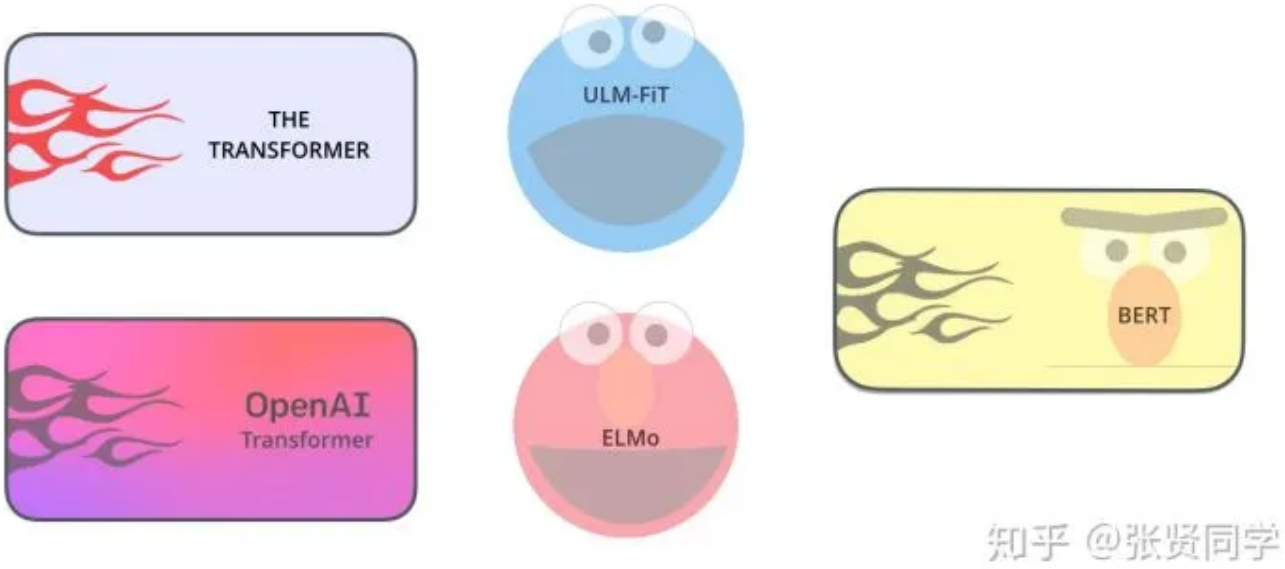
图解 BERT

机器学习研究组订阅 2020-10-20

来自 | 知乎
地址 | https://zhuanlan.zhihu.com/p/266364526
作者 | 张贤同学

> 本文翻译自：**jalammar.github.io/illu**。
> 通俗易懂，非常适合刚刚开始了解 Bert 的同学。
> BERT 来源于 Transformer，如果你不知道 Transformer 是什么，你可以查看 **图解 Transformer**。

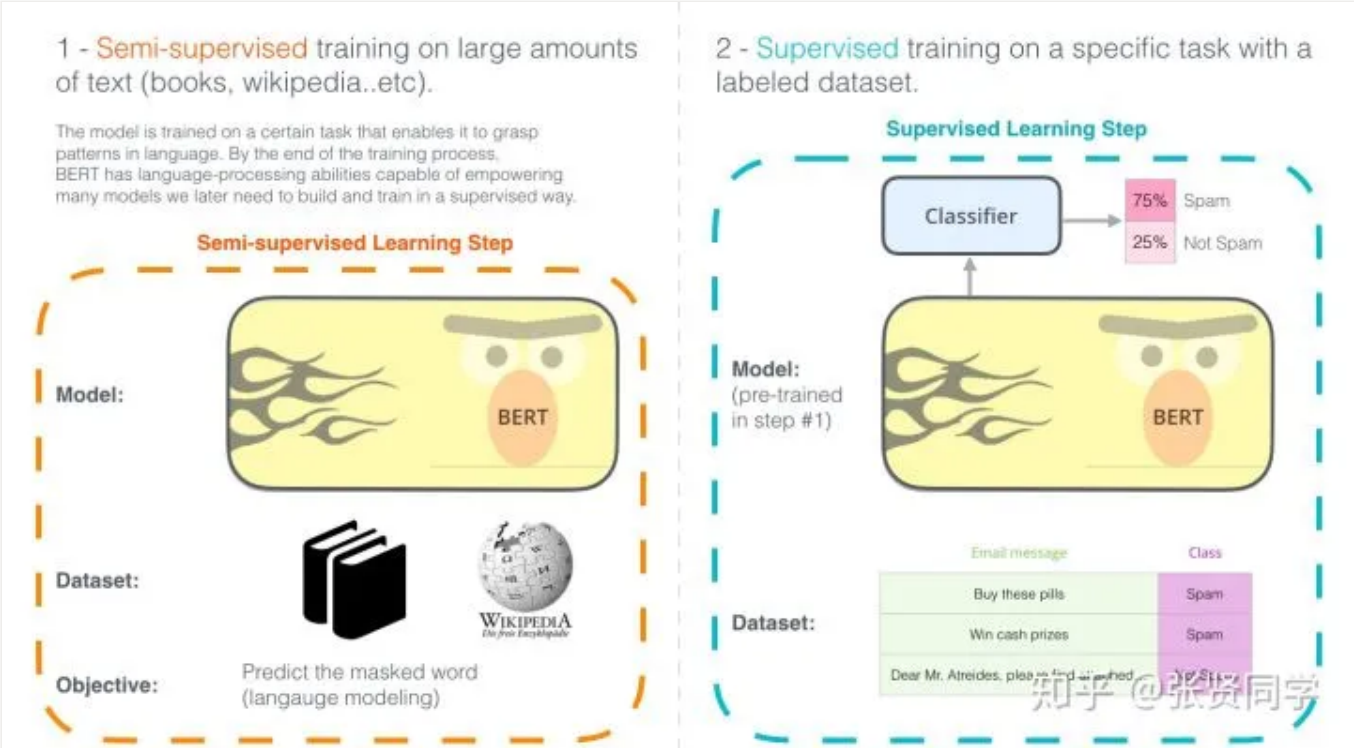
2018 年是机器学习模型处理文本（或者更准确地说，自然语言处理或 NLP）的转折点。我们对这些方面的理解正在迅速发展：如何最好地表示单词和句子，从而最好地捕捉基本语义和关系？此外，NLP 社区已经发布了非常强大的组件，你可以免费下载，并在自己的模型和 pipeline 中使用（今年可以说是 NLP 的 ImageNet 时刻，指的是多年前类似的发展也加速了机器学习在计算机视觉任务中的应用）。



ULM-FiT 与 Cookie Monster（饼干怪兽）无关。但我想不出别的了...

[BERT](github.com/google-resea) 的 [发布](ai.googleblog.com/2018/) 是这个领域发展的最新的里程碑之一，这个事件 [标志着](twitter.com/lmthang/sta) NLP 新时代的开始。BERT 模型打破了基于语言处理的任务的几个记录。在 BERT 的论文发布后不久，这个团队还公开了模型的代码，并提供了模型的下载版本，这些模型已经在大规模数据集上进行了预训练。这是一个重大的发展，因为它使得任何一个构建构建机器学习模型来处理语言的人，都可

以将这个强大的功能作为一个现成的组件来使用，从而节省了从零开始训练语言处理模型所需要的时间、精力、知识和资源。



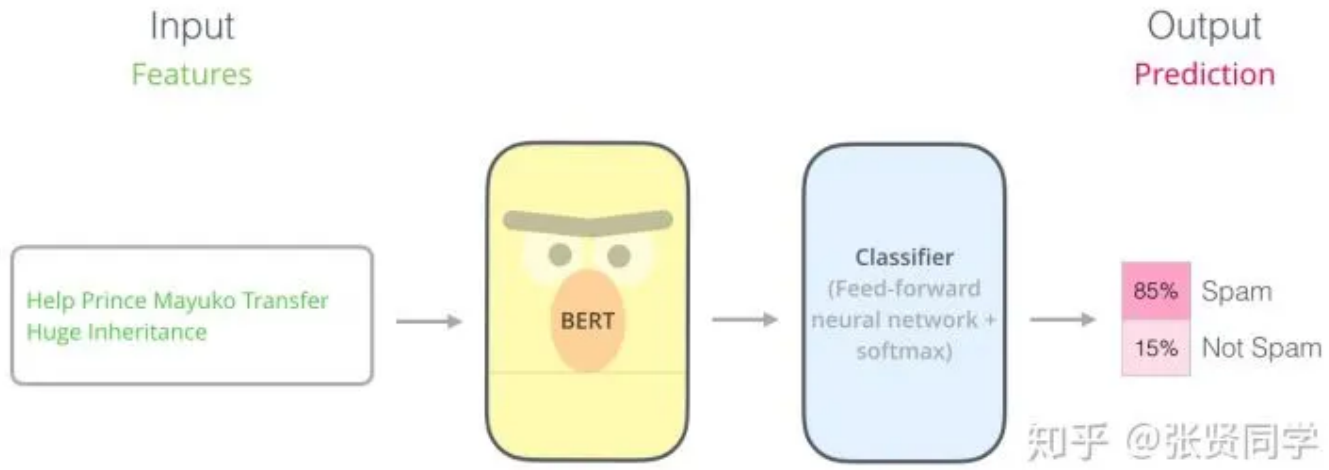
BERT 开发的两个步骤。第 1 步，你可以下载预训练好的模型（这个模型是在无标注的数据上训练的）。然后在第 2 步只需要关心模型微调即可。（图书图标 **来源**）

BERT 建立在 NLP 社区最近出现的一些顶尖的思想之上，包括但不限于： **Semi-supervised Sequence Learning** (**Andrew Dai** 和 **Quoc Le**), **ELMo** (**Matthew Peters** 和来自于 **AI2**、**UW CSE** 的研究), **ULMFiT** (fast.ai 的创始人 **Jeremy Howard** 和 **Sebastian Ruder**), the **OpenAI transformer** (OpenAI 研究员 **Radford**, **Narasimhan**, **Salimans** 和 **Sutskever**) 和 the Transformer (**Vaswani et al**).。

