

NLP Theories

NLP Theories	1
1. Common interview questions	3
1.1 Spelling correction in Search engine	3
1.2 End to end deep learning	4
1.3 Discriminative model vs Generative model	5
1.3.1 Discriminative model	5
1.3.2 Generative model	6
1.4 Feature selection in text categorization	6
2. Text Preprocessing	8
2.1 Data cleaning	8
2.2 Tokenization	9
2.2.1 Approaches	9
2.2.2 Tools	11
2.3. Remove stopwords	12
2.4 Bag of Words	12
2.4.1 Bag of words	12
2.4.2 TF-IDF	12
3. RNN	14
3.1 Architecture	14
3.2 Forward propagation	15
3.3 Backward propagation	15
3.4 Loss function	15
3.4.1 Cross Entropy	15
3.4.2 Binary_crossentropy	17
3.4.3 Categorical_crossentropy	17
3.5 Vanishing Gradients and Exploding Gradients	18
3.5.1 Overcome Vanishing Gradients	18
3.5.2 Overcome Exploding Gradients	18
4. LSTM	19
4.1 Architecture	19
4.1.1 Forget gate	20
4.1.2 Input gate	20
4.1.3 Output gate	21

5. GRU	22
5.1 Architecture	22
5.2 Difference with LSTM	23
6. Seq2Seq LM	24
6.1 Architecture	24
6.2 Beam Search	26
6.3 Bleu score	26
7. Attention	27
7.1 Scaled dot-product attention	29
7.2 Self - attention	30
7.3 Multi-head Attention	30
8. Transformer	31
8.1 Architecture	32
8.2 Encoder and Decoder stacks	32
8.3 Position encoding	33
8.4 Multi-head attention in Transformer	33
8.5 Position-wise Feed-Forward Networks	34
8.6 Add & Norm	34
8.6.1 Batch Norm vs Layer Norm	34
8.7 Universal Transformers	35
9. Overfitting	35
9.1 L1 regularization	35
9.2 L2 regularization	36
9.3 Dropout	36
9.4 Batch Normalization	36
9.5 Early stopping	39
10. Word Embedding	39
10.1 NNLM	40
10.2 Word2Vec	41
10.2.1 CBOW	41
10.2.2 SkipGram	42
10.2.3 Hierarchical softmax	43
10.2.4 Negative sampling	45
10.3 Fasttext	50
10.3.1 N-gram in Fasttext	52
10.3.2 Hashbucket	53

10.4 Matrix Factorization	53
10.4.1 LDA	54
10.4.2 LSA	55
10.5 GloVe	55
10.5.1 Word2Vec vs GloVe	56
10.6 BERT	57
10.6.2 Masked LM	59
10.6.3 Next Sentence Prediction (NSP)	60
10.6.4 Fine-tuning based on down-stream tasks	60
10.6.5 ELMo vs GPT vs BERT	62
10.6.6 Optimizations of BERT	63
11. Down-stream tasks	65
11.1 Machine Translation	65
11.2 Sentimental Analysis	66
11.3 Sequential tagging	68
11.3.1 HMM	69
11.3.2 Belief Networks	75
11.3.3 CRF	76
11.3.4 Bi-LSTM + CRF	77
11.4 Language Modeling	78
11.4.1 N-gram	78
11.4.2 BERT	83

1. Common interview questions

1.1 Spelling correction in Search engine

Google的拼写检查是基于贝叶斯方法。

用户输入单词时，可能拼错也可能拼对。如果把拼正确的情况记做c，拼错的情况记做w，那么拼写检查要做的事就是：在发生w的情况下，试图推断出c。换言之，已知w，然后在若干个备选方案中，找出可能性最大的c，也就是 $P(c|w)$ 的最大值。

根据贝叶斯定理，有：
$$P(c|w) = P(w|c) * P(c) / P(w)$$

由于对于所有备选的 c 来说，对应的都是同一个 w ，所以它们的 $P(w)$ 是相同的，因此我们只要最大化 $P(w|c)*P(c)$ 即可。

其中： $P(c)$ 表示某个正确的词的出现"概率"，它可以用"频率"代替。如果我们有一个足够大的文本库，那么这个文本库中每个单词的出现频率，就相当于它的发生概率。某个词的出现频率越高， $P(c)$ 就越大。比如在你输入一个错误的词"Julw"时，系统更倾向于去猜测你可能想输入的词是"July"，而不是"Jult"，因为"July"更常见。

$P(w|c)$ 表示在试图拼写 c 的情况下，出现拼写错误 w 的概率。为了简化问题，假定两个单词在字形上越接近，就有越可能拼错， $P(w|c)$ 就越大。举例来说，相差一个字母的拼法，就比相差两个字母的拼法，发生概率更高。你想拼写单词July，那么错误拼成Julw（相差一个字母）的可能性，就比拼成Jullw高（相差两个字母）。值得一提的是，一般把这种问题称为"编辑距离"。

所以，我们比较所有拼写相近的词在文本库中的出现频率，再从中挑出出现频率最高的一个，即是用户最想输入的那个词。

1.2 End to end deep learning

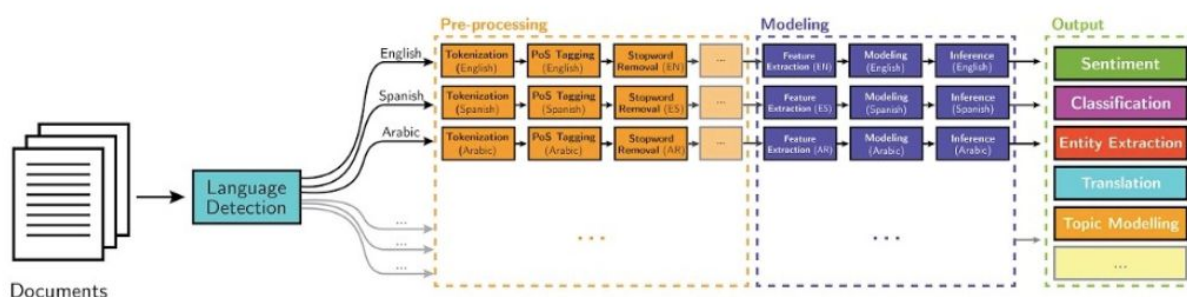
传统的机器学习流程往往由多个独立的模块组成，比如在一个典型的NLP问题中，包括分词、词性标注、句法分析、语义分析等多个独立步骤，每个步骤是一个独立的任务，其结果的好坏会影响下一个步骤，而影响整个训练的结果，这是非端到端的。

而深度学习模型在训练过程中，从输入端到输出端会得到一个预测结果，与真实结果相比较会得到一个误差，这个误差会在反向传播的过程中逐渐优化，知道模型收敛或达到预期效果，这就是端到端的。

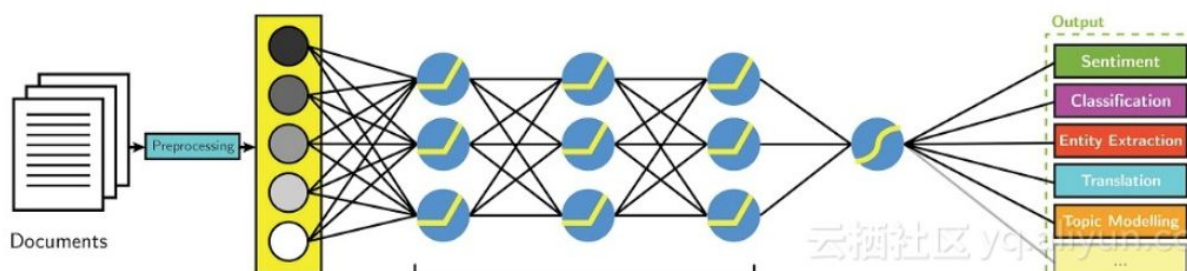
相比之下，端到端省去了每一个独立学习任务执行之前所需要做的数据标注，为样本做标注的代价是昂贵的，且易出错。缺点在于每类具体的问题可能都需要制定一个模型，如果输入或输出有变化可能要调整整个算法，有一定的难度。

近几年端到端的深度学习才开始在NLP大放异彩，主要是因为：

- 1) 大量的训练数据
- 2) 更快的机器和多核CPU/GPU/TPU
- 3) 性能高的新模型和算法：有效的端到端的联合系统学习、有效的使用上下文和任务间转换的学习方法，以及正则化优化方法



Deep Learning-based NLP



1.3 Discriminative model vs Generative model

1.3.1 Discriminative model

判别式模型是直接根据X特征来对Y模型进行建模训练，也就是说，这个训练过程是确定构建模型中的“复杂映射函数”中的参数，然后再去预测或是分类一批新的样本。

判别式模型的特征：

- 1) 对 $P(Y|X)$ 建模
- 2) 对所有样本只建立一个模型，确认总体判别边界
- 3) 观测到输入什么特征，就预测最可能的lable

4) 优点是，对数据没有生成式的严格，速度快，小数据量下准确率也高

常见的判别模型：SVM, LR, LDA（线性判别分析），NN, KNN, CRF, DT, Perception, Boosting等

1.3.2 Generative model

生成式模型的训练阶段对 $P(X,Y)$ 建模，需要确定维护这个联合概率分布的所有信息参数，然后对新的样本计算 $P(Y|X)$ ，导出Y，但这已不是建模阶段。

它的特点是：

- 1) 对 $P(X, Y)$ 建模
- 2) 在分类问题中，需要对每个标签Y都建模，最终选择最优概率的标签为结果，没有判别边界
- 3) 优点在于，包含的信息很全，不仅可以用来输入label，还可以干其他事。生成式模型关注结果是如何产生的，但是生成式模型需要非常充足的数据量以保证采样到了数据本来的面目，所以相比之下速度慢。

常见的生成模型：朴素贝叶斯，混合高斯模型，限制波兹曼机，HMM，LDA（主题文档生成模型）

生成算法尝试去找到这个数据是怎么产生的，然后再对一个信号进行分类。基于你的生成假设，那么哪个类别最有可能产生这个信号，这个信号就属于哪个类别。判别模型不关心数据是怎么生成的，它只关心信号之间的差别，然后用差别来简单对给定的一个信号进行分类。

