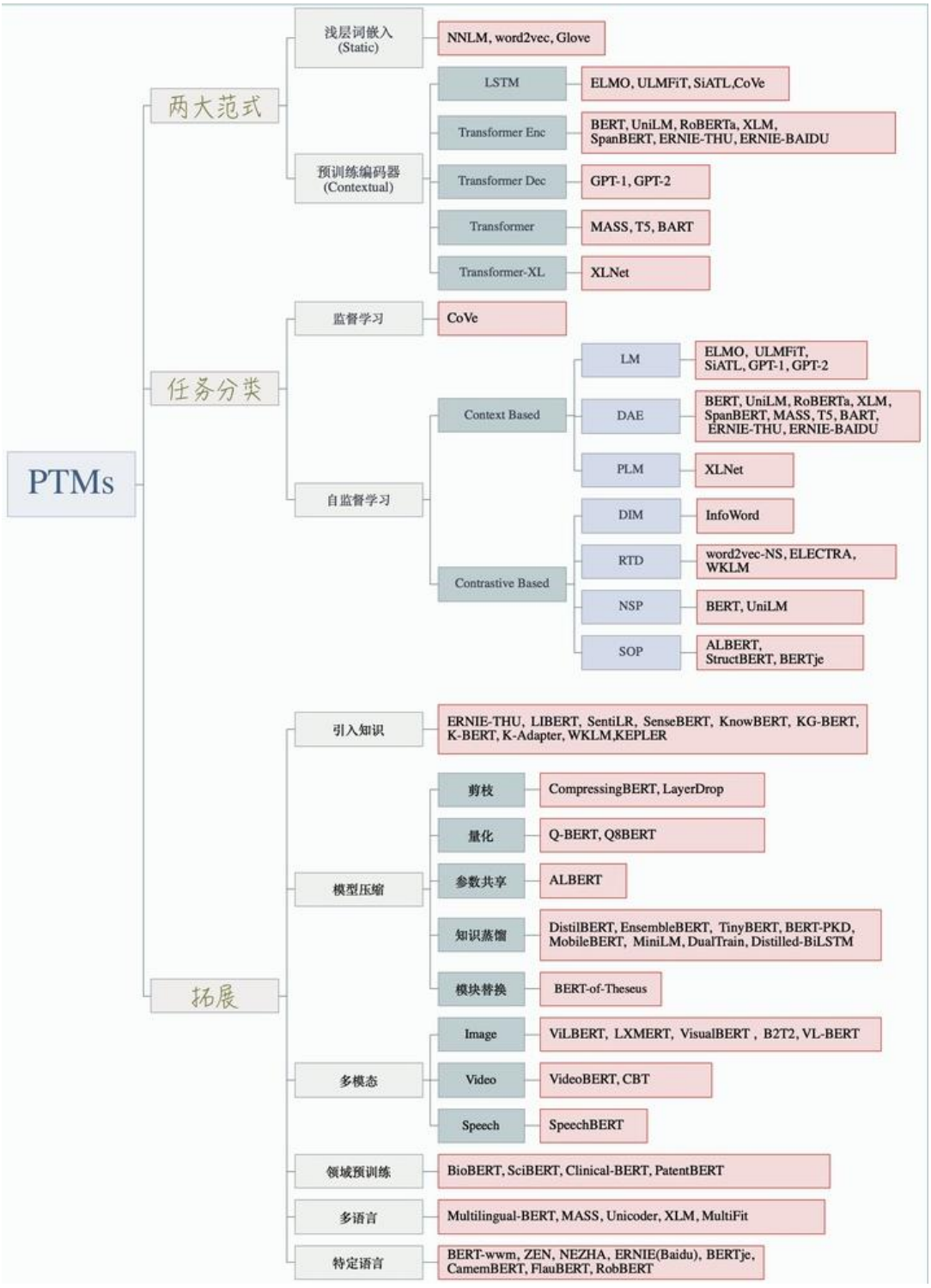
目录

一、PTM



1.为什么要进行预训练？

深度学习时代，为了充分训练深层模型参数并防止过拟合，通常需要更多标注数据喂养。在 NLP 领域，标注数据更是一个昂贵资源。PTMs 从大量无标注数据中进行预训练使许多 NLP 任务获得显著的性能提升。总的来看，预训练模型 PTMs 的优势包括：

在庞大的无标注数据上进行预训练可以获取更通用的语言表示，并有利于下游任

务；
为模型提供了一个更好的初始化参数，在目标任务上具备更好的泛化性能、并加速收敛；
是一种有效的正则化手段，避免在小数据集上过拟合（一个随机初始化的深层模型容易对小数据集过拟合）；

2.什么是词嵌入和分布式表示？PTMs 与分布式表示的关系？

词嵌入是自然语言处理（NLP）中语言模型与表征学习技术的统称。概念上而言，它是指把一个维数为所有词的数量的高维空间嵌入到一个维数低得多的连续向量空间中，每个单词或词组被映射为实数域上的向量，这也是分布式表示：向量的每一维度都没有实际意义，而整体代表一个具体概念。
分布式表示相较于传统的独热编码（one-hot）表示具备更强的表示能力，而独热编码存在维度灾难和语义鸿沟（不能进行相似度计算）等问题。传统的分布式表示方法，如矩阵分解（SVD/LSA）、LDA 等均是根据全局语料进行训练，是机器学习时代的产物。
PTMs 也属于分布式表示的范畴，本文的 PTMs 主要介绍深度学习时代、自 NNLM[2] 以来的“modern”词嵌入。

3.PTMs 有哪两大范式？对比不同的预训练编码器？

PTMs 的发展经历从浅层的词嵌入到深层编码两个阶段，按照这两个主要的发展阶段，我们归纳出 PTMs 两大范式：「浅层词嵌入」和「预训练编码器」。
3.1 浅层词嵌入（ Non-Contextual Embeddings）
浅层词嵌入，这一类 PTMs 范式是我们通常所说的“词向量”，其主要特点是学习到的是上下文独立的静态词嵌入，其主要代表为 NNLM[2]、word2vec（CBOW[3]、Skip-Gram[3]）、Glove[4]等。这一类词嵌入通常采取浅层网络进行训练，而应用于下游任务时，整个模型的其余部分仍需要从头开始学习。因此，对于这一范式的 PTMs 没有必要采取深层神经网络进行训练，采取浅层网络加速训练也可以产生好的词嵌入[3]。
浅层词嵌入的主要缺陷为：
词嵌入与上下文无关，每个单词的嵌入向量始终是相同，因此不能解决一词多义的问题。
通常会出现 OOV 问题，为了解决这个问题，相关文献提出了字符级表示或 sub-word 表示，如 CharCNN[5] 、FastText[6] 和 Byte-Pair Encoding [7]。

词嵌入	训练目标	全局/局部语料	特点
NNLM	语言模型	局部语料	基于语言模型进行训练的，词嵌入只不过是NNLM的一个产物而已；
word2vec	非语言模型（窗口上下文）	局部语料	1)为加速训练舍弃NNLM中的隐藏层、词嵌入直接sum; 2) 采用分层SoftMax(带权路径最小的哈夫曼树) 和负采样 进行运算优化; 3)损失函数：带权重的交叉熵，权重固定；
Glove	非语言模型（词共现矩阵）	全局语料	1) 基于全局语料构建词共现矩阵然后进行高效矩阵分解的方法; 2)损失函数：最小平方损失函数，权重可以做映射变换。

图 1: 常见的 3 种浅层词嵌入对比：NNLM、word2vec、Glove
图 1 给出了三种常见的浅层词嵌入之间的对比，Glove 可以被看作是更换了目标函数和权重函数的全局 word2vec。此外，相关文献也提出了句子和文档级别的嵌入方式，如 Skip-thought[8] 、Context2Vec[9] 等。
3.2 预训练编码器（Contextual Embeddings）

第二类 PTMs 范式为预训练编码器，主要目的是通过一个预训练的编码器能够输出上下文相关的词向量，解决一词多义的问题。这一类预训练编码器输出的向量称之为「上下文相关的词嵌入」。

编码器	PTMs代表	计算方式	特点
MLP	NNLM/word2vec	前馈+并行	不考虑序列（位置）信息，不能处理变长序列；
CNNs		前馈+并行	考虑序列（位置）信息，不能处理长距离依赖，聚焦于n-gram的局部上下文编码，pooling操作会导致序列（位置）信息丢失；
RNNs	ELMO	循环+串行	天然适合处理序列（位置）信息，但仍不能处理长距离依赖（由于BPTT导致的梯度消失等问题），故又称之为“较长的短期记忆单元(LSTM)”
Transformer	GPT（Decoder） BERT（Encoder）	前馈+并行	1）self-attention解决长距离依赖，无位置偏差； 2）self-attention可看作是权重动态调整的全连接网络；
Transformer-XL	XLNet	循环+串行	基于Transformer引入循环机制+相对位置编码，增强长距离建模能力；
长距离依赖建模能力			Transformer-XL > Transformer > RNNs > CNNs

PTMs 中预训练编码器通常采用 LSTM 和 Transformer（Transformer-XL），其中 Transformer 又依据其 attention-mask 方式分为 Transformer-Encoder 和 Transformer-Decoder 两部分。此外，Transformer 也可看作是一种图神经网络 GNN[10]。

这一类「预训练编码器」范式的 PTMs 主要代表有 ELMO[11]、GPT-1[12]、BERT[13]、XLNet[14]等。

4.PTMs 按照任务类型如何分类？

PTMs 按照任务类型可分为 2 大类：监督学习 和 无监督学习/自监督学习。

监督学习在 NLP-PTMs 中的主要代表就是 CoVe[15]，CoVe 作为机器翻译的 encoder 部分可以应用于多种 NLP 下游任务。除了 CoVe 外，NLP 中的绝大多数 PTMs 属于自监督学习。

综合各种自监督学习的分类方式，笔者将 NLP-PTMs 在自监督学习中分为两种类型：基于上下文（Context Based）和基于对比（Contrastive Based）。

4.1 基于上下文（Context Based）

基于上下文的 PTMs，主要基于数据本身的上下文信息构造辅助任务，在 NLP 中我们通常引入语言模型作为训练目标。PTMs 中的语言模型主要分为三大类：

语言模型		优点	缺点
LM	自回归语言模型	语言模型联合概率的无偏估计，考虑被预测单词之间的相关性，适合生成任务；	按照文本序列顺序拆解（从左至右分解），无法获取双向上下文信息表征；
DAE	自编码语言模型	本质为降噪自编码(DAE)特征表示，通过引入噪声[MASK]构建MLM获取双向上下文信息表征	1）引入独立性假设，为语言模型联合概率的有偏估计，没有考虑预测token之间的相关性； 2）预训练时的「MASK」噪声在finetune阶段不会出现，造成两阶段不匹配问题；
PLM	排列语言模型	综合了LM和DAE-LM两者的优点	收敛速度较慢，XL对于长序列建模效果并仅预测了排列后序列中的最后几个token；

图 3: 三类语言模型之间的对比

第一类：自回归语言模型（**LM**）

优点：

语言模型（language model，LM）联合概率的无偏估计，即为传统的语言模型，考虑被预测单词之间的相关性，天然适合处理自然生成任务；

缺点：

联合概率按照文本序列顺序拆解（从左至右分解），无法获取双向上下文信息表

征；
代表模型：ELMO、GPT-1、GPT-2[18]、ULMFiT[19]、SiATL[20]；
第二类：自编码语言模型（**DAE**）

优点：
本质为降噪自编码(DAE)特征表示，通过引入噪声[MASK]构建 MLM(Masked language model)，获取双向上下文信息表征（本文将自编码语言模型统一称为 DAE，旨在采用部分损坏的输入，旨在恢复原始的未失真输入）；如果当前 token 被预测，则 否则 ， 为原始文本被替换后的输入。
缺点：
引入独立性假设，为语言模型联合概率的有偏估计，没有考虑预测 token 之间的相关性；
预训练时的「MASK」噪声在 finetune 阶段不会出现，造成两阶段不匹配问题；为解决这一问题，在 15%被预测的 token 中，80%被替换为「MASK」，10%被随机替换，10%被替换为原词。
代表模型：BERT、MASS [21]、T5[22]、RoBERTa[23]、UniLM[24]、XLM[25]、SpanBERT[26]、ERNIE-Baidu[27][28]、E-BERT[29]、ERNIE-THU[30]、BART[31]。
BERT[13]是自编码语言模型的一个典型代表，但其采用的 MLM 策略和 Transformer-Encoder 结构，导致其不适合直接处理生成任务。为了解决这一问题，也可采用基于 Seq2Seq MLM 方法：encoder 部分采取 masked 策略，而 decoder 部分以自回归的方式预测 encoder 部分被 mask 的 token。此外，还有很多基于自编码语言模型的 PTMs 提出了不同的 MLM 增强策略，称之为 Enhanced Masked Language Modeling (E-MLM) 。

第三类：排列语言模型（**PLM**）
排列语言模型（PLM）综合了 LM 和 DAE-LM 两者的优点。严格来讲，PLM 和 LM 是标准的自回归语言模型，而 MLM 不是一个标准的语言模型，其引入独立性假设，隐式地学习预测 token（mask 部分本身的强相关性）之间的关系。如果衡量序列中被建模的依赖关系的数量，标准的自回归语言模型可以达到上界，不依赖于任何独立假设。LM 和 PLM 能够通过自回归方式来显式地学习预测 token 之间的关系。然而，LM 无法对双向上下文进行表征，借鉴 NADE[32]的思想，PLM 将这种传统的自回归语言模型（LM）进行推广，将顺序拆解变为随机拆解（从左至右分解），产生上下文相关的双向特征表示。
PLM 最为典型的代表就是 XLNet[14]，这是对标准语言模型的一个复兴[33]：提出一个框架来连接标准语言模型建模方法和预训练方法。
一个关键问题：为什么 PLM 可以实现双向上下文的建模？PLM 的本质就是语言模型联合概率的多种分解机制的体现，其将 LM 的顺序拆解推广到随机拆解。PLM 没有改变原始文本序列的自然位置，只是定义了 token 预测的顺序。PLM 只是针对语言模型建模不同排列下的因式分解排列，并不是词的位置信息的重新排列。
最后，我们对基于上述三类语言模型的 PTMs 进行总结：

语言模型	模型	编码器	主要亮点
LM	ELMO	LSTM	2个单向语言模型（前向和后向）的拼接
LM	ULMFiT	LSTM	引入逐层解冻解决finetune中的灾难性问题；
LM	SiATL	LSTM	引入逐层解冻+辅助LM解决finetune中的遗忘问题；
LM	GPT-1	Transformer-Decoder	首次将Transformer应用于预训练语言模型；
LM	GPT-2	Transformer-Decoder	没有特定模型的精调流程，生成任务取得很好效果；
DAE：MLM	BERT	Transformer-Encoder	MLM获取上下文相关的双向特征表示；
DAE：Seq2SeqMLM	MASS / T5	Transformer	改进BERT生成任务：统一为类似Seq2Seq的预训练框架；
DAE：E-MLM	UNILM	Transformer-Encoder	改进BERT生成任务：3个mask矩阵：LM/MLM/Seq2Seq LM；
DAE：E-MLM	RoBERTa	Transformer-Encoder	预训练过程中采取动态mask，不像BERT在预处理做静态mask；
DAE：E-MLM	XLM	Transformer-Encoder	在翻译语言模型的平行语料上执行MLM
DAE：E-MLM	SpanBERT	Transformer-Encoder	采取random span mask和span boundary objective 2个预训练目标；
DAE：E-MLM	ENRIE-BAIDU	Transformer-Encoder	mask实体和短语，2.0引入多任务进行增量学习；
DAE：E-MLM	ENRIE-THU/E-BERT	Transformer-Encoder	引入知识：将实体向量与文本表示融合；
DAE	BART	Transformer	采取Seq2Seq框架和5种DAE方法
PLM	XLNet	Transformer-XL	双向上下文表征+双注意力流

基于上下文（Context Based）的 3 种语言模型 PTMs 总结

4.2 基于对比（Contrastive Based）

基于对比（Contrastive Based），不同于 Context Based 主要基于数据本身的上下文信息构造辅助任利用，Contrastive Based 主要利用样本间的约束信息构造辅助任务，这类方法也是 Contrastive learning[34]（CTL）。CTL 假设观察到的文本对（正样本）在语义上比随机采样的文本（负样本）更相似。CTL 背后的原理是「在对比中学习」。相较于语言建模，CTL 的计算复杂度更低，因而在预训练中是理想的替代训练标准。

CTL 通过构建正样本（positive）和负样本（negative），然后度量正负样本的距离来实现自监督学习[17]:可以使用点积的方式构造距离函数，然后构造一个 softmax 分类器，以正确分类正样本和负样本。鼓励相似性度量函数将较大的值分配给正例，将较小的值分配给负例：

相似性度量函数通常可采取两种方式： 或

第一类： Deep InfoMax (DIM)

DIM 方法来源于 CV 领域，对于全局的特征（编码器最终的输出）和局部特征（编码器中间层的特征），DIM 需要判断全局特征和局部特征是否来自同一图像[17]。InfoWord [35]将 DIM 引入到 NLP 中，用 Mutual Information 的一个下界 InfoNCE 来重新解释 BERT 和 XLNET 的 objective，并提出一个新的 DIM objective 以最大化一个句子的 global representation 和其中一个 ngram 的 local representation 之间的 Mutual Information。

