须像上述这张表这样，先手动标记邮件是否为垃圾邮件。这样打标签的工作，需要人工介入。

聚类

聚类(也称“多分类”)与分类相似，又有不同。
例子：假设在100名学生的学校进行摸底考试，然后根据考试成绩把100名学生分成几组，根据分组结果，我们能得出某组偏重理科、某组偏重文科这样有意义的结论。

学生编号	英语分数	数学分数	语文分数	物理分数
A-1	100	98	89	96
A-2	77	98	69	98
A-3	99	56	99	61

Tips: 分类（Classification）和聚类（Clustering）的区别

分类是一种**监督学习**任务，其目标是预测给定输入数据的标签。在分类任务中，我们有一个标记的训练集，其中每个样本都有一个预定义的类别。我们的目标是训练一个模型，该模型能够根据输入数据的特征预测其类别。例如，我们可能有一个电子邮件数据集，我们的任务是预测每封电子邮件是否为垃圾邮件。

相比之下，**聚类**是一种**无监督学习**任务，其目标是发现输入数据中的自然分组，而无需任何预先定义的标签。在聚类任务中，我们的目标是根据输入数据的相似性将其分组。例如，我们可能有一个博客文章的数据集，我们的任务是将相似的文章分组，以便我们可以更容易地找到相关的内容。

总的来说，分类和聚类的主要区别在于是否有预定义的标签。分类任务需要预定义的标签，而聚类任务则不需要。

重新思考回归和分类问题

在机器学习中，回归问题和分类问题是两种主要的任务类型，它们各自解决不同的问题。回归问题是预测一个连续的输出值，例如预测股价。而分类问题是预测一个离散的输出值，例如判断一封电子邮件是否为垃圾邮件。
Tips: 这里，我们把二分类问题和多分类问题，统称为分类问题。
既然是处理连续和离散的问题，就是一个非常经典的数学问题。如果在丢失一些精度的情况下，连续问题是可以转化为离散的问题来处理的。上世纪70到80年代，轰轰烈烈的模拟电路往数字电路转变，背后的数学原理就是拉普拉斯变换。而我们在处理NLP的时候，用了几乎一样的思路。通常来说，在NLP领域的很多场景中模型最后所做的基本上都是一个分类任务，虽然表面上看起来不是。例如：

- 文本蕴含任务: 其实就是将两个序列拼接在一起，然后预测其所属的类别；
- 基于神经网络的序列生成模型（翻译、文本生成等）本质: 就是预测词表中下一个最有可能出现的词，此时的分类类别就是词表的大小。

> 通常来说，我们把几乎所有问题都最终转变成**分类问题。**即，在NLP邻域的许多场景中模型最后所做的基本上都是一个**分类任务。**

机器学习种类

机器学习的种类繁多，有各种维度的分类。本文大致会涉及以下几种，先给出简介。后文还有更详细的讨论。

1. **监督学习**：提供给算法的包含所需解决方案的训练数据，成为标签或标记。
2. **非监督学习**：训练数据都是未经标记的，算法会在没有指导的情况下自动学习。
3. **半监督学习**：有些算法可以处理部分标记的训练数据，通常是大量未标记的数据和少量标记的数据。
4. **自监督学习**：通过输入数据的某种形式自动生成标签，然后使用这些标签进行学习。这使得模型能够从大量的未标记数据中学习，同时也能利用监督学习的优点。它可以被看作是无监督学习和监督学习的结合。
5. **强化学习**：学习系统能够观测环境，做出选择，执行操作并获得回报，或者是以负面回报的形式获得惩罚。

通用人工智能

> 计算机科学是应用数学最强的分支。既然是应用类学科，学习起来就是有通用的套路。一言以蔽之，提出问题，然后解决问题。

在《大语言模型：革命前夜》里，我们已经介绍了Transformer模型，而时间节点也来到了2017年。我们站在2017年这个时间节点，看看人工智能解决了哪些问题，又有哪些问题亟待解决。

传统的人工智能在许多领域都取得了显著的成就：

1. **自然语言处理**：在自然语言处理方面取得了显著的进步，包括机器翻译、情感分析、文本生成等。
2. **计算机视觉**：在图像识别、目标检测、人脸识别等领域，传统人工智能已经达到了很高的精度。
3. **推荐系统**：在推荐系统中也发挥了重要作用，如电影推荐、音乐推荐等。
4. **游戏**：在游戏领域也取得了一些重要的成就，如围棋AI AlphaGo击败了世界冠军。

然而，传统人工智能也存在一些问题亟待解决：

1. **泛化能力**：传统人工智能通常只能在特定任务上表现良好，但在面对新的、未知的任务时，其性能可能会大打折扣。
2. **解释性**：许多传统人工智能模型，尤其是深度学习模型，其决策过程往往难以理解和解释。
3. **数据依赖**：传统人工智能通常需要大量的标注数据进行训练，这在许多情况下是不现实的。

总的来说，尽管传统人工智能在许多领域都取得了显著的成就，但它仍然面临着一些重要的挑战。这一阶段的传统人工智能，我们称之为“弱人工智能”（也称“窄人工智能”）。针对遇到的上述问题，大家自然而然地把目标设立在建立一个强人工智能，业界称之为通用人工智能（AGI）。理想的通用人工智能具备以下几个主要优点：

1. **广泛的应用领域**：AGI能够处理各种类型的任务，而不仅仅是特定的、预先定义的任务。这使得AGI可以在各种不同的领域中发挥作用，包括医疗、教育、科研和创意等。
2. **自我学习和适应能力**：AGI具有自我学习和适应新环境的能力。这意味着AGI可以通过经验学习，不断改进自己的性能，并适应各种未知的情况。
3. **独立处理任务**：与弱AI不同，AGI可以独立地处理各种任务。这使得AGI在处理复杂问题时具有更大的灵活性。
4. **创新和发现**：如果AGI能够实现，它可能会带来新的科学发现和技术进步。AGI的能力可能超越人类的知识和经验，从而在各种领域产生创新。
5. **高效和快速**：AGI有可能实现更高效和快速的业务处理，提供更高级别的决策支持。

总的来说，AGI的主要优点在于其广泛的应用领域、自我学习和适应能力、独立处理任务的能力，以及其潜在的创新和发现能力。然而，值得注意的是，尽管AGI具有巨大的潜力，但它仍然是一个正在研究和开发的领域，目前还没有完全实现。站在2023年这个时间节点，大家公认大语言模型（LLM: Large Language Model）是通往AGI之路的利器。

再度聚焦NLP

上文提到，传统人工智能在NLP，计算机视觉(CV)，推荐系统，游戏方面取得了很多显著的成就。事实上，NLP相对于其他三个方面，在产品成熟度来说，还存在较大的差距。但是解决NLP的重要性，缺是其他方面无法企及的。人类历史有 600 万年，但是人类文明历史大概就 6000 年，文明的大发展出现在近 2000 多年的原因，主要来自 3500 多年前人类发明了文字。所以，让AI理解并生成文字，这意味着AI可以用人类熟悉的方式（语言文字）与人类高效协作，这必将引爆生产力革命，并深远的影响几乎所有的领域。

提出问题

由于NLP领域要面对的问题非常复杂，人类在遇到庞大且复杂问题时，自然而然地会想到把一个大问题分解若干小问题。所以，一开始NLP就有很多细分领域，包括但不限于

- 命名实体识别（Named Entity Recognition，NER）
- 文本蕴含（Text Entailment）
- 常识推理（Common Sense Reasoning）
- 问答（Question Answering）
- 词性标注（POS Tagging）
- 情感分析（Sentiment Analysis，SA）
- 自然语言推理（Natural Language Inference，NLI）
- 总结摘要（Summarization）

  .....

