机器学习与深度学习面试系列二十一（语言预训练模型）

语言模型的任务形式是什么？如何帮助提升其他NLP任务的效果?

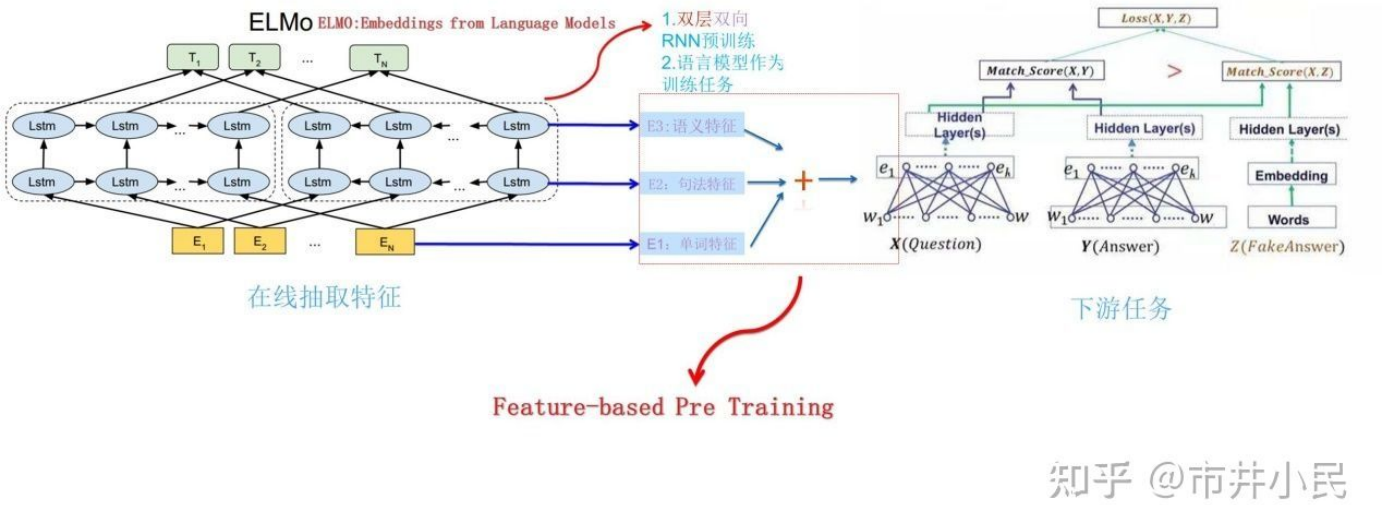
回想一个人类在婴儿阶段初学语言时，他们发现“妈”总是要和“妈”一起说，“爸”总是 要和“爸”一起说，才能完成对那两张最熟悉面孔的呼唤，随着成长，他们学会了短语，愚公总在移山，夸父永远逐日，女娲一生补天。渐渐地，他们又习得如何组成句子、篇章，“天若有情天亦老”的下句既可以是“月如无恨月长圆”，也可以是“人间正道是沧桑”。语言模型，从本质上讲便是要模拟人类学习语言的过程。从数学上讲，它是 一个概率分布模型，目标是评估语言中的任意一个字符串的生成概率 $P(S)$ ，其中 $S=(w_1,w_2,\ldots,w_n)$ 可以表示一个短语、句子、段落或文档， $w_i \in W$ ，W 为语言中所有词的集合(即词表)。利用条件概率， $P(S)$ 又可以表示为 $P(S)=P(w_1,w_2,\ldots,w_n)=P(w_1)\prod_{i=2}^{n}P(w_i|w_1,\ldots w_{i-1})$ ，语言模型的核心问题就是对 $P(w_i|w_1,\ldots w_{i-1})$ 建模。

传统的基于统计的语言模型引入了马尔可夫假设，认为任意一个词出现的概率只与它前面出 现的 N-1 个词有关，即 $P(w_i|w_1,\ldots w_{i-1})=P(w_i|w_{i-N+1},\ldots w_{i-1})$ ，这被称为 N 元(N-gram) 语言模型。实践中使用最多的是二元(bigram) 及三元(trigram)语言模型。深度学习普及后，更多基于神经网络的语言模型相继出现，突破了马尔可夫假设的限制，可以直接对 $P(w_i|w_1,\ldots w_{i-1})$ 建模，同时大幅提升了语言模型的效果。

语言模型是自然语言处理中的核心任务，一方面它可以评估语言的生成概率，直接用于生成符合人类认知的语言。另一方面由于语言模型的训练不依赖额外的监督信息，因此适合用于学习语句的通用语义表示。 目前非常普遍的做法就是利用语言模型在大规模公开语料库上预训练神经网络，然后在此基础上针对特定任务来对模型进行微调。比如我们之前所说的词嵌入也是这样的使用套路，通过word2vec、fastText等预训练模型得到word的词向量，然后到具体任务中进行使用。但是我们之前文章中介绍的词嵌入方法，均为静态化方法，也就是对于一个单词它的词向量就固定下来了，对于多义词的表达不够准确。例如bank，既有银行的意思，又有河岸的意思，如果在所有句子中都用同一个词向量来表达，显然是不合适的。解决这个多义词的问题，需要使用“动态词向量”，就是对一个词嵌入的生成应当考虑其上下文语义环境，同一个word在不同的语义环境中有不同的词向量。其中比较具有代表性的模型包括：ELMO(Embeddings from Language Model)、GPT(Generative Pre-trained Transformer)、Bert(Bidirectional Encoder Representations from Transformers)。

ELMO模型是怎样的?