第二类：Replaced Token Detection (RTD)

噪声对比估计（Noise-Contrastive Estimation，NCE）[36]通过训练一个二元分类器来区分真实样本和假样本，可以很好的训练词嵌入。RTD 于与 NCE 相同，根据上下文语境来预测 token 是否替换 。

word2vec[3]中的 negative sampling 可看作是 RTD，负样本从词表中进行带权采样。ELECTRA[37]提出了一种新的预训练任务框架，构建生成器-判别器，生成器通过 MLM 任务对被 mask 的 token 进行预测，迭代器判断原始句子中的每个 token 是否被 replace 过。生成器相当于对输入进行了筛选，使判别器的任务更难，从而学习到更好的表示。生成器-判别器共享 embedding，生成器部分采用 small-bert，判别器部分对每一个 token 采用 sigmoid 计算 loss。finetune 阶段只采用判别器部分。RTD 也被看作解决 MLM 中「MASK」在预训练和 finetune 间差异的一种手段。

WKLM[38]在实体 level 进行替换，替换为具有相同实体类型的实体名称。
第三类：Next Sentence Prediction (NSP)
NSP 区分两个输入句子是否为训练语料库中的连续片段，第二个句子 50%为第一句子实际的连续片段，50%从其他语料随机选择。NSP 可以引导模型理解两个输入句子之间的关系，从而使对此信息敏感的下游任务受益，如 QA 任务。而 RoBERTa[23]表明：NSP 在对单个文档中的文本块进行训练时，去除 NSP 任务或在下游任务上可以稍微提高性能。
第四类：Sentence Order Prediction (SOP)
SOP 使用同一文档中的两个连续片段作为正样本，而相同的两个连续片段互换顺序作为负样本。NSP 融合了主题预测和相关性预测，主题预测更容易，这使得模型进行预测时仅依赖于主题学习。与 NSP 不同，SOP 使用同一文档中的两个连续段作为正样本，但顺序互换为负样本。采取 SOP 任务的 PTMs 有 ALBERT[39]、StructBERT[40]、BERTje[41]。

图 5 对上述基于对比（Contrastive Based）的四类 PTMs 进行了总结：

Contrastive Based方法	特点	PTMs
DIM：Deep InfoMax	最大化全局特征和局部特征间的互信息	InfoWord
RTD：Replaced Token Detection	根据上下文语境来预测token是否替换	word2vec-ns/ELECTRA/WKLM
NSP：Next Sentence Prediction	区分两个输入句子是否为语料库中的连续片段	[illegible]
SOP：Sentence Order Prediction	相关性预测，将两个连续片段互换顺序	ALBERT/StructBERT/BERTje

图 5: 基于对比（Contrastive Based）的 PTMs 总结

5.PTMs 有哪些拓展？

5.1 引入知识
PTMs 通常从通用大型文本语料库中学习通用语言表示，但是缺少特定领域的知识。PTMs 中设计一些辅助的预训练任务，将外部知识库中的领域知识整合到 PTMs 中被证明是有效的[1]。
ERNIE-THU[30]将在知识图谱中预先训练的实体嵌入与文本中相应的实体提及相结合，以增强文本表示。由于语言表征的预训练过程和知识表征过程有很大的不同，会产生两个独立的向量空间。为解决上述问题，在有实体输入的位置，将实体向量和文本表示通过非线性变换进行融合，以融合词汇、句法和知识信息。
LIBERT[42]（语言知识的 BERT）通过附加的语言约束任务整合了语言知识。
SentiLR[43]集成了每个单词的情感极性，以将 MLM 扩展到标签感知 MLM（LA-MLM），ABSA 任务上都达到 SOTA。
SenseBERT[44] 不仅能够预测被 mask 的 token，还能预测它们在给定语境下的实际含义。使用英语词汇数据库 WordNet 作为标注参照系统，预测单词在语境中的实际含义，显著提升词汇消歧能力。
KnowBERT[45] 与实体链接模型以端到端的方式合并实体表示。
KG-BERT[46]显示输入三元组形式，采取两种方式进行预测：构建三元组识别和关系分类，共同优化知识嵌入和语言建模目标。这些工作通过实体嵌入注入知识图的结构信息。
K-BERT[47]将从 KG 提取的相关三元组显式地注入句子中，以获得 BERT 的扩展树形输入。
K-Adapter[48]通过针对不同的预训练任务独立地训练不同的适配器来注入多种

知识，从而可以不断地注入知识，以解决注入多种知识时可能会出现灾难性遗忘问题。
此外，这类 PTMs 还有 WKLM[38]、KEPLER[49]和[50]等。
5.2 模型压缩
由于预训练的语言模型通常包含至少数亿个参数，因此很难将它们部署在现实应用程序中的在线服务和资源受限的设备上。模型压缩是减小模型尺寸并提高计算效率的有效方法。
5 种 PTMs 的压缩方法为：
pruning（剪枝）：将模型中影响较小的部分舍弃。
如 Compressing BERT[51]，还有结构化剪枝 LayerDrop [52]，其在训练时进行 Dropout，预测时再剪掉 Layer，不像知识蒸馏需要提前固定 student 模型的尺寸大小。
quantization（量化）：将高精度模型用低精度来表示；
如 Q-BERT[53]和 Q8BERT[54]，量化通常需要兼容的硬件。
parameter sharing （参数共享）：相似模型单元间的参数共享；
ALBERT[39]主要是通过矩阵分解和跨层参数共享来做到对参数量的减少。
module replacing（模块替换）：
BERT-of-Theseus[55]根据伯努利分布进行采样，决定使用原始的大模型模块还是小模型，只使用 task loss。
knowledge distillation （知识蒸馏）：通过一些优化目标从大型、知识丰富、fixed 的 teacher 模型学习一个小型的 student 模型。蒸馏机制主要分为 3 种类型：
从软标签蒸馏：DistilBERT [56]、EnsembleBERT[57]
从其他知识蒸馏：TinyBERT[58]、BERT-PKD、MobileBERT[59] 、 MiniLM[60] 、 DualTrain[61]
蒸馏到其他结构：Distilled-BiLSTM[62]

知识蒸馏PTMs	主要方法
DistilBERT	软标签蒸馏，KL散度作为loss
TinyBERT	层与层蒸馏：embedding/hidden state/self-attention distributions
BERT-PKD	层与层蒸馏：hidden state
MobileBERT	软标签蒸馏+层与层蒸馏：hidden state/self-attention distributions
MiniLM	Self-attention distributions /self- attention value relation.
DualTrain	Dual Projection
Distilled-BiLSTM	软标签蒸馏，将Transformer蒸馏到LSTM
EnsembleBERT	取多个Ensemble模型的软标签进行蒸馏

图 6: 不同的知识蒸馏 PTMs
5.3 多模态
随着 PTMs 在 NLP 领域的成功，许多研究者开始关注多模态领域的 PTMs，主要为通用的视觉和语言特征编码表示而设计。多模态的 PTMs 在一些庞大的跨模式数据语料库（带有文字的语音、视频、图像）上进行了预训练，如带有文字的语音、视频、图像等，主要有 VideoBERT[63]、CBT[64] 、UniViLM[65]、 ViL-BERT[66] 、

LXMERT[67]、 VisualBERT [68]、 B2T2[69] 、Unicoder-VL[70] 、UNITER [71]、VL-BERT[72] 、 SpeechBERT[73]。

5.4 领域预训练

大多数 PTM 都在诸如 Wikipedia 的通用语料中训练，而在领域化的特定场景会收到限制。如基于生物医学文本的 BioBERT[74]，基于科学文本的 SciBERT[75]，基于临床文本的 Clinical-BERT[76]。一些工作还尝试将 PTMs 适应目标领域的应用，如医疗实体标准化[77]、专利分类 PatentBERT [78]、情感分析 SentiLR[79]关键词提取[80]。

5.5 多语言和特定语言

学习跨语言共享的多语言文本表示形式对于许多跨语言的 NLP 任务起着重要的作用。

Multilingual-BERT[81]在 104 种 Wikipedia 文本上进行 MLM 训练（共享词表），每个训练样本都是单语言文档，没有专门设计的跨语言目标，也没有任何跨语言数据，M-BERT 也可以很好的执行跨语言任务。

XLM [25]通过融合跨语言任务（翻译语言模型）改进了 M-BERT，该任务通过拼接平行语料句子对进行 MLM 训练。

Unicoder[82]提出了 3 种跨语言预训练任务：1)cross-lingual word recovery； 2) cross-lingual paraphrase classification;3) cross-lingual masked language model.

虽然多语言的 PTMs 在跨语言上任务表现良好，但用单一语言训练的 PTMs 明显好于多语言的 PTMs。此外一些单语言的 PTMs 被提出：BERT-wwm[83]， ZEN[84], NEZHA[85] , ERNIE-Baidu[27][28], BERTje[86], CamemBERT[87], FlauBERT [88], RobBERT [89]。

6.如何对 PTMs 进行迁移学习？

PTMs 从大型语料库中获取通用语言知识，如何有效地将其知识适应下游任务是一个关键问题。迁移学习的方式主要有归纳迁移（顺序迁移学习、多任务学习）、领域自适应（转导迁移）、跨语言学习等。NLP 中 PTMs 的迁移方式是顺序迁移学习。

1、如何迁移？

1）选择合适的预训练任务：语言模型是 PTM 是最为流行的预训练任务；同的预训练任务有其自身的偏置，并且对不同的任务会产生不同的效果。例如，NSP 任务可以使诸如问答（QA）和自然语言推论（NLI）之类的下游任务受益。

2）选择合适的模型架构：例如 BERT 采用的 MLM 策略和 Transformer-Encoder 结构，导致其不适合直接处理生成任务。

3）选择合适的数据：下游任务的数据应该近似于 PTMs 的预训练任务，现在已有有很多现成的 PTMs 可以方便地用于各种特定领域或特定语言的下游任务。

4）选择合适的 layers 进行 transfer：主要包括 Embedding 迁移、top layer 迁移和 all layer 迁移。如 word2vec 和 Glove 可采用 Embedding 迁移，BERT 可采用 top layer 迁移，Elmo 可采用 all layer 迁移。

5）特征集成还是 fine-tune？对于特征集成预训练参数是 freeze 的，而 fine-tune 是 unfreeze 的。特征集成方式却需要特定任务的体系结构，fine-tune 方法通常比特征提取方法更为通用和方便。

2、fine-tune 策略：通过更好的微调策略进一步激发 PTMs 性能

两阶段 fine-tune 策略：如第一阶段对中间任务或语料进行 finetune，第二阶段再

对目标任务 fine-tune。第一阶段通常可根据特定任务的数据继续进行 fine-tune 预训练。
多任务 fine-tune：MTDNN[90]在多任务学习框架下对 BERT 进行了 fine-tune，这表明多任务学习和预训练是互补的技术。
采取额外的适配器：fine-tune 的主要缺点是其参数效率低，每个下游任务都有自己的 fine-tune 参数。因此，更好的解决方案是在固定原始参数的同时，将一些可 fine-tune 的适配器注入 PTMs。
逐层阶段：逐渐冻结而不是同时对所有层进行 fine-tune，也是一种有效的 fine-tune 策略。

7.PTMs 还有哪些问题需要解决？

虽然 PTMs 已经在很多 NLP 任务中显示出了他们强大的能力，然而由于语言的复杂性，仍存在诸多挑战。综述论文给出了五个未来 PTMs 发展方向的建议。

1、**PTMs 的上限**

目前，PTMs 并没有达到其上限。大多数的 PTMs 可通过使用更长训练步长和更大数据集来提升其性能。目前 NLP 中的 SOTA 也可通过加深模型层数来更进一步提升。这将导致更加高昂的训练成本。因此，一个更加务实的方向是在现有的软硬件基础上，设计出更高效的模型结构、自监督预训练任务、优化器和训练技巧等。例如， ELECTRA [37]就是此方向上很好的一个解决方案。

2、**面向任务的预训练和模型压缩**

在实践中，不同的目标任务需要 PTMs 拥有不同功能。而 PTMs 与下游目标任务间的差异通常在于两方面：模型架构与数据分布。尽管较大的 PTMs 通常情况下会带来更好的性能表现，但在低计算资源下如何使用是一个实际问题。例如，对于 NLP 的 PTM 来说，对于模型压缩的研究只是个开始，Transformer 的全连接架构也使得模型压缩具有挑战性。

3、**PTMs 的架构设计**

对于 PTMs，Transformer 已经被证实是一个高效的架构。然而 Transformer 最大的局限在于其计算复杂度（输入序列长度的平方倍）。受限于 GPU 显存大小，目前大多数 PTM 无法处理超过 512 个 token 的序列长度。打破这一限制需要改进 Transformer 的结构设计，例如 Transformer-XL[92]。

4、**finetune 中的知识迁移**

finetune 是目前将 PTM 的知识转移至下游任务的主要方法，但效率却很低，每个下游任务都需要有特定的 finetune 参数。一个可以改进的解决方案是固定 PTMs 的原始参数，并为特定任务添加小型的 finetune 适配器，这样就可以使用共享的 PTMs 服务于多个下游任务。

5、**PTMs 的解释性与可靠性**

PTMs 的可解释性与可靠性仍然需要从各个方面去探索，它能够帮助我们理解 PTM 的工作机制，为更好的使用及性能改进提供指引。

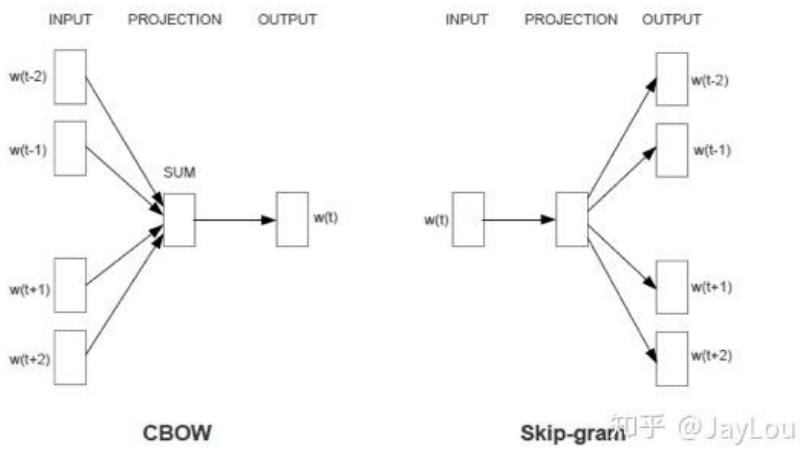
写在最后：本文总结 与 原综述论文[1]的一些不同之处：
本文定义了 PTMs 两大范式：浅层词嵌入和预训练编码器。不同于原文，XLNet 在原综述论文中被归为 Transformer-Encoder，本文认为将其归为 Transformer-XL 更合适。
本文 PTMs 按照自监督学习的分类不同于原文。本文按照 基于上下文（Context Based）和基于对比（Contrastive Based）两种方式归类；将原文的 LM、MLM、

DAE、PLM 归为 Context Based；
本文将原文 MLM 和 DAE 统一为 DAE；
其他：1）在 3.1.2 的 E-MLM 段落中，可以将 StructBERT 拿出来，只放在 SOP；2）3.1.5 对 ELECTRA 的描述，应采取 ELECTRA 原文中的主要方法（参数共享），两阶段的方法只是一种实验尝试；3）在 puring 部分可以补充 LayerDrop；4）应将 UniLM 归为 MLM；；

二、Word2vec

1、word2vec 的两种模型分别是什么？

word2Vec 有两种模型：CBOW 和 Skip-Gram：
CBOW 在已知 context(w) 的情况下，预测 w；
Skip-Gram 在已知 w 的情况下预测 context(w) ；



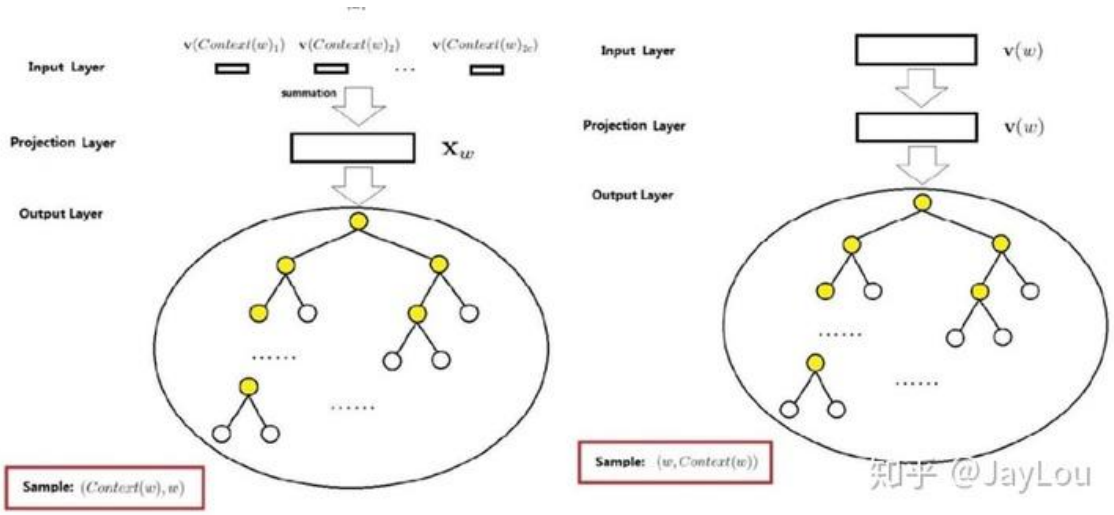
word2vec

与 NNLM 相比，word2vec 的主要目的是生成词向量而不是语言模型，在 CBOW 中，投射层将词向量直接相加而不是拼接起来，并舍弃了隐层，这些牺牲都是为了减少计算量，使训练更加