看着一长串的术语，相信你和我一样，一种深深的无力感。从审美的角度看，这些解决方法，很多都是很繁琐的中间任务，不够优雅（Elegence）。有没有更优雅的方式来解决这些问题呢?
针对上面例举的NLP任务，大致上可以分为两大类：

1. 自然语言理解（NLU）：让AI理解人类的语言，即让机器会“听”和“读”人类的语言。
2. 自然语言生成（NLG）：让AI用人类的语言表达，即让计算机会“说”和“写”人类的语言。

笔者所在的软件行业，都是采用软件工程的方法来开发软件的。软件工程有很多方法论，比如现在一般用敏捷(Agile)开发流程，如果你入行有段时间了，你会听过很多方法论，比如RUP，CMM/CMMI等。但无论是谁，一定听过**瀑布模型**这一方法论。瀑布模型几乎影响了所有后续软件工程方法论，导致所有方法论的介绍章节第一章的标题就是“传统瀑布模型的缺点”。而在基于神经网络的机器学习方法论里，和**瀑布模型**同等地位的就是**监督学习**。
Tips: 我没有任何贬低监督学习的意思，相反在我眼里，瀑布模型的地位是崇高的，几乎所有获得巨大成功的软件都出自瀑布模型。我估计我是全网唯一吹瀑布模型的人。

监督学习，特别是在CV邻域，获得了巨大成功的。但是在解决NLP过程中，遇到了下面两个痛点：

- **痛点一：数据标注困难**：NLP的数据标注工作量非常大，相比CV的标注，对数据和人员要求都高一些。
- **痛点二：模型泛化性差**：只能针对特定任务训练，导致训练好的模型只能针对特定任务起作用。

至此，问题提出来了，现在该寻求办法来解决上面这两个痛点了。

解决问题

正是应了那句话：

> 天才的设计，来自于剽窃!

针对**痛点一：数据标注困难**问题，大家第一个想到Copy解决CV问题时的思路，做一个通用的标注好了的语言库，类似于ImageNet的东西。ImageNet是包含了超过1400万张手动标注图像、涵盖 2 万多个类别的大型图像数据库，正是由于ImageNet的出现，帮助AI顺利的解决了CV邻域绝大多数的问题。曾经有很多人花费了多精力朝这个方向努力，可惜的是，直到2023年，虽然产生了一些比较优秀的数据集，但NLP的ImageNet时刻始终没有到来。

既然此路不通，再换条路。能否设计一套免标注的训练方法，或者少量标注训练数据的方法。所幸的是，少标注训练数据这条路走通了，发展出了**自监督学习**。后文会详细讲解，这里先按下不表。

针对**痛点二：模型泛化性差**问题, 我先卖个关子。其实，泛化性差这是一个各个领域通用的问题。再思考一下这句话“天才的设计，来自于剽窃！”。

> 解决问题的思路就转变成：找到相似的问题，然后借鉴该相似问题的解决思路，最后把该解决思路应用于原问题上来。

其实这就是应用科学一个通用的解决问题方法。现实的例子很多，我这里给出一个软件开发过程中一个常见的例子——代码重用。希望你能用这种思维方式来帮助解决你自己在工作学习中遇到的问题。

初始阶段（Phrase#0）：程序员张三为项目A开发了一个User service。虽然user svc出色的完成了任务，但该User Svc泛化性差，只能专门服务于项目A。

- **Phrase#1** - 过了一段时间，项目B，有了一个类似的需求需要开发User Service。基于User Service的口碑，项目B的负责人决定复用（reuse）项目A的User Service。
    - 步骤1：张三移除了原先User Service专门服务于项目A的代码，加入了拓展机制，使其能满足各个不同项目的需求。
    - 步骤2：项目组B的程序员李四，复用了新的User svc，并基于其拓展机制，实现了项目B的特殊需求，并部署在项目B中。这时，项目A和项目B里面各自都有一个User svc。
- **Phrase#2** - 又过了一段时间，项目C/D/E/...都有了类似的需求，各个项目基于User svc在项目A/B中的良好的口碑，都想用这个User Svc。
    - 张三所在的公司决定把这个User Svc做成一个Common svc，部署在公司统一平台里。这时，全公司就一个User svc，供所有项目统一使用。

看完例子，我们回到**痛点二：模型泛化性差**问题，我们其实遇到和上面例子中初始阶段（Phrase#0）相似的问题。现在相似的问题找到了，我们研究一下这个问题的解决办法可不可以借鉴。
借鉴Phrase#1中的两步骤的解决方法，机器学习把学习过程分成了两个阶段：

- 第一阶段叫**预训练（Pre-train）**，解决问题的思路类似Phrase#1-步骤1
- 第二阶段叫**微调（Fine-tune）**，解决问题的思路类似Phrase#1-步骤2

这就是鼎鼎大名的两阶段训练法，正式名称叫**语言建模预训练-微调（Pre-train LM and Fine-tune）**。而第一阶段的产物，称之为**预训练模型（PTM, Pre-Train Model）**.

借鉴Phrase#2的解决问题的思路，发展出了**上下文学习(ICL，In-Context Learning)**。这是更加牛逼的技术，我们后面还会遇到，届时再详细介绍。

迁移学习（Transfer Learning）

回到上面提到的例子，还有一个不起眼的一个动作——复用（Reuse），在机器学习中，对应的概念就是**迁移学习**。在深度学习中，预训练又是迁移学习的主要方法，而迁移学习正是大语言模型诞生的两大基石之一。

我们这里不介绍迁移学习的晦涩的定义。让我们直接以图像识别为例来说明迁移学习。

假设你已经有一个在大量的猫和狗图片上训练过的深度学习模型，这个模型已经学会了识别猫和狗的特征。现在，你想要训练一个新的模型来识别狮子和老虎，但是你只有很少量的狮子和老虎的图片。

在这种情况下，你可以使用迁移学习。你可以将原来识别猫和狗的模型的一部分（通常是前面的层）复制过来，作为新模型的一部分。这样，新模型就可以利用原模型学到的特征，比如边缘、纹理、形状等，来帮助识别狮子和老虎。这样，即使你只有少量的狮子和老虎的图片，也可以训练出一个性能不错的模型。

这个和代码reuse像不像，毕竟太阳底下没有新鲜事。

大规模化（Scaling）

既然提到了两大基石之一，我们就介绍一下另一大基石——大规模化。
大规模化需要三个要素：

1. 计算机硬件算力的突飞猛进。过去4年，增加了10倍以上。
2. Transformer模型的诞生。该模型利用硬件的并行性来训练比以前更具表现力的模型。
3. 更多训练数据的可用性

随着这三大要素的日益精进，让大规模化成为了可能。加之，迁移学习相关技术的日益完善，人工智能开启了大语言模型的时代。

大语言模型

语言模型（Language Model，LM）是自然语言处理领域的核心问题，它的任务是预测一个句子在语言中出现的概率。语言模型起源于语音识别，如今已经扩展到机器翻译、信息检索、问答、文摘等众多NLP领域。

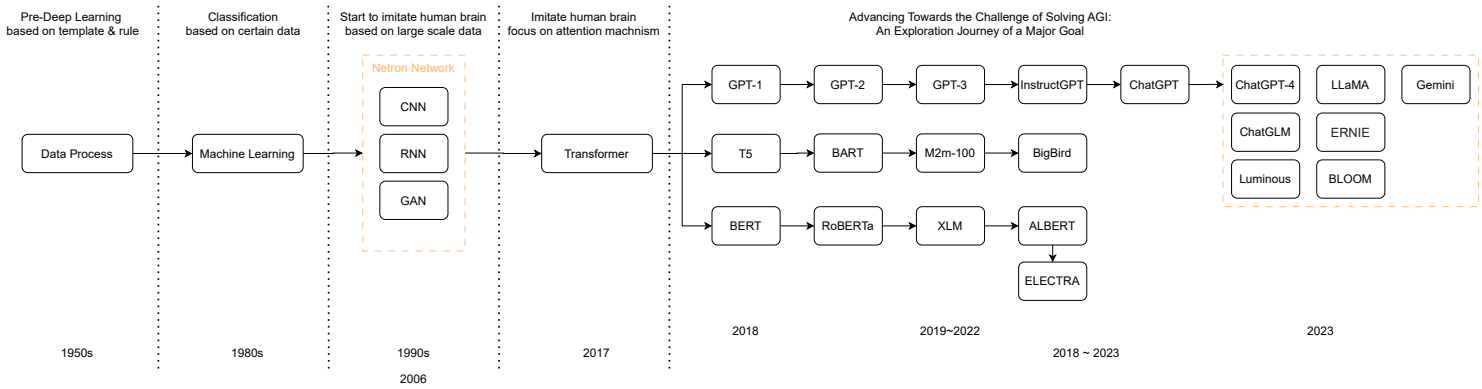
简单来说，语言模型是这样一个模型：对于任意的词序列，它能够计算出这个序列是一句话的概率。例如，词序列A：“张三|的|文章|真|水|啊”，这个明显是一句话，一个好的语言模型也会给出很高的概率，再看词序列B：“张三|的|睡觉|啤酒|好快”，这明显不是一句话，如果语言模型训练的好，那么序列B的概率就很小很小。