ELMO：训练好之后如何使用？



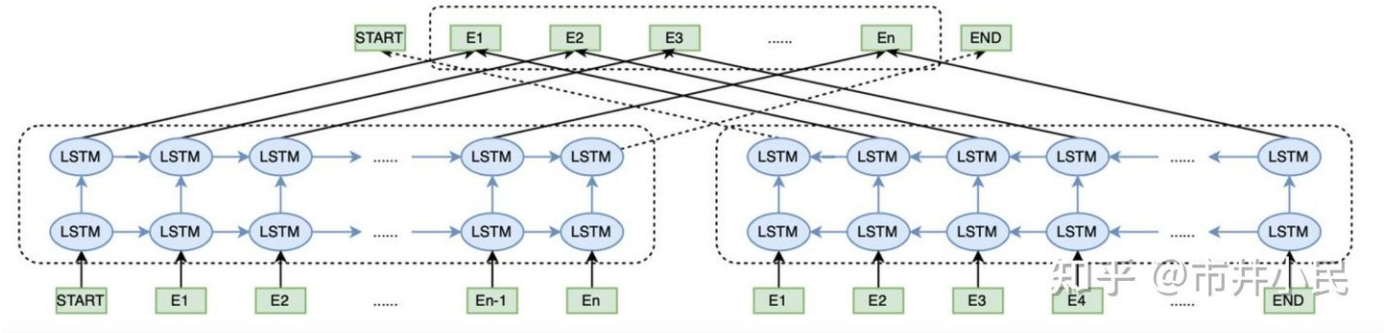
ELMO采用了典型的两阶段过程，第一个阶段是利用语言模型进行预训练；第二个阶段是在做下游任务时，从预训练网络中提取对应单词的网络各层的Word Embedding作为新特征补充到下游任务中。

上图左侧展示的是其预训练过程，它的网络结构采用了双层双向LSTM，目前语言模型训练的任务目标是根据单词 W_i 的上下文去正确预测单词 W_i ， W_i 之前的单词序列Context-before称为上文，之后的单词序列Context-after称为下文。图中左端的前向双层LSTM代表正方向编码器，输入的是从左到右顺序的除了预测单词外 W_i 的上文Context-before；右端的逆向双层LSTM代表反方向编码器，输入的是从右到左的逆序的句子下文Context-after；每一层的编码都是这一层Context-before编码和Context-after编码的拼接。使用这个网络结构利用大量语料做语言模型任务就能预先训练好这个网络，如果训练好这个网络后，输入一个新句子 $Snew$ ，句子中每个单词都能得到对应的三个Embedding：最底层是单词的Word Embedding，往上走是第一层双向LSTM中对应单词位置的Embedding，这层编码单词的句法信息更多一些；再往上走是第二层LSTM中对应单词位置的Embedding，这层编码单词的语义信息更多一些。

下游任务的使用过程，比如上图中右侧我们的下游任务是QA问题，此时对于问句X，我们可以先将句子X作为预训练好的ELMO网络的输入，这样句子X中每个单词在ELMO网络中都能获得对应的三个Embedding，之后给予这三个Embedding中的每一个Embedding一个权重a，这个权重由下游特定任务学习得来，根据各自权重累加求和，将三个Embedding整合成一个。然后将整合后的这个Embedding作为X句在自己任务的那个网络结构中对应单词的输入，以此作为补充的新特征给下游任务使用。

ELMO预训练的目标是什么？

将ELMO输出的向量映射到 vocab_size的长度，softmax后，取出概率最大的元素对应的下标，作为对下一个字的预测。相当于做一个分类，类别数量是词表大小。



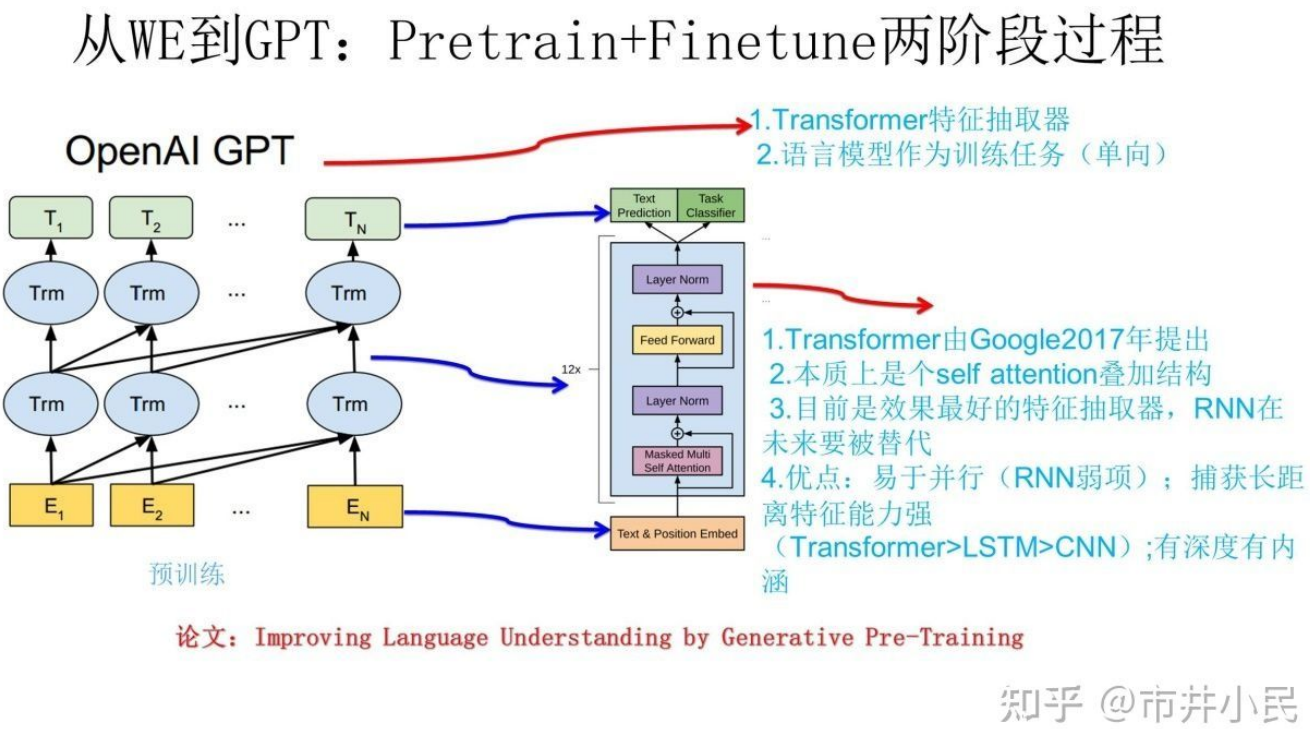
为什么ELMO用两个单向的LSTM代替一个双向的LSTM呢?