2、word2vec 的两种优化方法是什么？它们的目标函数怎样确定的？

训练过程又是怎样的？

不经过优化的 CBOW 和 Skip-gram 中 ,在每个样本中每个词的训练过程都要遍历整个词汇表，也就是都需要经过 softmax 归一化，计算误差向量和梯度以更新两个词向量矩阵（这两个词向量矩阵实际上就是最终的词向量，可认为初始化不一样），当语料库规模变大、词汇表增长时，训练变得不切实际。为了解决这个问题，word2vec 支持两种优化方法：hierarchical softmax 和 negative sampling。此部分仅做关键介绍，数学推导请仔细阅读《word2vec 中的数学原理详解》。
（1）基于 hierarchical softmax 的 CBOW 和 Skip-gram



基于 hierarchical softmax 的 CBOW 和 Skip-gram

hierarchical softmax 使用一颗二叉树表示词汇表中的单词，每个单词都作为二叉树的叶子节点。对于一个大小为 V 的词汇表，其对应的二叉树包含 V-1 非叶子节点。假如每个非叶子节点向左转标记为 1，向右转标记为 0，那么每个单词都具有唯一的从根节点到达该叶子节点的由｛0 1｝组成的代号（实际上为哈夫曼编码，为哈夫曼树，是带权路径长度最短的树，哈夫曼树保证了词频高的单词的路径短，词频相对低的单词的路径长，这种编码方式很大程度减少了计算量）。

CBOW 中的目标函数是使条件概率 最大化，其等价于：

$$\begin{aligned}\mathcal{L} &= \sum_{w\in\mathcal{C}} \log \prod_{j=2}^{l^w} \left\{ [\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]^{1-d_j^w} \cdot [1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)]^{d_j^w} \right\} \\ &= \sum_{w\in\mathcal{C}} \sum_{j=2}^{l^w} \left\{ (1-d_j^w) \cdot \log[\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)] + d_j^w \cdot \log[1-\sigma(\mathbf{x}_w^\top \theta_{j-1}^w)] \right\}\end{aligned}$$

Skip-gram 中的目标函数是使条件概率 最大化，其等价于：

$$\begin{aligned}\mathcal{L} &= \sum_{w\in\mathcal{C}} \log \prod_{u\in Context(w)} \prod_{j=2}^{l^u} \left\{ [\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)]^{1-d_j^u} \cdot [1-\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)]^{d_j^u} \right\} \\ &= \sum_{w\in\mathcal{C}} \sum_{u\in Context(w)} \sum_{j=2}^{l^u} \left\{ (1-d_j^u) \cdot \log[\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)] + d_j^u \cdot \log[1-\sigma(\mathbf{v}(w)^\top \theta_{j-1}^u)] \right\}.\end{aligned}$$

（2）基于 negative sampling 的 CBOW 和 Skip-gram

negative sampling 是一种不同于 hierarchical softmax 的优化策略，相比于 hierarchical softmax，negative sampling 的想法更直接——为每个训练实例都提供负例。

对于 CBOW，其目标函数是最大化：

$$g(w) = \prod_{u\in\{w\}\cup NEG(w)} p(u|Context(w)),$$

$$p(u|Context(w)) = \left[\sigma(\mathbf{x}_w^\top \theta^u)\right]^{L^w(u)} \cdot \left[1 - \sigma(\mathbf{x}_w^\top \theta^u)\right]^{1-L^w(u)}$$

$$\begin{aligned}
\mathcal{L} &= \log G = \log \prod_{w \in \mathcal{C}} g(w) = \sum_{w \in \mathcal{C}} \log g(w) \\
&= \sum_{w \in \mathcal{C}} \log \prod_{u \in \{w\} \cup NEG(w)} \left\{ \left[\sigma(\mathbf{x}_w^\top \theta^u)\right]^{L^w(u)} \cdot \left[1 - \sigma(\mathbf{x}_w^\top \theta^u)\right]^{1-L^w(u)} \right\} \\
&= \sum_{w \in \mathcal{C}} \sum_{u \in \{w\} \cup NEG(w)} \left\{ L^w(u) \cdot \log \left[\sigma(\mathbf{x}_w^\top \theta^u)\right] + \left[1 - L^w(u)\right] \cdot \log \left[1 - \sigma(\mathbf{x}_w^\top \theta^u)\right] \right\}
\end{aligned}$$

对于 Skip-gram，同样也可以得到其目标函数是最大化：

$$g(w) = \prod_{\tilde{w} \in Context(w)} \prod_{u \in \{w\} \cup NEG^{\tilde{w}}(w)} p(u|\tilde{w}),$$

$$p(u|\tilde{w}) = \left[\sigma(\mathbf{v}(\tilde{w})^\top \theta^u)\right]^{L^w(u)} \cdot \left[1 - \sigma(\mathbf{v}(\tilde{w})^\top \theta^u)\right]^{1-L^w(u)}$$

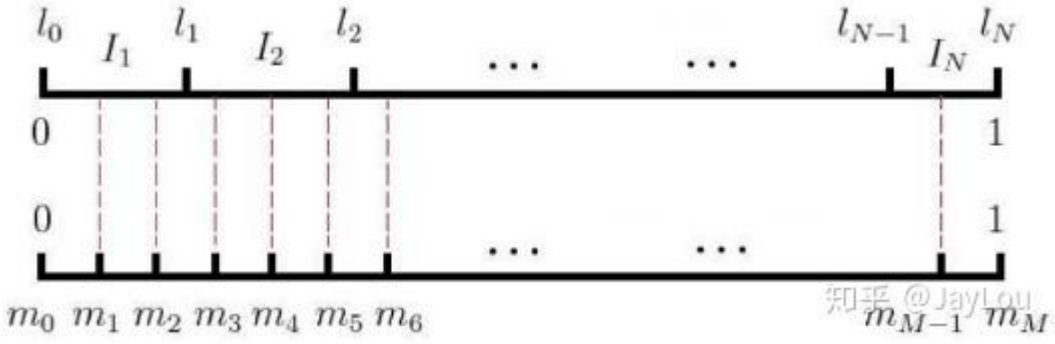
$$\begin{aligned}
\mathcal{L} &= \log G = \log \prod_{w \in \mathcal{C}} g(w) = \sum_{w \in \mathcal{C}} \log g(w) \\
&= \sum_{w \in \mathcal{C}} \log \prod_{\tilde{w} \in Context(w)} \prod_{u \in \{w\} \cup NEG^{\tilde{w}}(w)} \left\{ \left[\sigma(\mathbf{v}(\tilde{w})^\top \theta^u)\right]^{L^w(u)} \cdot \left[1 - \sigma(\mathbf{v}(\tilde{w})^\top \theta^u)\right]^{1-L^w(u)} \right\} \\
&= \sum_{w \in \mathcal{C}} \sum_{\tilde{w} \in Context(w)} \sum_{u \in \{w\} \cup NEG^{\tilde{w}}(w)} \\
&\quad \left\{ L^w(u) \cdot \log \left[\sigma(\mathbf{v}(\tilde{w})^\top \theta^u)\right] + \left[1 - L^w(u)\right] \cdot \log \left[1 - \sigma(\mathbf{v}(\tilde{w})^\top \theta^u)\right] \right\}.
\end{aligned}$$

负采样算法实际上就是一个带权采样过程，负例的选择机制是和单词词频联系起来的。

$$l_0 = 0, \cdots, l_k = \sum_{j=1}^{k} len(w_j), \quad k = 1, 2, \cdots, N$$

$$\text{Table}(i) = w_k, \quad \text{where} \quad m_i \in I_k, \quad i = 1, 2, \cdots, M-1$$

具体做法是以 N+1 个点对区间 [0,1] 做非等距切分，并引入的一个在区间 [0,1] 上的 M 等距切分，其中 M >> N。源码中取 M = 10^8。然后对两个切分做投影，得到映射关系：采样时，每次生成一个 [1, M-1] 之间的整数 i，则 Table(i) 就对应一个样本；当采样到正例时，跳过（拒绝采样）。



3、word2vec 和 tf-idf 相似度计算时的区别？

word2vec 1、稠密的 低维度的 2、表达出相似度； 3、表达能力强；4、泛化能力强；

4、word2vec 和 NNLM 对比有什么区别？

1）其本质都可以看作是语言模型；
2）词向量只不过 NNLM 一个产物，word2vec 虽然其本质也是语言模型，但是其专注于词向量本身，因此做了许多优化来提高计算效率：
与 NNLM 相比，词向量直接 sum，不再拼接，并舍弃隐层；
考虑到 sofmax 归一化需要遍历整个词汇表，采用 hierarchical softmax 和 negative sampling 进行优化，hierarchical softmax 实质上生成一颗带权路径最小的哈夫曼树，让高频词搜索路劲变小；negative sampling 更为直接，实质上对每一个样本中每一个词都进行负例采样；

5、word2vec 负采样有什么作用？

负采样这个点引入 word2vec 非常巧妙，两个作用，1.加速了模型计算，2.保证了模型训练的效果，一个是模型每次只需要更新采样的词的权重，不用更新所有的权重，那样会很慢，第二，中心词其实只跟它周围的词有关系，位置离着很远的词没有关系，也没必要同时训练更新，作者这点非常聪明。

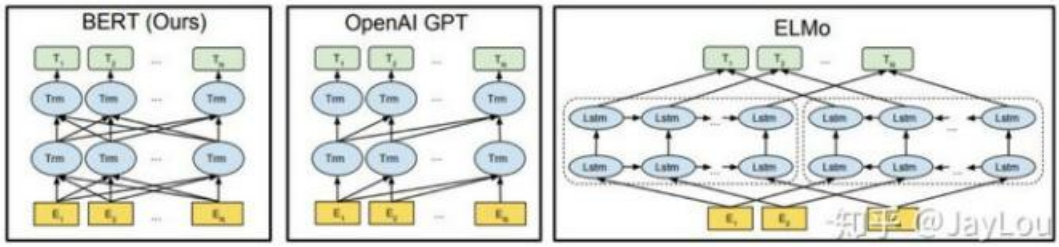
6、word2vec 和 fastText 对比有什么区别？

1）都可以无监督学习词向量， fastText 训练词向量时会考虑 subword；
2）fastText 还可以进行有监督学习进行文本分类，其主要特点：
结构与 CBOW 类似，但学习目标是人工标注的分类结果；
采用 hierarchical softmax 对输出的分类标签建立哈夫曼树，样本中标签多的类别被分配短的搜寻路径；
引入 N-gram，考虑词序特征；
引入 subword 来处理长词，处理未登陆词问题；

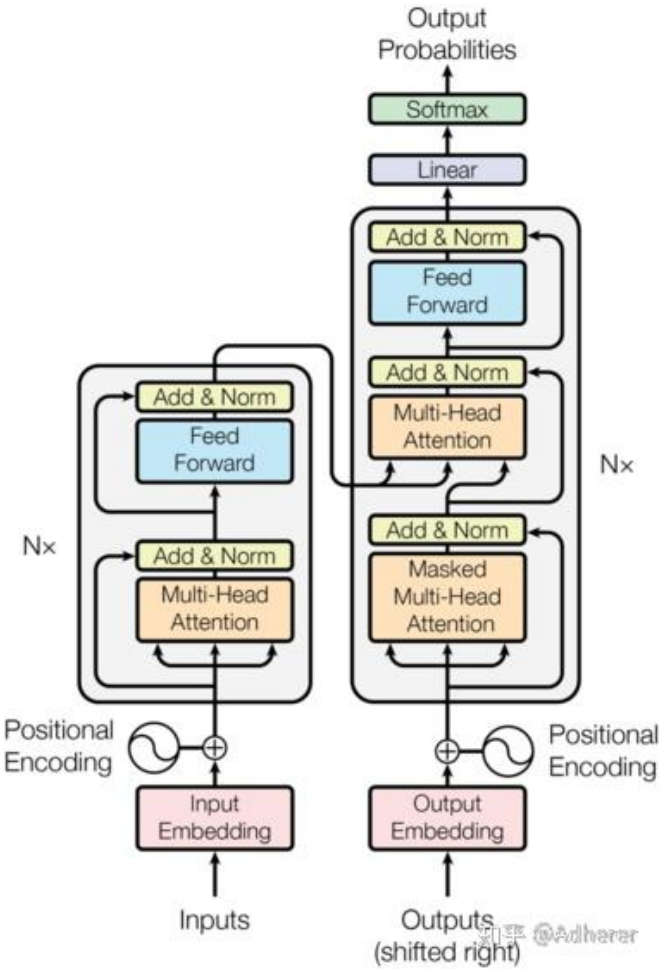
7、glove 和 word2vec、 LSA 对比有什么区别？

1）glove vs LSA
LSA（Latent Semantic Analysis）可以基于 co-occurance matrix 构建词向量，实质上是基于全局语料采用 SVD 进行矩阵分解，然而 SVD 计算复杂度高；
glove 可看作是对 LSA 一种优化的高效矩阵分解算法，采用 Adagrad 对最小平方损失进行优化；
2）word2vec vs LSA
主题模型和词嵌入两类方法最大的不同在于模型本身。
主题模型是一种基于概率图模型的生成式模型。其似然函数可以写为若干条件概率连乘的形式，其中包含需要推测的隐含变量(即主题)
词嵌入模型一般表示为神经网络的形式，似然函数定义在网络的输出之上。需要学习网络的权重来得到单词的稠密向量表示。
3）word2vec vs glove
word2vec 是局部语料库训练的，其特征提取是基于滑窗的；而 glove 的滑窗是为

了构建 co-occurance matrix，是基于全局语料的，可见 glove 需要事先统计共现概率；因此，word2vec 可以进行在线学习，glove 则需要统计固定语料信息。
word2vec 是无监督学习，同样由于不需要人工标注；glove 通常被认为是无监督学习，但实际上 glove 还是有 label 的，即共现次数[公式]。
word2vec 损失函数实质上是带权重的交叉熵，权重固定；glove 的损失函数是最小平方损失函数，权重可以做映射变换。
总体来看，glove 可以被看作是更换了目标函数和权重函数的全局 word2vec。



三、Transformer



1.Transformer 的结构是什么样的？

Transformer 本身还是一个典型的 encoder-decoder 模型，如果从模型层面来看，Transformer 实际上就像一个 seq2seq with attention 的模型，下面大概说明一下 Transformer 的结构以及各个模块的组成。

(1) **Encoder 端 & Decoder 端总览**

Encoder 端由 N(原论文中 N=6)个相同的大模块堆叠而成，其中每个大模块又由两个子模块构成，这两个子模块分别为多头 self-attention 模块，以及一个前馈神经网络模块；

需要注意的是，Encoder 端每个大模块接收的输入是不一样的，第一个大模块(最底下的那个)接收的输入是输入序列的 embedding(embedding 可以通过 word2vec 预训练得来)，其余大模块接收的是其前一个大模块的输出，最后一个模块的输出作为整个 Encoder 端的输出。

Decoder 端同样由 N(原论文中 N=6)个相同的大模块堆叠而成，其中每个大模块则由三个子模块构成，这三个子模块分别为多头 self-attention 模块，多头 Encoder-Decoder attention 交互模块，以及一个前馈神经网络模块；

同样需要注意的是，Decoder 端每个大模块接收的输入也是不一样的，其中第一个大模块(最底下的那个)训练时和测试时的接收的输入是不一样的，并且每次训练时接收的输入也可能是不一样的(也就是模型总览图示中的"shifted right"，后续会解释)，其余大模块接收的是同样是其前一个大模块的输出，最后一个模块的输出作为整个 Decoder 端的输出。

对于第一个大模块，简而言之，其训练及测试时接收的输入为：

训练的时候每次的输入为上次的输入加上输入序列向后移一位的 ground truth(例如每向后移一位就是一个新的单词，那么则加上其对应的 embedding)，特别地，当 decoder 的 time step 为 1 时(也就是第一次接收输入)，其输入为一个特殊的 token，可能是目标序列开始的 token(如<BOS>)，也可能是源序列结尾的 token(如<EOS>)，也可能是其它视任务而定的输入等等，不同源码中可能有微小的差异，其目标则是预测下一个位置的单词(token)是什么，对应到 time step 为 1 时，则是预测目标序列的第一个单词(token)是什么，以此类推；

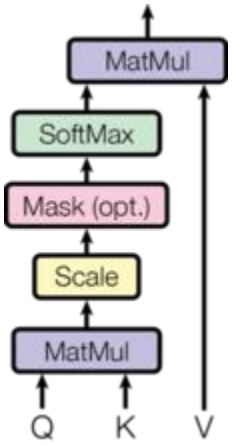
这里需要注意的是，在实际实现中可能不会这样每次动态的输入，而是一次性把目标序列的 embedding 通通输入第一个大模块中，然后在多头 attention 模块对序列进行 mask 即可