更为正式的定义是，假设我们要为中文创建一个语言模型，V表示词典，V={猫,狗,机器,学习,语言,模型,...}，$w_i \in V$ 。语言模型就是这样一个模型：给定词典V，能够计算出任意单词序列 $w_1, w_2, ..., w_n$ 是一句话的概率 $p(w_1, w_2, ..., w_n)$ ，其中，$p \geq 0$。

语言模型的发展先后经历了文法规则语言模型、统计语言模型、神经网络语言模型。其中，统计语言模型中最常用的是n-gram model，它引入了马尔可夫假设：当前词的出现概率仅与前 n-1个词有关。神经网络语言模型则基于深度学习架构，如转化器，这有助于它们在各种NLP任务上取得令人印象深刻的表现。

我们先一起来回顾一下人工智能的发展史。

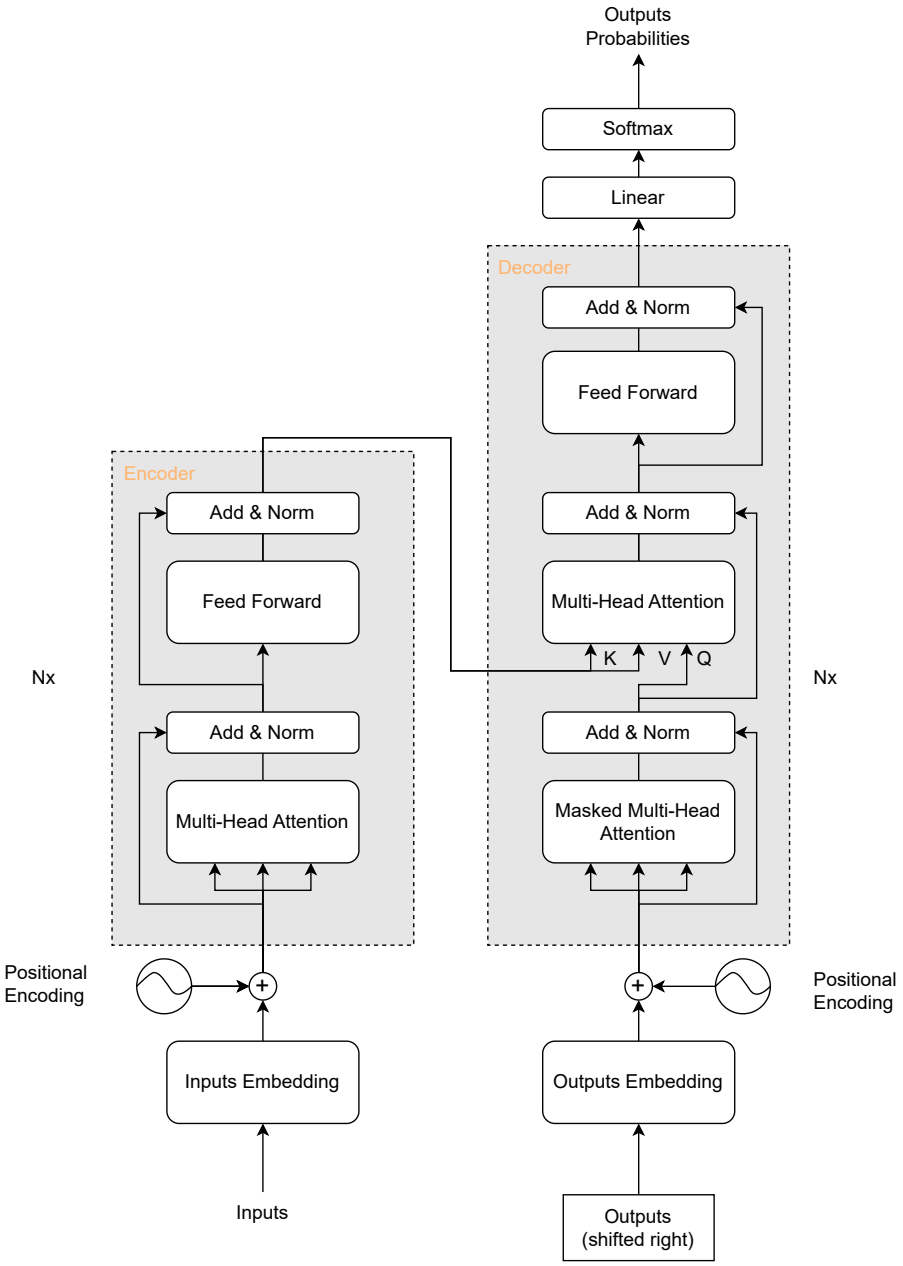
人工智能发展史



- 从上世纪50年代到80年代，我们称之为“前深度学习时代”，随着计算机的诞生和发展，人工智能主要利用各类模板或者规则来解决数据处理的问题。
- 80年代，诞生了机器学习。那时候的人工智能主要是基于一定量的数据，处理分类问题（模式识别）。
- 90年代起，AI迎来了飞速的发展。值得一提的是2006年，神经网络技术获得突破，AI/ML成为热门，热度一直延续至今
- 2017年，诞生了Transformer模型，成为至今最好的特征提取器。
- 2018~2023年，业界在使用Transformer模型过程中，沿着三条路径展开了探索，诞生了很多大语言模型。最终ChatGPT发布，并在2023年初成功“破圈”，进入公众视野。迎来“AIGC”时代。

2018，Transformer模型线路之争

从人工智能发展史，我们可以看到，业界在2018年，针对Transformer模型，产生了三条不同的发展路径。让我们再度聚焦Transformer模型，看看业界是如何把Transfomer模型玩出花的。



总所周知，在Transformer模型的左面是Encoder，右边是Decoder。发生在2018年的三条线路之争，对的应产品就是：

1. Encoder：OpenAI的GPT-1
2. Encoder+Decoder：Google的T5
3. Decoder：Google的BERT

现实发生的故事很精彩，我们讲重点介绍BERT和GPT系列。

Decoder线路的集大成者：BERT

BERT简介

BERT（Bidirectional Encoder Representations from Transformers）是一种用于自然语言处理的预训练技术，由Google提出。强调了不再像以往一样采用传统的单向语言模型或者把两个单向语言模型进行浅层拼接的方法进行预训练，而是采用新的masked language model（MLM），以致能生成深度的双向语言表征。BERT模型的预训练可以使用较少的资源和较小的数据集在下游任务上进行微调，以改进在这些任务上的效能。这使得BERT模型的适用性更广，并且不需要做更多重复性的模型训练工作。

BERT 模型可以作为公认的里程碑式的模型，但是它最大的优点不是创新，而是集大成者，具体如下：

- BERT采用两段式训练方法：第一阶段，使用大规模无标签语料，训练出基础模型。第二阶段，使用少量带标签数据微调。

- 参考了ELMO模型的双向编码思想，提出Bidirectional双向编码。
- 借鉴了GPT用Transformer作为特征提取器的思路，采用了Transformer中的Encoder编码器。
- 采用了类似word2vec所使用的CBOW方法，提出MLM。

Tips：本文没有讲解CBOW，请读者自行查阅。思路类似于英语考试题型“完型填空”。

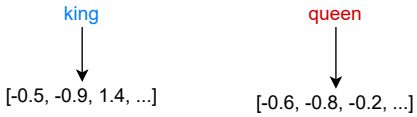
学习BERT其实一点不难，就两部分有点改进工作，一个改进在名字里，即Bidirectional，另一个改进在架构图里，即在Input Embedding处。

改进一：双向（Bidirectional）语言表征

什么是双向（Bidirectional）语言表征，为什么BERT要用这个方法？老套路，提出问题，然后解决问题。

提出问题

上一篇我们介绍了词嵌入（Word Embedding）方法在Transformer模型中的应用。我们来看看在NLP预训练过程中，遇到了什么问题。
处理NLP问题时，第一步就是人类语言用数学表达出来，以便于计算机处理。我们利用词嵌入先把词表示成向量，例子如下：



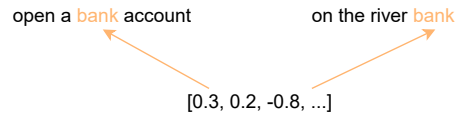
无论是word2vec 还是GloVe。这些词嵌入（Word Embeddings）方法都是在语料库中词之间的共现（co-occurrence）统计进行预训练的。如果两个不同词在两句话，上下文相似，当我们用向量表示king时，这两个词的距离相近。
Tips：请参考附录关于向量余弦章节。这两个词距离相近，用数学表达，就是余弦夹角小，即余弦值近1。