用双向的模型结构去训练语言模型会导致“看到自己”或“看到答案”的问题。例如在正向网络中，模型的训练目标就是要得到下一个位置的词，如果引入了反向，那模型就已经“看到了”下一个位置的词。

ELMO模型有什么局限性?

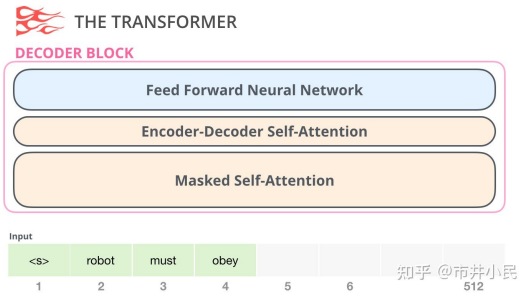
1. 特征抽取器选择方面，ELMO使用了LSTM而不是新贵Transformer，很多研究已经证明了Transformer提取特征的能力是要远强于LSTM的。如果ELMO采取Transformer作为特征提取器，那么估计Bert的反响远不如现在的这种火爆场面。
2. ELMO采取双向拼接这种融合特征的能力可能比Bert一体化的融合特征方式弱，但是，这只是一种从道理推断产生的怀疑，目前并没有具体实验说明这一点。

GPT模型是怎样的?



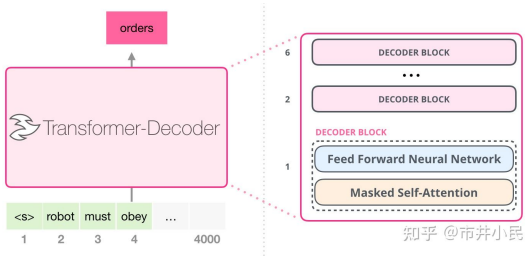
GPT是“Generative Pre-Training”的简称，从名字看其含义是指的生成式的预训练。GPT也采用两阶段过程，第一个阶段是利用语言模型进行预训练，第二阶段通过Fine-tuning的模式解决下游任务。上图展示了GPT的预训练过程，其实和ELMO是类似的，主要不同在于两点：

- 特征抽取器不是用的RNN，而是用的Transformer，它的特征抽取能力要强于RNN
- GPT的预训练虽然仍然是以语言模型作为目标任务，但是采用的是单向的语言模型。ELMO在做语言模型预训练的时候，预测单词 W_i 同时使用了上文和下文，而GPT则只采用Context-before这个单词的上文来进行预测，而抛开了下文。这个选择现在看不是个太好的选择，原因很简单，它没有把单词的下文融合进来，这限制了其在更多应用场景的效果，比如阅读理解这种任务，在做任务的时候是可以允许同时看到上文和下文一起做决策的。如果预训练时候不把单词的下文嵌入到Word Embedding中，是很吃亏的，白白丢掉了很多信息。