而在测试的时候，是先生成第一个位置的输出，然后有了这个之后，第二次预测时，再将其加入输入序列，以此类推直至预测结束。

(2) **Encoder 端各个子模块**

多头 self-attention 模块

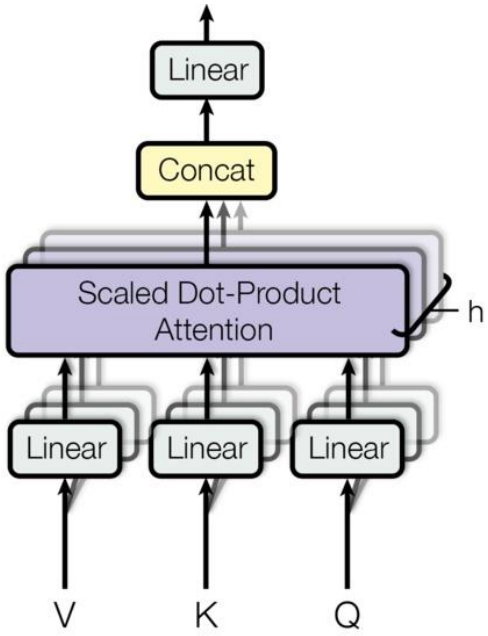
在介绍 self-attention 模块之前，先介绍 self-attention 模块，图示如下：



上述attention可以被描述为**将query和key-value键值对的一组集合映射到输出**，其中 query，keys，values和输出都是向量，其中 query和keys的维度均为 d_k ，values的维度为 d_v (论文中 $d_k = d_v = d_{\text{model}}/h = 64$)，输出被计算为values的加权和，其中分配给每个value的权重由query与对应key的相似性函数计算得来。这种attention的形式被称为“Scaled Dot-Product Attention”，对应到公式的形式为：

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

而多头self-attention模块，则是将 Q, K, V 通过参数矩阵映射后(给 Q, K, V 分别接一个全连接层)，然后再做self-attention，将这个过程重复(原论文中)次，最后再将所有的结果拼接起来，再送入一个全连接层即可，图示如下：



对应到公式的形式为：

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O$$
$$\text{where head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)$$

其中 $W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}, W^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$

前馈神经网络模块

前馈神经网络模块(即图示中的 Feed Forward)由两个线性变换组成，中间有一个 ReLU 激活函数，对应到公式的形式为：

论文中前馈神经网络模块输入和输出的维度均为　　，其内层的维度

(3) Decoder 端各个子模块

多头 self-attention 模块

Decoder 端多头 self-attention 模块与 Encoder 端的一致，但是需要注意的是 Decoder 端的多头 self-attention 需要做 mask，因为它在预测时，是“看不到未来的序列的”，所以要将当前预测的单词(token)及其之后的单词(token)全部 mask 掉。

多头 Encoder-Decoder attention 交互模块

多头 Encoder-Decoder attention 交互模块的形式与多头 self-attention 模块一致，唯一不同的是其　矩阵的来源，其　矩阵来源于下面子模块的输出(对应到图中即为 masked 多头 self-attention 模块经过 Add & Norm 后的输出)，而　矩阵则来源于整个 Encoder 端的输出，仔细想想其实可以发现，这里的交互模块就跟 seq2seq with attention 中的机制一样，目的就在于让 Decoder 端的单词(token)给予 Encoder 端对应的单词(token)“更多的关注(attention weight)”

前馈神经网络模块

该部分与 Encoder 端的一致

(4).其他模块

Add & Norm 模块

Add & Norm模块接在Encoder端和Decoder端每个子模块的后面，其中Add表示残差连接，Norm表示LayerNorm，残差连接来源于论文Deep Residual Learning for Image Recognition，LayerNorm来源于论文Layer Normalization，因此Encoder端和Decoder端每个子模块实际的输出为：$\mathrm{LayerNorm}(x+\mathrm{Sublayer}(x))$，其中 $\mathrm{Sublayer}(x))$ 为子模块的输出。

Positional Encoding

Positional Encoding 添加到 Encoder 端和 Decoder 端最底部的输入 embedding。Positional Encoding 具有与 embedding 相同的维度d_{model}　，因此可以对两者进行求和。

具体做法是使用不同频率的正弦和余弦函数，公式如下：

$$PE_{(pos,2i)} = \sin\left(\text{pos}\ /10000^{2i/\ \text{d}_{\text{model}}}\right)$$
$$PE_{(pos,2i+1)} = \cos\left(\text{pos}\ /10000^{2i/d_{\text{model}}}\right)$$

其中 pos 为位置，i 为维度，之所以选择这个函数，是因为任意位置 PE_{pos+k} 可以表示为 PE_{pos} 的线性函数，这个主要是三角函数的特性：

$$\sin(\alpha+\beta) = \sin(\alpha)\cos(\beta)+\cos(\alpha)\sin(\beta)$$
$$\cos(\alpha+\beta) = \cos(\alpha)\cos(\beta)-\sin(\alpha)\sin(\beta)$$

需要注意的是，Transformer 中的 Positional Encoding 不是通过网络学习得来的，而是直接通过上述公式计算而来的，论文中也实验了利用网络学习 Positional Encoding，发现结果与上述基本一致，但是论文中选择了正弦和余弦函数版本，因为三角公式不受序列长度的限制，也就是可以对 比所遇到序列的更长的序列 进行表示。

2.Transformer Decoder 端的输入具体是什么？

见上述 Encoder 端 & Decoder 端总览中，对 Decoder 端的输入有详细的分析

3.Transformer 中一直强调的 self-attention 是什么？self-attention 的计

算过程？为什么它能发挥如此大的作用？self-attention 为什么要使用 Q、K、V，仅仅使用 Q、V/K、V 或者 V 为什么不行？

self-attention，也叫 intra-attention，是一种通过自身和自身相关联的 attention 机制，从而得到一个更好的 representation 来表达自身，self-attention 可以看成一般 attention 的一种特殊情况。在 self-attention 中，$Q = K = V$ ，序列中的每个单词(token)和该序列中其余单词(token)进行 attention 计算。self-attention 的特点在于无视词(token)之间的距离直接计算依赖关系，从而能够学习到序列的内部结构，实现起来也比较简单，值得注意的是，在后续一些论文中，self-attention 可以当成一个层和 RNN，CNN 等配合使用，并且成功应用到其他 NLP 任务。

关于 self-attention 的计算过程问题 1 中有详细的解答

关于 self-attention 为什么它能发挥如此大的作用，在上述 self-attention 的介绍中实际上也有所提及，self-attention 是一种自身和自身相关联的 attention 机制，这样能够得到一个更好的 representation 来表达自身，在多数情况下，自然会对下游任务有一定的促进作用，但是 Transformer 效果显著及其强大的特征抽取能力是否完全归功于其 self-attention 模块，还是存在一定争议的，参见论文：How Much Attention Do You Need?A Granular Analysis of Neural Machine Translation Architectures，如下例子可以大概探知 self-attention 的效果：

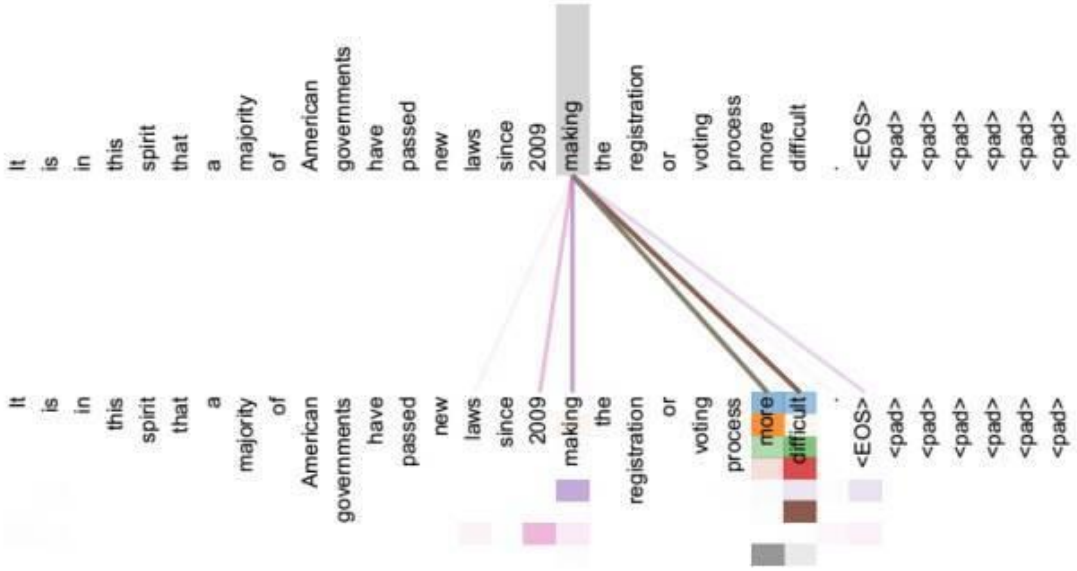


图 1 可视化 self-attention 实例

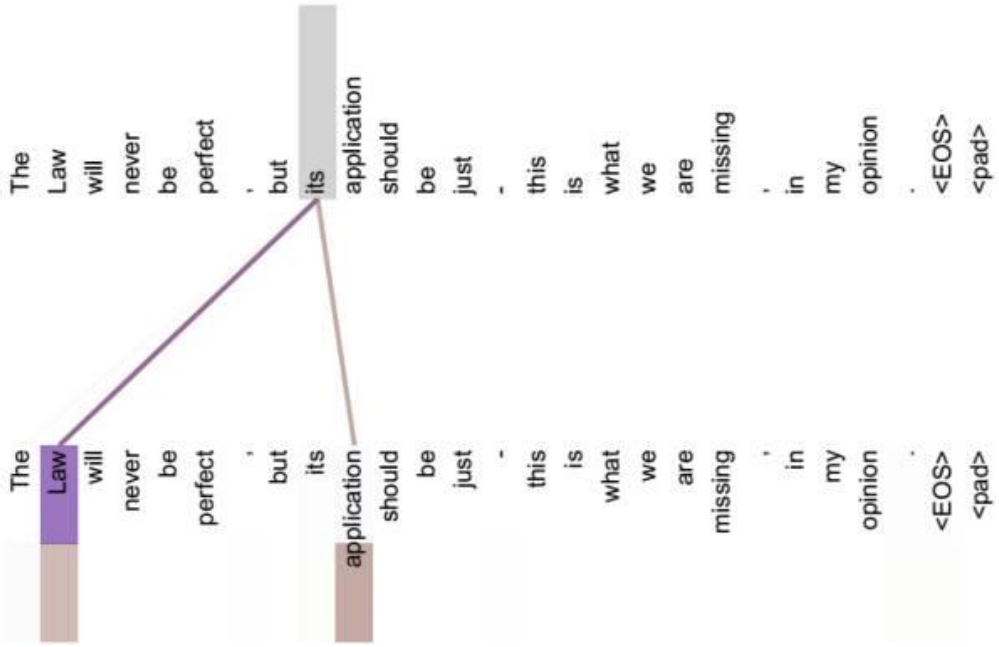


图 2 可视化 self-attention 实例

从两张图（图 1、图 2）可以看出，self-attention 可以捕获同一个句子中单词之间的一些句法特征（比如图 1 展示的有一定距离的短语结构）或者语义特征（比如图 1 展示的 its 的指代对象 Law）。

很明显，引入 Self Attention 后会更容易捕获句子中长距离的相互依赖的特征，因为如果是 RNN 或者 LSTM，需要依次序序列计算，对于远距离的相互依赖的特征，要经过若干时间步步骤的信息累积才能将两者联系起来，而距离越远，有效捕获的可能性越小。

但是 Self Attention 在计算过程中会直接将句子中任意两个单词的联系通过一个计算步骤直接联系起来，所以远距离依赖特征之间的距离被极大缩短，有利于有效地利用这些特征。除此外，Self Attention 对于增加计算的并行性也有直接帮助作用。这是为何 Self Attention 逐渐被广泛使用的主要原因。

关于 self-attention 为什么要使用 Q、K、V，仅仅使用 Q、V/K、V 或者 V 为什么不行？

这个问题我觉得并不重要，self-attention 使用 Q、K、V，这样三个参数独立，模型的表达能力和灵活性显然会比只用 Q、V 或者只用 V 要好些，当然主流 attention 的做法还有很多种，比如说 seq2seq with attention 也就只有 hidden state 来做相似性的计算，处理不同的任务，attention 的做法会有细微的不同，但是主体思想还是一致的，不知道有没有论文对这个问题有过细究，有空去查查~

其实还有个小细节，因为 self-attention 的范围是包括自身的(masked self-attention 也是一样)，因此至少是要采用 Q、V 或者 K、V 的形式，而这样“询问式”的 attention 方式，个人感觉 Q、K、V 显然合理一些。

4.Transformer 为什么需要进行 Multi-head Attention？这样做有什么好处？Multi-head Attention 的计算过程？各方论文的观点是什么？

原论文中说到进行 Multi-head Attention 的原因是将模型分为多个头，形成多个子空间，可

以让模型去关注不同方面的信息，最后再将各个方面的信息综合起来。其实直观上也可以想到，如果自己设计这样的一个模型，必然也不会只做一次 attention，多次 attention 综合的结果至少能够起到增强模型的作用，也可以类比 CNN 中同时使用多个卷积核的作用，直观上讲，多头的注意力有助于网络捕捉到更丰富的特征/信息。关于 Multi-head Attention 的计算过程在 1 中也有详细的介绍，但是需要注意的是，论文中并没有对 Multi-head Attention 有很强的理论说明，因此后续有不少论文对 Multi-head Attention 机制都有一定的讨论，一些相关工作的论文如下(还没看，先攒着)：

Multi-head Attention 机制相关的论文：

A Structured Self-attentive Sentence Embedding

对 Multi-head Attention 机制进行分析的论文：

Analyzing Multi-Head Self-Attention: Specialized Heads Do the Heavy Lifting, the Rest Can Be Pruned

Are Sixteen Heads Really Better than One?

What Does BERT Look At? An Analysis of BERT's Attention

A Multiscale Visualization of Attention in the Transformer Model

Improving Deep Transformer with Depth-Scaled Initialization and Merged Attention

5.Transformer 相比于 RNN/LSTM，有什么优势？为什么？

(1).RNN系列的模型，并行计算能力很差

RNN系列的模型时刻隐层状态的计算，依赖两个输入，一个是 T 时刻的句子输入单词 X_t ，另一个是 $T-1$ 时刻的隐层状态 S_{t-1} 的输出，这是最能体现RNN本质特征的一点，RNN的历史信息是通过这个信息传输渠道往后传输的。而RNN并行计算的问题就出在这里，因为 T 时刻的计算依赖 $T-1$ 时刻的隐层计算结果，而 $T-1$ 时刻的计算依赖 $T-2$ 时刻的隐层计算结果，如此下去就形成了所谓的序列依赖关系。

(2).Transformer的特征抽取能力比RNN系列的模型要好

上述结论是通过一些主流的实验来说明的，并不是严格的理论证明，具体实验对比可以参见：

放弃幻想，全面拥抱Transformer：自然语言处理三大特征抽取器（CNN/RNN/TF）比较

但是值得注意的是，并不是说Transformer就能够完全替代RNN系列的模型了，任何模型都有其适用范围，同样的，RNN系列模型在很多任务上还是首选，熟悉各种模型的内部原理，知其然且知其所以然，才能遇到新任务时，快速分析这时候该用什么样的模型，该怎么做好。

6.Transformer 是如何训练的？测试阶段如何进行测试呢？

Transformer 训练过程与 seq2seq 类似，首先 Encoder 端得到输入的 encoding 表示，并将其输入到 Decoder 端做交互式 attention，之后在 Decoder 端接收其相应的输入(见 1 中有详细分析)，经过多头 self-attention 模块之后，结合 Encoder 端的输出，再经过 FFN，得到 Decoder 端的输出之后，最后经过一个线性全连接层，就可以通过 softmax 来预测下一个单词(token)，然后根据 softmax 多分类的损失函数，将 loss 反向传播即可，所以从整体上来说，Transformer 训练过程就相当于一个有监督的多分类问题。

需要注意的是，Encoder 端可以并行计算，一次性将输入序列全部 encoding 出来，但 Decoder

端不是一次性把所有单词(token)预测出来的，而是像 seq2seq 一样一个接着一个预测出来的。而对于测试阶段，其与训练阶段唯一不同的是 Decoder 端最底层的输入，详细分析见问题 1。