这种基于词共现统计的方法，在一些上下文相识句子中，会得到一模一样的词向量。在下面这个例子里，king和queen的词向量是一致的。

这个问题，尽管这被GloVe方法考虑更大范围词频共现后一定程度解决了（没有完全解决）。然而，另一个致命问题来了，一词多义。

问题1：如何解决“一词多义”问题

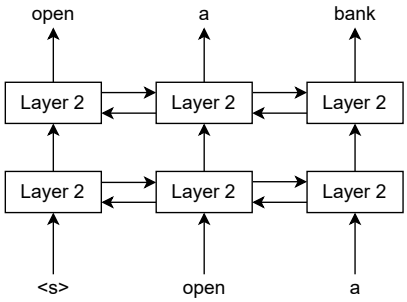
请看下面这两句，用WordEmbedding的方法，两个bank只能表示成一组相同的向量，无法处理“一词多义”这种情况。



解决这个问题的大致思路，我估计大家都能想到，就是把bank表示成多个向量。这样的词表示法有个专门术语：CWRs(Contextual Word Representations), 截止到2023年，这种解决思路是业界主流。

问题2：如何避免“See themselves”问题

在讲解本系列文章的时候，我举得都是相对简单基础的模型。实际上，业界还是涌现出很多改进的模型。比如基于RNN/LSTM的双向模型，而且这些双向模型效果比基础模型要好的。这样在预训练阶段，就带来了“see themselves”的问题，如下图所示：



具体的说，在双向模型中，单词可以“看到自己(see themselves)”的。这是因为双向模型在处理每个单词时，会同时考虑该单词前后的上下文信息。然而，这种方式可能会导致一种情况，即模型在预测某个单词时，实际上已经“看到”了这个单词，从而影响了模型的预测能力。
解决这个问题，大致思路有两个。

1. 从根本上解决：去掉双向模型，改为单向模型。
2. 另一种办法：在预训练阶段，人为用Mask遮住一些词。类似于“Masked MultiHeadsAttention”思路。

方法二，听起来更像一种Workaround，但是结合上下文，却更加符合人类理解语言的方式。所以，上面两种方法孰优孰劣在2017~2018年这个时间节点，还真不好说。

至此，问题提出来了，即解决“一词多义”问题，并同时避免“see themselves”问题。

解决问题

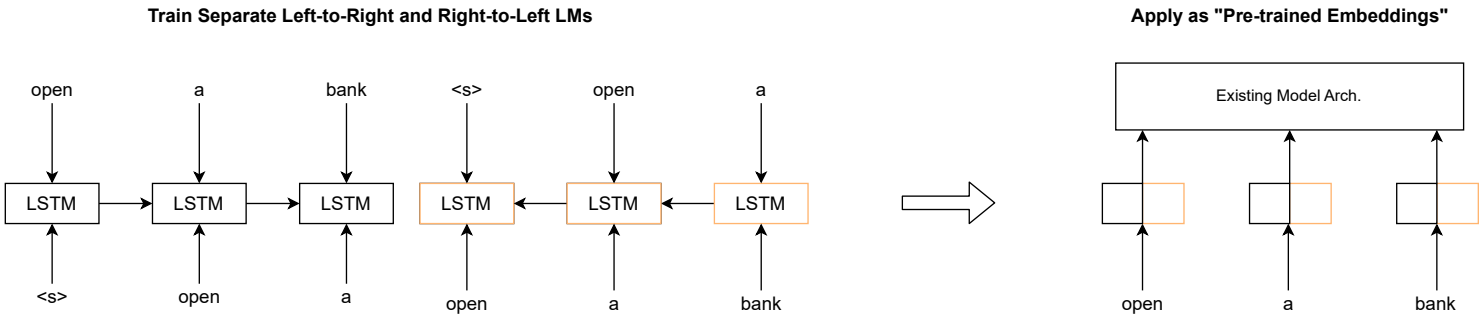
解决“一词多义”问题，并同时避免“see themselves”问题，业界还是花了不少精力的。我们一起来考古一下，看看别人是怎么解决这个问题的。

方案1：半监督学习（2015）

第一种解决方案，其实没什么好讲的，叫半监督学习。其实就是在Fine-tune阶段，人为的把这些问题标注出来。人们用这个办法在2015年，是基于LSTM的。
详见Semi-Supervised Sequence Learning, Google, 2015

方案2：ELMo（2018）

这个需要提一下，ELMO是2018年度NAACL最佳论文，解决的办法也很巧妙，可惜的是还是用了LSTM模型，没有用到最新的Transformer模型。
解决一词多义，用的是CWRs。ELMo用了BiLSTM。在预训练阶段，对于任何一个输入文本，用一个”从左到右“的单向LSTM模型和一个”从右到左“的单向LSTM模型，从而避免了“see themselves”问题，如下图所示：



这样每个词的词向量都有了上下文的信息。这样用ELMo得到的文本向量就可以作为其他模型的预训练嵌入（Pre-trained Embeddings）了。具体步骤就不详细展开了，大家专注于理解思路即可。如果大家对ELMo有兴趣，请自行在网上搜索。

方案3：GPT-1 (2018)

这个就是大名鼎鼎的ChatGPT。用的办法是CWRs和单向从左到右的模型。后面会详细讨论GPT，这里就先按下不表了。

BERT的解决方案：MLM+NSP

语言掩码模型（MLM，Masked Language Modeling）

前文提到BERT借鉴了Word2Vec的类似于“完型填空”的CBOW思路，提出了MLM。一句话，解释MLM方法：就是随机去掉句子中的部分token（单词），然后模型来预测被去掉的token是什么。这样实际上已经不是传统的神经网络语言模型(类似于生成模型)了，而是单纯作为分类问题。

- 分类问题：根据这个时刻的hidden state来预测这个时刻的token应该是什么。
- 生成模型：预测下一个时刻的词的概率分布了。

BERT具体采用的方法是，随机选择15%的tokens出来（这里和CBOW的100%不一样），但是并非把它们全部都Mask掉。
这样设计MLM训练方法会引入一些弊端：就是在第二阶段（Fine tune阶段），输入的文本中将没有Mask，进而导致产生训练和预测数据偏差导致的性能损失。

为了解决上述弊端，MLM又做了下面这些事情。

- 其中的 80% 被替换为 [MASK]，例如：went to the store -> went to the [MASK]
- 其中的 10% 被替换为一个随机token，例如：went to the store -> went to the runing
  - 这样做，是为了让BERT学会根据上下文信息纠错
- 剩余的 10% 不变，例如：went to the store ->went to the store
  - 这样做，是为了缓解训练和预测数据偏差导致的性能损失

这个80-10-10的组合是怎么定出来的呢？答案是Google团队尝试了不同的组合，结果80-10-10效果最好。

下句预测（NSP，Next Sentence Prediction）

在很多自然语言处理的下游任务中，如问答和自然语言推断，都基于两个句子做逻辑推理，而语言模型并不具备直接捕获句子之间的语义联系的能力，或者可以说成单词预测粒度的训练到不了句子关系这个层级，为了学会捕捉句子之间的语义联系，BERT采用了下句预测（NSP）作为无监督预训练的一部分。

因此BERT想用预训练阶段的NSP任务来解决这个痛点。NSP预训练任务所准备的数据，是从单一语种的语料库中取出两个句子S_i和S_j，其中50%的情况下B就是实际跟在A后面的句子，50%的情况下B是随机取的。这样语言模型就是在面对一个二元分类问题进行预训练，例如：

```
INPUT: [CLS] the man went to [MASK] store [SEP]
       he bought a gallon [MASK] milk [SEP]
LABEL: IsNext

INPUT: [CLS] the man [MASK] to the store [SEP]
       penguin [MASK] are flight ##less birds [SEP]
LABEL: NotNext
```