1.4 Feature selection in text categorization

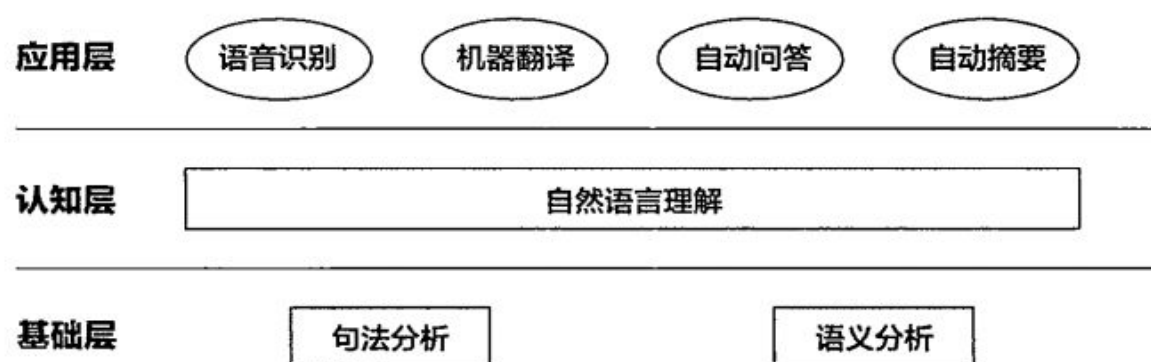
文本分类中，常见的特征选择方法有以下六种：

- 1) DF (Document Frequency) 文档频率：统计特征词出现的文档数量，用来衡量某个特征词的重要性
- 2) MI (Mutual Information) 互信息法：互信息法用来衡量特征词与文档类别直接的信息量。如果某个特征词的频率很低，那么互信息得分就会很大，因此互信息法倾向于“低频”的特征词，相对的词频很高的词，得分就会变低，如果这词携带了很高的信息量，衡量某个特征词的重要性。
- 3) IG (Information Gain) 信息增益法：通过某个特征词的缺失与存在的两种情况下，预料中前后信息的增加，衡量某个特征词的重要性。

4) CHI (Chi-square) 卡方检验法：利用了统计学中“假设检验”的基本思想：首先假定特征词与类别是不相关的，如果利用CHI分布计算出的简言之偏离阈值越大，那么更有信息否定原假设，接受原假设的备择假设：特征词与类别有很高的相关度。

5) WLLR (Weighted Log Likelihood Ration) 加权对数似然

6) WFO (Weighted Frequency and Odds) 加权频率和可能性



通常NLP的问题的解决方案分三种：

1) 基于规则

这是NLP最古老的方法，正则表达式和上下文语法是NLP基于规则的方法的教科书实例。

它倾向于专注于模式匹配或解析；通常可以被认为是“填补空白”的方法；低精度，高召回率

2) 基于统计：机器学习

方法包括概率建模，似然最大化和线性分类器等。特点如下：

- 训练数据：带有标记的语料库
- 特征工程：单词类型、周围单词、大写、复数等
- 训练参数模型，然后拟合测试数据（典型的机器学习系统）
- 推理（将模型应用于测试数据），其目的在于找到最可能的单词、下一个单词、最佳类别等
- 语义槽填充

3) 神经网络（端到端模型）

与传统机器学习类似，但有如下差异：

- 特征工程一般被跳过，因为网络将学习重要的特征
- 输入端是原始数据的矢量表示，直接送到神经网络中后，输出目标数据
- 需要非常大的训练语料库

NLP任务的主要类别：

1) 文本分类

表示：一揽子单词（不保留单词顺序）

目标：预测标签、类别、情绪

应用：过滤垃圾邮件、根据主导内容对文档进行分类

2) 单词序列任务

表示：序列（保留字顺序）

目标：语言建模（预测上一个/下一个单词）、文本生成

应用：翻译、聊天机器人、词性标记（按顺序预测每个单词的POS标签）、命名实体识别

3) 文本含义任务

表示：单词向量、单词到向量的映射也称为词嵌入

目标：我们如何表达意义？

应用：找到相似的单词(类似的向量)，句子嵌入（与词嵌入相对），主题建模，搜索，问答

4) 序列到序列的任务

机器翻译、摘要、简化、问答系统等

这种系统的特征在于编码器和解码器，他们互补的工作以找到文本的隐藏表示，并使用该隐藏表示

2. Text Preprocessing

2.1 Data cleaning

去除标点符号、字母、数字等。

2.2 Tokenization

分词就是将句子、段落、文章这种长文本，分解为以字词为单位的数据结构，方便后续的处理分析工作。

分词可以将复杂问题转化为数学问题；词是一个比较合适的粒度；深度学习时代，部分任务中也可以「分字」。

根据应用的不同，汉语分词的颗粒度大小应该不同。比如，在机器翻译中，颗粒度应该大一些，如“北京大学”就不能分为两个词，比如联想公司作为整体，很容易找到对应的英文Lenovo，如果分词时将它们分开，很可能翻译失败。而在语音识别和搜索中，“北京大学”一般被分成两个词。因此，不同的应用应该有不同分词系统。一般更好的方法是一个分词器同时支持不同层次的词的切分。

中文分词的方法可以用于英语处理中，主要是手写识别体中。因为在识别手写体时，单词之间的空间不是很清楚，中文分词法可以帮助判别英语单词的边界。

颗粒度的不一致性可以用来度量分词器的好坏，对于某些应用，需要尽可能的找到各种复合词，而不是将其切分。因此，需要花一些功夫做数据挖掘工作，不断完善复合词的词典，这是近年来中文分词主要花精力的地方。

2.2.1 Approaches

1) 基于语法规则

基于语法规则的分词法基本思想是在分词的同时进行句法、语义分析，利用句法信息和语义信息来进行词性标注，以解决分词歧义现象。因为现有的语法知识、句法规则十分笼统、复杂，基于语法和规则的分词法所能达到的精确度远远还不能令人满意，目前这种分词系统应用较少。

2) 基于词典

优点：速度快、成本低

缺点：适应性不强，不同领域效果差异大。

基本思想是基于词典匹配，将待分词的中文文本根据一定规则切分和调整，然后跟词典中的词语进行匹配，匹配成功则按照词典的词分词，匹配失败通过调整或者重新选择，如此反复循环即可。可以进一步分为最大匹配法，最大概率法，最短路径法等。最大匹配法指的是按照一定顺序选取字符串中的若干个字当做一个词，去词典中查找。根据扫描方式可细分为：正向最大匹配，反向最大匹配，双向最大匹配，最小切分。最大概率法指的是一个待切分的汉字串可能包含多种分词结果，将其中概率最大的那个作为该字串的分词结果。最短路径法指的是在词图上选择一条词数最少的路径。

3) 基于统计

优点：适应性较强

缺点：成本较高，速度较慢。统计语言模型很大程度上是依照“大众的想法”，或者“多数句子的用法”，而在特定情况下可能是错的。

基本思想是：根据字符串在语料库中出现的统计频率来决定其是否构成词。词是字的组合，相邻的字同时出现的次数越多，就越有可能构成一个词。因此字与字相邻共现的频率或概率能够较好的反映它们成为词的可信度。其对歧义词和未登录词的识别都具有良好的效果常用的方法有HMM（隐马尔科夫模型），MAXENT（最大熵模型），MEMM（最大熵隐马尔科夫模型），CRF（条件随机场）。比如stanford、Hanlp分词工具是基于CRF算法。

4) 基于深度学习

优点：准确率高、适应性强

缺点：成本高，速度慢

例如有人尝试使用双向LSTM+CRF实现分词器，其本质上是序列标注，所以有通用性，命名实体识别等都可以使用该模型，据报道其分词器字符准确率可高达97.5%。

常见的分词器都是使用机器学习算法和词典相结合，一方面能够提高分词准确率，另一方面能够改善领域适应性。

2.2.2 Tools

大部分大公司都有自己的分词工具，而不是用开源工具

下面排名根据 GitHub 上的 star 数排名：

- Hanlp
- Stanford 分词
- ansj 分词器
- 哈工大 LTP
- KCWS分词器
- jieba

精确模式(默认)：试图将句子最精确切开，适合文本分析。

全模式：把句子中可以成词的词语都扫描出来，速度很快，但是不能解决歧义。

搜索引擎模式：在精确模式的基础上，对长词再次划分，提高召回率，适合用于搜索引擎分词。

- IK
- 清华大学THULAC

- ICTCLAS

中英文分词的3个典型区别:

- 分词方式不同, 中文更难
- 英文单词有多种形态, 需要词性还原和词干提取
- 中文分词需要考虑粒度问题

中文分词的难点: 歧义消除、未登录词识别、错别字、谐音字规范化、分词粒度问题等

2.3. Remove stopwords

英文可以直接在nltk library 中import stopwords, 中文需要自己构造停用词, 网上有下载的停用词表。

2.4 Bag of Words

2.4.1 Bag of words

词袋模型能够把一个句子转化为向量表示, 是比较简单直白的一种方法, 它不考虑句子中单词的顺序, 只考虑词表 (vocabulary) 中单词在这个句子中的出现次数。scikit-learn中的CountVectorizer()函数实现了BOW模型

2.4.2 TF-IDF

Term frequency-inverse document frequency

TF-IDF是对搜索关键词的重要性的度量, 是用于信息检索与数据挖掘的常用加权技术, 常用于挖掘文章中的关键词, 在工业界常用于最开始的文本数据清洗。

如果和BOW一样，只考虑关键词的频率，那么停止词和一些通用词的重要性很大（在文件中出现频率高），但是他们并不是对文本主题起到作用的关键词，于是需要给汉语中的每一个词一个权重，这个权重的设定要满足（1）一个词预测主题的能力越强，权重越大，反之越小。（2）停止词的权重为零。

IDF的概念其实就是一个特定条件下关键词的概率分布的交叉熵。

算法步骤

1) 计算词频，并标准化：词频（TF）=某个词在文章中出现的次数/文章的总词数

2) 逆文档频率（IDF）= $\log(\text{语料库的总文档数}/\text{包含该词的文档数}+1)$

那么，如果一个关键词只在很少的文档中出现，通过它就很容易锁定搜索目标，它的权重也就更大，反之，如果一个词在大量文档中出现，看它依然不清楚要找什么内容，它的权重就应该小。比如，一共有D=10亿个文档，“的”字在每个文档中都出现了，那么它的IDF就为0。为了避免分母为0（即所有文档都不包含该词），给分母加了1。