7.Transformer 中的 Add & Norm 模块，具体是怎么做的？

见 1 其他模块的叙述，对 Add & Norm 模块有详细的分析

8.为什么说 Transformer 可以代替 seq2seq？

这里用代替这个词略显不妥当，seq2seq 虽已老，但始终还是有其用武之地，seq2seq 最大的问题在于将 Encoder 端的所有信息压缩到一个固定长度的向量中，并将其作为 Decoder 端首个隐藏状态的输入，来预测 Decoder 端第一个单词(token)的隐藏状态。在输入序列比较长的时候，这样做显然会损失 Encoder 端的很多信息，而且这样一股脑的把该固定向量送入 Decoder 端，Decoder 端不能够关注到其想要关注的信息。上述两点都是 seq2seq 模型的缺点，后续论文对这两点有所改进，如著名的 Neural Machine Translation by Jointly Learning to Align and Translate，虽然确确实实对 seq2seq 模型有了实质性的改进，但是由于主体模型仍然为 RNN(LSTM)系列的模型，因此模型的并行能力还是受限，而 transformer 不但对 seq2seq 模型这两点缺点有了实质性的改进(多头交互式 attention 模块)，而且还引入了 self-attention 模块，让源序列和目标序列首先“自关联”起来，这样的话，源序列和目标序列自身的 embedding 表示所蕴含的信息更加丰富，而且后续的 FFN 层也增强了模型的表达能力，并且 Transformer 并行计算的能力是远远超过 seq2seq 系列的模型，因此我认为这是 transformer 优于 seq2seq 模型的地方。

9.Transformer 中句子的 encoder 表示是什么？如何加入词序信息的？

Transformer Encoder 端得到的是整个输入序列的 encoding 表示，其中最重要的是经过了 self-attention 模块，让输入序列的表达更加丰富，而加入词序信息是使用不同频率的正弦和余弦函数，具体见 1 中叙述。

10.Transformer 如何并行化的？

Transformer的并行化我认为主要体现在self-attention模块，在Encoder端Transformer可以并行处理整个序列，并得到整个输入序列经过Encoder端的输出，在self-attention模块，对于某个序列 $x_1, x_2, \ldots, x_n$ ，self-attention模块可以直接计算 x_i, x_j 的点乘结果，而RNN系列的模型就必须按照顺序从 x_1 计算到 x_n 。

11.self-attention 公式中的归一化有什么作用？

首先说明做归一化的原因，随着 d_k 的增大， $q \cdot k$ 点积后的结果也随之增大，这样会将softmax函数推入梯度非常小的区域，使得收敛困难(可能出现梯度消失的情况)(为了说明点积变大的原因，假设 q 和 k 的分量是具有均值0和方差1的独立随机变量，那么它们的点积 $q \cdot k = \sum_{i=1}^{d_k} q_i k_i$ 均值为0，方差为 d_k)，因此为了抵消这种影响，我们将点积缩放 $\frac{1}{\sqrt{d_k}}$ ，

四、Glove

GloVe 的全称叫 Global Vectors for Word Representation，它是一个基于全局词频统

计（count-based & overall statistics）的词表征（word representation）工具。

1、GloVe 构建过程是怎样的？

（1）根据语料库构建一个共现矩阵，矩阵中的每一个元素 代表单词 和上下文单词 在特定大小的上下文窗口内共同出现的次数。
（2）构建词向量（Word Vector）和共现矩阵之间的近似关系，其目标函数为：
这个 loss function 的基本形式就是最简单的 mean square loss，只不过在此基础上加了一个权重函数 ：
根据实验发现 的值对结果的影响并不是很大，原作者采用了 。而 时的结果要比 时要更好。下面是 时 的函数图象，可以看出对于较小的 ，权值也较小。这个函数图像如下所示：

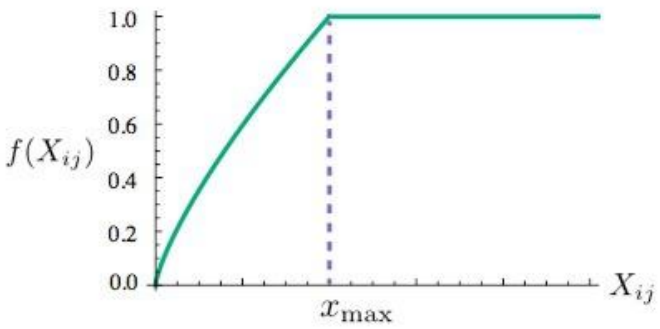


Figure 1: Weighting function f with $\alpha = 3/4$.

2、GloVe 的训练过程是怎样的？

1. 实质上还是监督学习：虽然glove不需要人工标注为无监督学习，但实质还是有label就是 $log(X_{ij})$ 。
2. 向量 w 和 $\tilde{w}$ 为学习参数，本质上与监督学习的训练方法一样，采用了AdaGrad的梯度下降算法，对矩阵 X 中的所有非零元素进行随机采样，学习曲率（learning rate）设为0.05，在vector size小于300的情况下迭代了50次，其他大小的vectors上迭代了100次，直至收敛。
3. 最终学习得到的是两个词向量是 $\tilde{w}$ 和 w ，因为 X 是对称的（symmetric），所以从原理上讲 $\tilde{w}$ 和 w ，是也是对称的，他们唯一的区别是初始化的值不一样，而导致最终的值不一样。所以这两者其实是等价的，都可以当成最终的结果来使用。但是为了提高鲁棒性，我们最终会选择两者之和 $w + \tilde{w}$ 作为最终的vector（两者的初始化不同相当于加了不同的随机噪声，所以能提高鲁棒性）。

3、Glove 损失函数是如何确定的？（来自 GloVe 详解）

- X_{ij}表示单词j出现在单词i的上下文中的次数；
- X_i表示单词i的上下文中所有单词出现的总次数，即$X_i = \sum^k X_{ik}$；
- $P_{ij} = P(j|i) = X_{ij}/X_i$，即表示单词$j$出现在单词$i$的上下文中的概率；

有了这些定义之后，我们来看一个表格：

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

理解这个表格的重点在最后一行，它表示的是两个概率的比值（ratio），**我们可以使用它观察出两个单词i和j相对于单词k哪个更相关（relevant）。**比如，ice和solid更相关，而stream和solid明显不相关，于是我们会发现$P(solid|ice)/P(solid|steam)$比1大更多。同样的gas和steam更相关，而和ice不相关，那么$P(gas|ice)/P(gas|steam)$就远小于1；当都有关（比如water）或者都没有关(fashion)的时候，两者的比例接近于1；这个是很直观的。因此，**以上推断可以说明通过概率的比例而不是概率本身去学习词向量可能是一个更恰当的方法**，因此下文所有内容都围绕这一点展开。

于是为了捕捉上面提到的概率比例，我们可以构造如下函数：

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}, \tag{4}$$

其中，函数F的参数和具体形式未定，它有三个参数w_i, w_j和$\tilde{w}_k$，w和$\tilde{w}$是不同的向量；
因为向量空间是线性结构的，所以要表达出两个概率的比例差，最简单的办法是作差，于是我们得到：

$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}, \tag{5}$$

这时我们发现公式5的右侧是一个数量，而左侧则是一个向量，于是我们把左侧转换成两个向量的内积形式：

$$F((w_i - w_j)^T \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}, \tag{6}$$

我们知道X是个对称矩阵，单词和上下文单词其实是相对的，也就是如果我们做如下交换：$w \leftrightarrow \tilde{w}_k$，$X \leftrightarrow X^T$公式6应该保持不变，那么很显然，现在的公式是不满足的。为了满足这个条件，首先，我们要求函数F要满足同态特性（homomorphism）：

$$F((w_i - w_j)^T \tilde{w}_k) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)}, \tag{7}$$

结合公式6，我们可以得到：

$$F(w_i^T \tilde{w}_k) = P_{ik} = \frac{X_{ik}}{X_i}, \tag{8}$$

然后，我们令$F = \exp$，于是我们有：

$$w_i^T \tilde{w}_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i), \tag{9}$$

此时，我们发现因为等号右侧的$\log(X_i)$的存在，公式9是不满足对称性（symmetry）的，而且这个$\log(X_i)$其实是跟k独立的，它只跟i有关，于是我们可以针对w_i增加一个bias term b_i把它替换掉，于是我们有：

$$w_i^T \tilde{w}_k + b_i = \log(X_{ik}), \tag{10}$$

但是公式10还是不满足对称性，于是我们针对w_k增加一个bias term b_k，从而得到公式1的形式：

$$w_i^T \tilde{w}_k + b_i + b_k = \log(X_{ik}), \tag{11}$$

五、BERT

1. Bert 的基本原理

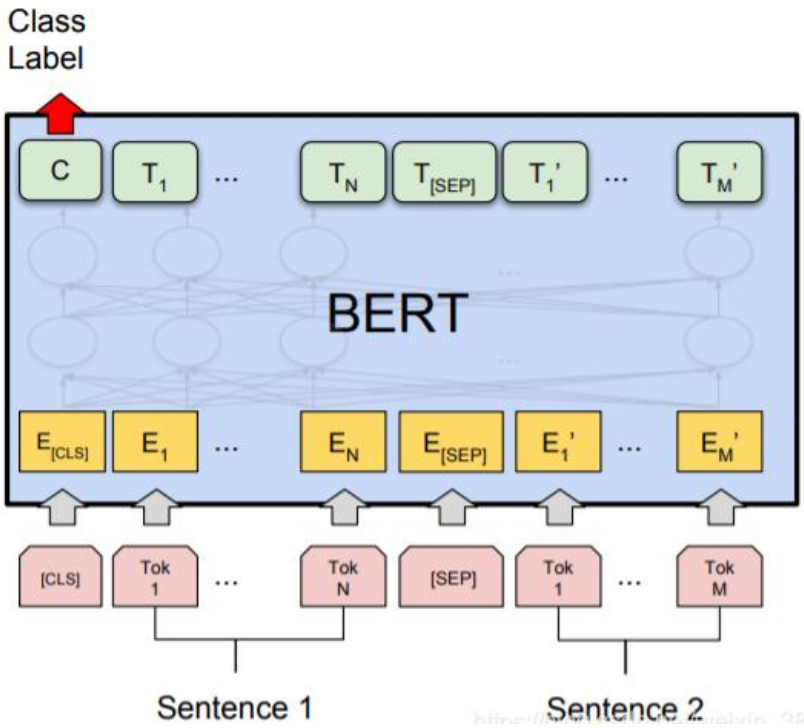
Bert 的参考论文《Pre-training of Deep Bidirectional Transformers for Language Understanding》，设计了两个任务来预训练该模型：

第一个任务是采用 Mask LM 的方式来训练语言模型，通俗地说就是在输入一句话时，随机地选一些要预测的词，然后用一个特殊的符号[MASK]来代替它们，之后让模型根据所给的标签去学习这些地方该填的词。

第二个任务在双向语言模型的基础上额外增加一个句子级别的连续性预测任务，即预测输入 Bert 的两端文本是否为连续的文本，引入这个任务可以更好地让模型学到连续的文本片段之间的关系。

其次，Bert 用的 fine-tune 主要应用在以下 4 大场景中：

- 句子语义相似度的任务



- 多标签分类的任务

多标签分类任务，即 MultiLabel，指的是一个样本可能同时属于多个类，即有多个标签。如一件 L 尺寸的衣服，该样本至少有两个标签----型号：L，类型：冬装。

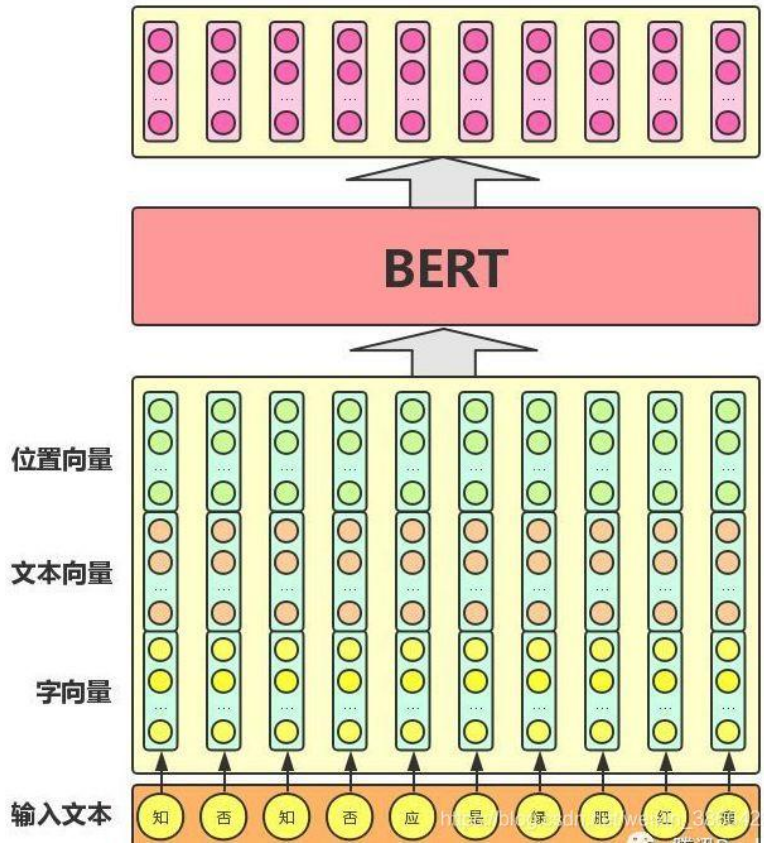
对于多标签任务，通常做法是给它训练几个分类模型即可，然后再一一判断在该类别中，其属于哪个子类别，但是这种做法太 low，而且没有考虑到多个任务之间的依赖关系，其实可以用一个模型解决的。

利用 Bert 模型解决多标签分类问题时，其输入与普通单标签分类问题一致，得到 Bert 输出的 embedding 后，后接全连接层，softmax 分类，这样会得到 loss1,loss2,loss3，最后将所有的 loss 加起来。这种做法相当于将 n 个分类模型的特征提取层参数共享，得到一个共享的表示，最后再做多标签分类任务。

- 针对翻译的任务
- 文本生成的任务

2. Bert 模型的输入和输出

Bert 模型的主要输入是文本中各个字/词(token)的原始词向量，该向量可以是随机初始化的也可以利用 Word2Vector 等算法进行预训练作为初始值；输出是文本中各个字/词(token)融合了全文语义信息后的向量表示：



此外，模型的输入除了字向量（token embedding）之外，还包含另外两个部分：

- 文本向量（segment embedding）：该向量的取值在模型训练过程中自动学习，用于刻画文本的全局语义信息，并与单字/词的语义信息相融合。
- 位置向量（position embedding）：由于出现在文本不同位置的字/词所携带的语义信息存在差异，因此 Bert 模型对不同位置的字/词分别附加一个不同的向量以作区分。

最后，Ber 模型将字向量、文本向量和位置向量的加和作为模型的输入。特别地，在目前的 Bert 模型中，文章作者还将英文词汇作为进一步切割，划分为更细粒度的语义单位（WordPiece），如将 playing 分别为 play 和##ing；此外，对于中文，目前作者未对文本进行分词，而实直接将单字作为构成文本的基本单位。

需要注意的是，上图中只是简单介绍了单个句子输入 Bert 模型中的表示，实际上，在做 Next Sentence Prediction 任务时，在第一个句子的首部会加上一个**[CLS]** token，在两个句子中间以及最后一个句子的尾部会加上一个**[SEP]** token。

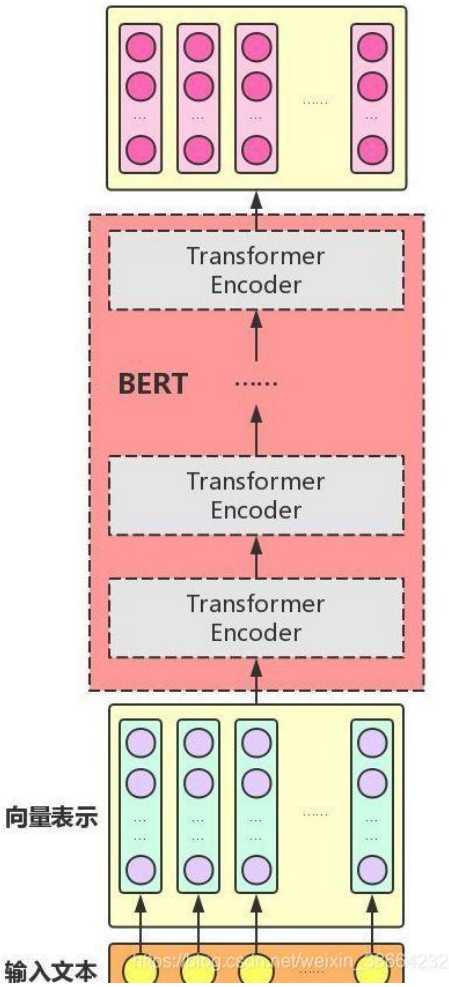
3. Bert 模型中的 transformer 架构

Bert 只使用了 transformer 的 encoder 模块，原论文中，作者分别用了 12 层和 24 层 Transformer Encoder 组装了两套 Bert 模型，分别是：

- $Bert_{BASE}: L=12, H=768, A=12, Total Parameters=110M$
- $Bert_{LARGE}: L=24, H=1024, A=16, Total Parameters=340M$

其中，层（即 Transformer Encoder 块）的数量为 L，隐藏层的维度是 H，自注意头的数量是 A。在所有例子中，我们将前馈/过滤器（即 Transformer Encoder 端的 feed-forward 层）的维度设置为 4H，即当 H=768 时是 3072；当 H=1024 时是 4096.