你需要注意一些事情，才能理解 BERT 是什么。因此，在介绍模型本身涉及的概念之前，让我们先看看如何使用 BERT。

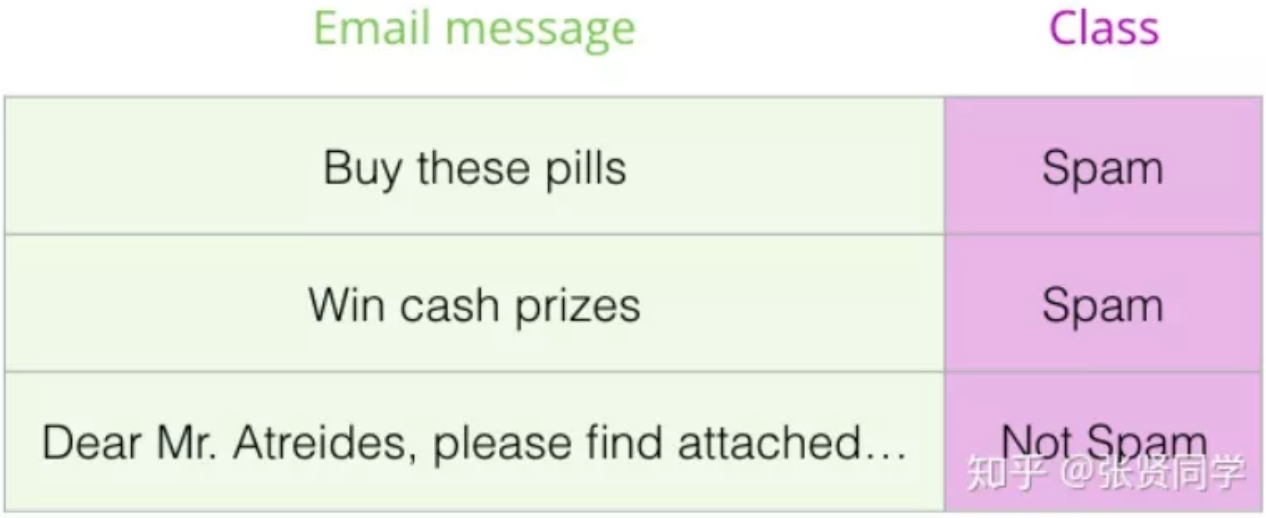
示例：句子分类

使用 BERT 最直接的方法就是对一个句子进行分类。这个模型如下所示：



为了训练这样一个模型，你主要需要训练分类器（上图中的 Classifier），在训练过程中 BERT 模型的改动很小。这个训练过程称为微调，它起源于 [Semi-supervised Sequence Learning](arxiv.org/abs/1511.0143) 和 ULMFiT。

对于不熟悉这个主题的人来说，既然我们在讨论分类器，那么我们说的是机器学习的监督学习领域。这意味着我们需要一个带有标签的数据集来训练这样一个模型。对于这个垃圾邮件分类器的示例，带有标签的数据集包括一个邮件内容列表和一个标签（每个邮件是“垃圾邮件”或者“非垃圾邮件”）。



这种用例的 其他示例，包括：

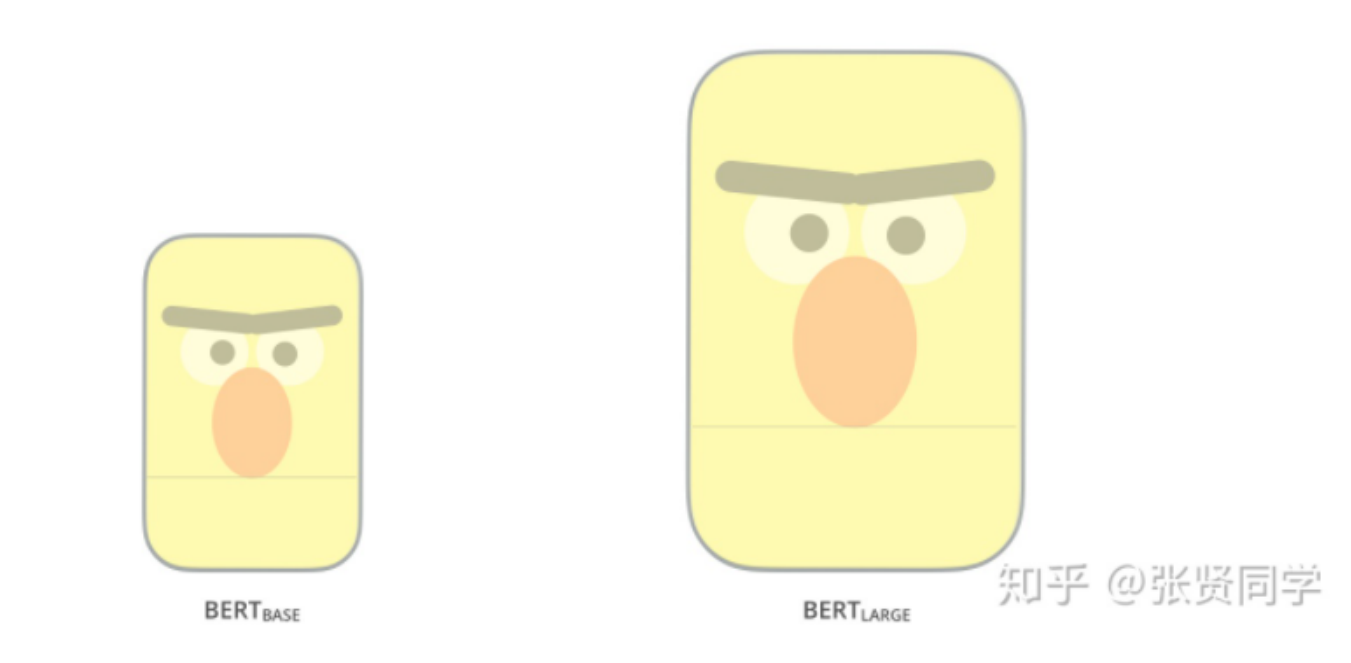
- **语义分析**
  输入：电影或者产品的评价。输出：这个评价是正面的还是负面的？
  数据集示例：**SST**

- **事实核查**

输入：句子。输出：“索赔”或者“不索赔”
更具雄心/更加具有未来感的示例：
输入：索赔判决。输出："True" 或者 "False"
Full Fact 是一个组织，构建了自动化事实核查工具，方便了其他人使用。他们的 pipeline 的一部分是阅读新的文章，并且检测 claims（将文本分类为 "claim" 或者 "not claim"），然后可以对其进行事实核查（现在是由人工完成的，希望以后可以由 ML 来完成）。
参考视频：**Sentence embeddings for automated factchecking - Lev Konstantinovskiy**。

模型架构

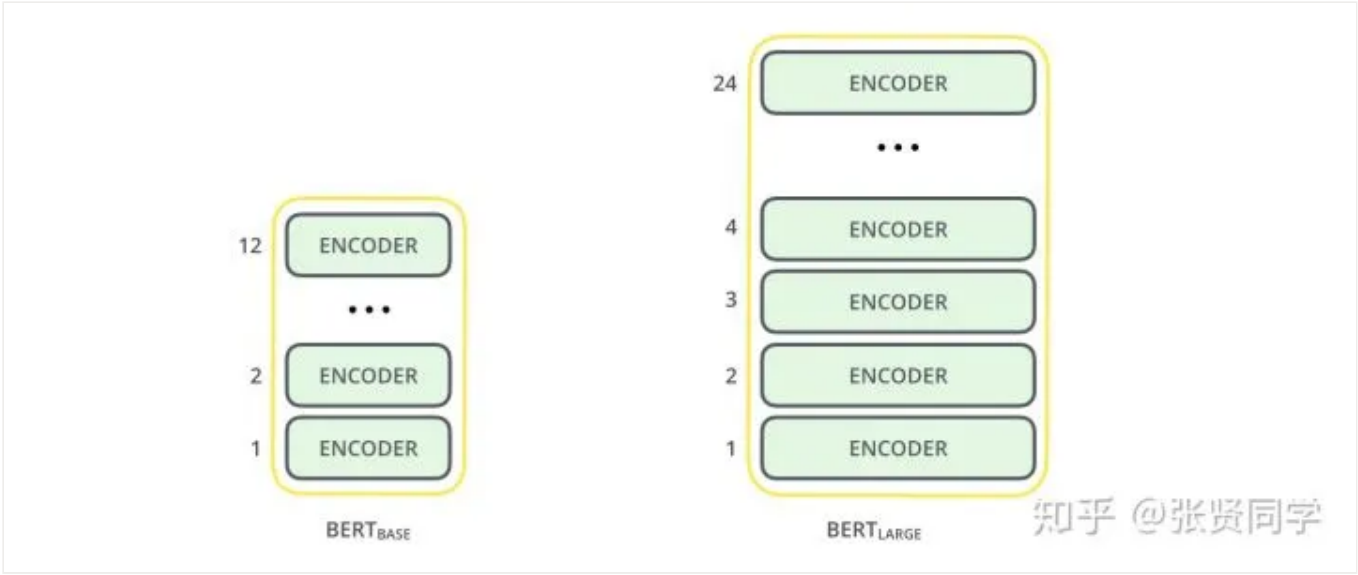
现在你已经通过示例了解了如何使用 BERT，让我们更深入地了解以下它的工作原理。



论文里介绍了两种不同模型大小的 BERT：

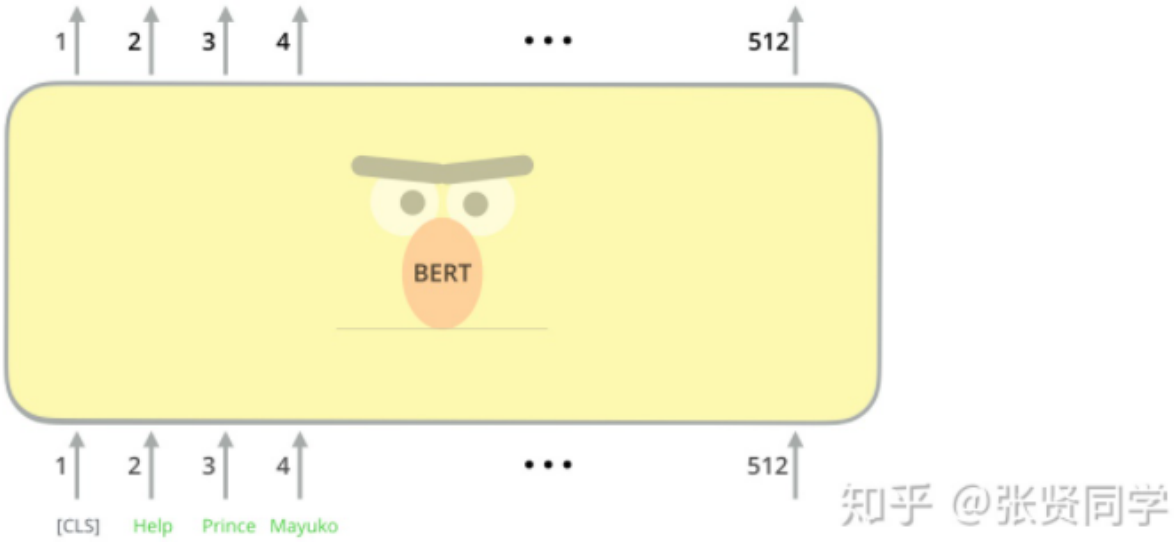
- BERT BASE - 与 OpenAI 的 Transformer 大小相当，以便比较性能
- BERT LARGE - 一个非常巨大的模型，它达到了论文中所说的最先进的结果