3) 计算 $TF-IDF=TF*IDF$

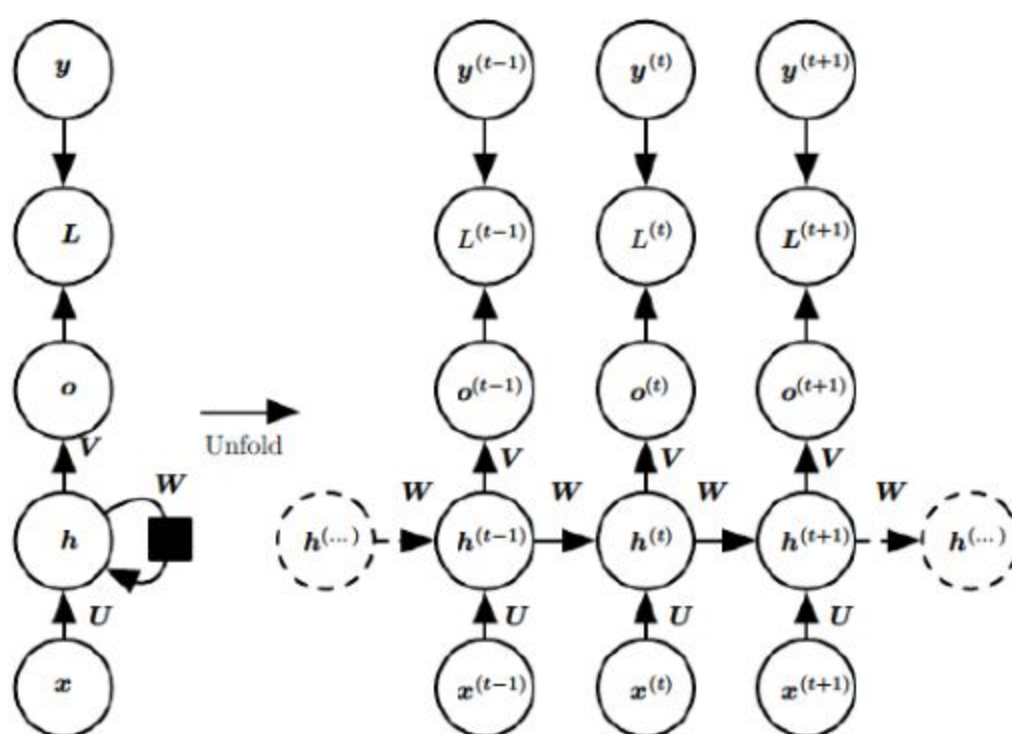
计算出每个文档的每个词的TF-IDF 值，然后按降序排列，取排在最前面的几个词。

TF-IDF的优点是简单快速容易理解，缺点是有时用词频来衡量文章中的一个词的重要性不够全面，有时重要的词可能出现的不够多，而且无法体现位置信息，无法体现词在上下文的重要性。

3. RNN

3.1 Architecture

DNN和CNN的算法都是前向反馈的，模型的输出和模型本身没有关联关系。而RNN的输出和模型间有反馈。



上图中左边的是RNN没有按时间展开的图，右边是按照时间序列展开的图。其中， $x^{(t)}$ 代表在序列索引号 t 时训练样本的输入， $h^{(t)}$ 代表在序列索引号 t 时模型的隐藏状态， $h^{(t)}$ 由 $h^{(t-1)}$ 和 $x^{(t)}$ 共同决定， $o^{(t)}$ 代表在序列索引号 t 时模型的输出，它只由当前的隐藏状态 $h^{(t)}$ 决定， $L^{(t)}$ 代表在序列索引号 t 时模型的损失函数， $y^{(t)}$ 代表在序列索引号 t 时训练样本序列的真实输出。 U , W , V 这三个矩阵是模型的线性关系参数，在整个RNN网络中是共享的。

3.2 Forward propagation

$$h^{(t)} = \sigma(z^{(t)}) = \sigma(Ux^{(t)} + Wh^{(t-1)} + b)$$

其中， σ 为RNN的激活函数，一般为tanh， b 为线性关系的偏差

$$o^{(t)} = Vh^{(t)} + c$$

$$\hat{y}^{(t)} = \sigma(o^{(t)})$$

通常由于RNN是识别类的分类模型，所以一般这个激活函数都是softmax，通过损失函数，比如logloss，我们可以量化模型在当前位置的损失，即 $\hat{y}^{(t)}$ 与 $y^{(t)}$ 的差距。

3.3 Backward propagation

和DNN一样，RNN通过梯度下降法一轮轮迭代，得到适合RNN模型的参数 U, W, V, b, c ，这些参数在序列的各个位置是共享的，反向传播时我们更新相同的参数。

3.4 Loss function

3.4.1 Cross Entropy

它用来衡量两个取值为正数的函数的相似性，对于概率分布或者概率密度函数，如果取值均大于零，交叉熵可以度量两个随机分布的差异性。Q相对于P的交叉熵定义如下：

$$\begin{aligned} H(P, Q) &= -\sum_x P(x) \log Q(x) = -\sum_x P(x) [\log P(x) + \log Q(x) - \log P(x)] = -\sum_x P(x) [\log P(x) + \log P(x)/Q(x)] \\ &= -\sum_x P(x) \log P(x) - \sum_x P(x) \log P(x)/Q(x) = H(P) + D_{KL}(P||Q) \end{aligned}$$

$D_{KL}(P||Q)$ 是KL离散度，也是P与Q的相对熵。而 $H(P)$ 是无法优化的，所以当训练模型以减少交叉熵其实就是减少KL离散度。

作为损失函数一般简化为：

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)}))]$$

对于两个完全相同的函数，他们的交叉熵为零；

交叉熵越大，两个函数差异越大，相反，交叉熵越小，两个函数的差异越小；

NLP中经常用交叉熵来衡量两个常用词在不同文本中的分布概率，看他们是否同义，或者根据两篇文章中不同词的分布，看他们的内容是否相近等。TF-IDF也由它而来。

Maximum Entropy

最大熵代表了整体分布的信息，通常具有最大熵的分布作为该随机变量的分布。保留全部的不确定性，将风险降到最小。投资所说的不要把所有鸡蛋放在一个篮子里，利用的就是最大熵模型这个原理。

$$P(w_3 | w_1, w_2, subject) = \frac{e^{(\lambda_1(w_1, w_2, w_3) + \lambda_2(subject, w_3))}}{Z(w_1, w_2, subject)}$$

对任何一组不自相矛盾的信息，这个最大熵模型不仅存在，而且唯一，并且他们都有同一个简单的形式：指数函数。

以上公式是根据上下文和主题预测下一个词的最大熵模型，其中w3是要预测的词，w1, w2是它的前两个字，也就是其上下文的一个大致估计，subject表示主题。z是归一化因子，保证概率加起来等于一。

最大熵模型在NLP常用来做词性标注、句法分析、语言模型和机器翻译。

最大熵模型可以将各种信息整合到一个统一的模型中。它是唯一一种既可以满足各个信息源的限制条件，同时又可以保证平滑性的模型。但其计算量巨大。

3.4.2 Binary_crossentropy

二分类的对数损失函数，与sigmoid相对应的损失函数。

二进制交叉熵的通用公式：

$$H_p(q) = -\frac{1}{N} \sum_{i=1}^N y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))$$

3.4.3 Categorical_crossentropy

多分类的对数损失函数，与softmax分类器相对应的损失函数，理同上。

此损失函数与上一类同属对数损失函数，sigmoid和softmax的区别主要是，sigmoid用于二分类，softmax用于多分类。

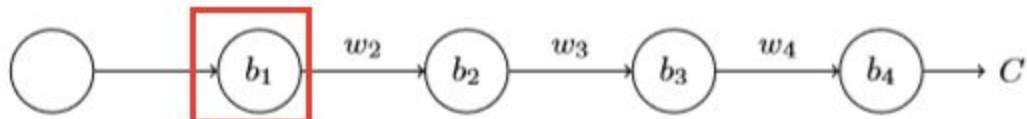
当使用categorical_crossentropy损失时，你的目标值应该是分类格式（即，如果你有10个类，每个样本的目标值应该是一个10维的向量，这个向量除了表示类别的那个索引为1，其他均为0，即onehotencoding后的向量模式）。为了将整数目标值转换为分类目标值，你可以使用Keras实用函数to_categorical。

3.5 Vanishing Gradients and Exploding Gradients

RNN is good for short-term dependency, but it can't handle long-term dependencies well.

神经网络中的梯度不稳定性在于前面层的梯度是后面层上梯度的乘积。

以3层隐藏网络为例，可以推算出以下损失函数对 b_1 的偏导。

$$\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}$$


除开最后一项，整个公式可以看成是多个 $w_j \times \sigma'(z_j)$ 的乘积。

而sigmoid导数函数最大值为0.25，那么 $|w_j \times \sigma'(z_j)|$ 的值肯定小于0.25，这就是梯度消失的原因。

同理，如果 $|w_j \times \sigma'(z_j)|$ 每个都大于1，那么越靠近输入层的学习速度越快，越接近输出层的学习速度会消失，这也就是梯度爆炸问题。为了解决这个问题，可以用relu替代sigmoid作为激活函数。

3.5.1 Overcome Vanishing Gradients

- 1) 使用Relu替代sigmoid激活函数
- 2) 使用Xavier初始化
- 3) 使用LSTM或者GRU代替RNN

3.5.2 Overcome Exploding Gradients

可以用梯度裁剪 (gradient clipping)的方法解决梯度爆炸的问题。具体步骤如下：

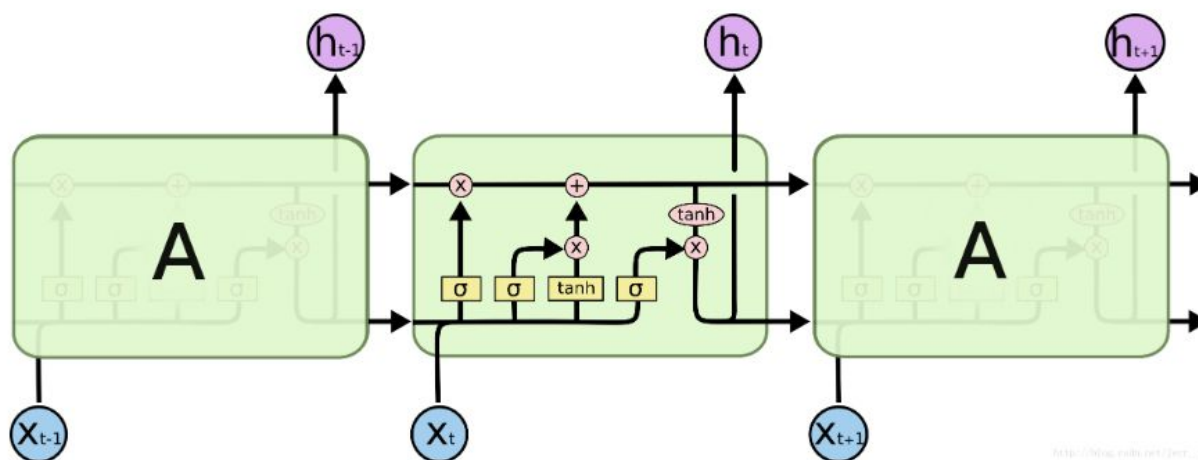
- 1) 首先设置一个梯度阈值：clip_gradient
- 2) 在反向传播中求出各个参数的梯度，这里不直接用梯度去更新参数，而是求这些梯度的L2范数
- 3) 比较梯度的L2范数 $\|g\|$ 与clip_gradient的大小
- 4) 如果前者大，求缩放因子 $\text{clip_gradient} / \|g\|$ ，梯度越大，缩放因子越小，这样就很好地控制了梯度的范围
- 5) 最后将梯度乘以缩放因子便得到最后所需的梯度