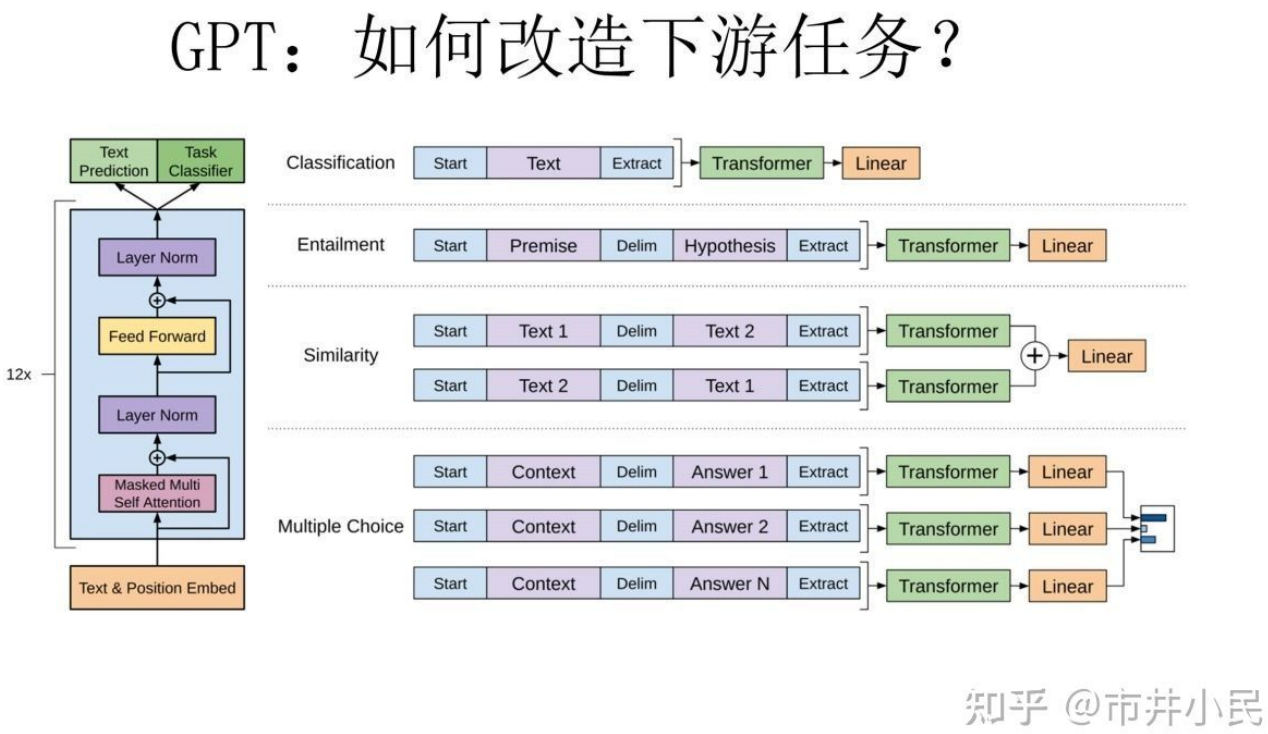


GPT的训练目标与ELMO一致，即预测下一个位置的word。GPT模型的整体结构可以为Transformer的Decoder，只是放弃了第二个自我注意层(*Encoder-Decoder Self-Attention*)。如上图所示，Decoder测一个block内有两个自注意力层，其中第一个自注意力的输入是将输入句子当前位置后面部分进行masked，来“欺骗”模型，防止模型“看到”下一位置词的正确答案（与ELMO中使用两个单向的LSTM是解决同一个问题，只是这里只考虑上文不考虑下文，所以仅需要遮盖后面

的词）。第二个自注意力层是从Encoder侧得到的结果计算Q和K，这里没有Encoder，直接放弃。[1]



GPT模型怎么迁移到下游任务?



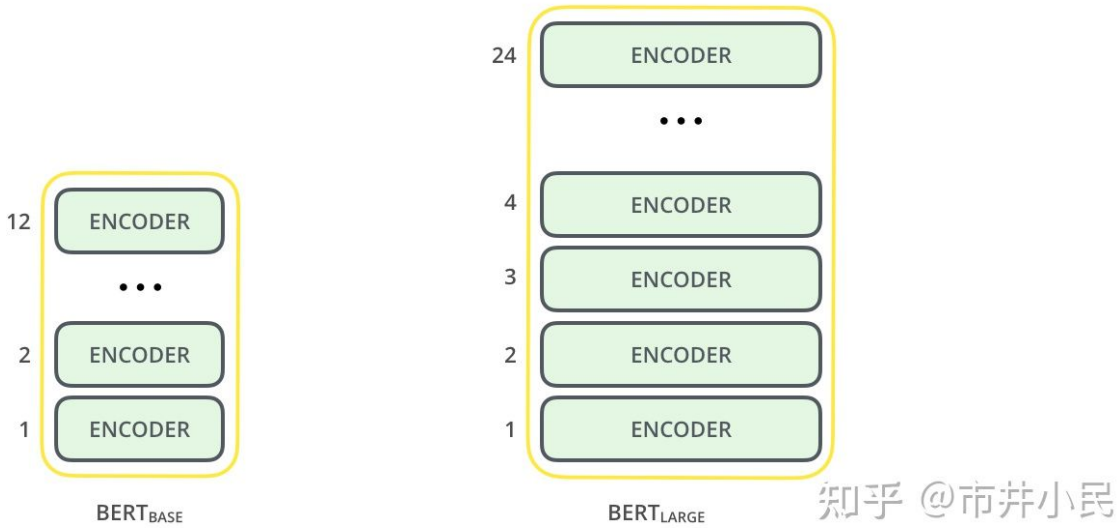
GPT原论文[2]给了一个改造施工图如上，其实也很简单：对于分类问题，不用怎么动，加上一个起始和终结符号即可；对于句子关系判断问题，比如Entailment，两个句子中间再加个分隔符即可；对文本相似性判断问题，把两个句子顺序颠倒下做出两个输入即可，这是为了告诉模型句子顺序不重要；对于多项选择问题，则多路输入，每一路把文章和答案选项拼接作为输入即可。从上图可看出，这种改造还是很方便的，不同任务只需要在输入部分施工即可。

GPT的缺陷?

GPT是一个单向语言模型，没有考虑下文对词向量的影响。

Bert模型是怎样的?

Bert采用和GPT完全相同的两阶段模型，首先是语言模型预训练；其次是使用Fine-Tuning模式解决下游任务。和GPT的最主要不同在于在预训练阶段采用了类似ELMO的双向语言模型，当然另外一点是语言模型的数据规模要比GPT大。Bert模型可以看作是Transformer的Encoder侧，基本版本有12层Encoder Block，large版本有24层Encoder Block。[3]



Bert模型的训练目标是什么?

根据我们在GPT中所说，如果我们的目标是预测下一个位置的词，那就需要对当前位置后面的词做遮盖，这样就不能满足Bert想要的双向语言模型要求了，如果采用ELMO中的思路，那应该train两个 transformer 的Encoder，一个负责正向，一个负责反向。但是Bert没有采取这种方式，而是换了个训练目标：Masked 语言模型和Next Sentence Prediction。