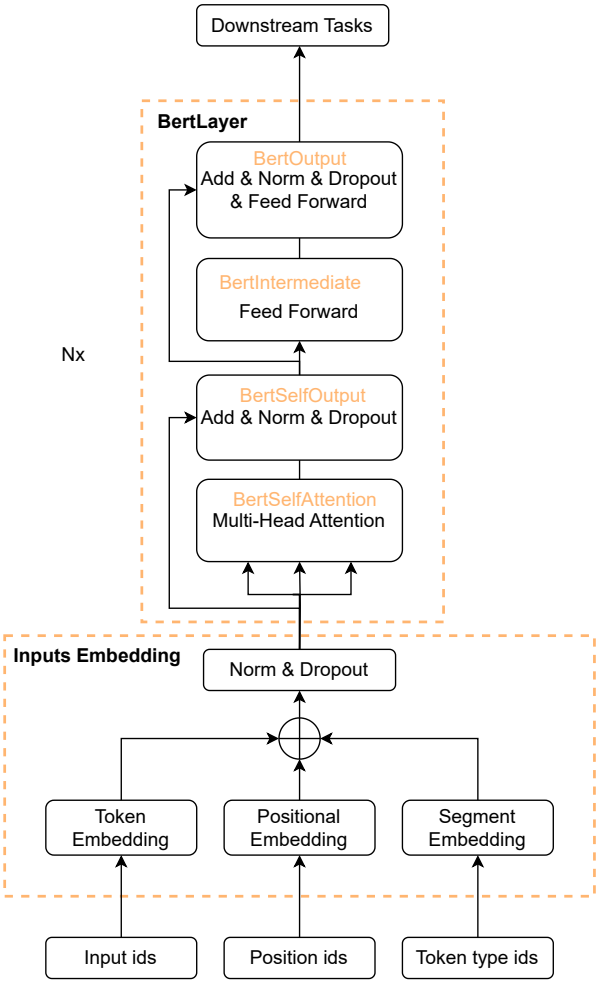
其中，[CLS]表示"classification"，用于类别预测，结果为1，表示输入为连续句对（sentence pair）；结果为0，表示输入为随机句对。
[SEP]表示"separate"，即分隔符，用于断句。这样的预训练任务，让BERT在词维度的语言知识外，也让BERT学习到一些句子维度的语言结构。

Tips: NSP效果一般，后续的BERT改进模型中，都不约而同的弱化或者直接放弃了NSP。

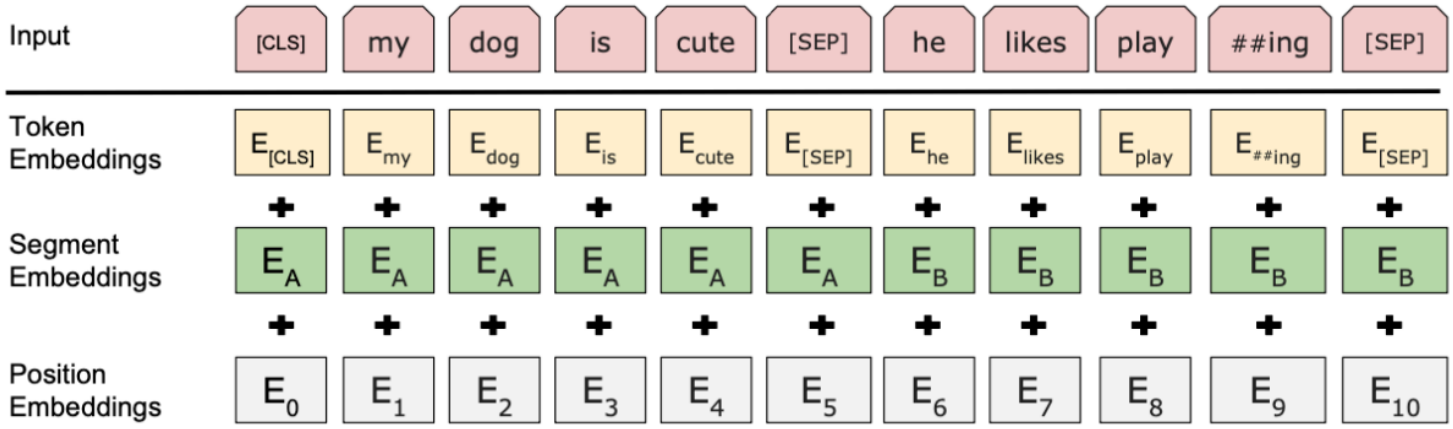
改进二：输入表示（Input Representation）

前文提到BERT的另外一项改进可以从架构图里找到，我们一起来看看BERT的架构图。

BERT架构图



和标准的Transformer Encoder比较，差别比较大的就是Input这部分。我们一起来讨论一下BERT的Input部分。
Input Embedding部分，引用一下原论文的这张图，如图所示：



和大多数NLP深度学习模型一样，BERT将输入文本中的每一个词（token)送入Token Embeddings层从而将每一个词转换成向量形式。在Transformer模型中，有Positional Embeddings和Token Embeddings。BERT又多了一个嵌入层Segment Embeddings。

Inputs Embeddings

1. **Token Embeddings**：Token Embeddings是BERT的一个重要组成部分，它将每个词（token）转换为一个固定维度的向量。在BERT中，每个词会被转换成768维的向量表示。这个过程实际上是建立一个从one-hot编码到768维稠密向量的映射。每个词的词向量最初都是随机生成的，在神经网络训练的过程中，这些词向量会不断优化。
2. **Segment Embeddings**：Segment Embeddings用于帮助BERT区分输入序列中的不同句子。例如，对于一对句子（"I like cats", "I like dogs"），Segment Embeddings层只有两种向量表示。前一个向量是把0赋给第一个句子中的各个token，后一个向量是把1赋给第二个句子中的各个token。如果输入仅仅只有一个句子，那么它的segment embedding就是全0。
3. **Positional Embeddings**：Positional Embeddings用于补充文本输入的时序性信息。BERT能够处理最长512个token的输入序列。论文作者通过让BERT在各个位置上学习一个向量表示来讲序列顺序的信息编码进来。这意味着Position Embeddings实际上就是一个大小为 (512, 768) 的lookup表，表的第一行是代表第一个序列的第一个位置，第二行代表序列的第二个位置，以此类推。

就拿上面这个例子来解释一下上面这3种Embeddings吧。
Input部分为两句话：my dog is cute 和 he likes playing

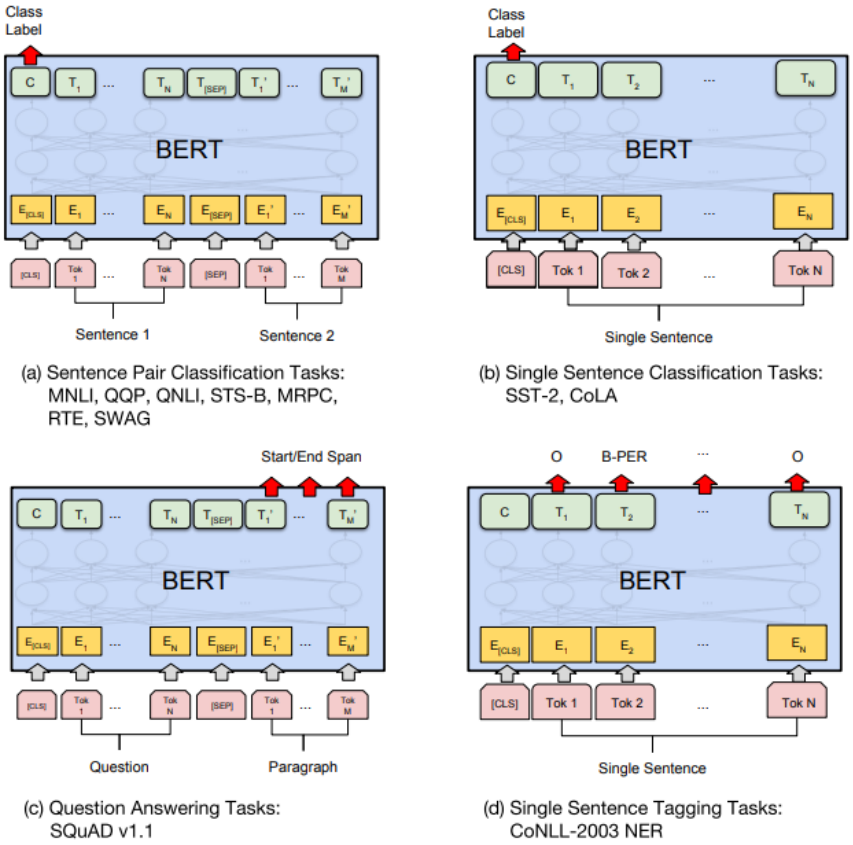
- NLP在处理语言之前，先做分词(Tokenize)，再建立词表，最后转化为Token ids。BERT的分词用的是开源的Workpieces，词表也是直接用的开源vocab.txt。
  - Token Embeddings将两个input被表示为11个Token，即[CLS]my dog is cute[SEP]he likes play ##ing[SEP]
- Segment Embeddings把input1赋值为0（对应图中E_A）；input2赋值为1（对应图中E_B）。
  - 对应Segment Embedding编码为00000011111
- Positional Embeddings直接给这11个token，独一无二的位置编码。