图示如下：



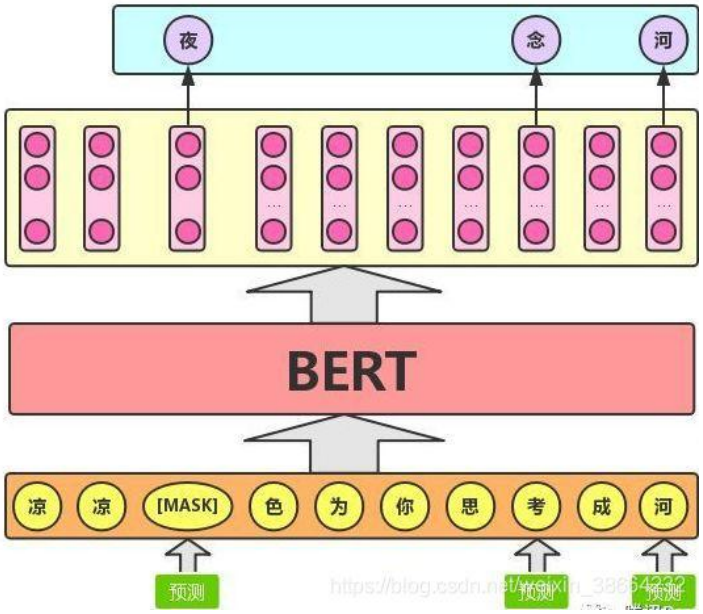
注意：与 transformer 本身的 encoder 端相比，Bert 的 Transformer Encoder 端输入的向量中多了 Segent Embedding 向量。

4. Bert 模型的训练过程

论文中，作者提出了两个预训练任务：Maked LM 和 Next Sentence Prediction。

4.1 Makded LM

Masked LM 的任务描述为：给定一句话，随机 Mask 掉这句话中的一个或几个词，要求根据剩余词汇预测被 Mask 掉的几个词是什么，如下图所示：



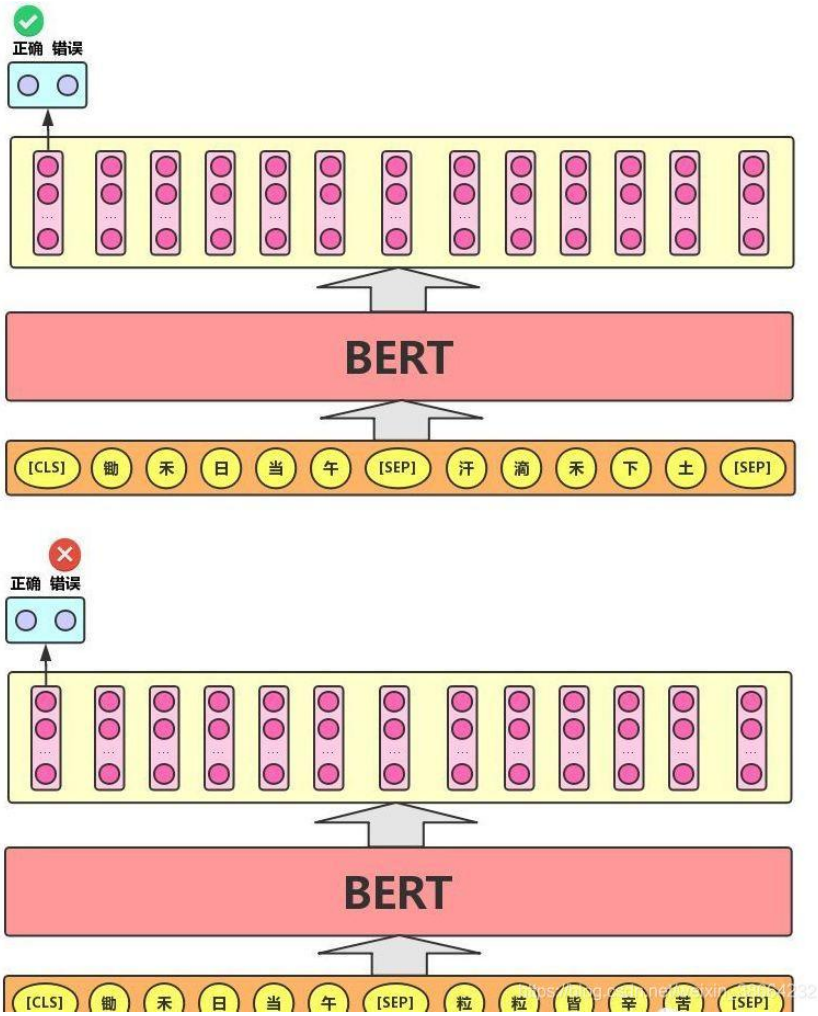
具体来说就是，在一句话中随机选择 15%的词汇用于预测。对于原句中被 Mask 掉的词汇，80%的情况会使用一个特殊符号**[MASK]**替换，10%的情况下采用一个**任意词**替换，剩余 10%情况下保持**原词汇不变**。

这么做的优点是：在后续微调的过程中，语句并不会出现[MASK]标记，当预测词汇时，模型并不知道输入对应位置的词汇是否为正确的词汇（10%的概率），这就迫使模型更多地依赖于上下文信息去预测词汇，并且赋予模型一定的纠错能力。

缺点：由于每批次数据中，只有 15%的标记被预测，这意味着模型需要更多的预训练步骤来收敛。

4.2 Next Sentence Prediction

Next Sentence Prediction 的任务描述为：给定一篇文章中的两句话，判断第二句话在文本中是否紧跟在第一句话之后，如下图所示：



Next Sentence Prediction 任务实际上是段落重排序的简化版：只考虑两句话，判断是否是一篇文章中的前后句。在实际预训练过程中，文章作者从文本语料库中随机选择 50%正确语句对和 50%错误语句对进行训练，与 Masked LM 任务相结合，让模型能够更准确地刻画语句乃至整篇文章层面的语义信息。

5. Bert 两个预训练任务的损失函数

Bert 的损失函数由两部分组成，第一部分来自 Masked LM（单词级别分类任务），另一部分是 Next Sentence Prediction（句子级别的分类任务）。通过这两个任务的联合学习，可以使得 Bert 学习到的表征既有 token 级别信息，同时也包含了句子级别的语义信息。具体损失函数如下：

$$L(/theta, /theta_1, /theta_2) = L_1(/theta, /theta_1) + L_2(/theta, /theta_2) /colorGreen$$

其中，$/theta$ 是 Bert 中 encoder 部分的参数，$/theta_1$ 是 Mask-LM 任务中在 encoder 上所接的输出层中的参数；$/theta_2$ 则是在句子预测任务中在 encoder 接上的分类器参数。因此在第一部分的损失函数中，如果被 mask 的词集合为 M，因为它是一个词典|V| 上的多分类问题，那么具体说来

有：

$$L_1(/theta, /theta_1) = -/sum_{i=1}^{M} logp(m = m_i|/theta, /theta_1), m_i/in[1, 2, ..., |V|]$$

在句子预测任务中，也是一个分类问题的损失函数：

$$L_2(/theta, /theta_2) = -/sum_{j=1}^{N} logp(n = n_i|/theta, /theta_2), n_i/in[IsNext, NotNext]$$

因此，两个任务联合学习的损失函数是：

$$L(/theta, /theta_1, /theta_2) = -/sumlogp(m = m_i|/theta, /theta_1) - /sum_{j=1}^{N} logp(n = n_1|/theta, /theta_2)$$

具体的训练工程实现细节方面，Bert 还利用了一系列策略，使得模型更易于训练，比如对于学习率的 warm-up 策略，使用的激活函数不再是普通的 ReLu，而实 GeLu，也使用了 dropout 等常见的训练技巧。

6. Bert vs mask

6.1 Bert 模型为什么要用 mask?

原因就是使用 mask 是为了引入噪声，属于增强模型鲁棒性的一种手段。通过上面的内容可以知道，通过在输入 X 句子中随机 mask 掉一部分单词，然后在预训练过程中根据上下文单词来预测这些 mask 掉的单词，这种模式就是典型的 Denosing Autoencoder 的思路，那些被 mask 掉的单词就是在输入侧加入的所谓噪音。类似 Bert 这种预训练模式，被称为 DAE LM。

DAE LM 预训练模式的优点：它能比较自然地融入双向语言模型，同时看到被预测单词的上文和下文。缺点：主要在输入侧引入[Mask]标记，导致预训练阶段和 Fine-Tune 阶段不一致问题。（这个问题的解决参考 6.2 内容）

6.2 如何使用 mask?

在给定的句子 X 中，会随机 mask 15%的词，然后让 Bert 来预测这些 mask 的词，如同如同上述 6.1 所述，在输入侧引入[Mask]标记会导致预训练和 Fine-Tune 阶段不一致问题，论文中作者采取了如下措施：

如果某个 token 在被 mask 掉的 15%里，则按照下面的方式随机的执行：

- 80%的概率替换成[MASK]；
- 10%的概率替换成随机的一个词；
- 10%的概率它本身不变

这样做的好处是，Bert 并不知道[MASK]替换掉的是这 15%token 中的哪一个词，而且任何一个词都可能是被替换掉的，这样强迫模型在编码当前时刻的时候不能太依赖于当前的词，而要考虑它的上下文，甚至对其上下文进行“纠错”。

6.3 其 mask 相对于 CBOW 有什么异同点?

相同点：CBOW 的核心思想是，给定上下文，根据它的上文 context-before 和下文 context-after 去预测 input word。Bert 本质上也是如此。

不同点：首先，在 CBOW 中，每个单词会称为 input word，而 Bert 中只有 15%的词会称为 input word。其次，对于数据输入部分，CBOW 中的输入数据只有待预测单词的上下文，而 Bert 的输入是带有[MASK] token 的“完整”句子，也就是说 Bert 在输入端将待预测的 input word 用[MASK] token 代替了。

另外，通过 CBOW 模型训练后，每个单词的 word embedding 是唯一的，因此并不能很好的处理一词多异的问题；而 Bert 模型得到的 word embedding （token embedding）融合了上下文信息，就算是同一个单词，在不同的上下文环境下，得到的 word embedding 是不一样的。

7. Bert vs ELMO vs GPT

7.1 为什么 Bert 比 ELMO 效果好?

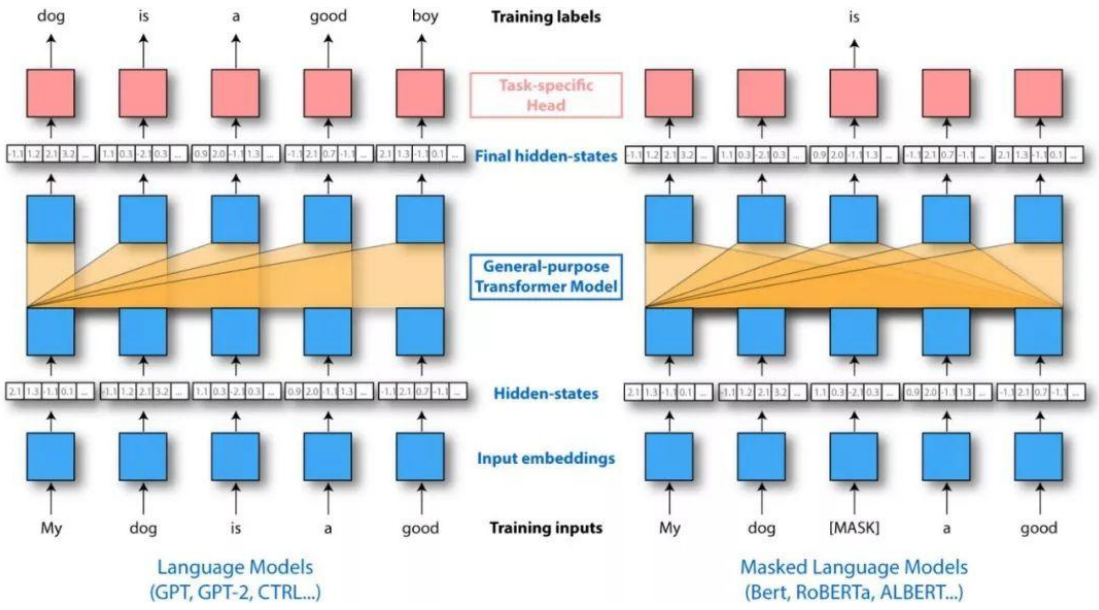
从网络结构以及最后的实验效果来看，BERT 比 ELMo 效果好主要集中在以下几点原因：

1. LSTM 抽取特征的能力远弱于 Transformer
2. 拼接方式双向融合的特征融合能力偏弱(没有具体实验验证，只是推测)
3. 其实还有一点，BERT 的训练数据以及模型参数均多余 ELMo，这也是比较重要的一点

7.2 ELMO 与 Bert 的区别是什么?

ELMo 模型是通过语言模型任务得到句子中单词的 embedding 表示，以此作为补充的新特征给下游任务使用。因为 ELMO 给下游提供的是每个单词的特征形式，所以这一类预训练的方法被称为“Feature-based Pre-Training”。而 BERT 模型是“基于 Fine-tuning 的模式”，这种做法和图像领域基于 Fine-tuning 的方式基本一致，下游任务需要将模型改造成 BERT 模型，才可利用 BERT 模型预训练好的参数。

7.3 Bert 和 GPT 有什么不同?



- GPT 不是双向的，没有 masking 概念
- Bert 在训练中加入下一个句子预测任务，所以它有 sengment 嵌入

7.4 ELMO、GPT、Bert 三者之间有什么区别？

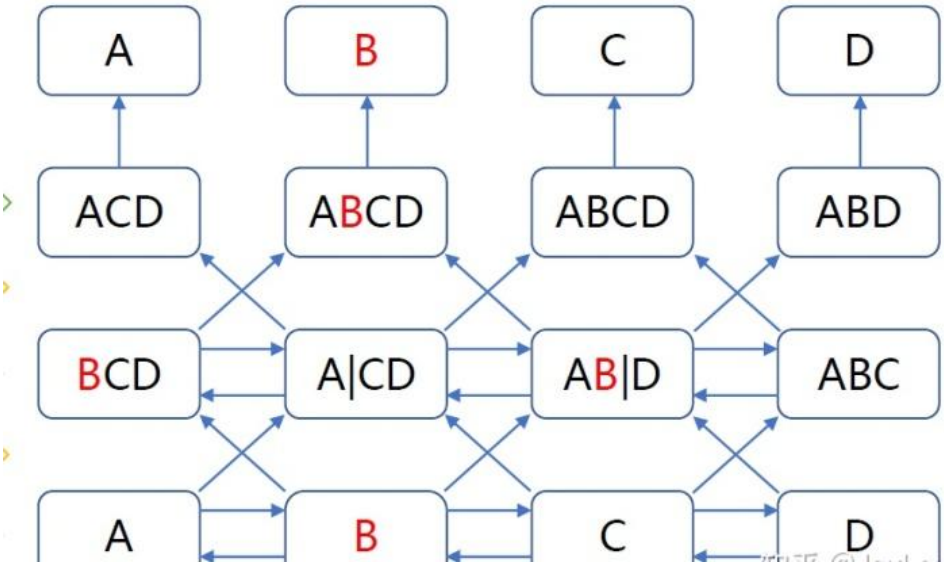
下面从几个方面对这三者进行对比：
（1）特征提取器：elmo 采用 LSTM 进行提取，GPT 和 bert 则采用 Transformer 进行提取。很多任务表明 Transformer 特征提取能力强于 LSTM，elmo 采用 1 层静态向量+2 层 LSTM，多层提取能力有限，而 GPT 和 bert 中的 Transformer 可采用多层，并行计算能力强。
（2）单/双向语言模型：
GPT 采用单向语言模型，elmo 和 bert 采用双向语言模型。但是 elmo 实际上是两个单向语言模型（方向相反）的拼接，这种融合特征的能力比 bert 一体化融合特征方式弱。
GPT 和 bert 都采用 Transformer，Transformer 是 encoder-decoder 结构，GPT 的单向语言模型采用 decoder 部分，decoder 的部分见到的都是不完整的句子；bert 的双向语言模型则采用 encoder 部分，采用了完整句子。