4. LSTM

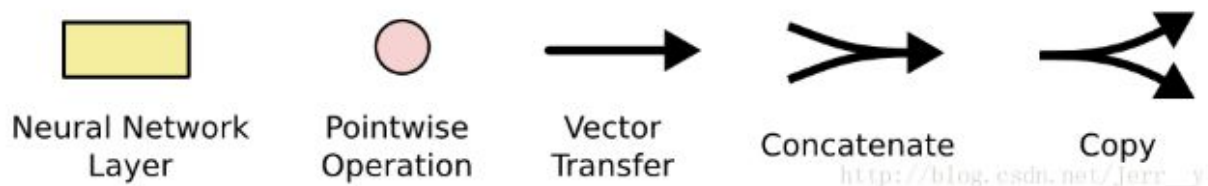
传统的RNN有两个问题，一是梯度爆炸与消失，二是长期记忆容易被短期记忆掩盖，针对梯度消失，LSTM采用门机制解决，针对短期记忆覆盖长期记忆问题，LSTM采用一个cell state来保存长期记忆，再配合门机制对信息进行过滤，从而达到对长期记忆的控制。

4.1 Architecture

和普通的RNN一样，LSTM也是由完全相同结构的模块进行复制而成的，只是LSTM的模块内不再是一个单一的tanh层，而是用了四个相互作用的层，如下：



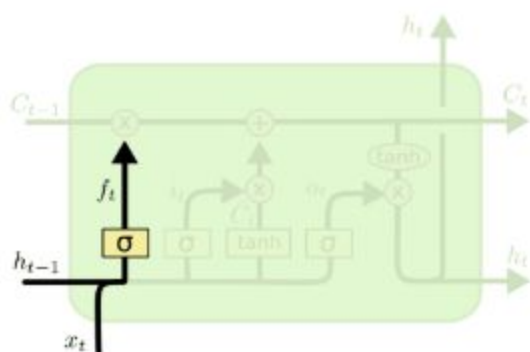
其中，



LSTM通过门的结构，来实现对长期记忆的保留。门可以实现选择性的让信息通过，主要是通过sigmoid的神经层和一个逐点相乘的操作来实现的。

4.1.1 Forget gate

首先是 LSTM 要决定让哪些信息继续通过这个cell，这是通过一个叫做“forget gate layer”的sigmoid 神经层来实现的。它的输入是 $h_{(t-1)}$ 和 x_t ，输出是一个数值都在 0, 1 之间的向量（向量长度和 cell 的状态 $c_{(t-1)}$ 一样，表示让 $c_{(t-1)}$ 的各部分信息通过的比重。0 表示“不让任何信息通过”，1 表示“让所有信息通过”。

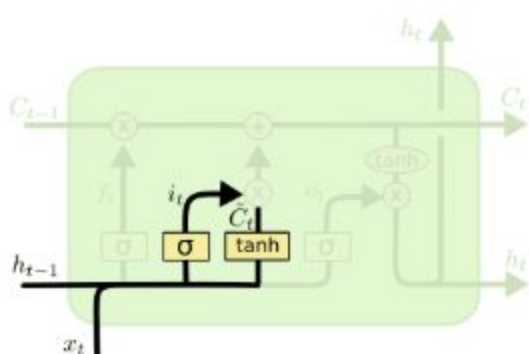


$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

<https://china.csdn.net/jk/>

4.1.2 Input gate

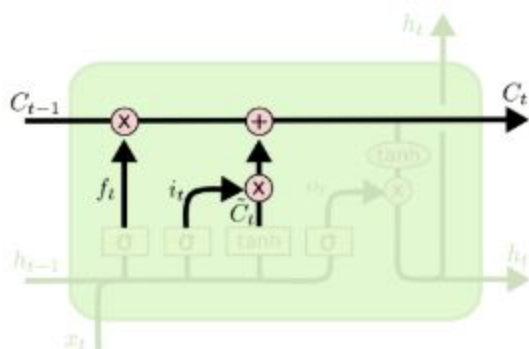
下一步是决定让多少新的信息加入到 cell 状态 中来。实现这个需要包括两个步骤：首先，一个叫做“input gate layer”的 sigmoid 层决定哪些信息需要更新；一个 tanh 层生成一个向量，也就是备选用来更新的内容 \tilde{C}_t 。在下一步，我们把这两部分联合起来，对 cell 的状态进行一个更新。



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

更新cell状态：即把 $c_{(t-1)}$ 更新为 c_t 。从结构图中应该能一目了然，首先我们把旧的状态 $c_{(t-1)}$ 和 f_t 相乘，把一些不想保留的信息忘掉，然后加上 $i_t * \tilde{C}_t$ 。这部分信息就是我们要添加的新内容。



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

4.1.3 Output gate

最后，我们需要来决定输出什么值了。这个输出主要是依赖于cell的状态 C_t ，但是又不仅仅依赖于 C_t ，而是需要经过一个过滤的处理。首先，我们还是使用一个sigmoid层来决定 C_t 中的哪部分信息会被输出。接着，我们把 C_t 通过一个tanh层（把数值都归到 -1 和 1 之间），然后把 tanh 层的输出和 sigmoid 层计算出来的权重相乘，这样就得到了最后输出的结果。

在语言模型例子中，假设我们的模型刚刚接触了一个代词，接下来可能要输出一个动词，这个输出可能就和代词的信息相关了。比如说，这个动词应该采用单数形式还是复数的形式，那么我们就得把刚学到的和代词相关的信息都加入到 cell 状态中来，才能够进行正确的预测。

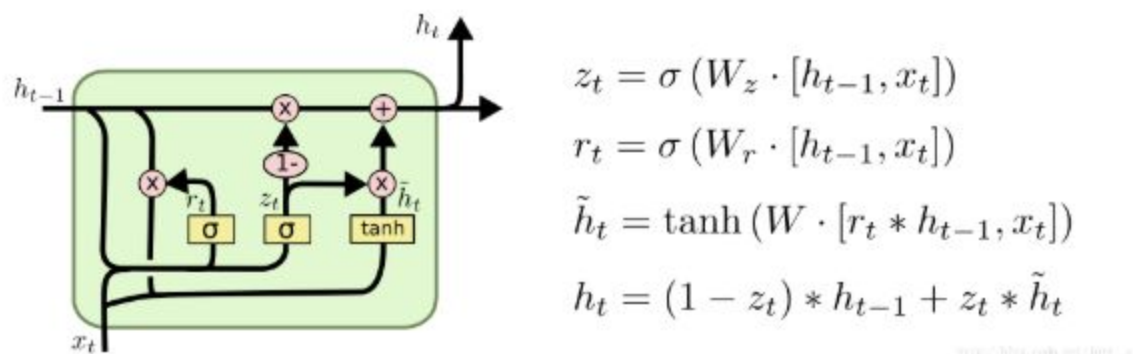
c_t 本质是0-t时刻的全局信息，而 h_t 表示的是在0-(t-1)时刻的全局信息的影响下，当前t时刻的上下文表示。具体到公式中， h_t 本质是先将 c_t 经过tanh激活函数压缩为(-1,1)之间的数值，再通过输出门对 c_t 进行过滤，来获得当前单元的上下文信息， h_t 是全局信息 c_t 的一部分信息；而对于全局信息 c_t ，是由上一时刻的全局信息 c_{t-1} 与当前时刻信息 x_t 通过输入门与遗忘门结合而成的。

5. GRU

Gated Recurrent Unit

5.1 Architecture

只有两个门：重置门（reset gate）和更新门（update gate）。同时在这个结构中，把细胞状态和隐藏状态进行了合并。最后模型比标准的 LSTM 结构要简单，而且这个结构后来也非常流行。



其中， r_t 表示重置门， z_t 表示更新门。重置门决定是否将之前的状态忘记。当 r_t 趋于0的时候，前一个时刻的状态信息 h_{t-1} 会被忘掉，隐藏状态 h_t 会被重置为当前输入的信息。更新门决定是否要将隐藏状态更新为新的状态 h_t 。

5.2 Difference with LSTM

- 1) LSTM选择暴露部分信息（ h_t 才是真正的输出， c_t 只是作为信息载体，并不输出），而GRU选择暴露全部信息。
- 2) 在LSTM中，通过遗忘门和传入门控制信息的保留和传入；GRU则通过重置门来控制是否要保留原来隐藏状态的信息，但是不再限制当前信息的传入。
- 3) 在LSTM中，虽然得到了新的细胞状态 C_t ，但是还不能直接输出，而是需要经过一个过滤的处理。同样，在GRU中，虽然(2)中我们也得到了新的隐藏状态 h_t ，但是还不能直接输出，而是通过更新门来控制最后的输出。
- 4) 由于GRU参数更少，收敛速度更快，可以大大加速迭代过程。实际表现上效果差不多，主要依据具体的任务和数据集而定，他们的表现差距远没有调参所带来的效果明显，所以一般可以选择GRU作为基本的单元，再尝试其他优化技巧，比如将 \tanh 改成 \tanh 的变体，或在权重初始上下功夫，如果还不行，再换成LSTM看会不会有惊喜。