最后把一个把这三种向量相加，形成单一序列。然后经过Layer Norm和Dropout，输出到Encoder Layer。
至此，我们介绍完BERT在Pre-tain阶段的任务了。下一章节，我们来看看如何用BERT来处理第二阶段任务。

微调BERT模型

本文开始阶段就提到NLP最终把所有问题都转化成一个分类问题。BERT也是如此，将微调训练的NLP任务分为四类：

1. 句子对分类任务(Sentence Pair Classification)
2. 单句分类任务(Single Sentence Classification)
3. 文本问答任务(Question Answering Tasks)
4. 单句标注任务(Single Sentence Tagging)



(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG

(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

上图引自BERT原论文，如图所示，前2类是基于句子级别（sentence level）的任务；后两类基于词级别(token level)的任务。我们先一起来看一下涉及到的数据集，如果想从事NLP相关工作，最好还是熟悉一下这些数据集。

- **MNLI(Multi-Genre Natural Language Inference)** 是一个自然语言推断任务的数据集。MNLI数据集是通过众包方式对句子对进行文本蕴含标注的集合。给定前提语句和假设语句，任务是预测前提语句是否包含假设（entailment）、与假设矛盾(contradiction)或者两者都不(中立，neutral)。前提语句是从数十种不同来源收集的，包括转录的语音、小说和政府报告。
- **QQP (Quora Question Pairs)**: 这是一个包含大约40万对问题的数据集，目标是确定一对问题是否具有相同的意思。这个数据集来自Quora，一个用户可以提出问题并得到其他用户的回答的网站。
- **QNLI (Question Natural Language Inference)**: 这个数据集是从斯坦福问答数据集（SQuAD）派生出来的自然语言推理数据集。任务是确定上下文句子是否包含问题的答案。
- **STS-B (Semantic Textual Similarity Benchmark)**: 这个数据集包含了一对句子，这些句子已经被人工标记为在0（没有语义相似性）到5（语义等价）之间的相似性。
- **MRPC (Microsoft Research Paraphrase Corpus)**: 这个数据集包含了一对句子，目标是确定一对句子是否彼此是释义。
- **RTE (Recognizing Textual Entailment)**: 这个数据集包含了一对句子，目标是确定一个句子是否能从另一个句子中推断出来。
- **SWAG（Situations With Adversarial Generations）** 是一个大规模的数据集，用于进行基于常识的推理任务。给定一个情境（一个问题或一句描述），任务是从给定的四个选项中预测最有可能的一个。
- **SST-2 (Stanford Sentiment Treebank)**: 这个数据集包含了电影评论的句子，已经被标记为正面或负面情感。
- **CoLA (Corpus of Linguistic Acceptability)**: 这个数据集包含了一些句子，已经被标记为语法正确或错误。
- **SQuAD（Stanford Question Answering Dataset）**：是一个由斯坦福大学创建的机器阅读理解数据集。SQuAD包含了一系列的英文阅读理解任务，现在做和英文的阅读理解相关所有任务，都用它。SQuAD有两个版本，SQuAD1.0和SQuAD2.0[2]。SQuAD1.0的数据集中，所有的答案都可以在文章中找到[2]。而SQuAD2.0增加了一些没有答案的问题。
- **CoNLL-2003 NER (Named Entity Recognition)**: 这个数据集包含了一些句子，已经被标记为命名实体识别的任务，包括位置（LOC），组织（ORG），人名（PER）和杂项（MISC）。

句子对的分类任务

举个单项选择题的例子，看我们是如何做fine tune的。

本文开头提到NLP问题最终都转成分类问题。四选一的选择题问题就是一个典型的分类任务（四分类问题），而关键的地方就在于如何构建模型的输入和输出。对照上面列出来的数据集，我们也很容易能想到采用SWAG这个数据集。

下面我们展示一下如何准备数据来进行fine tune的具体步骤。
原始数据中有两个样本，需要把多个样本构筑成一个batch，然后交给BERT模型做fine tune。整个过程如下：

```
The people are in robes. They
 A) are wearing colorful costumes. ## Correct
 B) are doing karate moves on the floor.
 C) shake hands on their hips.
 D) do a flip to the bag.


She smirks at someone and rides off. He
 A) similes and falls heavily.
 B) wears a bashful smile. ## Correct
 C) kneels down behind her.
 D) gives him a playful glance.
```

第一步，我们需要重构这个样本，改为：

```
[CLS]The people are in robes. They[SEP]are wearing colorful costumes.[SEP]
[CLS]The people are in robes. They[SEP]are doing karate moves on the floor.[SEP]
[CLS]The people are in robes. They[SEP]shake hands on their hips.[SEP]
[CLS]The people are in robes. They[SEP]do a flip to the bag.[SEP]


[CLS]She smirks at someone and rides off. He[SEP]smiles and falls heavily.[SEP]
[CLS]She smirks at someone and rides off. He[SEP]wears a bashful smile.[SEP]
[CLS]She smirks at someone and rides off. He[SEP]kneels down behind her.[SEP]
[CLS]She smirks at someone and rides off. He[SEP]gives him a playful glance.[SEP]
```

第二步，分词(Tokenize)和填充(Padding)

为了让训练更加高效，还需要把样本对齐，即每句话的Token数一样。用的方法也挺简单的，就是在短句后面加padding，用[PAD]表示。在BERT中，还有两种特殊标记，分别是[UNK]表示Unkown，[MASK]表示遮蔽语言模型中的单词。
Tips：实际过程中，先做分词转换得到token id后，再做padding。为了更好的可读性，本文先讲padding。

```
[CLS]The people are in robes. They[SEP]are wearing colorful costumes.[SEP][PAD][PAD][PAD]
[CLS]The people are in robes. They[SEP]are doing karate moves on the floor.[SEP]
[CLS]The people are in robes. They[SEP]shake hands on their hips.[SEP][PAD][PAD]
[CLS]The people are in robes. They[SEP]do a flip to the bag.[SEP][PAD]


[CLS]She smirks at someone and rides off. He[SEP]smiles and falls heavily.[SEP][PAD]
[CLS]She smirks at someone and rides off. He[SEP]wears a bashful smile.[SEP][PAD]
[CLS]She smirks at someone and rides off. He[SEP]kneels down behind her.[SEP][PAD]
[CLS]She smirks at someone and rides off. He[SEP]gives him a playful glance.[SEP]
```

再结合vocab.txt，把词用Token id表示，结果如下：

```
[101, 1996, 2111,..., 2027, 102, 2024, 4147,...,  102,    0,    0,   0]
[101, 1996, 2111,..., 2027, 102, 2024, 2725,..., 1996, 2723, 1012, 102]
[101, 1996, 2111,..., 2027, 102, 6073, 2398,..., 2006,  102,    0,   0]
[101, 1996, 2111,..., 2027, 102, 2079, 1037,...,  200, 1012,  102,   0]


[101, 2016, 15081, ..., 102,  8451, 1998,  4112, 4600, 1012,  102,   0]
[101, 2016, 15081, ..., 102,  6181, 4877,  2091, 2369, 1012,  102,   0]
[101, 2016, 15081, ..., 102,  3957, 2032,  1037, 6054, 1012,  102,   0]
[101, 2016, 15081, ..., 102, 11651, 1037, 24234, 3993, 2868, 1012, 102]
```