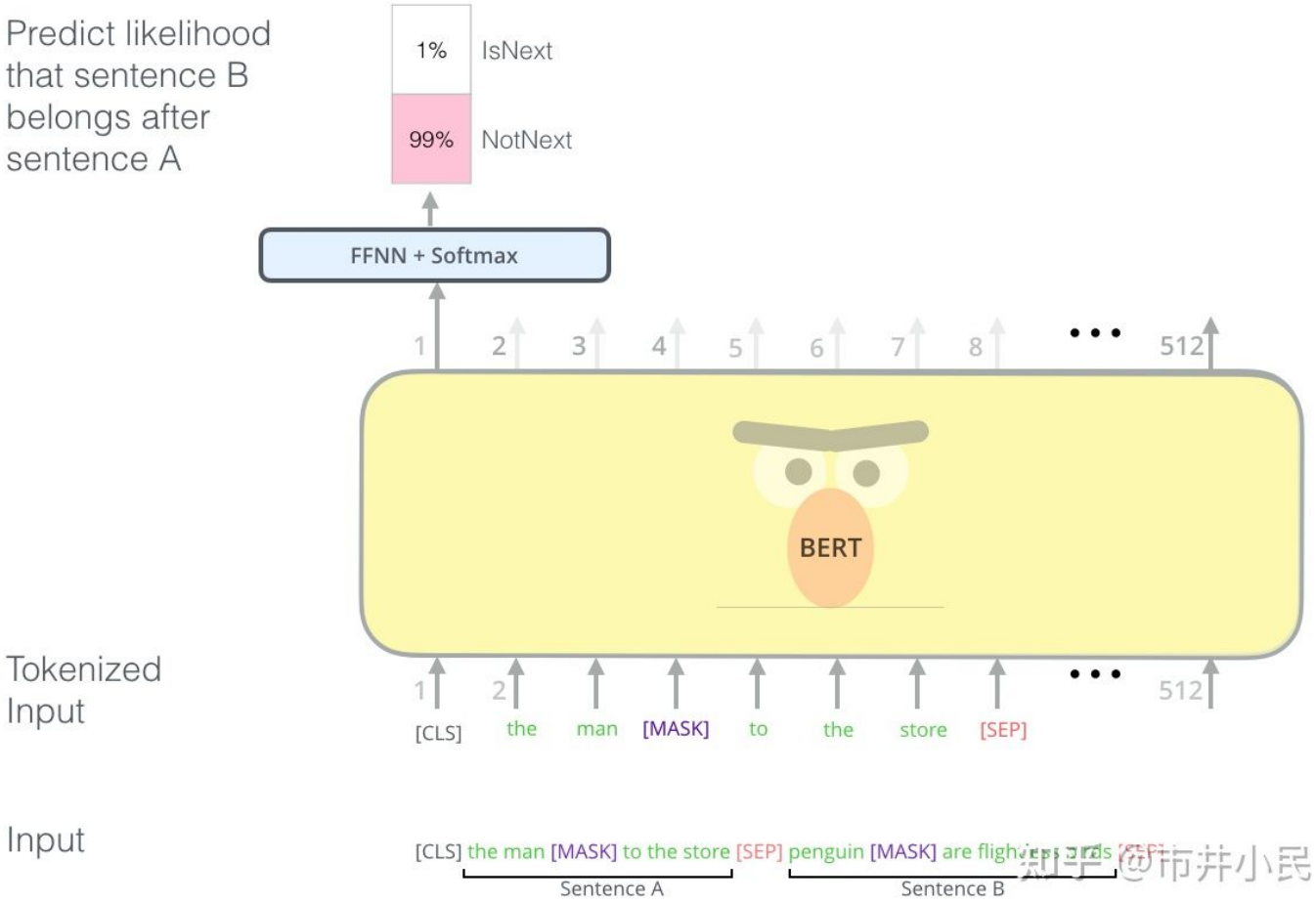
Masked 语言模型

Masked语言模型上面讲了，本质思想其实和word2vec的CBOW模型类似，就是做完形填空，但是细节方面有改进。随机选择语料中15%的单词，把它抠掉，也就是用[Mask]掩码代替原始单词，然后要求模型去正确预测被抠掉的单词。但是这里有个问题：训练过程大量看到[mask]标记，但是真正后面用的时候是不会有这个标记的，这会引导模型认为输出是针对[mask]这个标记的，但是实际使用又见不到这个标记，这自然会有问题。为了避免这个问题，Bert改造了一下，15%的被上天选中要执行[mask]替身这项光荣任务的单词中，只有80%真正被替换成[mask]标记，10%被狸猫换太子随机替换成另外一个单词，10%情况这个单词还待在原地不做改动。这就是Masked双向语音模型的具体做法。

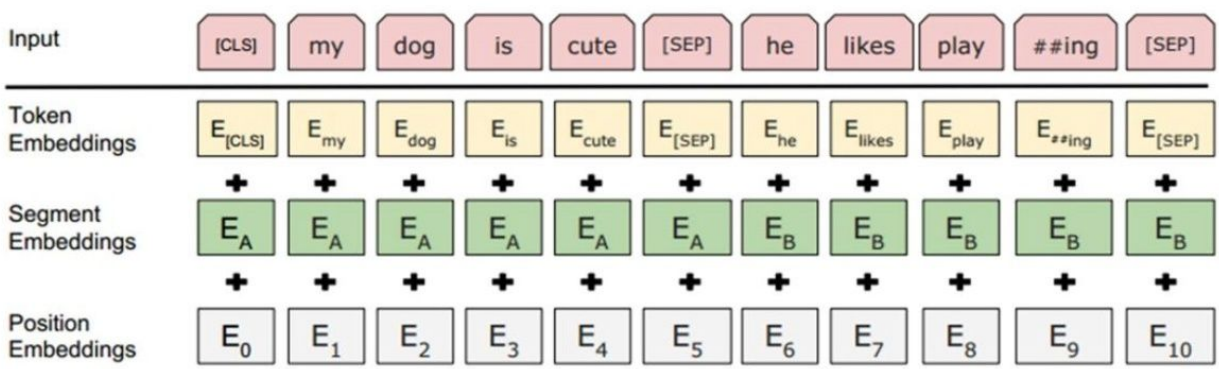
Next Sentence Prediction

做语言模型预训练的时候，分两种情况选择两个句子，一种是选择语料中真正顺序相连的两个句子；另外一种是第二个句子从语料库中抛色子，随机选择一个拼到第一个句子后面。我们要求模型除了做上述的Masked语言模型任务外，附带再做个句子关系预测，判断第二个句子是不是真的是

第一个句子的后续句子。之所以这么做，是考虑到很多NLP任务是句子关系判断任务，单词预测粒度的训练到不了句子关系这个层级，增加这个任务有助于下游句子关系判断任务。所以可以看到，它的预训练是个多任务过程。这也是Bert的一个创新。