BERT 基本上是一个训练好的 Transformer 的 decoder 的栈。关于 Transformer 的介绍，可以阅读我之前的文章 **图解 Transformer**，这篇文章介绍了 Transformer 模型 - BERT 的一个基本概念，以及我们接下来将要讨论的概念。



2 种大小的 BERT 模型都有大量的 encoder 层（论文里把这些层称为 Transformer Blocks）- BASE 版本由 12 层，Large 版本有 20 层。同时，它也有更大的前馈神经网络（分别有 768 个和 1024 个隐藏层单元）和更多的 attention heads（分别有 12 个和 16 个），超过了原始 Transformer 论文中的默认配置参数（6 个 encoder 层， 512 个隐藏层单元和 8 个 attention heads）。

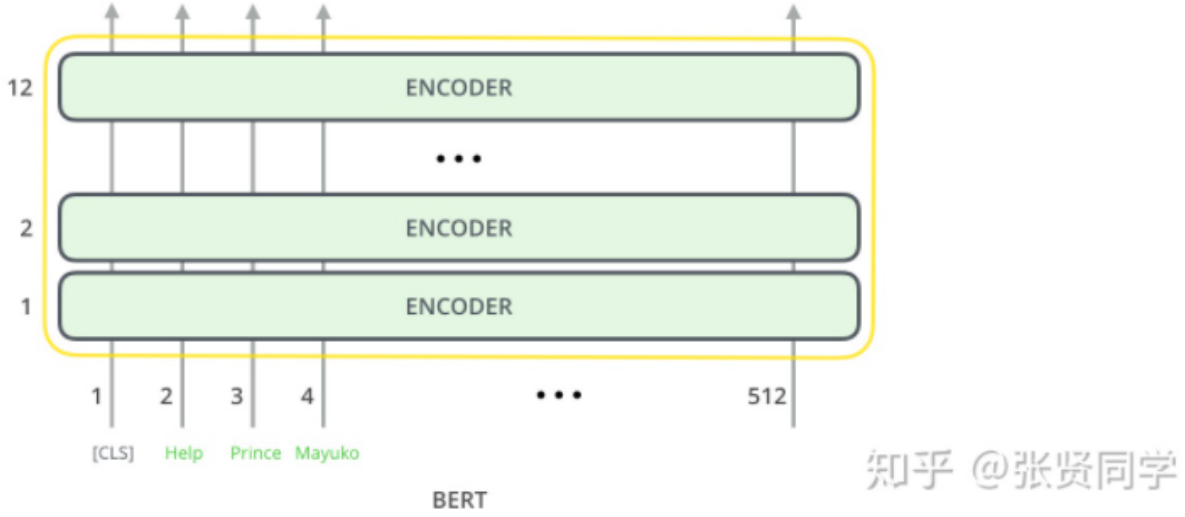
模型输入



第一个输入的 token 是特殊的 [CLS]，我会在后面解释。这里的 CLS 的含义是分类。

就像 Transformer 中普通的 encoder 一样，BERT 将一个单词序列作为输入，这些单词不断向上流动。每一层都用用到了 self-attention，并通过一个前馈神经网络，将结果传给下一个

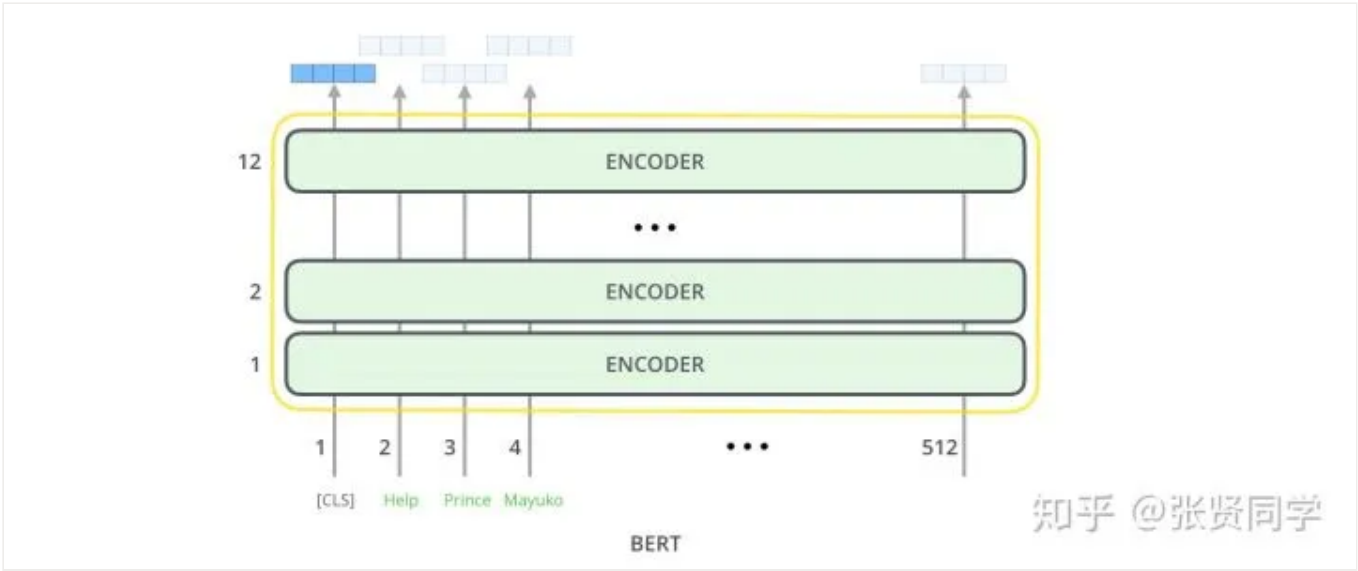
encoder。



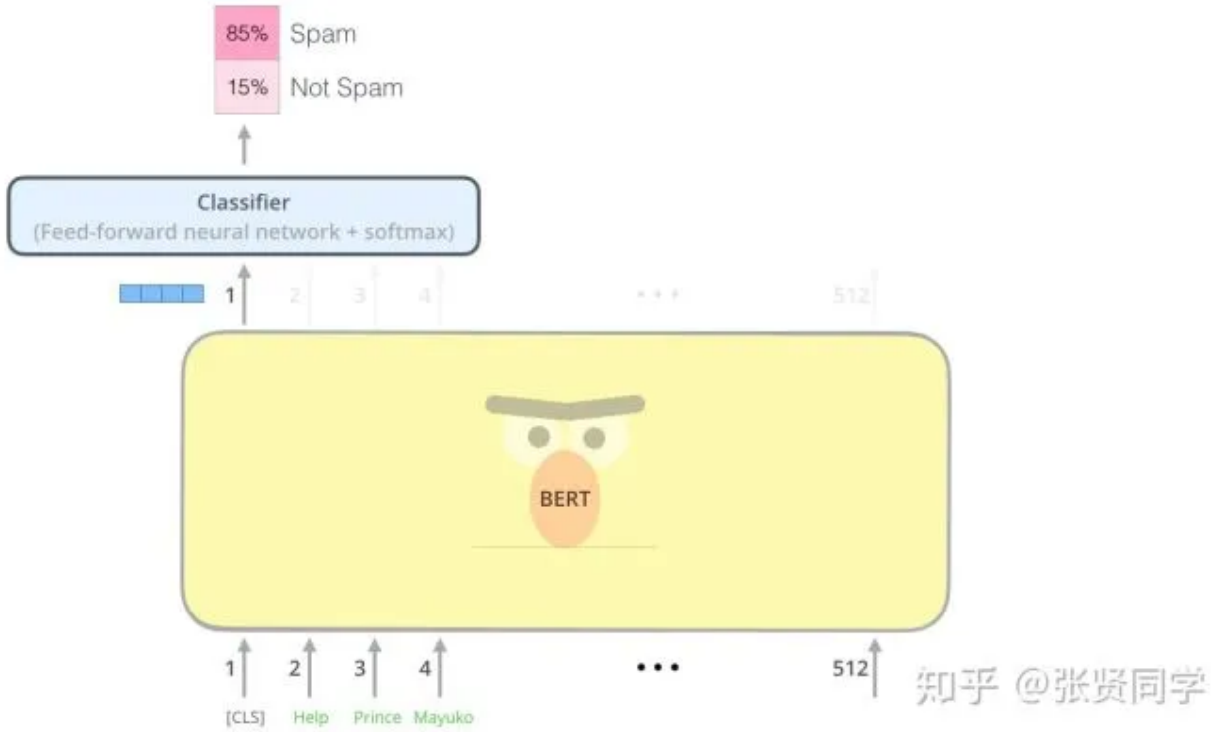
在模型架构方面，到目前为止，和 Transformer 是相同的（除了模型大小，因为这是我们可以设置的参数）。在模型的输出上，我们可以发现有一些不同。

模型输出

每个为止输出一个大小为 hidden_size（在 BERT Base 中是 768）的向量。对于上面提到的句子分类的示例，我们只关注第一个位置（第一个位置的输入是 [CLS]）的输出。