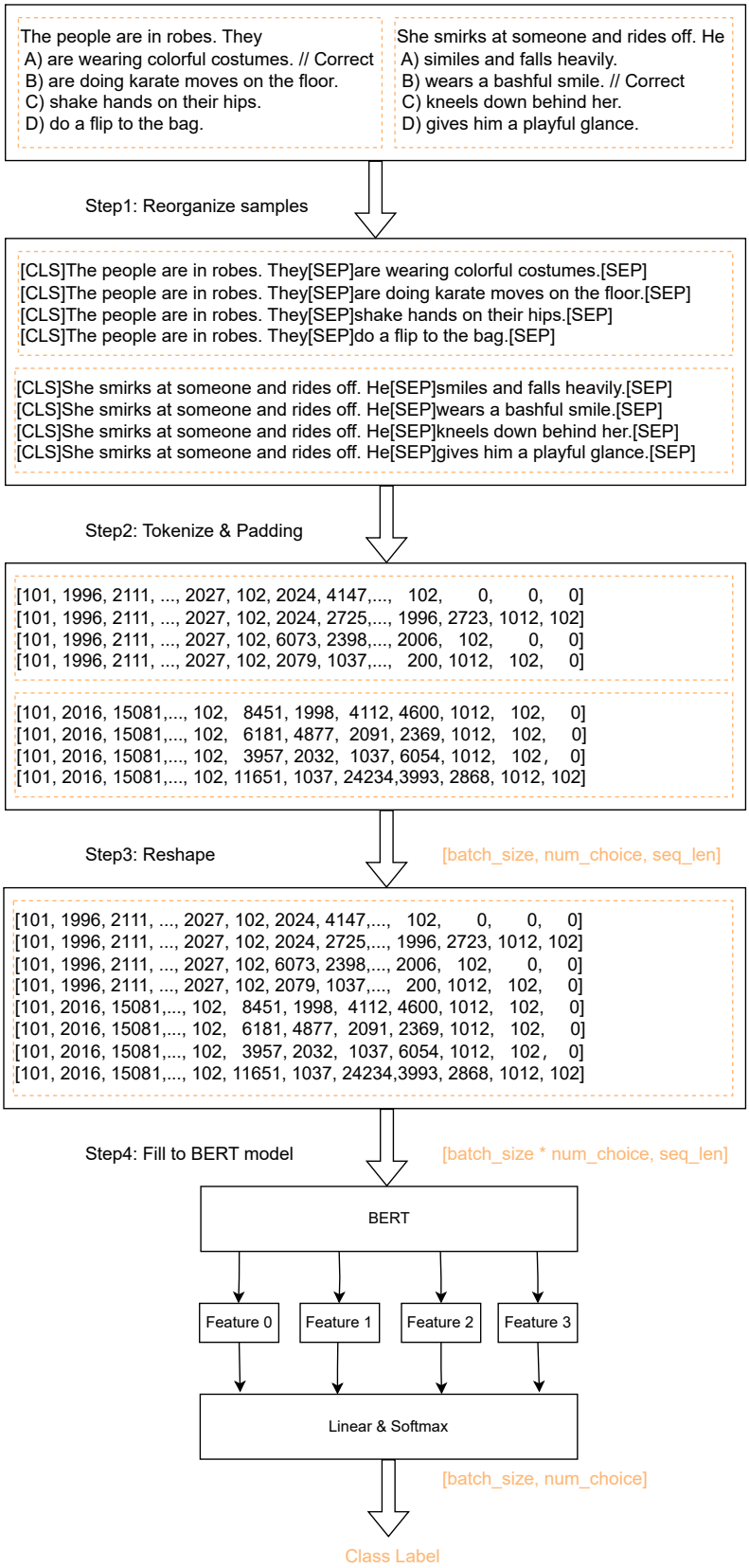
第二步结束后，我们得到一个[batch_size, num_choice, seq_len]的三维张量([2,4,18])

第三步，合并这2个已经对齐了的样本，变成一个batch。这个过程叫Reshape。此时我们得到一个[batch_size * num_choice, seq_len]二维张量([8,18])

```
[101, 1996, 2111,..., 2027, 102, 2024, 4147,...,  102,    0,    0,   0]
[101, 1996, 2111,..., 2027, 102, 2024, 2725,..., 1996, 2723, 1012, 102]
[101, 1996, 2111,..., 2027, 102, 6073, 2398,..., 2006,  102,    0,   0]
[101, 1996, 2111,..., 2027, 102, 2079, 1037,...,  200, 1012,  102,   0]
[101, 2016, 15081, ..., 102,  8451, 1998,  4112, 4600, 1012,  102,   0]
[101, 2016, 15081, ..., 102,  6181, 4877,  2091, 2369, 1012,  102,   0]
```

```
[101, 2016, 15081, ..., 102,  3957, 2032,  1037, 6054, 1012,  102,   0]
[101, 2016, 15081, ..., 102, 11651, 1037, 24234, 3993, 2868, 1012, 102]
```

第四步，输入到BERT模型，进行fune tune。上述四步全过程如下图所示：

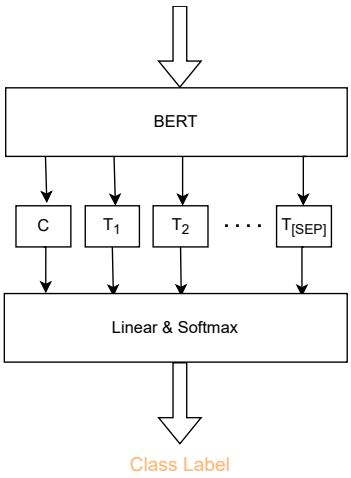


单句的分类任务

BERT的单句分类（Single Sentence Classification）是一种监督学习任务，用于确定给定文本的类别。

在单句分类任务中，BERT模型接收一个单独的句子作为输入，然后通过其内部的自注意力机制和Transformer架构，生成该句子的深度表示。然后，这个表示被送入一个顶层分类器（通常是一个全连接层），该分类器根据训练数据将句子分类到预定义的类别中。

例如，单句分类可以用于情感分析（将文本分类为“正面”或“负面”），主题分类（将新闻文章分类为“政治”，“体育”等类别），或者垃圾邮件检测（将电子邮件分类为“垃圾邮件”或“非垃圾邮件”）等任务。基本工作流程示意图如下：



文本回答任务

所谓问题回答指的就是同时给模型输入一个问题和一段描述，最后需要模型从给定的描述中预测出问题答案所在的位置（text span)。我们采用SQuAD这个数据集。

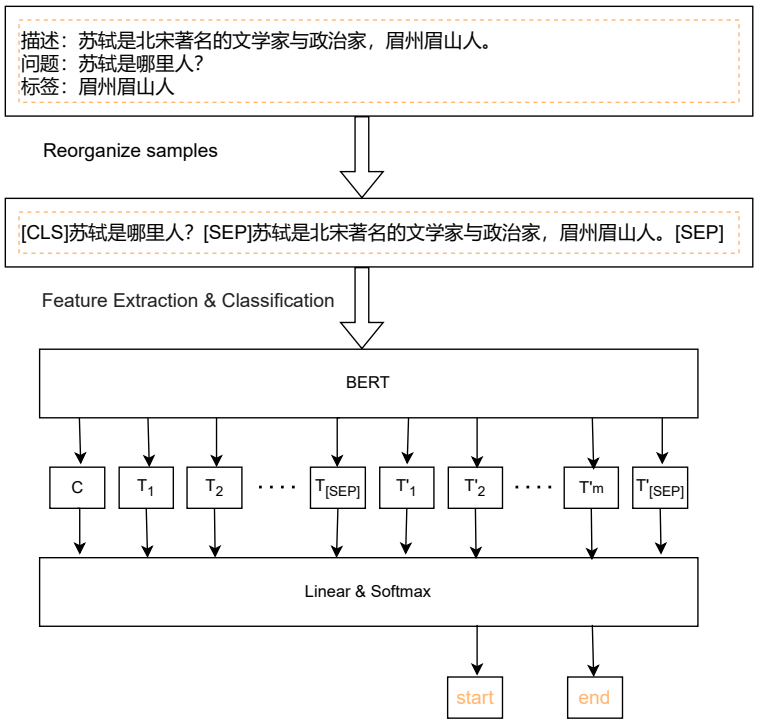
原始数据：

描述：苏轼是北宋著名的文学家与政治家，眉州眉山人。
问题：苏轼是哪里人？
标签：眉州眉山人

对于这样一个问题问答任务我们应该怎么来构建这个模型呢？
在做这个任务之前首先需要明白的就是：

1. 最终问题的答案一定是在给定的“描述”中；
2. 问题再描述中的答案一定是一段连续的字符，不能有间隔。

例如对于上面的描述内容来说，如果给出的问题是“苏轼生活在什么年代以及他是哪里人？”，那么模型最终并不会给出类似“北宋”和“眉州眉山人”这两个分离的 答案，最好的情况下便是给出“北宋著名的文学家与政治家，眉州眉山人”这一个连续的答案。
在有了这两个限制条件后，对于这类问答任务的本质也就变成了需要让模型预测得到答案在描述中的起始位置（start position）以及它的结束位置（end position）。所以，问题最终又变成了如何在BERT模型的基础上再构建一个分类器来对BERT最后一层输出的每个Token进行分类，判断它们是否属于start position或者是end position。原理，如下图所示：