5) LSTM更擅长捕捉短期文本语义理解, 而GRU擅长捕捉较远文本的语义理解, 且利用了重复词语和语义进行预测, 当然这也和具体应用和超参数的设定有关系 (

<https://distill.pub/2019/memorization-in-rnns/>) 。

6. Seq2Seq LM

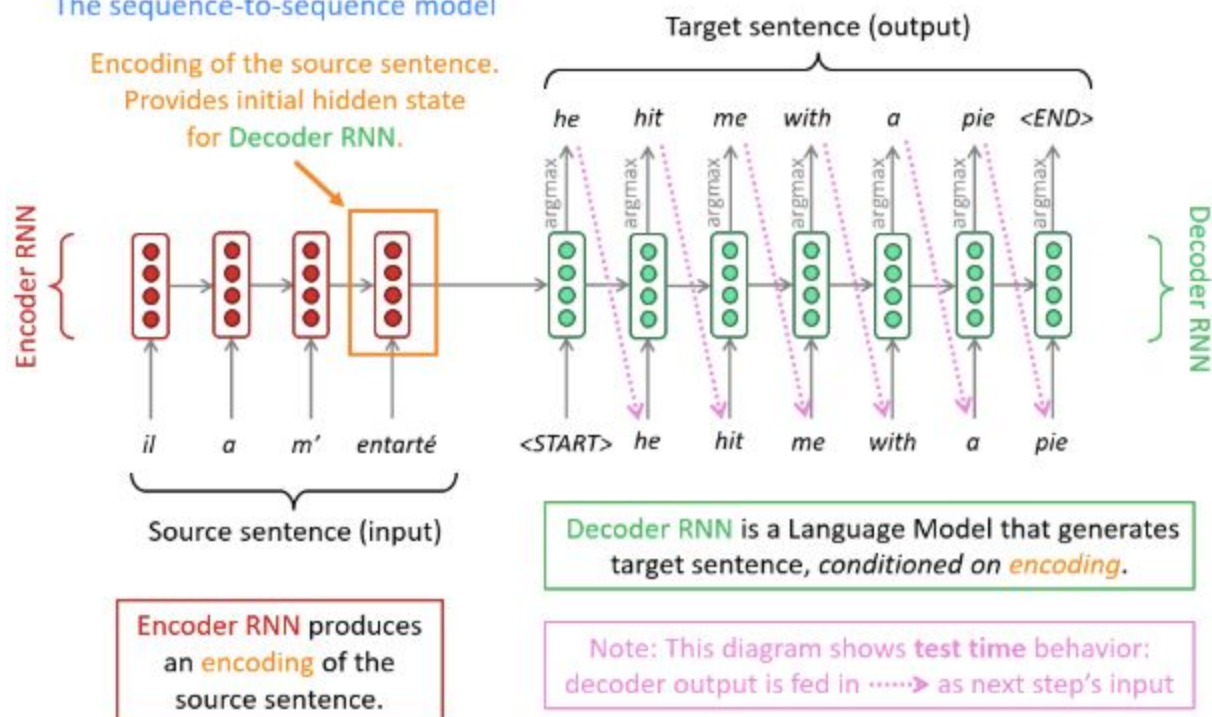
Seq2Seq 模型常用于机器翻译、自动摘要、对话系统等、language modeling, NER

6.1 Architecture

Seq2Seq在用作语言模型时, 它的NMT (Neural Machine Translation) 如下图所示, 左边红色部分是encoder RNN, 它负责对源语言进行编码, 学习源语言的隐含特征, 右边绿色部分是decoder RNN, 负责对目标语言进行解码。Encoder RNN 可以是任意一个RNN, 比如朴素RNN, LSTM或者GRU。Encoder RNN的最后一个神经元的隐状态作为Decoder RNN的初始隐状态, 因此Seq2Seq 模型是一个条件语言模型 (解码器在预测目标语言的下一个单词, 而这个预测是基于源语言的)。Decoder RNN在预测的时候, 需要把上一个神经元的输入作为下一个神经元的输出, 不断预测下一个词, 直到预测输出了结束标志符<END>。

Neural Machine Translation (NMT)

The sequence-to-sequence model



在机器翻译的任务中，我们常用的Seq2Seq模型如下图：

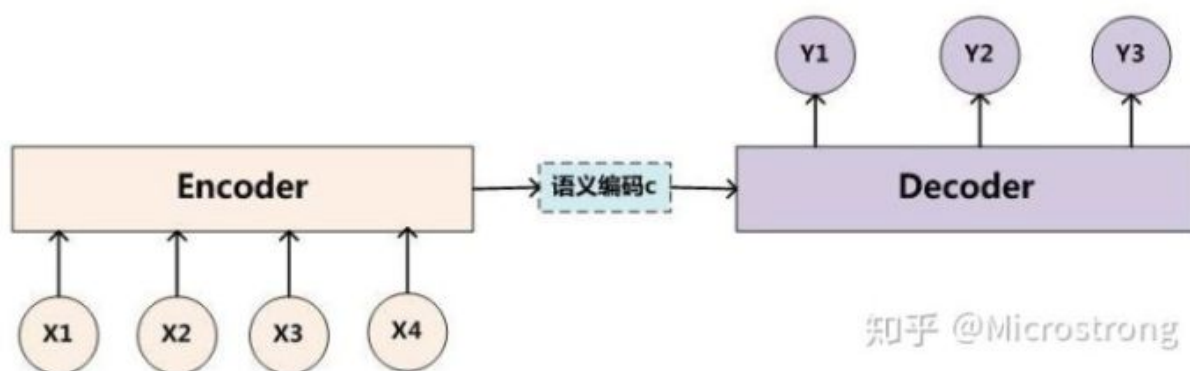


图3：NLP中的Encoder-Decoder框架

为方便阐述，其中的Encoder Decoder我用RNN。

在RNN中，当前时刻隐藏层的状态是由上一时刻隐藏层的状态和当前时刻的输入状态决定的，公式为： $h_t = f(h_{t-1}, x_t)$ 。在编码阶段，获得了各个时刻的隐藏层状态后，我们把这些隐藏层的状态

进行汇总，生成最后的语义编码向量C，公式： $C = q(h_1, h_2, h_3, \dots, h_{T_x})$ ，其中q表示某种非线性的神经网络，这里表示多层RNN。

在解码阶段，我们要根据给定的语义向量C和之前已经生成的输出序列 $y_1, y_2, y_3, \dots, y_{t-1}$ 来预测下一个输出的单词 y_t ，即公式：

$$y_t = \operatorname{argmax} P(y_t) = \prod_{t=1}^T p(y_t | y_1, y_2, \dots, y_{t-1}, C)$$

也可以写成： $y_t = g(y_1, y_2, \dots, y_{t-1}, C)$

在RNN中，以上公式可以表示为： $y_t = g(y_{t-1}, s_{t-1}, C)$

其中， s_{t-1} 表示Decoder中RNN神经元的隐藏层状态， y_{t-1} 表示前一时刻的输出，C表示语义向量

而g是一个非线性的神经网络，g一般是多层的RNN后接一个softmax层。

Encoder-Decoder的局限性在于编码器和解码器之间唯一的联系就是一个固定长度的语义向量C，也就是说，编码器需要将整个序列的信息压缩进一个固定长度的向量中。这样做有两个弊端：一是语义向量可能无法完全表示整个序列的信息，二是先输入到网络的内容携带的信息会被后输入的信息覆盖掉，输入信息越长，这个现象越严重。

6.2 Beam Search

在Seq2Seq预测的过程中，会用到Beam search的搜索策略，它是贪心策略和穷举策略的折中方案。贪心策略的预测过程在Decoder RNN的每一步都贪心的选择 \hat{y}_t 概率最大的那个词，但是贪心只能保证每一步是最优的，无法保证预测出来的句子是整体最优的。特别是如果在t时刻贪心选择的词语不是全局最优，会导致t时刻往后的所有预测词都是错误的，但是如果每步都穷举所有情况，时间复杂度太高。Beam Search在预测每一步时，都保留top-k高概率的词，作为下一个时间步的输入。k为beam size，k越大，得到更好结果的可能性越大，但是计算消耗也越大。Beam search不一定能找到全局最优。

Beam search结束时，需要从n条完全路径（抵达<END>标识符的路径）中选一个打分最高的路径作为最终结果。由于不同路径长度不同，而Beam Search打分是累加项，累加越多打分越低，所以要用长度对打分进行归一化

$$\frac{1}{t} \sum_{i=1}^t \log P_{LM}(y_i | y_1, \dots, y_{i-1}, x)$$

相对于统计语言模型(SMT), NMT的优势在于:性能好, 翻译流畅, 更好的利用上下文关系和短语相似性; 模型简单, 只需要一个神经网络, 端到端训练即可; 不需要很多人工干预和特征工程, 对于不同语言, 网络结构保持一样, 只需要变化词向量。而缺点在于: 难以调试和解释出错原因; 难以加入人工规则和控制输出结果。

6.3 Bleu score

Bilingual Evaluation Understudy

Bleu是机器翻译的评估指标, 衡量的是机器结果和人类的翻译结果(标注答案)的n-gram的重合, 重合越高, 打分越高。

Bleu的计算公式为:

$$BLEU = BP \times \exp\left(\sum_{n=1}^N w_n \log P_n\right)$$

$w_n = 1/N$, N 的上限取值为4, 即最多只统计4-gram的精度。

其中, 各阶N-gram的精度可以按照以下公式计算:

$$P_n = \frac{\sum_i \sum_k \min(h_k(c_i), \max_{j \in m} h_k(s_{ij}))}{\sum_i \sum_k h_k(c_i)}$$

$H_k(C_i)$ 表示 W_k 翻译选译文 C_i 中出现的次数, $H_k(S_{ij})$ 表示 W_k 在标准答案 S_{ij} 中出现的次数

其中, BP为长度惩罚因子(Brevity Penalty), 由于N-gram的匹配度可能会随着句子长度的变短而变好, 因此会存在这样的问题: 一个翻译引擎只翻译出了句子中部分句子且翻译的比较准确, 那么它的匹配度依然很高, 引入BP就可以避免这种评分的偏向性。

$$BP = \begin{cases} 1 & \text{if } l_c > l_s \\ e^{1 - \frac{l_s}{l_c}} & \text{if } l_c \leq l_s \end{cases}$$

BP的计算公式如上， l_c 代表机器翻译译文的长度， l_s 代表参考答案的有效长度，当存在多个参考译文时，选取和翻译译文最接近的长度。当翻译译文长度大于参考译文长度时，惩罚系数为1，意味着不惩罚，只有机器翻译译文长度小于参考答案才会计算惩罚因子。

虽然BLEU为MT提供了一套自动化的评价方法，但BLEU也不是完美的。因为它基于n-gram，假设机器翻译出来的是另一种优美的表达方法，但人给出的正确翻译结果是常规翻译，那么他们之间的n-gram的overlap可能不高，导致正确翻译的打分反而得分低。

7. Attention

注意力机制从本质上讲和人类的视觉选择注意力机制类似，核心目标也是从众多信息中选取对当前任务目标更关键的信息。目前大部分的注意力机制都附着在Encoder-Decoder框架下，不过Attention本身是一种思想，并不依赖于任何框架。

注意力机制除了用于机器翻译任务中，还被广泛用于很多其他NLP任务中，比如通过视觉注意生成图像标题：先使用CNN对图像进行编码，使用带有注意力记住的RNN来生成描述。

通过注意力机制，我们不再尝试将文本编码为固定长度的矢量，而是允许解码器在输出生成的每个步骤处理源语句的不同部分，让模型根据输入句子以及它到目前为止产生的内容来学习要注意的内容。

加入注意力机制的编码解码器可以解决传统的seq2seq LM的两个弊端（语义向量无法表示整个序列的信息以及长期记忆会被短期记忆覆盖）。其图解如下：

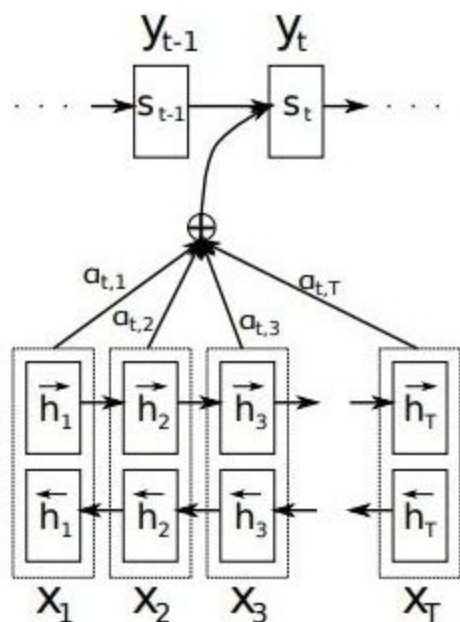


Figure 1: The graphical illustration of the proposed model trying to generate the t -th target word y_t given a source sentence (x_1, x_2, \dots, x_T) .