7.5 Bert 和 ALBert v2 有什么不同?

1. ALBert 中，嵌入矩阵分解（减少参数数量）
2. ALBert 中没有 dropout
3. ALBert 中实现了参数共享（有助于减少参数数量并进行正则化）

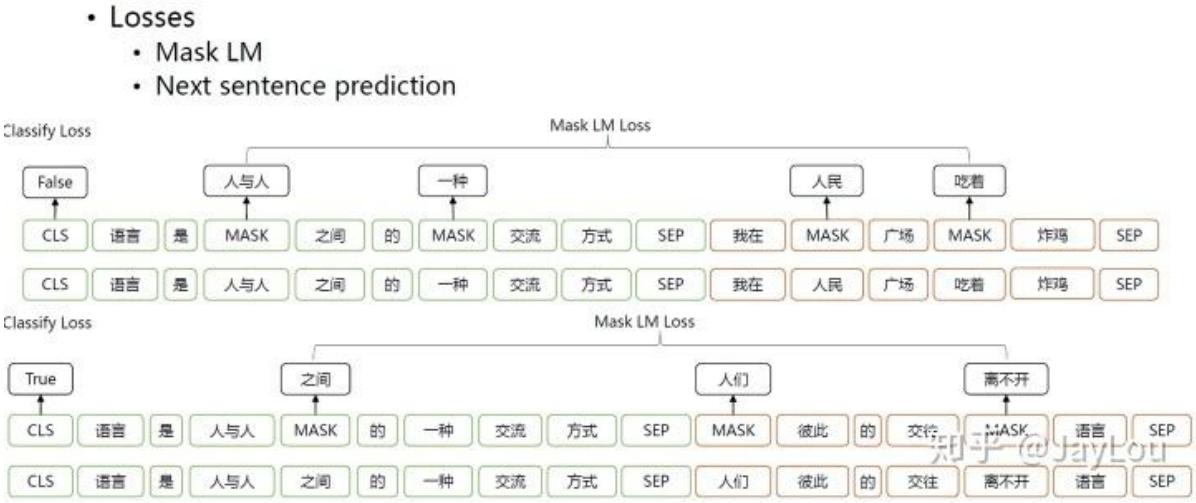
8、为什么 bert 采取的是双向 Transformer Encoder，而不叫 decoder？
BERT Transformer 使用双向 self-attention，而 GPT Transformer 使用受限制的 self-attention，其中每个 token 只能处理其左侧的上下文。双向 Transformer 通常被称为“Transformer encoder”，而左侧上下文被称为“Transformer decoder”，

decoder 是不能获要预测的信息的。
9、bert 构建双向语言模型不是很简单吗？不也可以直接像 elmo 拼接 Transformer decoder 吗？
BERT 的作者认为，这种拼接式的 bi-directional 仍然不能完整地理解整个语句的语义。更好的办法是用上下文全向来预测[mask]，也就是用“能/实现/语言/表征/../的/模型”，来预测[mask]。BERT 作者把上下文全向的预测方法，称之为 deep bi-directional。
10、bert 为什么要采取 Marked LM，而不直接应用 Transformer Encoder？
我们知道向 Transformer 这样深度越深，学习效果会越好。可是为什么不直接应用双向模型呢？因为随着网络深度增加会导致标签泄露。如下图：

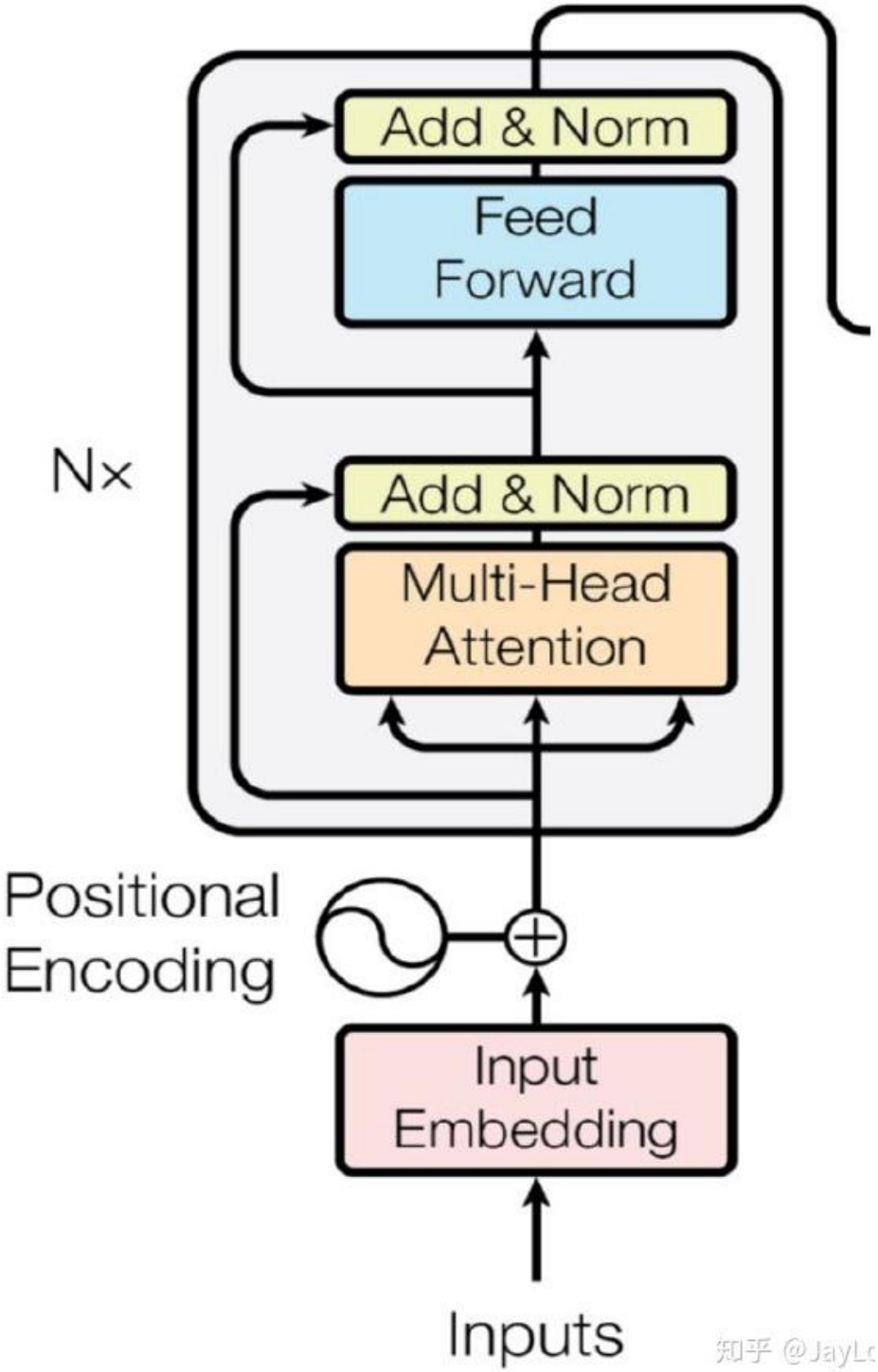


双向编码与网络深度的冲突
深度双向模型比 left-to-right 模型或 left-to-right and right-to-left 模型的浅层连接更强大。遗憾的是，标准条件语言模型只能从左到右或从右到左进行训练，因为双向条件作用将允许每个单词在多层上下文中间接地“see itself”。
为了训练一个深度双向表示（deep bidirectional representation），研究团队采用了一种简单的方法，即随机屏蔽（masking）部分输入 token，然后只预测那些被屏蔽的 token。论文将这个过程称为“masked LM”(MLM)。
11、bert 为什么并不总是用实际的[MASK]token 替换被“masked”的词汇？
NLP 必读 | 十分钟读懂谷歌 BERT 模型： 虽然这确实能让团队获得双向预训练模型，但这种方法有两个缺点。首先，预训练和 finetuning 之间不匹配，因为在 finetuning 期间从未看到[MASK]token。为了解决这个问题，团队并不总是用实际的[MASK]token 替换被“masked”的词汇。相反，训练数据生成器随机选择 15％的 token。例如在这个句子“my dog is hairy”中，它选择的 token 是“hairy”。然后，执行以下过程：
数据生成器将执行以下操作，而不是始终用[MASK]替换所选单词：
80％的时间：用[MASK]标记替换单词，例如，my dog is hairy → my dog is [MASK]
10％的时间：用一个随机的单词替换该单词，例如，my dog is hairy → my dog is apple
10％的时间：保持单词不变，例如，my dog is hairy → my dog is hairy. 这样做的

目的是将表示偏向于实际观察到的单词。
Transformer encoder 不知道它将被要求预测哪些单词或哪些单词已被随机单词替换，因此它被迫保持每个输入 token 的分布式上下文表示。此外，因为随机替换只发生在所有 token 的 1.5％（即 15％的 10％），这似乎不会损害模型的语言理解能力。
使用 MLM 的第二个缺点是每个 batch 只预测了 15％的 token，这表明模型可能需要更多的预训练步骤才能收敛。团队证明 MLM 的收敛速度略慢于 left-to-right 的模型（预测每个 token），但 MLM 模型在实验上获得的提升远远超过增加的训练成本。
bert 模型的主要创新点都在 pre-train 方法上，即用了 Masked LM 和 Next Sentence Prediction 两种方法分别捕捉词语和句子级别的 representation 。

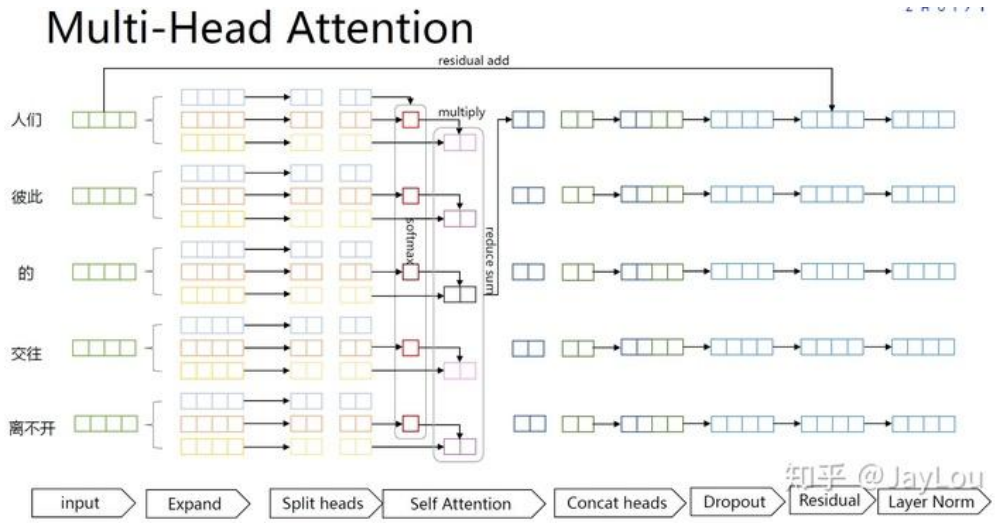


下面给出了 Transformer Encoder 模型的整体结构：



Transformer

Encoder



12. Bert 的局限性

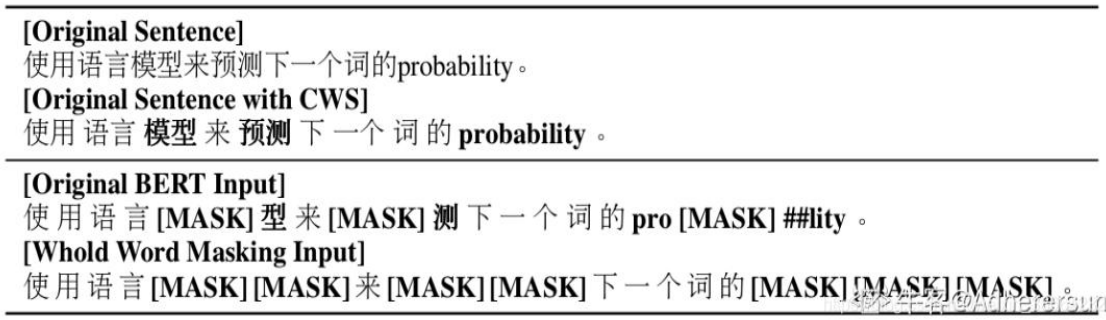
从 XLNet 论文中，提到了 BERT 的两个缺点，分别如下：

- BERT 在第一个预训练阶段，假设句子中多个单词被 Mask 掉，这些被 Mask 掉的单词之间没有任何关系，是条件独立的，然而有时候这些单词之间是有关系的，比如”New York is a city”，假设我们 Mask 住”New”和”York”两个词，那么给定”is a city”的条件下”New”和”York”并不独立，因为”New York”是一个实体，看到”New”则后面出现”York”的概率要比看到”Old”后面出现”York”概率要大得多。

  但是需要注意的是，这个问题并不是什么大问题，甚至可以说对最后的结果并没有多大的影响，因为本身 BERT 预训练的语料就是海量的(动辄几十个 G)，所以如果训练数据足够大，其实不靠当前这个例子，靠其它例子，也能弥补被 Mask 单词直接的相互关系问题，因为总有其它例子能够学会这些单词的相互依赖关系。

- BERT 的在预训练时会出现特殊的[MASK]，但是它在下游的 fine-tune 中不会出现，这就出现了预训练阶段和 fine-tune 阶段不一致的问题。其实这个问题对最后结果产生多大的影响也是不够明确的，因为后续有许多 BERT 相关的预训练模型仍然保持了[MASK]标记，也取得了很大的结果，而且很多数据集上的结果也比 BERT 要好。但是确确实实引入[MASK]标记，也是为了构造自编码语言模型而采用的一种折中方式。

- 另外还有一个缺点，是 BERT 在分词后做[MASK]会产生的一个问题，为了解决 OOV 的问题，我们通常会把一个词切分成更细粒度的 WordPiece。BERT 在 Pretraining 的时候是随机 Mask 这些 WordPiece 的，这就可能出现只 Mask 一个词的一部分的情况，例如：



probability 这个词被切分成"pro"、"#babi"和"#lity"3 个 WordPiece。有可能出现的一种随机 Mask 是把"#babi" Mask 住，但是"pro"和"#lity"没有被 Mask。这样的预测任务就变得容易了，因为在"pro"和"#lity"之间基本上只能是"#babi"了。这样它只需要记住一些词(WordPiece 的序列)就可以完成这个任务，而不是根据上下文的语义关系来预测出来的。类似的中文的词"模型"也可能被 Mask 部分(其实用"琵琶"的例子可能更好，因为这两个字只能一起出现而不能单独出现)，这也会让预测变得容易。

为了解决这个问题，很自然的想法就是词作为一个整体要么都 Mask 要么都不 Mask，这就是所谓的 Whole Word Masking。这是一个很简单的想法，对于 BERT 的代码修改也非常少，只是修改一些 Mask 的那段代码。

13. 从词袋模型 word2vec 改进了什么？从 word2vec 到 Bert 又改进了什么?

13.1 从词袋模型到 word2vec 改进了什么?

词袋模型(Bag-of-words model)是将一段文本（比如一个句子或是一个文档）用一个"装着这些词的袋子"来表示，这种表示方式不考虑文法以及词的顺序。**「而在用词袋模型时，文档的向量表示直接将各词的词频向量表示加和」**。通过上述描述，可以得出词袋模型的两个缺点：

- 词向量化后，词与词之间是有权重大小关系的，不一定词出现的越多，权重越大。
- 词与词之间是没有顺序关系的。

而 word2vec 是考虑词语位置关系的一种模型。通过大量语料的训练，将每一个词语映射成一个低维稠密向量，通过求余弦的方式，可以判断两个词语之间的关系，word2vec 其底层主要采用基于 CBOW 和 Skip-Gram 算法的神经网络模型。

因此，综上所述，词袋模型到 word2vec 的改进主要集中于以下两点：

- 考虑了词与词之间的顺序，引入了上下文的信息
- 得到了词更加准确的表示，其表达的信息更为丰富

13.2 从 word2vec 到 Bert 又改进了什么?

word2vec 到 BERT 的改进之处其实没有很明确的答案，如同上面的问题所述，BERT 的思想其实很大程度上来源于 CBOW 模型，如果从准确率上说改进的话，BERT 利用更深的模型，以及海量的语料，得到的 embedding 表示，来做下游任务时的准确率是要比 word2vec 高不少的。实际上，

这也离不开模型的“加码”以及数据的“巨大加码”。再从方法的意义角度来说，BERT 的重要意义在于给大量的 NLP 任务提供了一个泛化能力很强的预训练模型，而仅仅使用 word2vec 产生的词向量表示，不仅能够完成的任务比 BERT 少了很多，而且很多时候直接利用 word2vec 产生的词向量表示给下游任务提供信息，下游任务的表现不一定会很好，甚至会比较差。

14.BERT 模型可以使用无监督的方法做文本相似度任务吗?