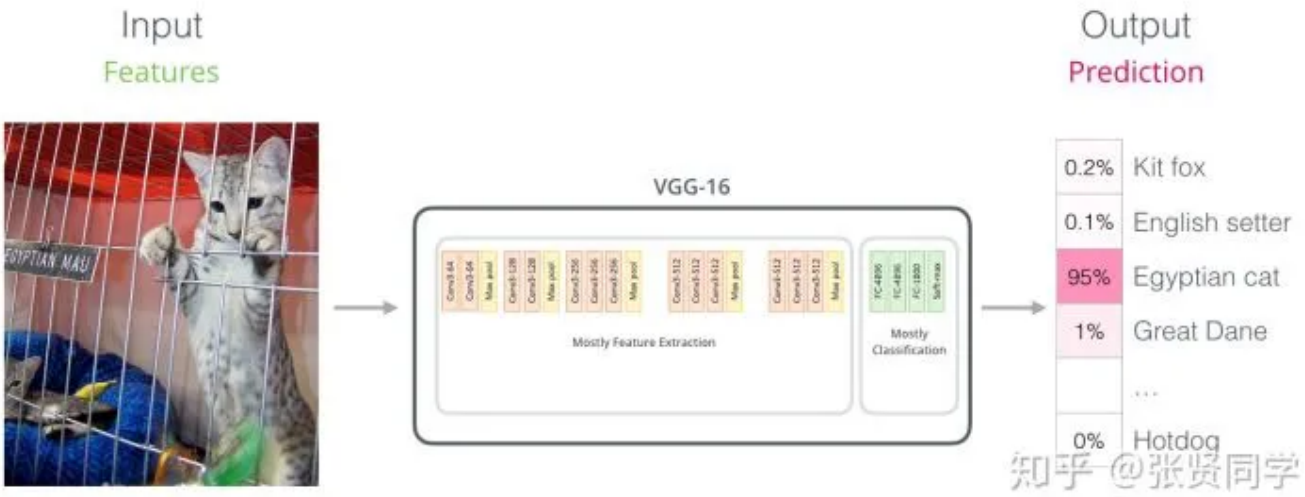
这个向量现在可以作为我们选择的分类器的输入。论文里用单层神经网络作为分类器，取得了很好的效果。



如果你有更多标签（例如你是一个电子邮件服务，标签有 “垃圾邮件”、“非垃圾邮件”、“社交”、“推广”），你只需要调整分类器网络，增加输出的神经元个数，然后经过 softmax 即可。

与 卷积神经网络进行对比

对于那些有计算机视觉背景的人来说，这个向量传递应该让人联想到 VGGNet 等网络的卷积部分和网络最后的全连接分类部分之间发生的事情。



词嵌入（Embedding）的新时代

这些新的发展带来了单词编码方式的新转变。到目前为止，词嵌入一直是 NLP 模型处理语言的主要主要组成部分。像 Word2Vec 和 Glove 这样的方法已经被广泛应用于此类任务。在我们讨论新的方法之前，让我们回顾一下它们是如何使用的。

回顾词嵌入

单词不能直接输入机器学习模型，而需要某种数值表示形式，以便模型能够在计算中使用。Word2Vec 表明我们可以使用一个向量（一组数字）来恰当地表示单词，以捕捉单词的语义以及单词和单词之间的关系（例如，判断单词是否相似或者相反，或者像 "Stockholm" 和 "Sweden" 这样的一对词，与 "Cairo" 和 "Egypt"这一对词，是否有同样的关系）以及句法、语法关系（例如，"had" 和 "has" 之间的关系与 "was" 和 "is" 之间的关系相同）。

这个领域的人很快意识到，相比于在小规模数据集上和模型一起训练词嵌入，更好的一种做法是，在大规模文本数据上预训练好词嵌入，然后拿来使用。因此，可以下载由 Word2Vec 和 GloVe 预训练好的单词列表及其词嵌入。下面是单词 "stick" 的 Glove 词嵌入示例（词嵌入大小是 200）

-0.34	-0.84	0.20	-0.26	-0.12	0.23	1.04	-0.16	0.31	0.06	0.30	0.33	-1.17	-0.30	0.03	0.09	0.35	-0.28	-(

单词 "stick" 的 Glove 词嵌入 - 一个由200个浮点数组成的向量（四舍五入到小数点后两位）。

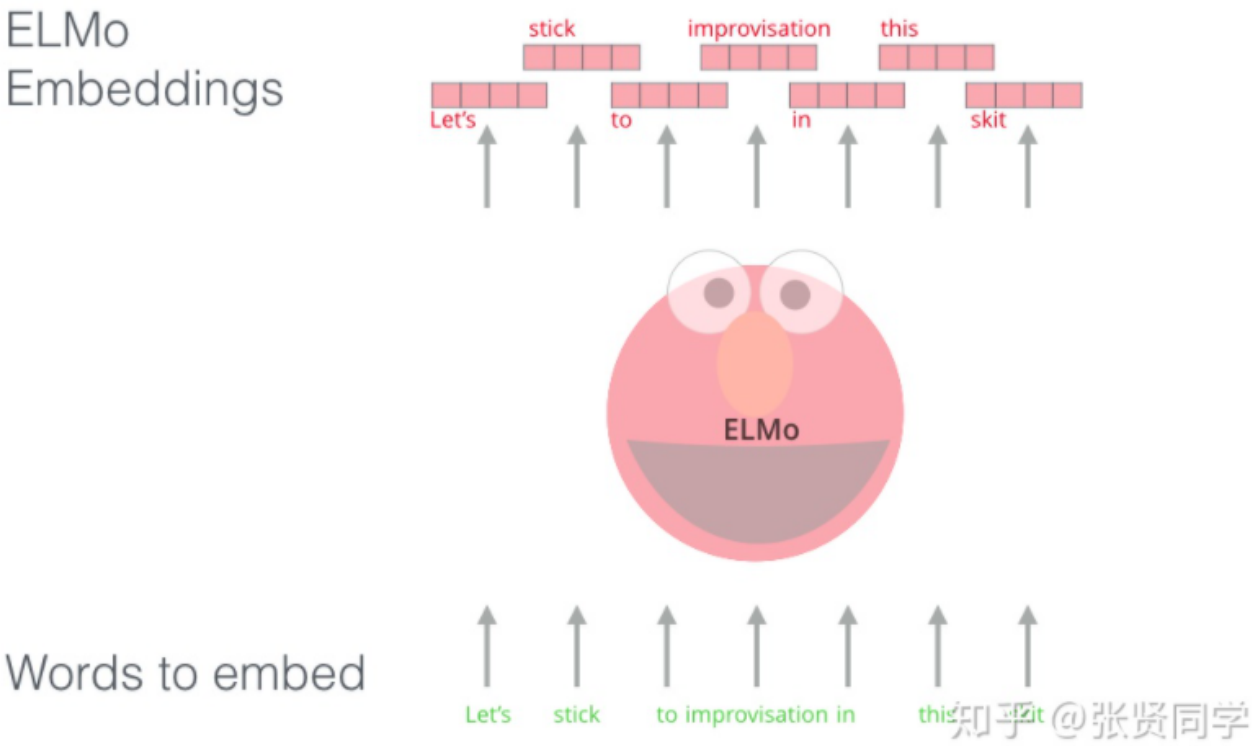
由于这些向量都很长，且全部是数字，所以在文章中我使用以下基本形状来表示向量：

ELMo：语境问题

如果我们使用 Glove 表示，那么不管上下文是什么，单词 "stick" 都只表示为同一个向量。"Wait a minute"，一些研究人员(**Peters et. al., 2017**, **McCann et. al., 2017**, **Peters et. al., 2018 in the ELMo paper**) 指出，"stick" 根据它的用法有多种含义。为什么不根据它使用的上下文来给出词嵌入呢？这样既能捕捉单词的语义信息，又能捕捉上下文的语义信息。于是，语境化的词嵌入应运而生。

语境化的词嵌入，可以根据单词在句子语境中的含义，赋予不同的词嵌入。你可以查看这个视频 **RIP Robin Williams**

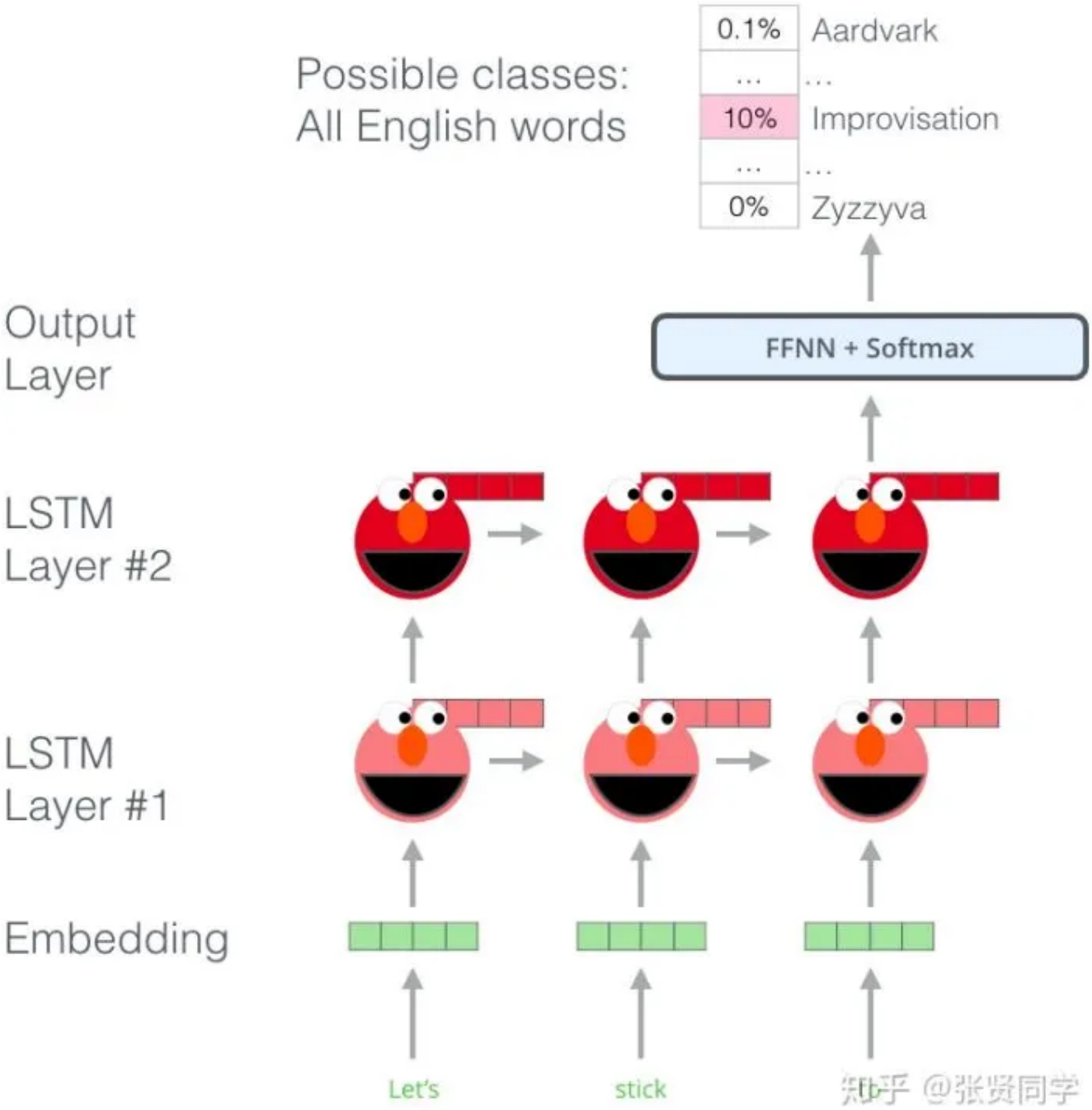
ELMo 没有对每个单词使用固定的词嵌入，而是在为每个词分配词嵌入之前查看整个句子。它使用在特定任务上经过训练的双向 LSTM 来创建这些词嵌入。



ELMo 在 NLP 背景的预训练中迈出了重要的一步。ELMo LSTM 会在一个大规模的 数据集上进行训练，然后我们可以将它作为其他语言处理模型的一个组件。

ELMo 的秘密是什么?