在重构样本时，我们把问题和描述合成一个句子，也引入了一个问题。如果这个合成的句子超过了BERT模型的最大长度（通常为512个tokens），怎么办？通常有下面这几种处理方法：

1. **截断**：这是最简单的方法，可以选择保留前512个tokens，或者保留后512个tokens，或者保留前N个和后K个tokens，其中N+K<=510。
2. **滑动窗口**：将长文本分割成多个小段，每个小段的长度不超过512，且相邻的小段有重叠部分。
3. **重新初始化Positional Embedding**：BERT模型中的Positional Embedding是一个可学习的参数，我们可以重新初始化一个更大的位置词表，然后将前512个向量用预训练模型中的进行替换，余下的通过在下游任务中微调或语料中训练得到。

推荐用第二种滑动窗口的办法。下面是一个使用滑动窗口处理长文本的例子：

```
from transformers import BertTokenizer

def sliding_window(text, max_len):
    tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
    tokens = tokenizer.tokenize(text)
    window_size = max_len - 2  # for [CLS] and [SEP]
    segments = []

    for i in range(0, len(tokens), window_size):
        segment = tokens[i:i+window_size]
        segments.append(segment)

    return segments

text = "Your long text here..."
max_len = 512
segments = sliding_window(text, max_len)

for segment in segments:
    print(segment)
```

这段代码首先将长文本tokenize成tokens，然后使用滑动窗口将tokens分割成多个小段，每个小段的长度不超过512。这样，我们就可以将每个小段分别输入到BERT模型中进行处理了。

如果输入采用了滑动窗口的方法，那么在推理时，我们需要对每个窗口分别进行推理，并将结果合并。具体的操作步骤如下：

1. **分割输入**：首先，我们需要将输入文本分割成多个小段，每个小段的长度不超过BERT模型的最大长度（通常为512个tokens），且相邻的小段有重叠部分。
2. **对每个窗口进行推理**：然后，我们将每个小段分别输入到BERT模型中进行推理。这一步的操作与之前的推理过程相同，只是输入变成了小段。
3. **合并结果**：最后，我们需要将每个小段的推理结果合并成完整的推理结果。具体的合并方式取决于你的任务。例如，如果你的任务是文本分类，那么你可以对每个小段的分类结果进行投票，选择得票最多的类别作为最终的分类结果。如果你的任务是问答，那么你可以选择得分最高的答案作为最终

的答案。

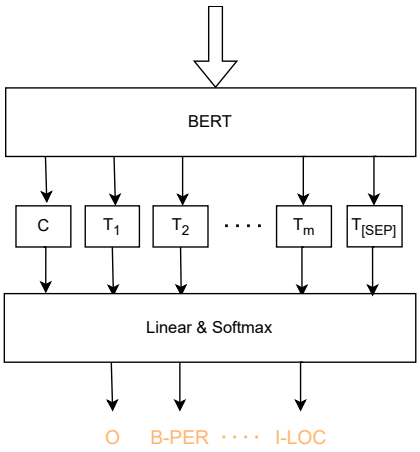
以下是一个使用滑动窗口处理长文本并进行推理的Python代码示例：

```
from transformers import BertForQuestionAnswering, BertTokenizer
import torch

# Load the fine-tuned BERT-large model
model = BertForQuestionAnswering.from_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')

# Load the tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-large-uncased-whole-word-masking-finetuned-squad')

# Your question and text
question = "Who was the first president of the United States?"
text = "Your long text here..."

# Package the question and text into the input, and use the sliding window to handle the long text
inputs = [tokenizer(question, segment, add_special_tokens=True, return_tensors="pt") for segment in sliding_window(text, max_len)]

# Perform inference on each window and get the prediction of the start and end
outputs = [model(**input) for input in inputs]
start_scores, end_scores = zip(*[(output.start_logits, output.end_logits) for output in outputs])

# Find the position with the highest score for the start and end
start_index = torch.argmax(torch.cat(start_scores))
```

这段代码首先加载了一个已经在SQuAD上fine-tuned的BERT模型和相应的tokenizer。然后，我们定义了一个问题和一段包含答案的文本。我们使用tokenizer将问题和文本打包到一个输入张量中，然后将其输入到模型中。模型返回每个可能的开始和结束位置的分数。我们选择分数最高的位置作为答案的开始和结束位置，然后使用tokenizer将这些位置的token转换回文本，得到我们的答案。这就是在使用滑动窗口处理长文本时，如何进行推理的一个例子。

单句标注任务

让我们通过一个命名实体识别（Named Entity Recognition，NER）的例子来详细说明单句标注（Single Sentence Tagging）。例如，对于句子“苹果公司在加利福尼亚州的库比蒂诺市设有总部”，模型可能会预测出“苹果公司”为“组织”，“加利福尼亚州”和“库比蒂诺市”为“地点”，其余单词为“其他”。具体工作流程，如下图所示：



这种标注方式被称为BIO标注法，广泛应用于各种序列标注任务中。BIO标注法是一种常用于序列标注任务（如命名实体识别）的标注方法。在BIO标注法中，每个标签由两部分组成：一个前缀（B、I或O）和一个类型（如PER、LOC等）。前缀表示当前词在实体中的位置，类型表示实体的类别。具体来说：

- **B（Begin）**：表示实体的开始。例如，在句子“我在**北京**工作”中，“北京”的标签就是B-LOC。
- **I (inside)**: 表示实体的持续。例如，在句子“我在**新疆乌鲁木齐市**工作”中，“新疆乌鲁木齐”的标签就是B-LOC I-LOC I-LOC。
- **O (Outside)**: 表示这个词不属于任何实体。

除了人名（PER），地名（LOC）和组织名（ORG）之外，命名实体识别（Named Entity Recognition，NER）通常还包括以下类型：

- TIME：时间表达式，如“下周一”、“20世纪90年代”等。

- DATE：日期表达式，如“2024年1月1日”、“上个月”等。
- MONEY：货币表达式，如“五美元”、“100人民币”等。
- PERCENT：百分比表达式，如“百分之五”、“20%”等。

至此，我们讲解完了BERT模型。

BERT的影响

从纯技术的角度来说，BERT没什么太大的创新，更像是把2018年前NLP领域重大进展放在一个模型里的集大成者。但是，BERT效果非常好，在机器阅读理解顶级水平测试SQuAD1.1中，BERT模型的表现甚至超越了人类（具体数据详见原论文）。对后面的发展产生了重大影响。
在笔者看来，最主要的以下三方面的影响：

1. **夯实预训练和微调的范式**：BERT模型的成功证明了预训练和微调的有效性。在预训练阶段，BERT模型在大规模无标签文本数据上进行训练，学习到丰富的语言表示。在微调阶段，BERT模型在特定任务的标注数据上进行训练，使得模型能够适应各种NLP任务。
2. **推动了Transformer的广泛应用**：BERT模型基于Transformer架构，这种架构能够捕获文本中的长距离依赖关系，并且计算效率高。BERT模型的成功推动了Transformer架构在NLP领域的广泛应用，让Transformer取代RNN，占据主导地位。
3. **促进了NLP领域的创新**：BERT开源并开放各种规格的模型下载成为了2018到几乎ChatGPT出现之前NLP领域研究的核心模型。BERT模型的出现引发了一系列的创新工作，包括对BERT模型的改进，以及模型轻量化技术。

小结

科学的重大进步，从不是通过一种直接的方式，而是一定要设立一个高度挑战性的目标，通过强大的动力促使技术革新，迫使科学家燃烧他们的想象力，使他们尽最大的可能完成他们的目标，这就是为什么我们说，一定要把路脚放的更远一点。

Reference

https://arxiv.org/abs/1810.04805
https://www.cs.princeton.edu/courses/archive/fall22/cos597G/lectures/lec02.pdf
https://nlp.stanford.edu/seminar/details/jdevlin.pdf
https://www.mikecaptain.com/2023/03/06/captain-aigc-2-llm/
http://www.evinchina.com/uploadfile/file/20230315/20230315094024075 39.pdf
https://developers.google.com/machine-learning/glossary/language