1.首先一点是在不 finetune 的情况下，cosine similairty 绝对值没有实际意义，bert pretrain 计算的 cosine similairty 都是很大的，如果你直接以 cosine similariy>0.5 之类的阈值来判断相似不相似那肯定效果很差。如果用做排序，也就是 cosine(a,b)>cosine(a,c)->b 相较于 c 和 a 更相似，是可以用的。总而言之就是你模型评价的标准应该使用 auc，而不是 accuracy
2.短文本（新闻标题）语义相似度任务用先进的 word embedding（英文 fasttext/glove，中文 tencent embedding）mean pooling 后的效果就已经不错；而对于长文本（文章）用 simhash 这种纯词频统计的完全没语言模型的简单方法也 ok
3.bert pretrain 模型直接拿来用作 sentence embedding 效果甚至不如 word embedding，cls 的 emebdding 效果最差（也就是你说的 pooled output）。把所有普通 token embedding 做 pooling 勉强能用（这个也是开源项目 bert-as-service 的默认做法），但也不会比 word embedding 更好。
4.用 siamese 的方式训练 bert，上层通过 cosine 做判别，能够让 bert 学习到一种适用于 cosine 作为最终相似度判别的 sentence embedding，效果优于 word embedding，但因为缺少 sentence pair 之间的特征交互，比原始 bert sentence pair fine tune 还是要差些。

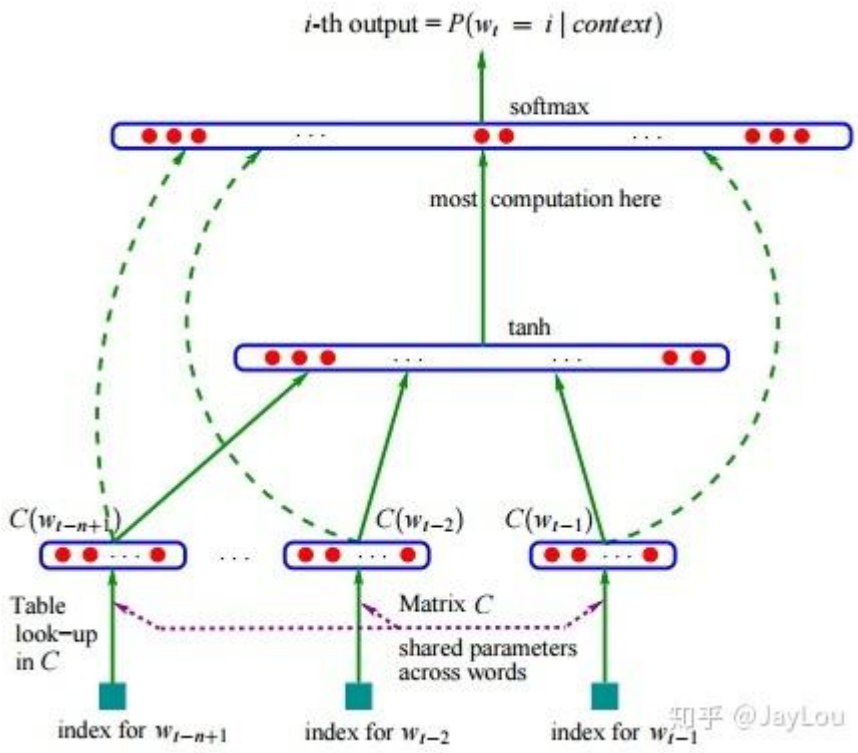
六、NLP 综合面试题

1.文本表示哪些方法？

下面对文本表示进行一个归纳，也就是对于一篇文本可以如何用数学语言表示呢？
基于 one-hot、tf-idf、textrank 等的 bag-of-words；
主题模型：LSA（SVD）、pLSA、LDA；
基于词向量的固定表征：word2vec、fastText、glove
基于词向量的动态表征：elmo、GPT、bert

2.怎么从语言模型理解词向量？怎么理解分布式假设？

上面给出的 4 个类型也是 nlp 领域最为常用的文本表示了，文本是由每个单词构成的，而谈起词向量，one-hot 是可认为是最为简单的词向量，但存在维度灾难和语义鸿沟等问题；通过构建共现矩阵并利用 SVD 求解构建词向量，则计算复杂度高；而早期词向量的研究通常来源于语言模型，比如 NNLM 和 RNNLM，其主要目的是语言模型，而词向量只是一个副产物。



NNLM

所谓分布式假设，用一句话可以表达：相同上下文语境的词有似含义。而由此引申出了 word2vec、fastText，在此类词向量中，虽然其本质仍然是语言模型，但是它的目标并不是语言模型本身，而是词向量，其所作的一系列优化，都是为了更快更好的得到词向量。glove 则是基于全局语料库、并结合上下文语境构建词向量，结合了 LSA 和 word2vec 的优点。

3.传统的词向量有什么问题？怎么解决？各种词向量的特点是什么？

上述方法得到的词向量是固定表征的，无法解决一词多义等问题，如“川普”。为此引入基于语言模型的动态表征方法：elmo、GPT、bert。

各种词向量的特点：

（1）One-hot 表示 ：维度灾难、语义鸿沟；

（2）分布式表示 (distributed representation) ：

矩阵分解（LSA）：利用全局语料特征，但 SVD 求解计算复杂度大；

基于 NNLM/RNNLM 的词向量：词向量为副产物，存在效率不高等问题；

word2vec、fastText：优化效率高，但是基于局部语料；

glove：基于全局预料，结合了 LSA 和 word2vec 的优点；

elmo、GPT、bert：动态特征；

4.什么 perplexity？它在 NLP 中的地位是什么？

perplexity 是一种表达模型在预测中出现的混乱程度的方法。熵越大=越混乱。使用 perplexity 来评估 NLP 中的语言模型。一个好的语言模型会给正确的预测赋予更高的概率。

5.使用 SVD 学习潜在特征和使用深度网络获取嵌入向量有什么区别？

SVD 使用输入的线性组合，而设逆境网络使用非线性组合。

6. transformer 的时间复杂度？

(序列长度**2)*hidden，当 hidden 尺寸大于序列长度时（通常是这种情况），transformer 速度比 LSTM 快。

7. 为什么 self-attention 这么牛 B？

在计算复杂性方面，当序列长度 n 小于表示维数 d 时，self-attention 层速度比 recurrent 层要快，实际情况也是这样，同时可以在机器翻译中使用最先进的模型来进行句子的比较，比如 word-piece 和 byte-pair 表示。

8. 在多任务学习中，软、硬参数共享的区别是什么？

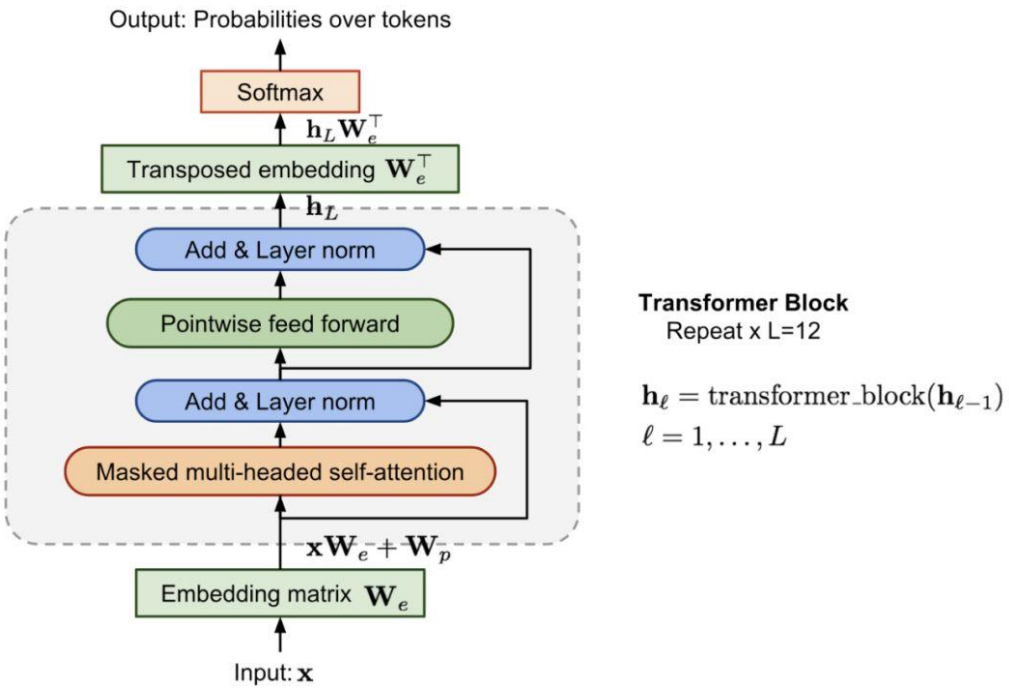
在硬共享中，我们一次训练所有的任务，并根据所有的损失更新权重。在软共享中，我们一次只训练一个任务。

9. BatchNorm 和 LayerNorm 的区别？

BatchNorm----为每一个 batch 计算每一层的均值和方差

LayerNorm----独立计算每一层每一个样本的均值和方差

10. 为什么 transformer 使用 LayerNorm，而不是 BatchNorm?



从 LayerNorm 的优点看，它对 batch 大小是健壮的，并且在样本级别而不是 batch 级别工作得更好。

11. 如果你知道你的训练数据有错误，你会对你的深度学习代码做什么改变？

我们可以做标签平滑，其中的平滑值是基于百分误差。如果任何特定的类有已知的误差，我们还可以使用类权值来修正损失。

12. 在 transformer 中使用最多的层是哪一层？

Dropout

13. 说一个不适用 dropout 的语言模型

ALBert v2：ALBert 中参数共享的正则化效果非常强，不需要 dropout(ALBert v1 中有 dropout)

14. 如何减少训练好的神经网络模型的推理时间？

1. 在 GPU/TPU/FPGA 上进行服务
2. 16 位量化，部署在支持 fp16 的 GPU 上提供服务
3. 剪枝以减少模型参数
4. 知识蒸馏（用于较小的 transformer 模型或简单的神经网络）
5. 采用分层 softmax

15.如何对中文分词问题用隐马尔可夫模型进行建模和训练？

场景描述

介绍隐马尔可夫模型之前，首先了解马尔科夫过程。马尔科夫过程是满足无后效性的随机过程。**当前状态仅与前一状态相关。**时间和状态都是离散的马尔科夫过程也叫马尔科夫链

隐马尔可夫模型是对含有未知参数（隐状态）的马尔科夫链进行建模的过程

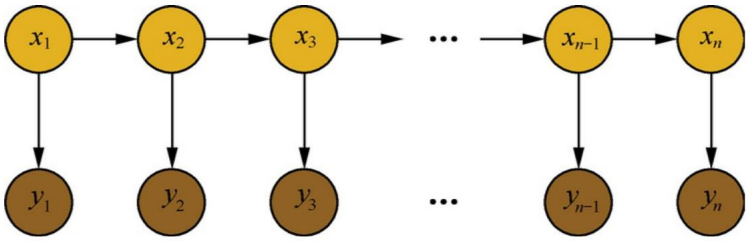


图6.5　隐马尔可夫模型

隐马尔可夫模型是对含有未知参数（隐状态）的马尔可夫链进行建模的生成模型，概率图模型如图6.5所示。在简单的马尔可夫模型中，所有状态对于观测者都是可见的，因此在马尔可夫模型中仅仅包括状态间的转移概率。而在隐马尔可夫模型中，隐状态x_i对于观测者而言是不可见的，观测者能观测到的只有每个隐状态x_i对应的输出y_i，而观测状态y_i的概率分布仅仅取决于对应的隐状态x_i。在隐马尔可夫模型中，参数包括了隐状态间的转移概率、隐状态到观测状态的输出概率、隐状态x的取值空间、观测状态y的取值空间以及初始状态的概率分布。

隐马尔可夫模型包括概率计算问题、预测问题、学习问题三个基本问题。

（1）概率计算问题：已知模型的所有参数，计算观测序列Y出现的概率，可使用前向和后向算法求解。

（2）预测问题：已知模型所有参数和观测序列Y，计算最可能的隐状态序列X，可使用经典的动态规划算法——维特比算法来求解最可能的状态序列。

（3）学习问题：已知观测序列Y，求解使得该观测序列概率最大的模型参数，包括隐状态序列、隐状态之间的转移概率分布以及从隐状态到观测状态的概率分布，可使用Baum-Welch算法进行参数的学习，Baum-Welch算法是最大期望算法的一个特例。

隐马尔可夫模型通常用来做序列标注问题，因此可以将分词问题转化为序列标注问题

下面回到开头的问题。隐马尔可夫模型通常用来解决序列标注问题，因此也可以将分词问题转化为一个序列标注问题来进行建模。例如可以对中文句子中的每个字做以下标注，B表示一个词开头的第一个字，E表示一个词结尾的最后一个字，M表示一个词中间的字，S表示一个单字词，则隐状态的取值空间为$\{B,E,M,S\}$。同时对隐状态的转移概率可以给出一些先验知识，B和M后面只能是M或者E，S和E后面只能是B或者S。而每个字就是模型中的观测状态，取值空间为语料中的所有中文字。完成建模之后，使用语料进行训练可以分有监督训练和无监督训练。有监督训练即对语料进行标注，相当于根据经验得到了语料的所有隐状态信息，然后就可以用简单的计数法来对模型中的概率分布进行极大似然估计。

16 最大熵隐马尔可夫模型为什么会产生标注偏置问题，如何解决？

实际上，在序列标注问题中，隐状态（标注）不仅和单个观测状态相关，还和观察序列的长度、上下文等信息相关。例如词性标注问题中，一个词被标注为动词还是名词，不仅与它本身以及它前一个词的标注有关，还依赖于上下文中的其他词，于是引出了最大熵马尔可夫模型（Maximum Entropy Markov Model，MEMM），如图6.6所示。最大熵马尔可夫模型在建模时，去除了隐马尔可夫模型中观测状态相互独立的假设，考虑了整个观测序列，因此获得了更强的表达能力。同时，隐马尔可夫模型是一种对隐状态序列和观测状态序列的联合概率$P(x,y)$进行建模的生成式模型，而最大熵马尔可夫模型是直接对标注的后验概率$P(y|x)$进行建模的判别式模型。

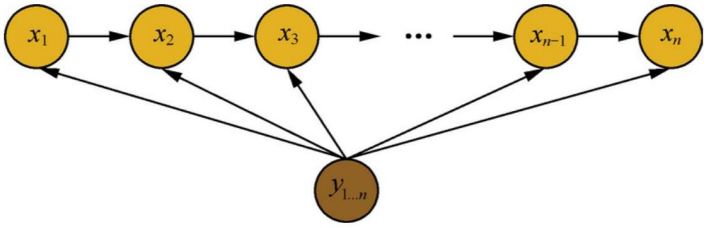


图6.6　最大熵马尔可夫模型

最大熵马尔可夫模型建模如下

$$p(x_{1\cdots n}|y_{1\cdots n})=\prod_{i=1}^{n}p(x_i\mid x_{i-1},y_{1\cdots n}),$$

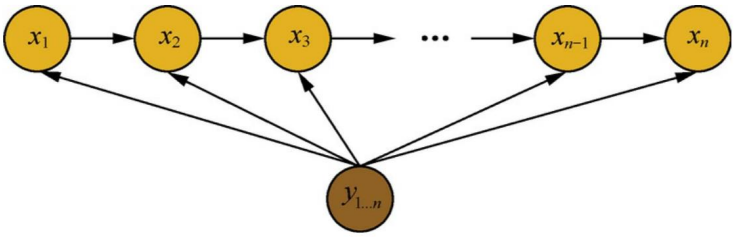


图6.6　最大熵马尔可夫模型

最大熵马尔可夫模型建模如下

$$p(x_{1\cdots n}|y_{1\cdots n})=\prod_{i=1}^{n}p(x_i\mid x_{i-1},y_{1\cdots n}),$$

其中$p(x_i \mid x_{i-1}, y_{1\cdots n})$会在局部进行归一化，即枚举$x_i$的全部取值进行求和之后计算概率，计算公式为

$$p(x_i|x_{i-1}, y_{1\cdots n}) = \frac{\exp(F(x_i, x_{i-1}, y_{1\cdots n}))}{Z(x_{i-1}, y_{1\cdots n})},$$

（6.23）

其中Z为归一化因子

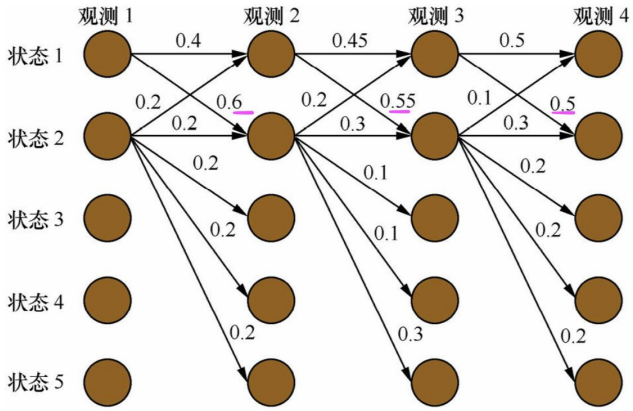
$$Z(x_{i-1}, y_{1\cdots n}) = \sum_{x_i} \exp(F(x_i, x_{i-1}, y_{1\cdots n})),$$

（6.24）

其中$F(x_i, x_{i-1}, y_{1\cdots n})$为$x_i, x_{i-1}, y_{1\cdots n}$所有特征的线性叠加。

标注偏置问题

最大熵马尔可夫模型存在标注偏置问题，如图6.7所示。可以发现，状态1倾向于转移到状态2，状态2倾向于转移到状态2本身。但是实际计算得到的最大概率路径是1->1->1->1，状态1并没有转移到状态2，如图6.8所示。这是因为，从状态2转移出去可能的状态包括1、2、3、4、5，概率在可能的状态上分散了，而状态1转移出去的可能状态仅仅为状态1和2，概率更加集中。由于局部归一化的影响，隐状态会倾向于转移到那些后续状态可能更少的状态上，以提高整体的后验概率。这就是标注偏置问题。



它是一种判别式的概率无向图模型，既然是判别式，那就是对条件概率分布建模。

条件随机场（Conditional Random Field，CRF）在最大熵马尔可夫模型的基础上，进行了全局归一化，如图6.9所示。

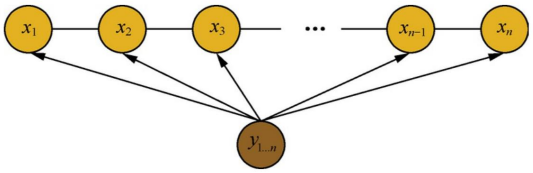


图6.9　条件随机场

条件随机场建模如下

$$p(x_{1...n}|y_{1...n}) = \frac{1}{Z(y_{1...n})}\prod_{i=1}^{n}\exp(F(x_i, x_{i-1}, y_{1...n})),$$

（6.25）

其中归一化因子$Z(y_{1...n})$是在全局范围进行归一化，枚举了整个隐状态序列$x_{1...n}$的全部可能，从而解决了局部归一化带来的标注偏置问题。