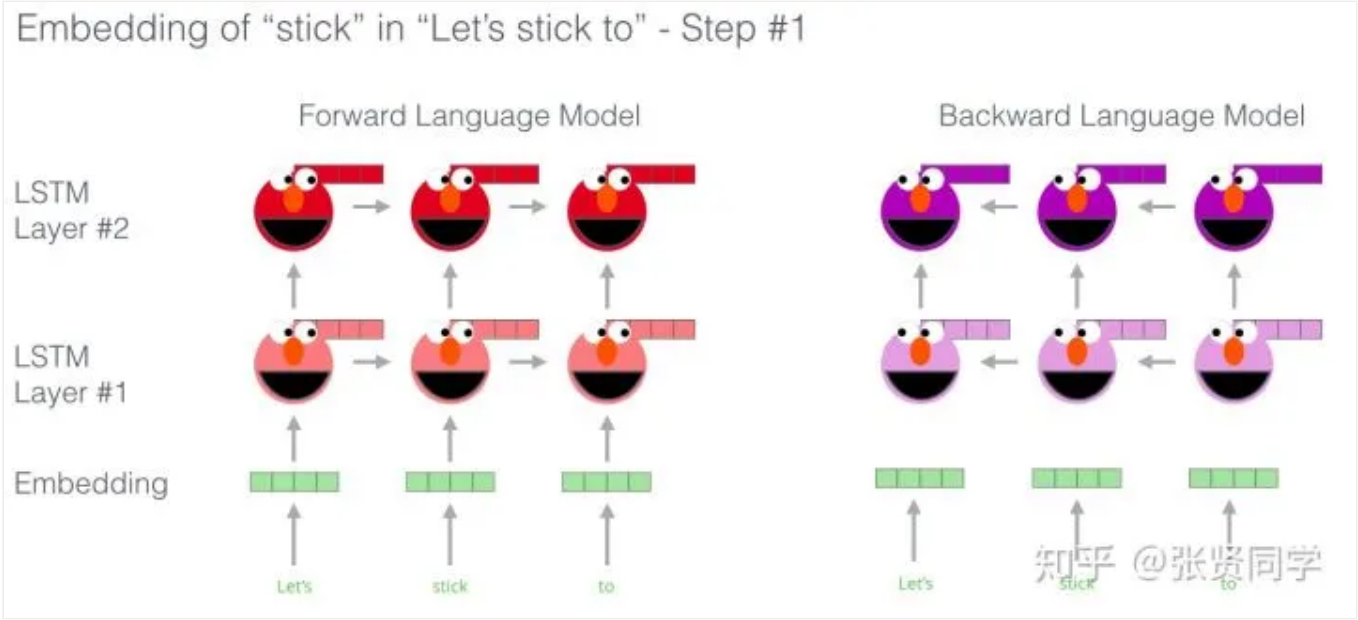
ELMo 通过训练，预测单词序列中的下一个词，从而获得了语言理解能力，这项任务被称为语言建模。要实现 ELMo 很方便，因为我们有大量文本数据，模型可以从这些不带标签的数据中学习。



ELMo 预训练过程的其中一个步骤：以 "Let’ s stick to" 作为输入，预测下一个最有可能的单词 - 这是一个语言建模任务。当我们在大规模数据集上训练时，模型开始学习语言模式。它

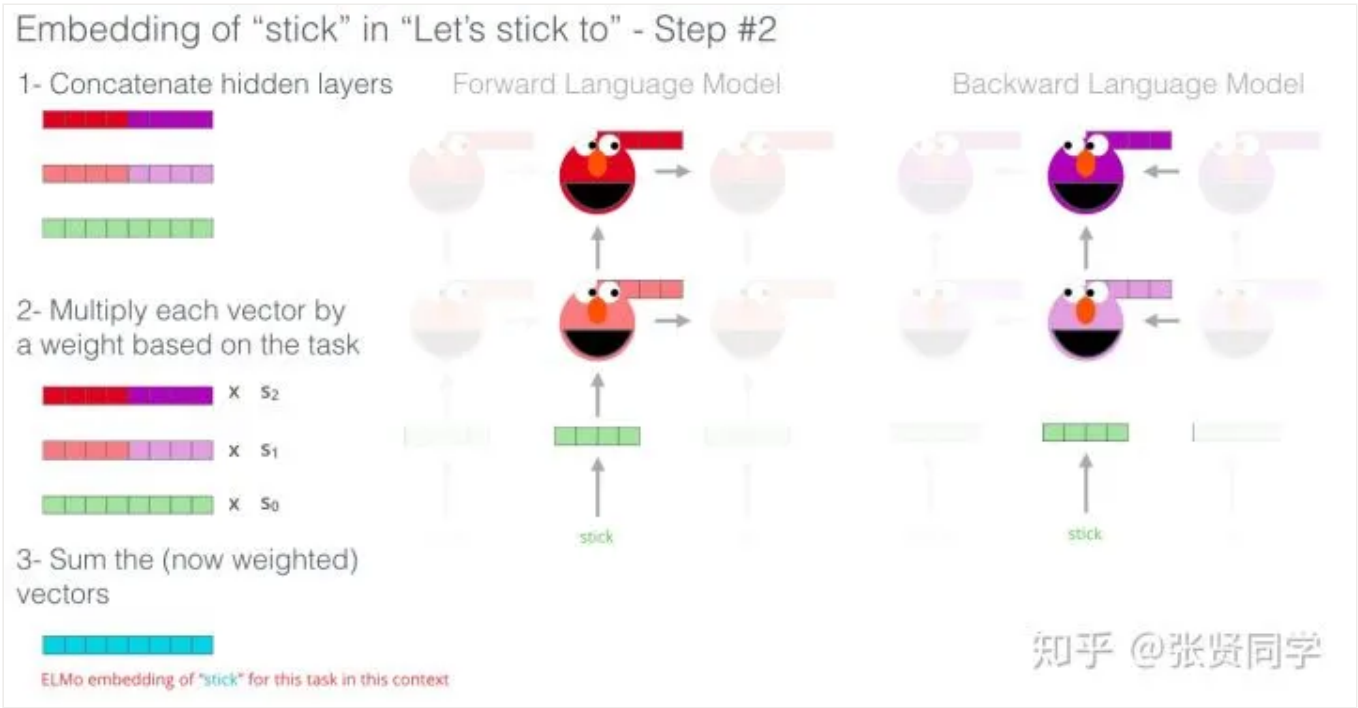
不太可能准确地猜出句子中的下一个词。更实际地说，在 "hang" 这样的词之后，模型将会赋予 "out" 更高的概率（拼写为 "hang out"），而不是 "camera"。

在上图中，我们可以看到 ELMo 头部突出的 LSTM 的每一步的隐藏层状态。在这个预训练过程完成后，这些隐藏层状态在词嵌入过程中非常有用。



关于 ELMo 的 **很棒的幻灯片**

ELMo 通过将隐藏层状态（以及初始化的词嵌入）以某种方式（向量拼接之后加权求和）结合在一起，实现了带有上下文的词嵌入。



ULM-FiT：NLP 领域的迁移学习

ULM-FiT 提出了一些方法来有效地利用模型在预训练期间学习到的东西 - 这些东西不仅仅是词嵌入和带有上下文信息的词嵌入。ULM-FiT 提出了一个语言模型和一种方法，可以有效地为各种任务微调这个语言模型。

NLP 最终可能找到了一种和计算机视觉的要迁移学习一样好的方法。

Transformer：超越 LSTM

Transformer 论文和代码的发布，以及它在机器翻译等任务上取得的成果，开始让这个领域的一些人认为它是 LSTM 的替代品。这是因为 Transformer 可以比 LSTM 更好地处理长期依赖。

Transformer 的 Encoder-Decoder 结构使得它非常适合机器翻译。但你怎么才能用它来做文本分类呢？你怎么才能使用它来预训练一个语言模型，并能够在其他任务上进行微调（下游任务是指那些能够利用预训练模型或者组件的监督学习任务）？