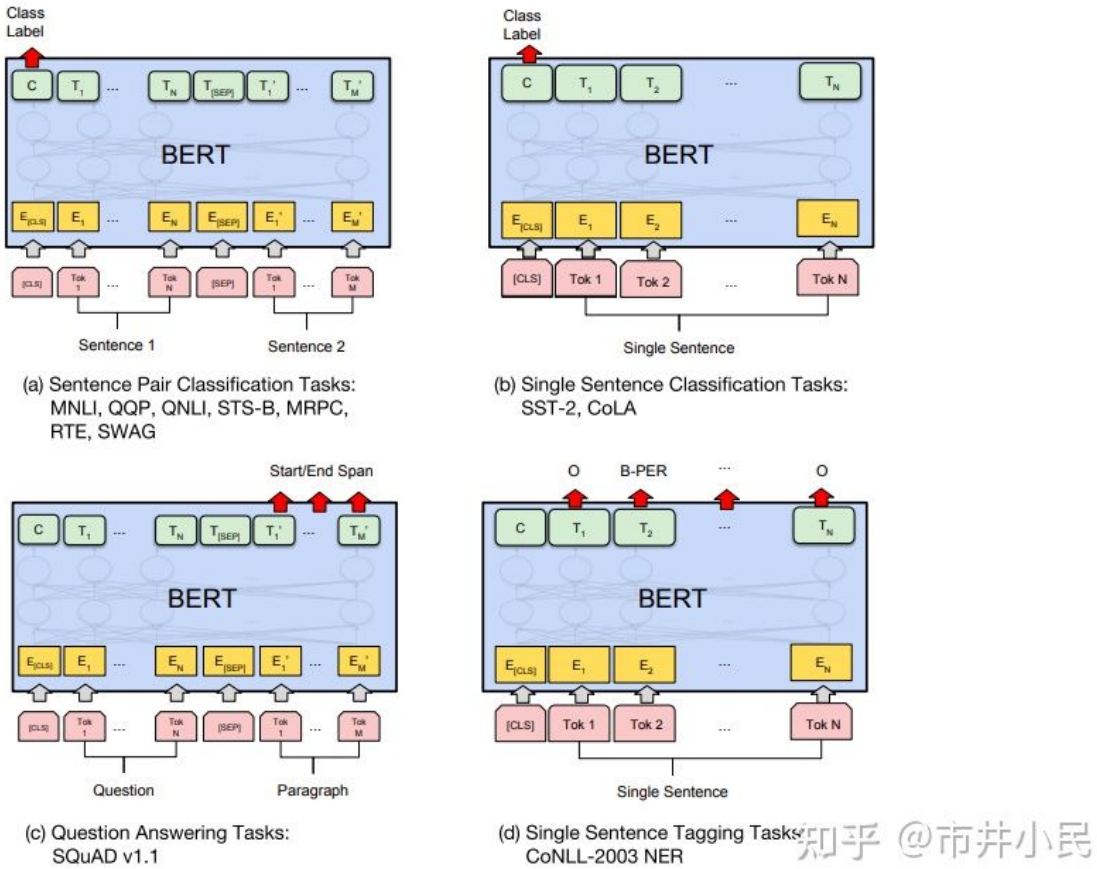
Bert：输入部分的处理



Bert的输入是个序列，两个句子通过分隔符分割，最前面和最后增加两个标识符号。每个单词有三个embedding：

- 位置信息embedding，这是transformer中自带的特性，因为NLP中单词顺序是很重要的特征，需要在这里对位置信息进行编码
- 单词embedding
- 第三个是句子embedding，因为训练数据都是由两个句子构成的，那么每个句子有个句子整体的embedding项对应给每个单词。把单词对应的三个embedding叠加，就形成了Bert的输入。

Bert模型如何迁移到下游任务?