图4: Seq-to-Seq with Attention (NMT)

加入注意力机制的机器翻译模型中，编码器用了双向神经网络， $h_j = [h_j^{\rightarrow}, h_j^{\leftarrow}]$ ，就包含了第 j 个输入的前后信息。

Attention机制最重要的步骤是如何在每一时刻产生不同的语言编码向量 c_i ，表示接下来输出的时候要重点关注输入序列中的哪些部分，然后根据关注的区域来产生下一个输出。模型可以形象化的表示为图5所示。

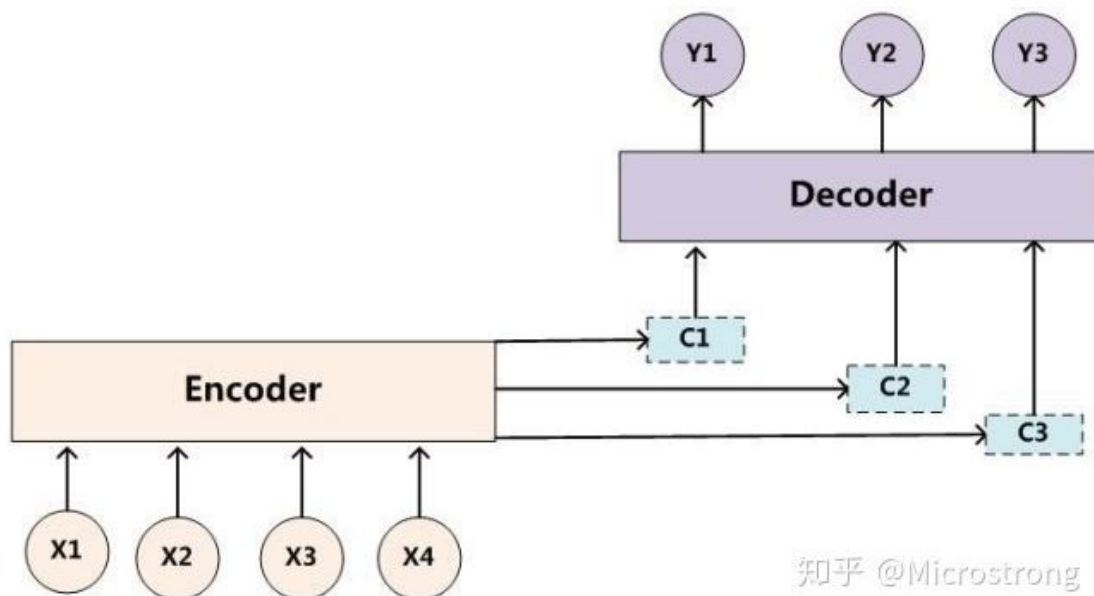


图5: Encoder-Decoder+Attention机制

相比于原始的Encoder-Decoder模型，加入Attention机制后最大的区别就是它不再要求编码器将所有输入信息都编码进一个固定长度的向量之中。而是，编码器需要将输入编码成一个向量的序列，在解码的时候，每一步都会选择性的从向量序列中挑选一个子集进行进一步处理。这样，在产生每一个输出的时候，都能够做到充分利用输入序列携带的信息。而且这种方法在翻译任务中取得了非常不错的成果。

7.1 Scaled dot-product attention

注意力机制简单来说就是seq2seq模型中的decoder在进行解码时，学习encoder输入中各词的权重分配。Dot-product attention将注意力表达成Q(query), K(key), V(value)三元组。其中Q是待生成的序列，V是输入的序列，K为权重。比如在机器翻译任务中，从源语言翻译成目标语言，V表示源语言序列，而Q就是待生成的目标语言序列，每产生Q中的一个翻译词时都需要计算与输入V中哪个词最优关系，这便是注意力权重K。

具体计算过程为：首先计算query和所有key之间的点乘，然后为了防止其结果过大，会除以一个尺度标度 $\sqrt{d_k}$ ，其中 d_k 为一个query和key向量的维度，再利用softmax将其结果归一化为概率分布

，让向量的每一维元素都是一个概率值，最后对Value vectors进行加权线性组合。在实际操作中，我们同时在一些列的queries中进行注意力权重的计算，将他们放在一个矩阵Q中。Keys and values也一样被放在两个矩阵K and V 中。最终输出的矩阵计算公式如下：

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

具体来说，在处理一个词时，我们需要计算句子中所有词与它的attention score，这就像将当前词作为搜索的query，去和句子中所有词（包含该词本身）的key去匹配，看看相关度有多高。

7.2 Self - attention

关于attention有很多应用，在非seq2seq任务中，比如文本分类，或者其他分类问题，会通过self attention来使用attention，它的好处在于：

- 1) 可以捕获同一个句子中单词之间的一些句法特征或语义特征
- 2) 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，需要依次序序列进行计算，对于远距离的相互依赖的特征，要经过若干时间步骤的信息累积才能将二者联系起来，而距离越远有效捕获的可能性越小，而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。
- 3) 增加计算的并行性

在self-attention中，其输入是独热编码，query和key，value都是相同的，即输入的句子序列信息（可以是词向量lookup后的序列信息，也可以先用cnn或者rnn进行一次序列编码后得到的处理过的序列信息）后面的步骤与上述的都是一样的：

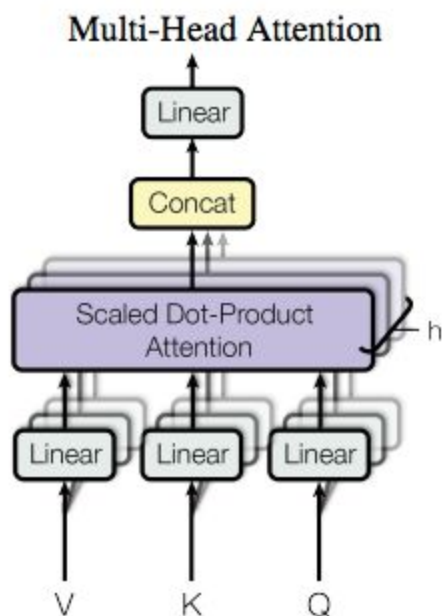
- 1) 首先建立句子序列中的每个词与句子其他词k的注意力权重 a_i
- 2) 然后将注意力权重向量进行softmax归一化，并与句子序列的所有时刻的信息（词向量或者rnn hidden state）进行线性加权。

7.3 Multi-head Attention

将很多个scaled dot-product attention单元堆叠，然后将结果级联到一起，再经过一个线性映射，就得到了multi-head attention。每个head学到的attention的侧重点可能略有不同，使得模型可以捕捉更多的特征。多头结构类似于CNN的多channel机制，是提高泛化性和健壮性的一个机制。

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$



以上是基本的multihead attention单元。

注意力机制的代价是需要计算输入和输出子的每个组合的注意力值，如果序列很长注意力机制会变得相当昂贵。

8. Transformer

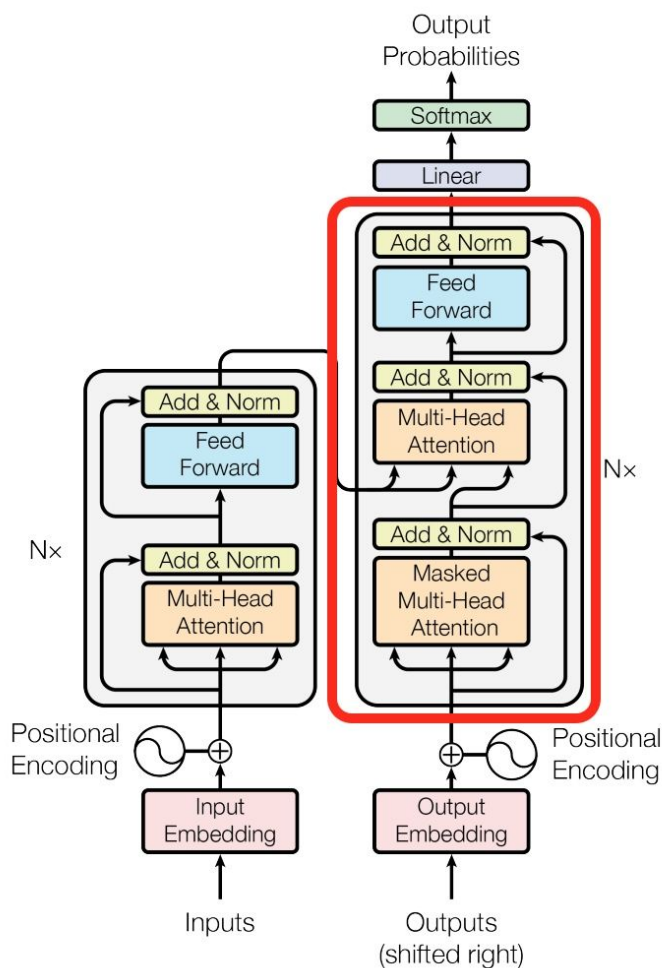
Why Transformer?