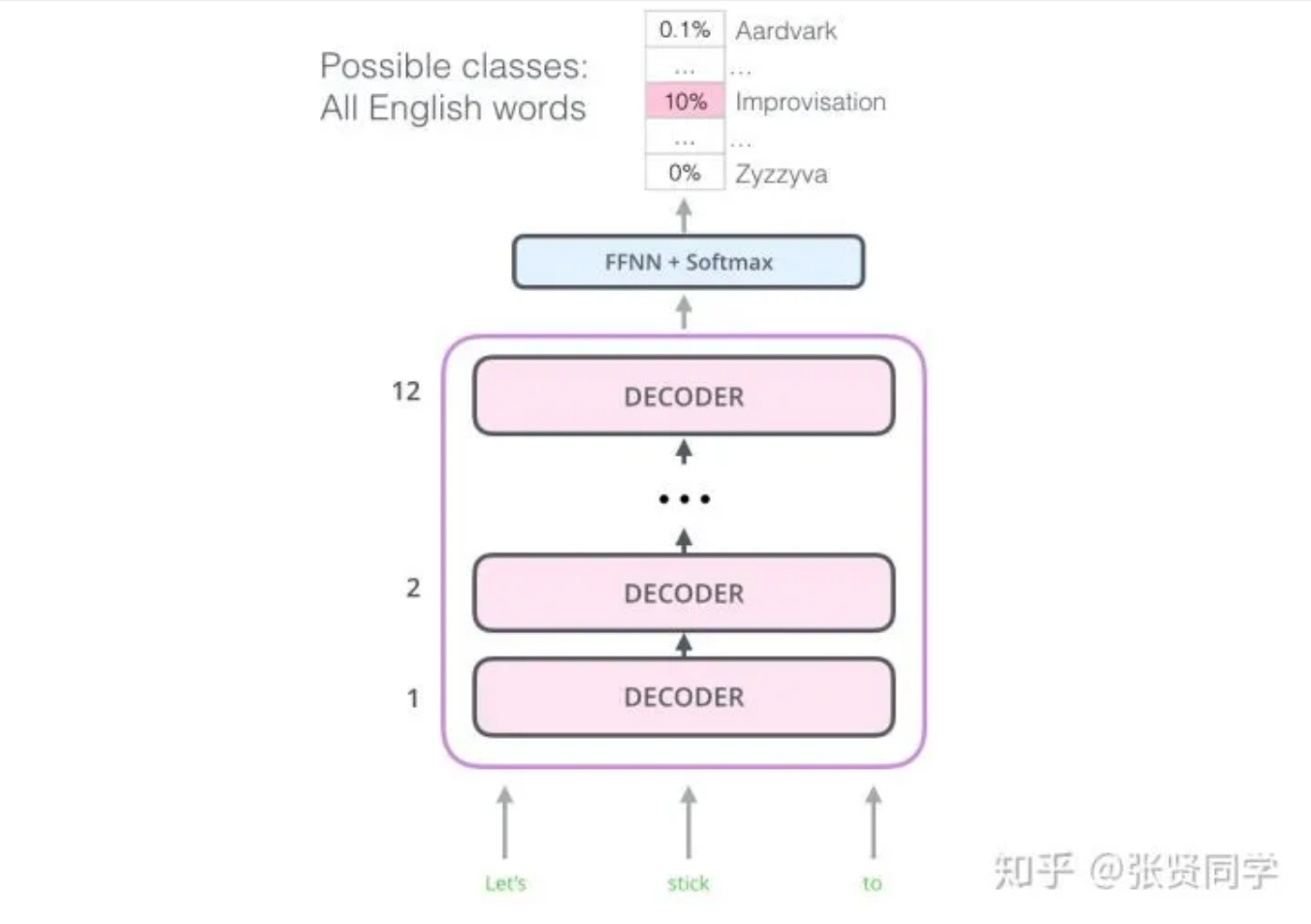
OpenAI Transformer：预训练一个 Transformer Decoder 来进行语言建模

事实证明，我们不需要一个完整的 Transformer 来进行迁移学习，也不需要一个完整的 Transformer 来作为一个 NLP 任务的可以微调的模型。我们只需要 Transformer 的 decoder 就可以了。decoder 是一个很好的选择，因为用它来做语言建模（预测下一个词）是很自然的，因为它就是为了 mask 未来的 token 而构建的 ，当你使用它来生成一个个词的翻译时，这是个很有价值的特点。

OpenAI Transformer 是由 Transformer 的 decoder 堆叠而成的

这个模型包括 12 个 decoder 层。因为在这个设置中没有 encoder，这些 decoder 层不会像普通的 Transformer 中的 decoder 层那样有 encoder-decoder attention 子层。不过，它仍然会有 self-attention 层（这些层使用了 maks，因此不会捕捉到未来的 token）。

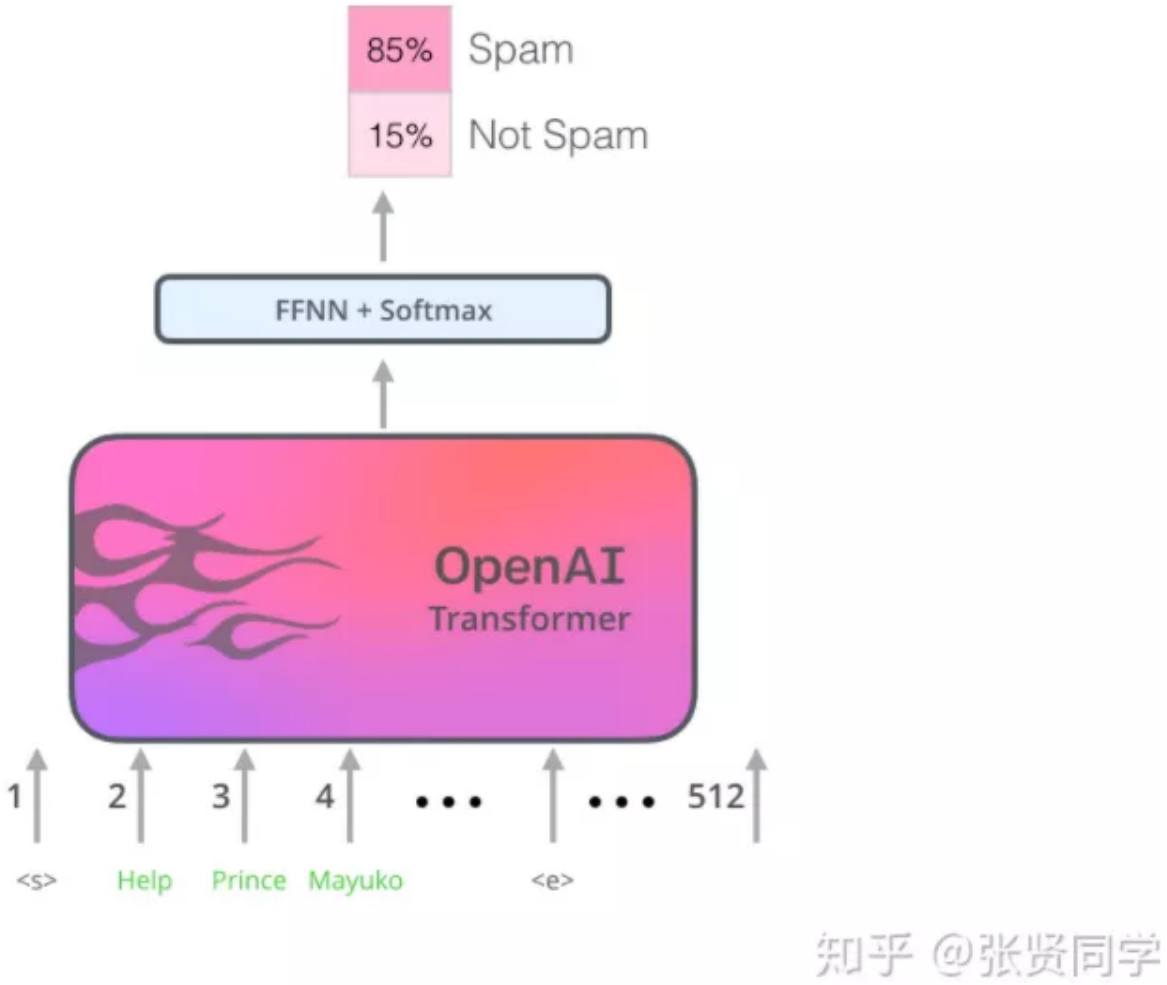
有了这个结构，我们可以继续在同样的语言建模任务上训练这个模型：使用大规模未标记的数据来预测下一个词。只是把 7000 本书的文字扔给它，然后让他学习。书籍对这种任务是非常有用的，因为书籍的数据允许模型将相关信息关联起来，即使它们被大量文本分隔开。如果你使用 tweets 或者文章来训练时，你是得不到这些信息的。



OpenAI Transformer 已经准备好在 7000 本书的组成的数据集中预测下一个单词

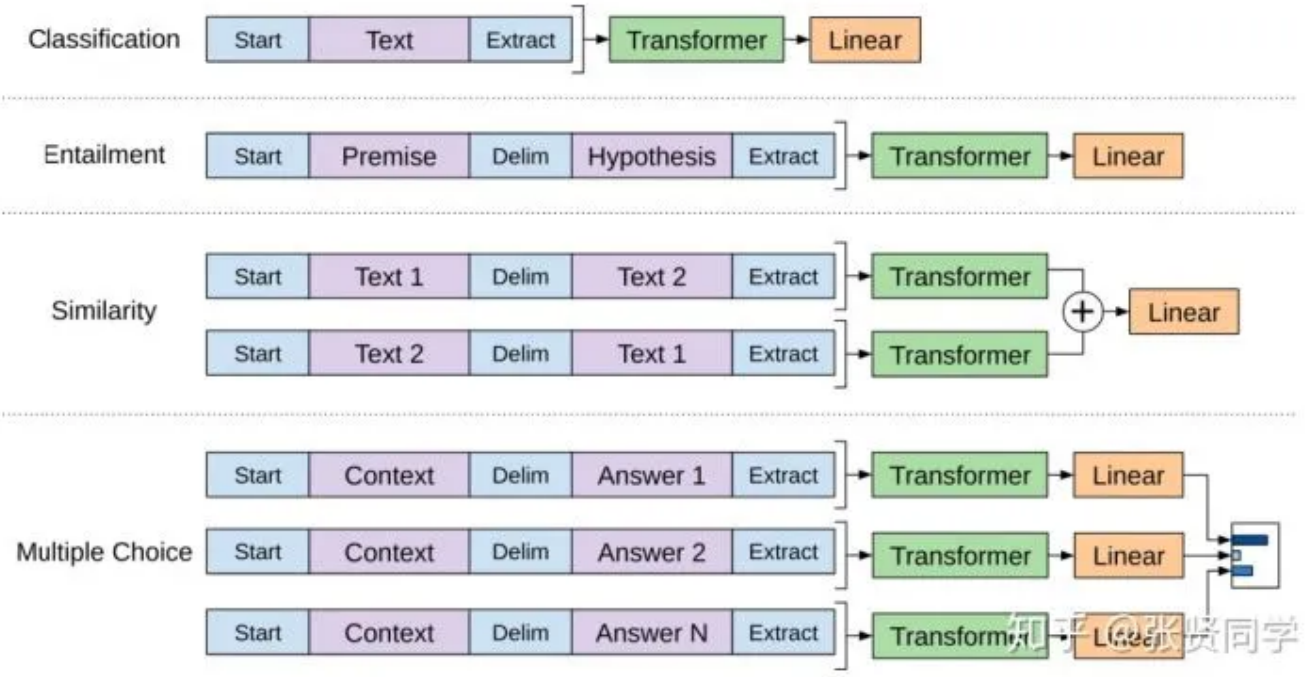
下游任务的迁移学习

现在，OpenAI Transformer 已经预训练好了，它的网络层已经调整到可以很好地处理语言，我们可以开始使用它来处理下游任务。让我们先看下句子分类（把电子邮件分类为 “ 垃圾邮件 “ 或者 ” 非垃圾邮件 “）：



使用 OpenAI Transformer 来做句子分类

OpenAI 的论文概述了一些列输入变换方法，来处理不同类型任务的输入。下面这种图片来源于论文，展示了执行不同任务的模型结构和输入变换。



这是不是很巧妙的做法?