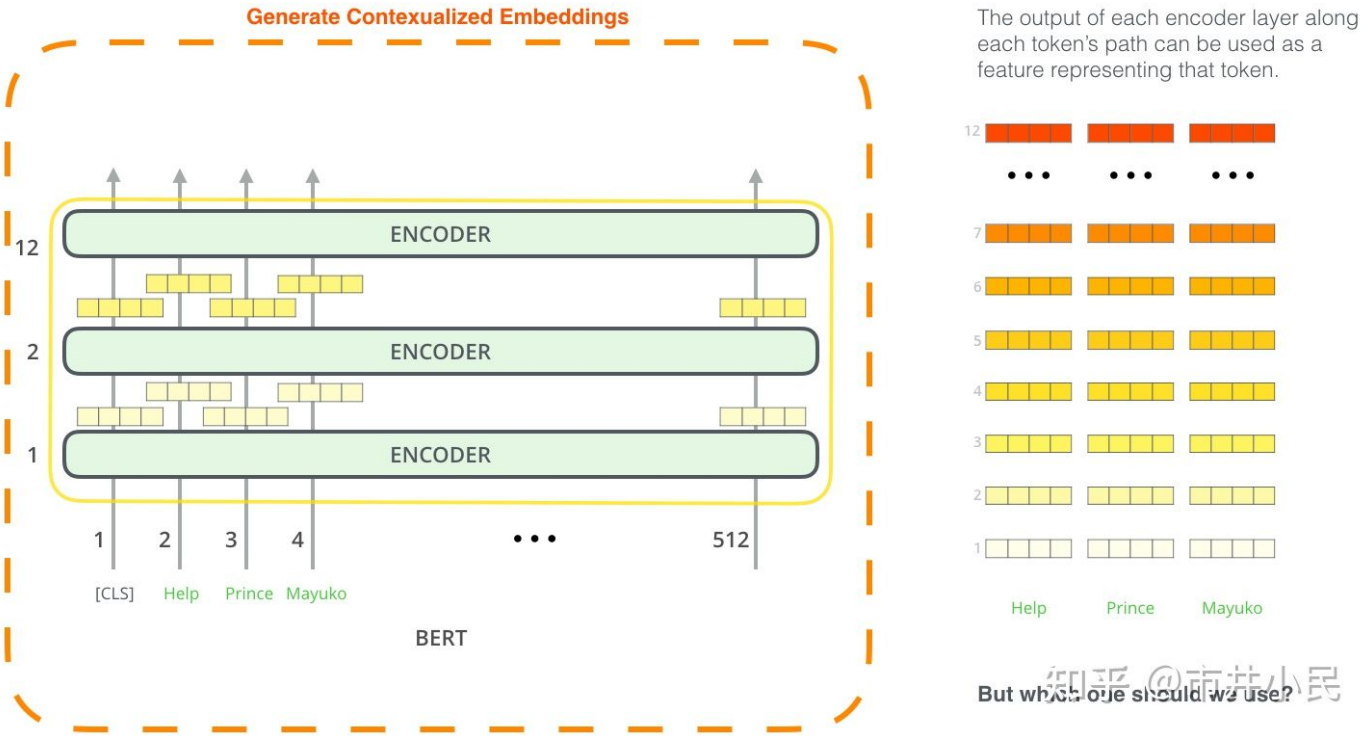
NLP主要的四类任务Bert都可以很好的迁移支持：

1. **序列标注**，这是最典型的NLP任务，比如中文分词，词性标注，命名实体识别，语义角色标注等都可以归入这一类问题，它的特点是句子中每个单词要求模型根据上下文都要给出一个分类类别。
2. **分类任务**，比如我们常见的文本分类，情感计算等都可以归入这一类。它的特点是不管文章有多长，总体给出一个分类类别即可。
3. **句子关系判断**，比如Entailment，QA，语义改写，自然语言推理等任务都是这个模式，它的特点是给定两个句子，模型判断出两个句子是否具备某种语义关系；
4. **生成式任务**，比如机器翻译，文本摘要，写诗造句，看图说话等都属于这一类。它的特点是输入文本内容后，需要自主生成另外一段文字。

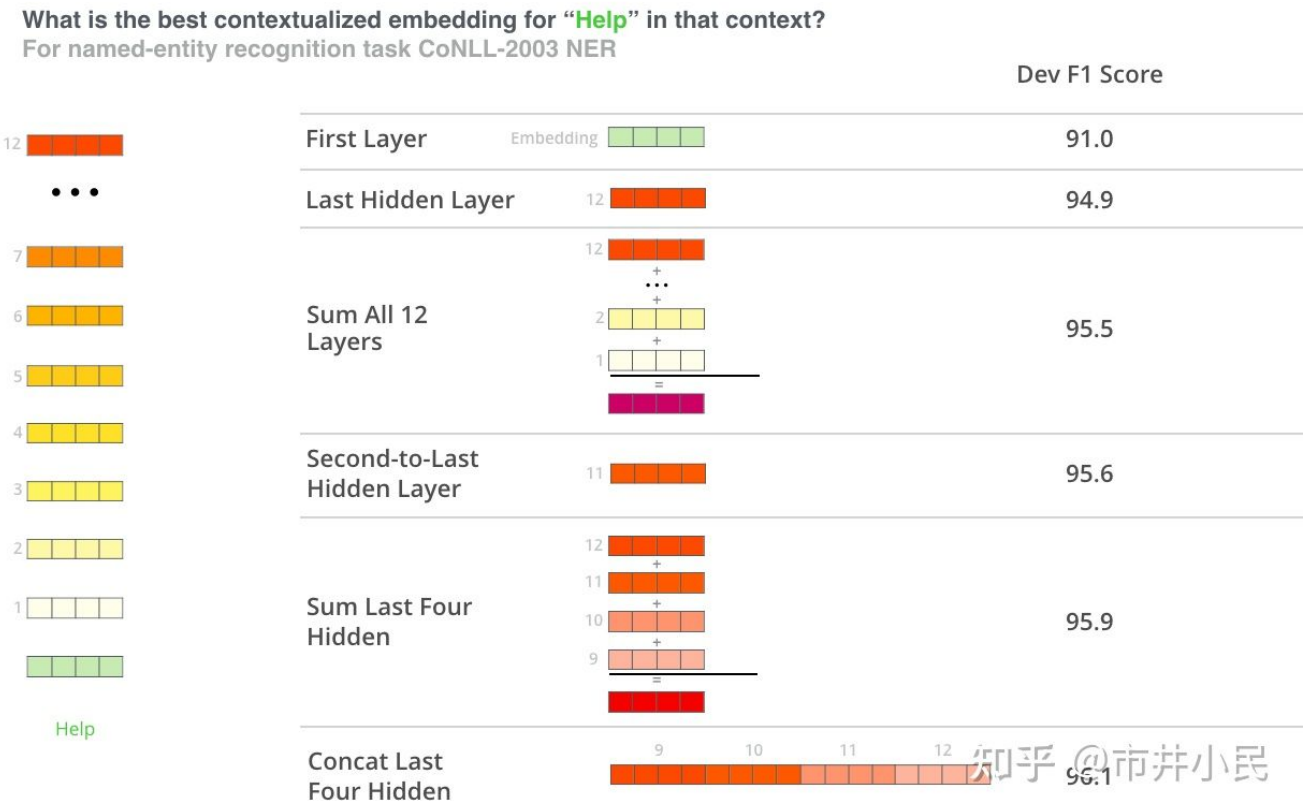
对于这四类下游NLP任务，Bert都可以改造输入输出部分就可以通过Bert预训练好的模型参数来支持：

1. 句子关系类任务，和GPT类似，加上一个起始和终结符号，句子之间加个分隔符即可。对于输出来说，把第一个起始符号对应的Transformer最后一层位置上面串接一个softmax分类层即可。
2. 对于分类问题，与GPT一样，只需要增加起始和终结符号，输出部分和句子关系判断任务类似改造；
3. 对于序列标注问题，输入部分和单句分类是一样的，只需要输出部分Transformer最后一层每个单词对应位置都进行分类即可。
4. 生成类任务外。尽管Bert论文没有提，其实对于机器翻译或者文本摘要，聊天机器人这种生成式任务，同样可以稍作改造即可引入Bert的预训练成果。只需要附着在S2S结构上，encoder部分是个深度Transformer结构，decoder部分也是个深度Transformer结构。根据任务选择不同的预训练数据初始化encoder和decoder即可。这是相当直观的一种改造方法。当然，也可以更简单一点，比如直接在单个Transformer结构上加装隐层产生输出也是可以的。

Bert模型如何用于特征提取?