- 1) 使用attention机制后，每层的复杂度比RNN小，且RNN是个递归模型， t 时刻的计算需要依赖 $t-1$ 时刻的状态，所以很难进行并行计算，而attention支持并行，因此可以用到更多数据
- 2) Attention机制解决了RNN里长距离依赖的问题，即便LSTM添加了遗忘门等机制，还是很难解决这一问题。

在self-attention的过程中，每个token只能看到前文的信息，不能看到后文的信息，所以transformer使用了sequence mask，在self-attention时将后文的所有token给屏蔽了，因此，transformer是个单向模型。

8.1 Architecture

Transformer的结构如下：



8.2 Encoder and Decoder stacks

Encoder 由6个完全相同的layers堆叠而成，每一层有两个子层。第一个子层是一个多头的自注意力机制，第二个子层是position-wise fully connected feed-forward network。在这两个子层的后面各添加了一个residual connection和layer normalization。

Decoder同样由六个相同的layers堆叠而成。除了两个在encoder的子层之外，decoder在前两个子层后添加了一个多头的注意力机制。和encoder一样，在每个子层后都添加了Add&Norm层。另外，在第一层的自注意力机制中，加入了mask以确保预测的时候只能用到此刻单词之前的信息，而不能用下文的信息。Decoder的输出对应i位置的输出次的概率分布，输入是encoder的输出和对应i-1位置的decoder的输出。

注意：编码可以并行计算，一次性全部encoding出来，但解码是和RNN一样，需要按照序列一个个地解出来，因为要用到上一个位置的输入当做attention的query。

8.3 Position encoding

由于attention机制中每个词都会关注前后所有的词，导致所有词在位置信息上其实是等价的，所以transformer中引用position encoding表达每个词在原始句子中的位置信息，来弥补这一缺陷。它被叠加在word embedding上，主要选取了三角函数的encoding方式，在偶数位置使用正弦编码，在奇数位置使用余弦编码。由于sin函数的特性，一定程度上还可以有效地表达句子中词的位置的周期性。

$$PE_{(pos, 2i)} = \sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos, 2i+1)} = \cos(pos/10000^{2i/d_{model}})$$

8.4 Multi-head attention in Transformer

Transformer中的多头注意力机制有三种不同的使用：

- 1) Encoder比较简单中使用的是自注意力机制。在自注意力机制中，所有的key, query, value的来源相同，在transformer的encoder中，均来自前一层的encoder输出。每一个位置的encoder可以参与之前一层encoder的所有位置。
- 2) 相同的，Decoder中使用的自注意力机制层的k, q, v均来自前一层decoder的输出，但加入了mask操作，即我们只能参与前面已经翻译过的输出的词语，因为翻译过程我们当前还并不知道下一个输出的词语，这是我们之后才会推测到的。
- 3) Encoder-decoder attention layer的query来自前一级的decoder层的输出，但其k, v来自于encoder的输出，这使得decoder的每一个位置都可以参与输入序列的每一个位置。

总结就是：k, v的位置总是相同的，q在encoder和第一级decoder中与k, v来源相同，在encoder-decoder attention layer中与k, v来源不同。

8.5 Position-wise Feed-Forward Networks

除了添加注意力机制层之外，encoder和decoder的每一层最面都添加了一样的fully-connected feed-forward network，用来增强网络的非线性：

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

注意，虽然这个线性转换在不同位置是一样的，但每一层会使用不同的参数。

8.6 Add & Norm

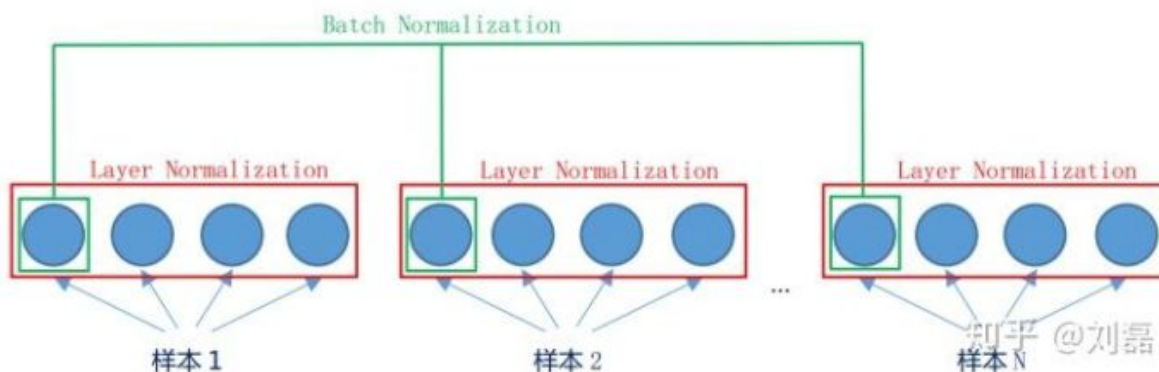
Add代表residual connection，是为了解决多层神经网络训练困难的问题，通过将前一层的信息无差的传递到下一层，可以有效地仅关注差异部分，类似于ResNet的思路。而Norm代表layer normalization，通过对层的激活值的归一化，加速模型的训练过程，收敛更快。

8.6.1 Batch Norm vs Layer Norm

Batch Norm是竖着来的，在各个维度做归一化，与batch size有关系，一般在CV中使用；

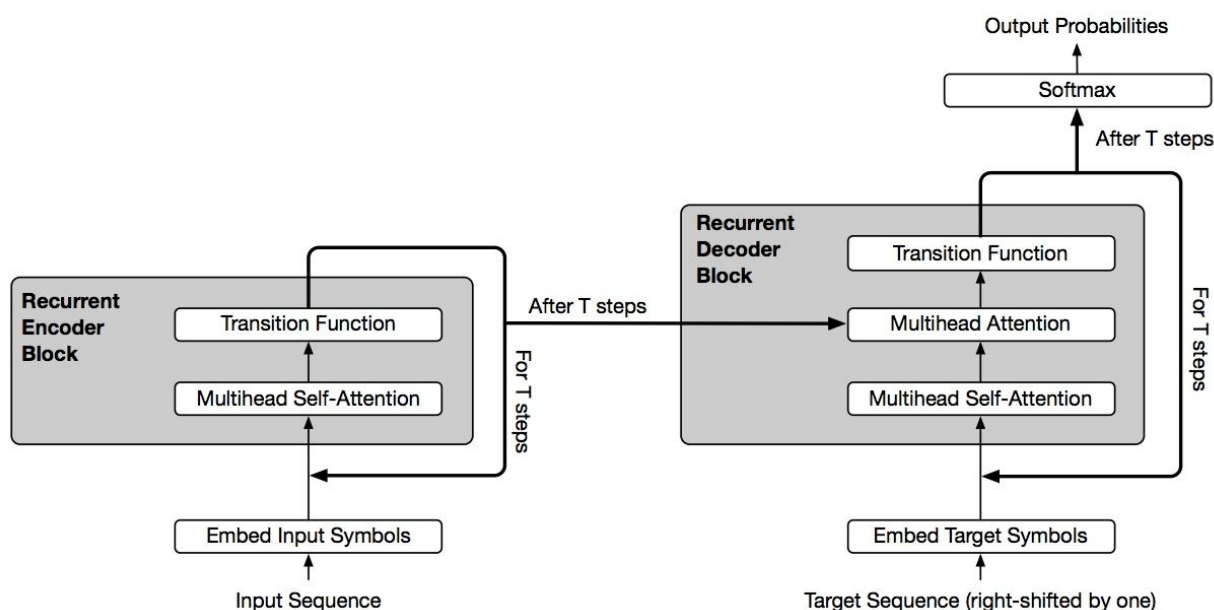
Layer Norm是横着来的，对一个样本，不同的神经元间做归一化，一般在NLP中使用。

下图显示了同一层的神经元的情况，假设这个mini-batch 有N个样本，则Batch Norm是对每一个维度进行归一，而Layer Norm对单个样本就可以处理。BN和LN都可以很好的抑制梯度消失和梯度爆炸的问题。



8.7 Universal Transformers

主要结合transformer的结构和RNN中循环归纳的优点，使得transformer能够适用于更多NLU的问题。其结构如下：



通过引入transition function，对attention可以进行多次循环。这一机制被用到很多应用上，如问答系统、根据主语推测谓语、根据上下文填充缺失单词、数字字符串运算处理、简易程序执行、机器翻译等场景。

9. Overfitting

9.1 L1 regularization

目标函数中增加所有权重 w 参数的绝对值之和，逼迫更多 w 为零(也就是变稀疏。L2因为其导数也趋0，奔向零的速度不如L1给力)。大家对稀疏规则化趋之若鹜的一个关键原因在于它能实现特征的自动选择。一般来说， x_i 的大部分元素(也就是特征)都是和最终的输出 y_i 没有关系或者不提供任何信息的，在最小化目标函数的时候考虑 x_i 这些额外的特征，虽然可以获得更小的训练误差，但在预测新的样本时，这些没用的特征权重反而会被考虑，从而干扰了对正确 y_i 的预测。稀疏规则化

的引入就是为了完成特征自动选择的光荣使命，它会学习地去掉这些无用的特征，也就是把这些特征对应的权重置为0。

9.2 L2 regularization

目标函数中增加所有权重 w 参数的平方之和，逼迫所有 w 尽可能趋向零但不为零。因为过拟合的时候，拟合函数需要顾及每一个点，最终形成的拟合函数波动很大，在某些很小的区间里，函数值的变化很剧烈，也就是某些 w 非常大。为此，L2正则化的加入就惩罚了权重变大的趋势。

9.3 Dropout

在训练的时候，让神经元以超参数 p 的概率被激活（也就是 $1-p$ 的概率被设置为0），每个 w 因此随机参与，使得任意 w 都不是不可或缺的，效果类似于数量巨大的模型集成。

类似于bagging中，我们定义 k 个不同的模型，从训练集有替换的采样构造 k 个不同的数据集，然后在训练集上训练模型，dropout是在指数级数量的神经网络上近似这个过程。Dropout与Bagging训练不太一样，在bagging的情况下，所有模型是独立的，而dropout下，模型是共享参数的，其中每个模型集成了神经网络参数的不同子集。

9.4 Batch Normalization

深度学习主要就是为了学习训练数据的分布，并在测试集上达到好的泛化效果，但是，如果每一个batch输入的数据都具有不同的分布，显然会给网络的训练带来困难。另一方面，数据经过一层网络计算后，上一层网络的输出数据经过这一层网络计算后，其数据分布会发生变化，即internal covariate shift，会给下一层的网络学习带来困难，batchnorm就是批规范化，就是为了解决这个分布变化问题。

而如果训练数据和测试数据存在分布的差异性，即covariate shift，也会给网络的泛化性和训练速度带来影响。Batchnorm可以做到数据归一化对数据做去相关性，突出分布的相对差异。

原理及源码

如果为了减少internal covariate shift，而对神经网络的每一层做归一化，假设每一层输出后的数据都归一化到0均值，1方差，但是，如果每一层的数据分布都是标准正态分布，导致其完全学习不到输入数据的特征，因为费劲学习到的特征分布都被归一化了，因此，直接对每一层做归一化

并不合理。于是我们想到加入可训练的参数做归一化，并在最后做缩放平移，也就是batchnorm实现的，具体过程如下：

之所以称之为batchnorm是因为所norm的数据是一个batch的，假设输入数据是 $\beta = x_{1...m}$ 共m个数据，输出是 $y_i = BN(x)$ ，batchnorm的步骤如下：