BERT：从 Decoder 到 Encoder

OpenAI Transformer 为我们提供了一个基于 Transformer 的可以微调的预训练网络。但是在从 LSTM 到 Transformer 的过程中，有些东西不见了。ELMo 的语言模型是双向的，但 OpenAI Transformer 只训练了一个前向的语言模型。我们是否可以构建一个基于 Transformer 的语言模型，它既向前看，又向后看（用技术术语来说 - 是否以左边和右边的上下文为条件的？）。

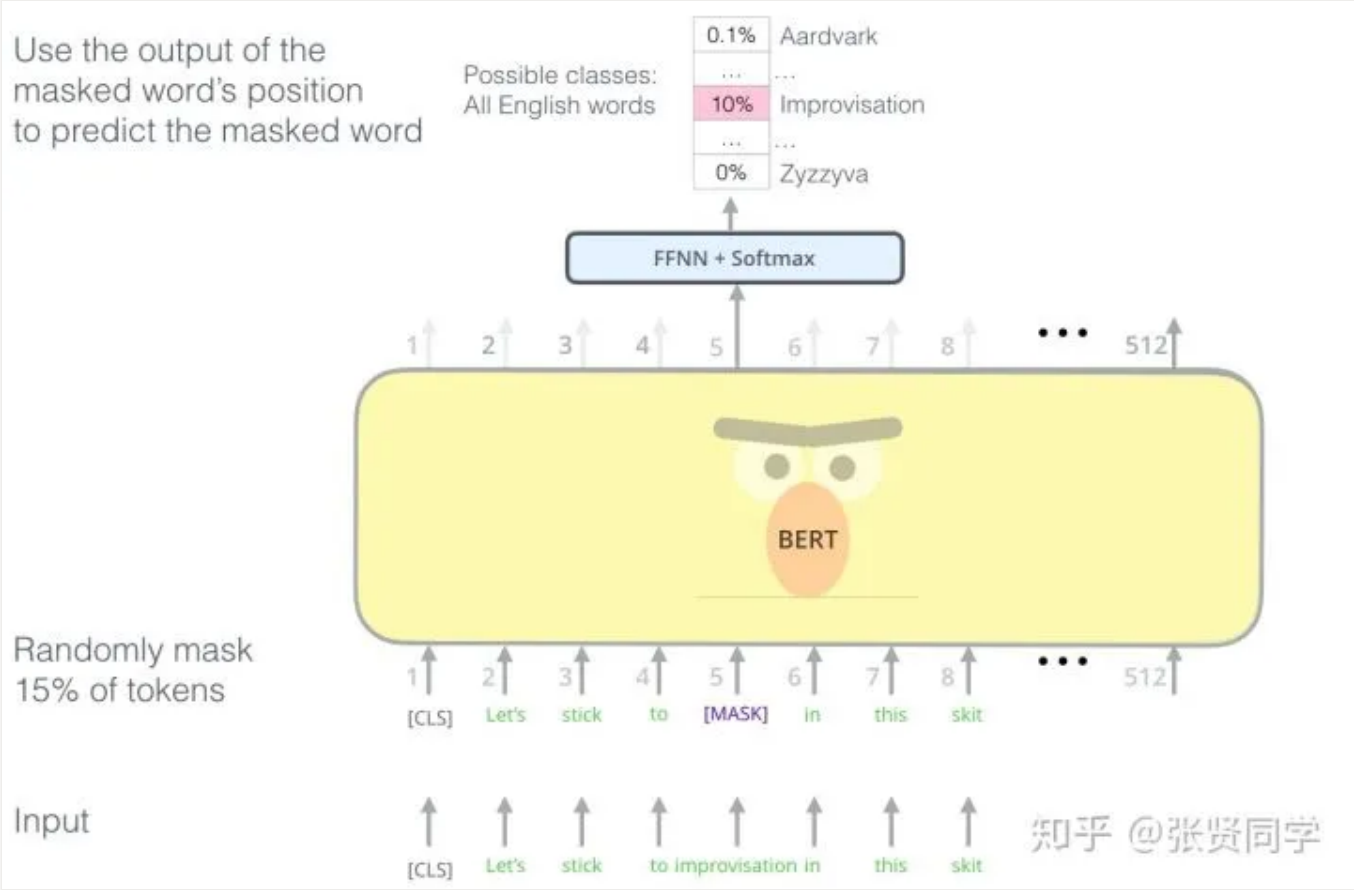
R-rated BERT 说，“拿着我的啤酒”。

Masked Language Model

BERT 说，“我们要用 Transformer 的 encoder”。

Ernie 说，” 这太疯狂了，每个人都知道双向条件允许每个词在多层上下文中看到自己 “。

BERT 自信地说，” 我们会使用 mask “。



BERT 巧妙的语言建模任务中，屏蔽了输入中 15% 的单词，并要求模型预测确实的单词

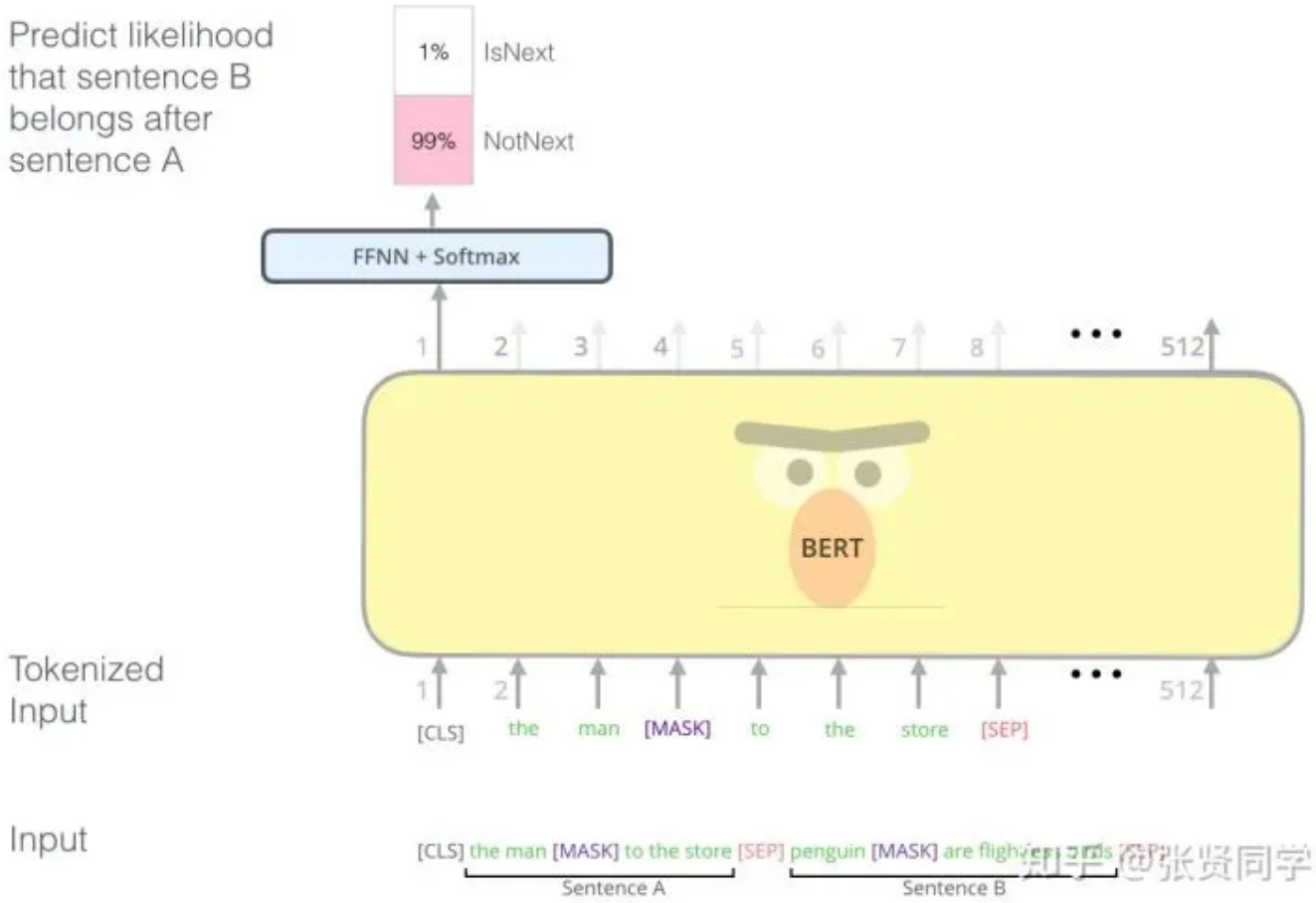
找到正确合适的任务来训练一个 Transformer 的 encoder 是一个复杂的障碍，BERT 通过使用早期文献中的 "masked language model" 概念（在这里被称为完形填空）来解决这个问题。

除了屏蔽输入中 15% 的单词外， BERT 还混合使用了其他的一些技巧，来改进模型之后的微调。有时它会随机地用一个词替换另一个词，然后要求模型预测这个位置的正确单词。

两个句子的任务

如果你回顾 OpenAI Transformer 在处理不同任务时所做的输入变换，你会注意到有些任务需要模型对两个句子做一些智能的判断（例如，它们是不是同一句话的不同解释？将一个维基百科条目作为输入，再将一个相关的问题作为另一个输入，我们可以回答这个问题吗？）。