微调不是Bert的唯一使用方法，我们也可以利用Bert训练得到的词嵌入直接应用到别的模型中。这里就有一个问题，Bert模型有12或24层，这么多层的隐藏层输入，我们到底应该用哪一层呢？这其实是取决于具体任务的，因为不同的任务对词向量的抽象表示层次要求不同，越低层的输出越偏向于句法的抽象表示，越高层越偏向于语义的表示。到底应该选择哪一层，个人觉得就是个哲学问题(手动捂脸)，得靠猜，又不是瞎猜，是有启发性的猜，这可能就是经验吧。



例如在这个任务中，将最后四层隐节点的输出作为词向量的表达效果最好。

Bert模型如何得到句向量?

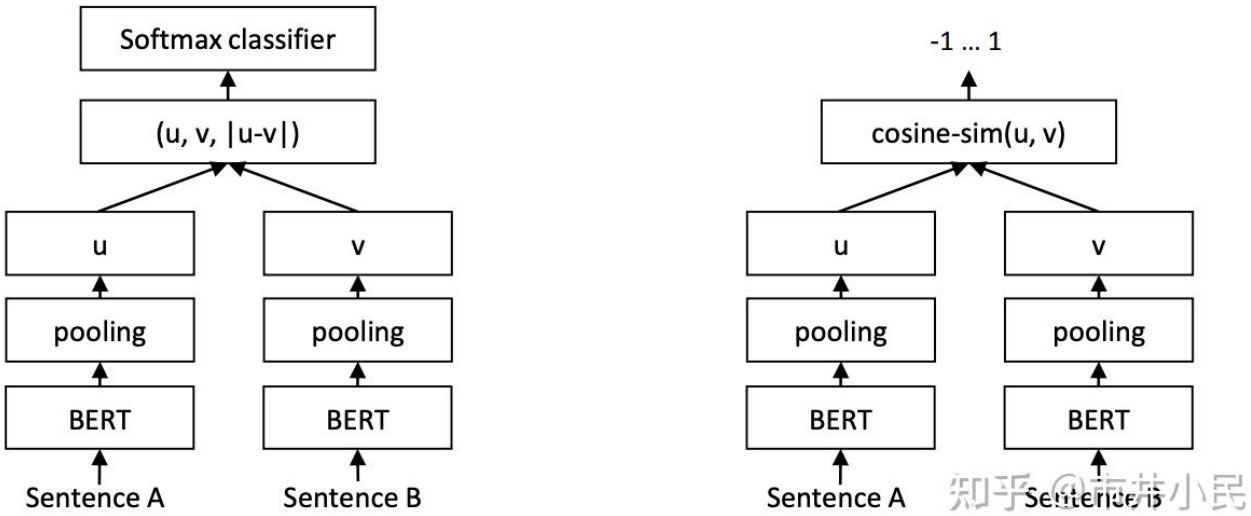
基本方法有两种：

1. 从句首的[CLS]得到的向量直接作为句向量。
2. 各个词向量做平均或最大，当作句向量，通常为平均。

还有一些传统的句向量生成方法，如：

- 直接取句中所有词的 GloVe embeddings，做平均
- InferSent - Glove，输入是一对句子，分别是前提 premise 与假设 hypothesis，然后将得到的句子向量通过一些向量操作后得到句子对的混合语义特征，最后接上全连接层并做数据集上的三分类任务 (entailment 蕴含, contradiction 矛盾, neutral 中性) 。
- Universal Sentence Encoder。这是Google 于 2018 年初发布的通用句子编码器，综合利用无监督训练数据和有监督训练数据，进行多任务训练，从而学习一个通用的句子编码器。

选择什么句向量方法，取决于特定的任务。在一些具体的任务如句子相似度比较上，上述方法的表现均不太好。所以在2019年德国达姆施塔特工业大学提出了一种基于Bert的句向量的方法：基于孪生网络的BERT网络[4](SBERT)。它从Bert得到词向量后，先取平均作为句向量，然后放到分类任务和语义相似度任务中继续做微调，得到更优的句向量。

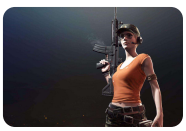


在很多任务上，SBERT的表现都超过了上述方法。值得一提的是，没有免费午餐定理告诉我们，机器学习与深度学习中没有一种模型适用于所有问题，在下面任务中SBERT的句向量表达得到了很好的效果，但是它所附带的词向量在很多场景中却不如原生的BERT。

Model	STS12	STS13	STS14	STS15	STS16	STSb	SICK-R	Avg.
Avg. GloVe embeddings	55.14	70.66	59.73	68.25	63.66	58.02	53.76	61.32
Avg. BERT embeddings	38.78	57.98	57.98	63.15	61.06	46.35	58.40	54.81
BERT CLS-vector	20.16	30.01	20.09	36.88	38.08	16.50	42.63	29.19
InferSent - Glove	52.86	66.75	62.15	72.77	66.87	68.03	65.65	65.01
Universal Sentence Encoder	64.49	67.80	64.61	76.83	73.18	74.92	**76.69**	71.22
SBERT-NLI-base	70.97	76.53	73.19	79.09	74.30	77.03	72.91	74.89
SBERT-NLI-large	72.27	**78.46**	**74.90**	80.99	76.25	**79.23**	73.75	76.55
SRoBERTa-NLI-base	71.54	72.49	70.80	78.74	73.69	77.77	74.46	74.21
SRoBERTa-NLI-large	**74.53**	77.00	73.18	**81.85**	**76.82**	79.10	74.29	**76.68**

本文重点参考了 @张俊林 大佬的文章，整篇文章行云流水，高屋建瓴。

张俊林：从Word Embedding到Bert模型—自然语言处理中的预训练技术...
zhuanlan.zhihu.com



参考

1. ^ https://jalammar.github.io/illustrated-gpt2/
2. ^ https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf
3. ^ http://jalammar.github.io/illustrated-bert/
4. ^ https://arxiv.org/pdf/1908.10084.pdf