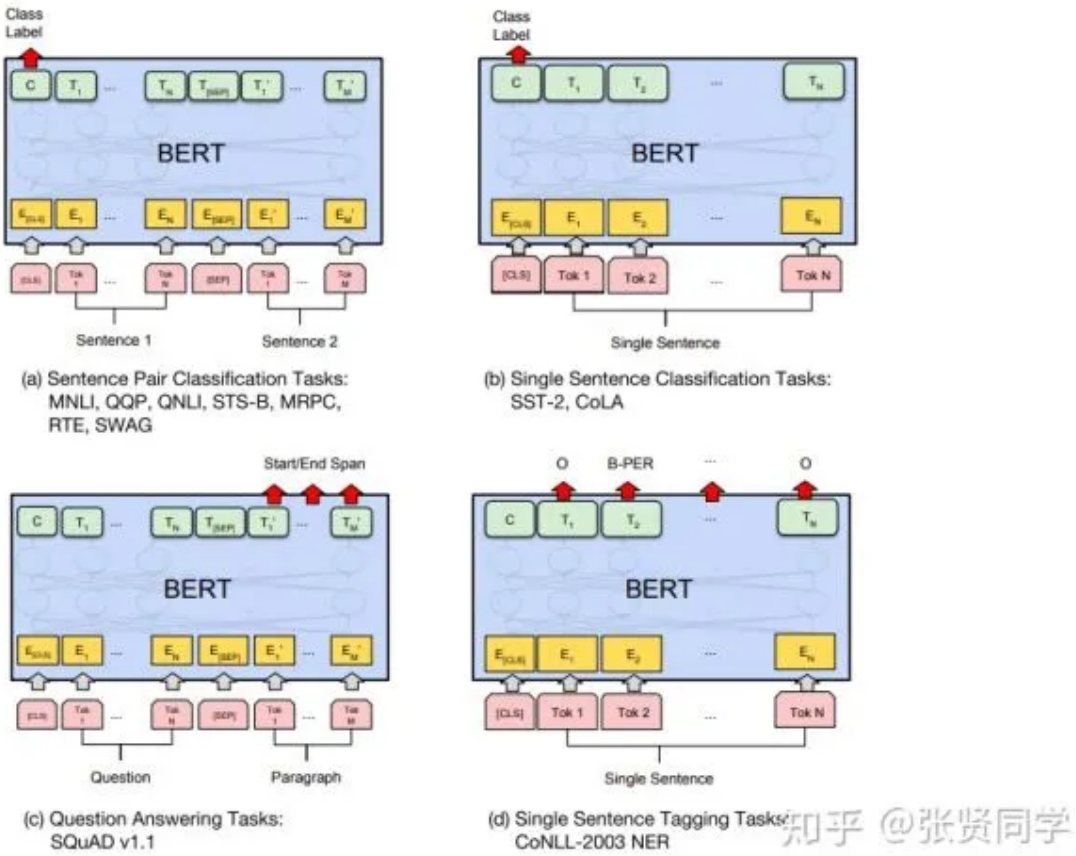
为了让 BERT 更好地处理多个句子之间的关系，预训练过程还包括一个额外的任务：给出两个句子（A 和 B），B 是否是 A 后面的相邻句子？



BERT 预训练的第 2 个任务是两个句子的分类任务。在此图中，tokenization 被简化了，因为 BERT 实际上使用了 WordPieces 作为 token，而不是使用单词本身 -- 因此有些词被拆分成了更小的块

特定任务的模型

BERT 的论文展示了将 BERT 用于多种任务的方法。



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

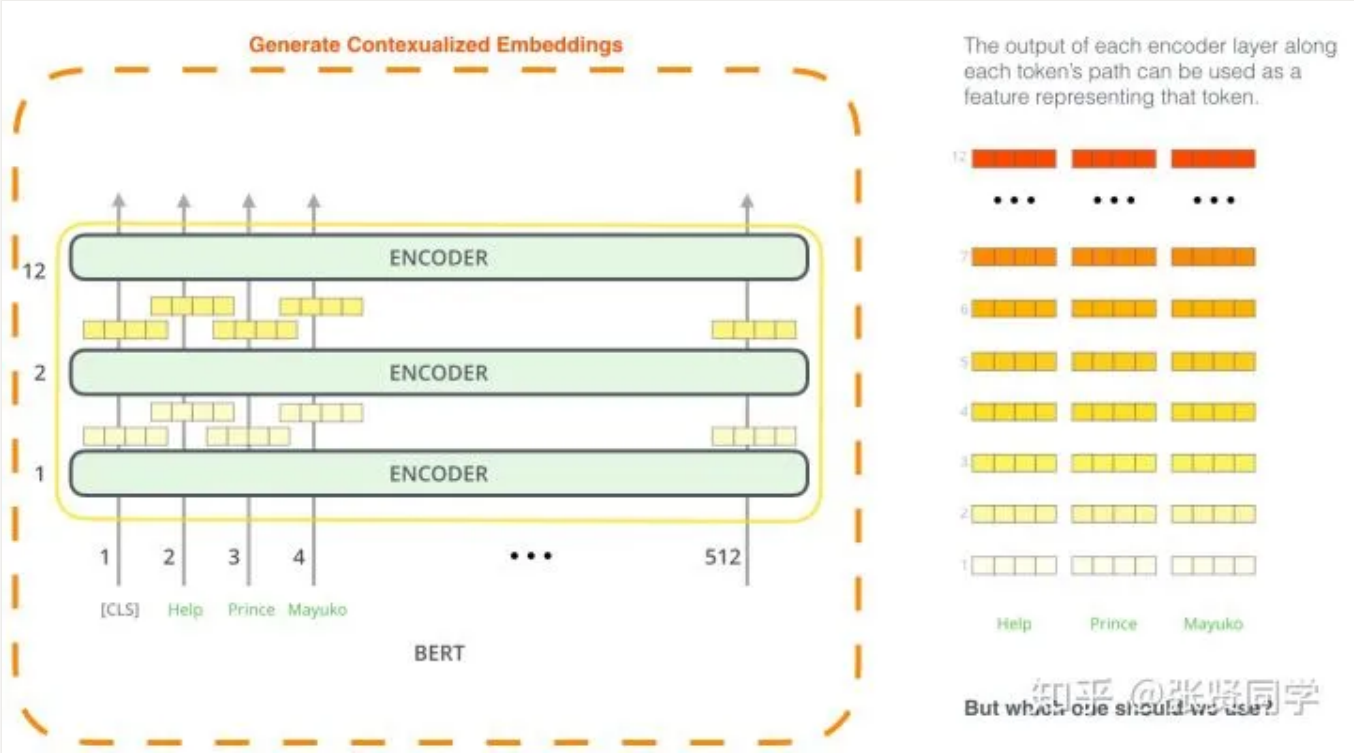
(b) Single Sentence Classification Tasks: SST-2, CoLA

(c) Question Answering Tasks: SQuAD v1.1

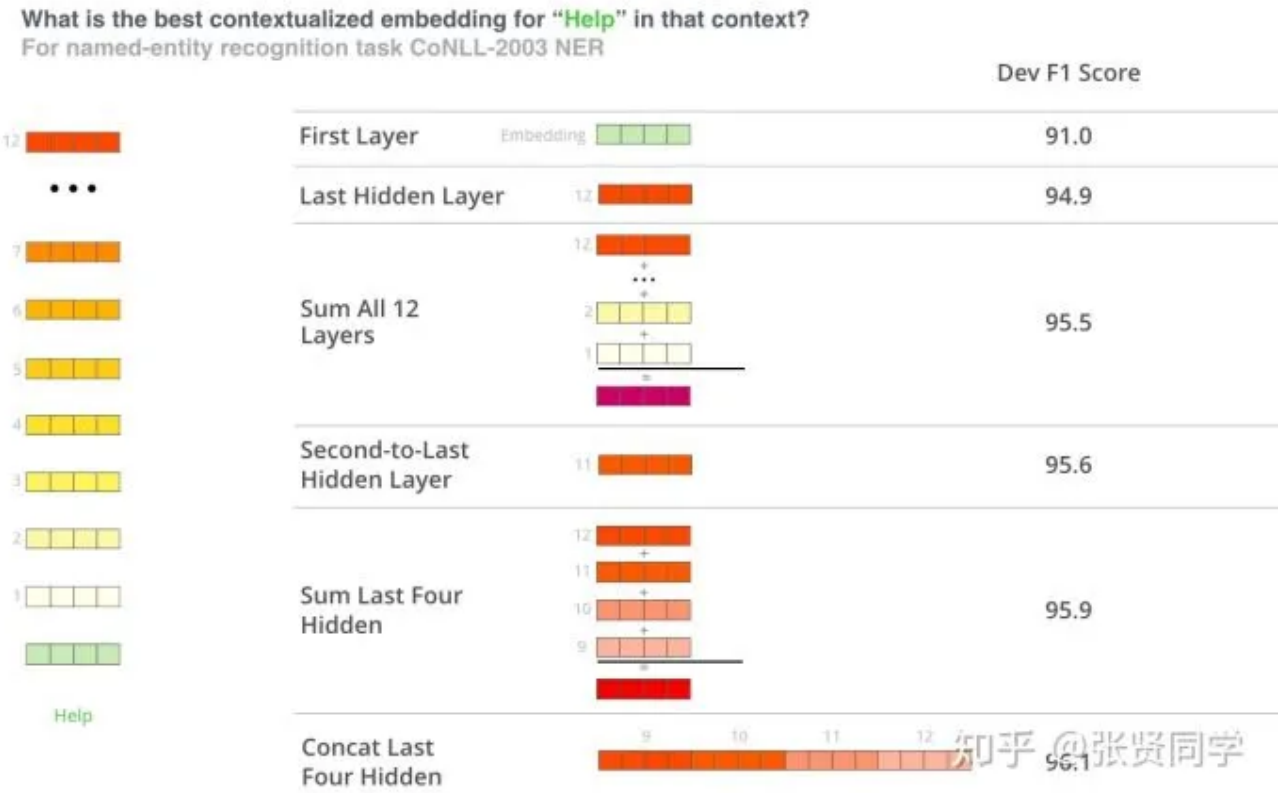
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

将 BERT 用于特征提取

微调的方法不是使用 BERT 的唯一方法。就像 ELMo，你可以使用预训练的 BERT 来创建语境化的词嵌入。然后你可以把这些词嵌入用到你现有的模型中 -- 论文里展示了这种方法在命名实体识别这样的任务中产生的结果，接近于微调的 BERT 产生的结果。



哪种向量最适合作为上下文词嵌入？我认为这取决于任务。论文里验证了 6 种选择（微调模型的分数为 96.4）：



如何使用 BERT

尝试 BERT 的最佳方式是通过托管在 Google Colab 上的 **BERT FineTuning with Cloud TPUs**。如果你之前从来没有使用过 Cloud TPU，那这也是一个很好的尝试开端，因为 BERT 代码可以运行在 TPU、CPU 和 GPU。

下一步是查看 **BERT 仓库** 中的代码：

- 模型是在 **modeling.py**（`class BertModel`）中定义的，和普通的 Transformer encoder 完全相同。
- **run_classifier.py** 是微调网络的一个示例。它还构建了监督模型分类层。如果你想构建自己的分类器，请查看这个文件中的 `create_model()` 方法。
- 可以下载一些预训练好的模型。这些模型包括 BERT Base、BERT Large，以及英语、中文和包括 102 种语言的多语言模型，这些模型都是在维基百科的数据上进行训练的。
- BERT 不会将单词作为 token。相反，它关注的是 WordPiece。**tokenization.py** 就是 tokenizer，它会将你的单词转换为适合 BERT 的 wordPiece。

致谢

感谢 **Jacob Devlin**、**Matt Gardner**、**Kenton Lee**、**Mark Neumann** 和 Matthew Peters](twitter.com/mattthemath) 为这篇文章的早期版本提供了反馈。

转自：机器学习算法与自然语言处理