- 1.先求出此次批量数据 x 的均值， $\mu_\beta = \frac{1}{m} \sum_{i=1}^m x_i$
- 2.求出此次batch的方差， $\sigma_\beta^2 = \frac{1}{m} \sum_{i=1}^m m(x_i - \mu_\beta)^2$
- 3.接下来就是对 x 做归一化，得到 x_i^-
- 4.最重要的一步，引入缩放和平移变量 γ 和 β ，计算归一化后的值， $y_i = \gamma x_i^- + \beta$

在加入两个额外参数 γ 和 β 后，相当于对归一化后的数据做了缩放和平移，极端的情况：如果 γ 和 β 分别等于此batch的标准差和均值，那么输出值就还原到了归一化之前的值，也就是BN没有起作用。 γ 和 β 分别称为缩放参数和平移参数，这就保证了每一次数据经过归一化后还保留原有的学习来的特征，同时又完成归一化的操作，加速训练。一般BN层放在激活函数层前。

```

1] def batchnorm_train(x, gamma, beta, bn_param):
    """
    x: 输入数据, 设shape(B, L)
    gamma: 缩放因子
    beta: 平移因子
    bn_param: batchnorm所需要的一些参数, 包含以下:
    eps: 接近0 的数, 防止分母出现0
    momentum: 动量参数, 一般为0.9, 0.99, 0.999
    running_mean: 滑动平均的方式计算新的均值, 训练时计算, 为测试数据做准备
    running_var: 滑动平均的方式计算新的方差, 训练时计算, 为测试数据做准备
    """

    running_mean = bn_param['running_mean'] # shape = [B]
    running_std = bn_param['running_std'] # shape = [B]
    res = 0.

    x_mean = x.mean(axis= 0) #计算x的均值
    x_std= x.std(axis = 0) #计算x的标准差
    x_normalized = (x-x_mean)/(x_std+eps) #归一化
    res = gamma*x_normalized + beta #缩放平移

    running_mean = momentum*running_mean + (1-momentum)*x_mean
    running_std = momentum*running_std + (1-momentum)*x_std

    #记录新的值
    bn_param['running_mean'] = running_mean
    bn_param['running_std'] = running_std

    return res, bn_param

```

注意，在训练中，可以每次训练给一个批量，然后计算批量的均值方差，但是在测试的时候，每次输入一张图片，无法计算批量的均值方差，于是需要记录running_mean, running_std在测试集中使用。


```
def batchnorm_test(x, gamma, beta, bn_param):
    """
    参数和训练时的batchnorm一样
    """
    running_mean = bn_param['running_mean'] # shape = [B]
    running_std = bn_param['running_std'] # shape = [B]
    res = 0.

    x_normalized = (x-running_mean)/(running_std+eps) #归一化
    res = gamma*x_normalized + beta #缩放平移

    return res, bn_param
```

使用BN的好处

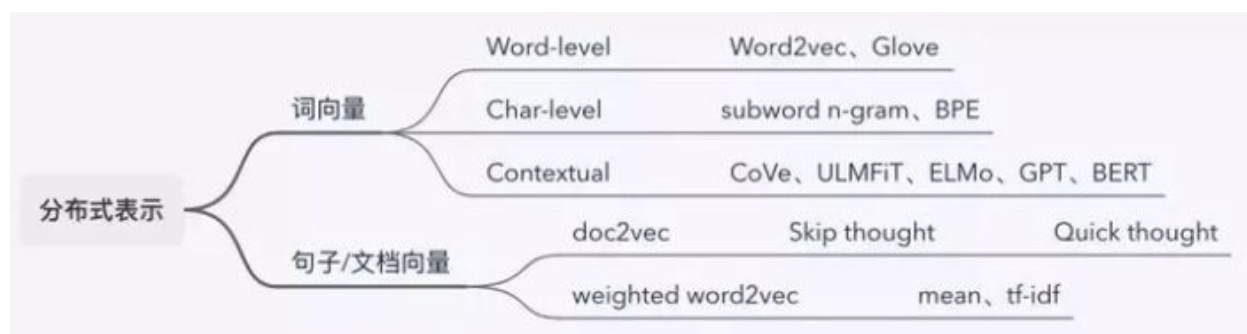
深度学习网络会用随机梯度下降法进行训练，虽然高效，但是问题在于需要人为去选择参数，比如学习率、参数初始化、dropout比例等，这些参数的选择对结果至关重要，而使用BN的好处就在于省去了很多调参的过程：

- 1) 可以选择较大的学习率，而不必担心错过local optimum
- 2) 可以移除dropout和L2正则项，或者选择更小的L2参数，因为BN可以有效提高网络泛化能力
- 3) 可以把训练数据彻底打乱，以防止每批训练的时候，每一个样本都经常被挑选到

9.5 Early stopping

理论上可能的局部极小值数量随参数的数量呈指数增长，到达某个精确的最小值是不良泛化的一个来源。实践表明，追求细粒度极小值具有较高的泛化误差。这是直观的，因为我们通常会希望我们的误差函数是平滑的，精确的最小值处所见相应误差曲面具有高度不规则性，而我们的泛化要求减少精确度去获得平滑最小值，所以很多训练方法都提出了提前终止策略。典型的方法是根据交叉验证提前终止：若每次训练前，将训练数据划分为若干份，取一份为测试集，其他为训练集，每次训练完立即拿此次选中的测试集自测。因为每份都有一次机会当测试集，所以此方法称之为交叉验证。交叉验证的错误率最小时可以认为泛化性能最好，这时候训练错误率虽然还在继续下降，但也得终止继续训练了。

10. Word Embedding



NNML, Word2Vec和Transformer是NLP界里程碑式的模型，在他们之前，针对不同的NLP任务，通常采用各不相同的模型架构去处理，比如，NER一般用HMM或者CRF，文本分类一般用BOW或LDA提取特征，再用LR或者SVM做分类。

传统的one-hot encoding 中，将每个单词转化成一个大小为 $|V|$ 的稀疏向量， V 为词库大小。向量中所有位置的都是0，除了单词本身的位置是1，而独热编码无法获取词与词之间的关联性（语义相似度），因为每个词向量的内积都为0。

不同于one-hot encoding，分布式词嵌入可以知道词之间的关系，这是基于分布式假设（出现在相同上下文下的词的意思应该相近）。可用相似度来度量词之间的语义相关性，所有学习word embedding的方法都是在用数学的方法建模词和上下文之间的关系。

词嵌入（word embedding）是一种词的类型表示，具有相似意义的词具有相似的表示，是将词汇映射到实数向量的方法总称。分布式实际上就是求一个映射函数，这个映射函数将每个词原始的one-hot表示压缩投射到一个较低维度的空间并一一对应。所以分布式可以表示确定的词。

模型的目的在于如何通过有限的上下文和样本词，让计算机自动学习从输入空间到嵌入空间的映射函数。词向量的分布式表示的核心思想由两部分组成：选择一种方式描述上下文；选择一种模型刻画目标词与其上下文之间的关系。

10.1 NNLM

NN language model

通过神经网络对语言模型进行建模，训练完后就得到了语言模型，也顺便得到了词向量。也就是说，用这种方法，词向量是一个有用的副产品（神经网络的权重）。

RNN的隐藏层重复利用，参数共享，故而可以减少需要学习的参数。RNN的优势在于能充分利用所有上下文信息来预测下一次词。但弱点是，由于参数共享且连乘，很容易导致梯度爆炸或者梯度消失，难以捕捉长距离信息，难以优化。

NNLM其核心思想就是基于上下文，先用向量代表各个词，然后通过一个预测目标函数学习这些向量的参数。RNNLM的网络主体是一种单隐层前馈神经网络，网络的输入和输出均为词向量。

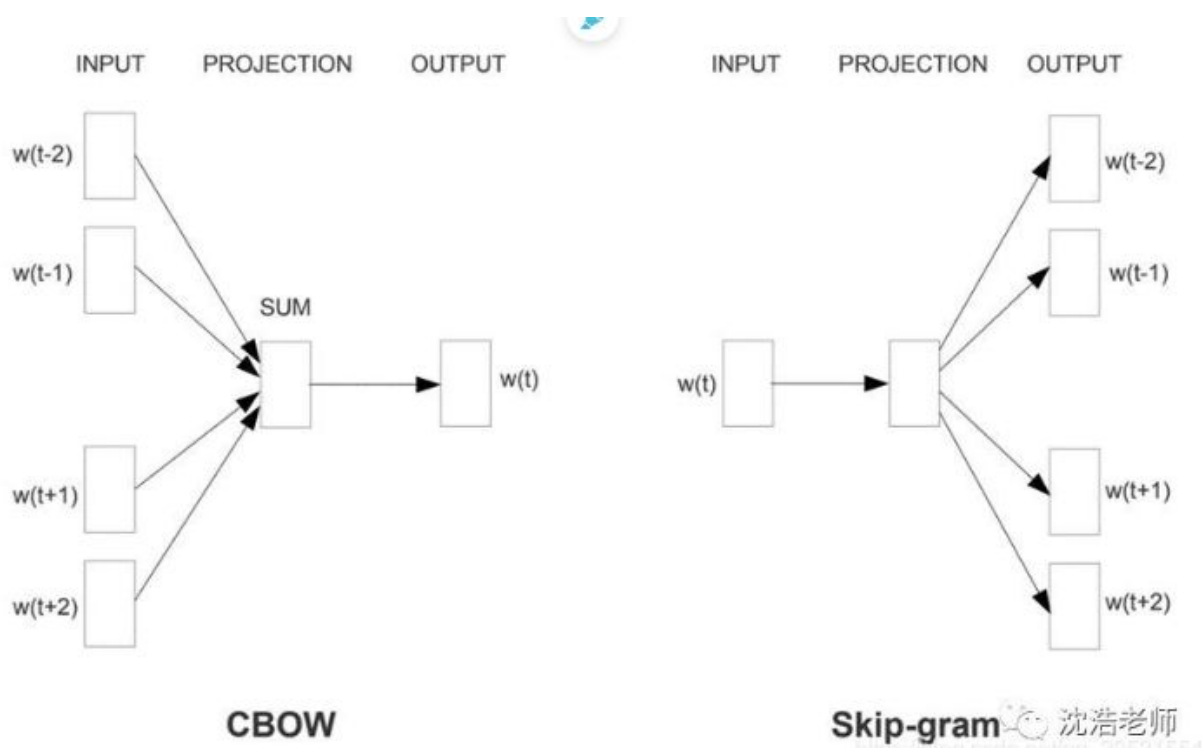
10.2 Word2Vec

Word2Vec是个三层的浅层神经网络模型，学习到的是静态词向量（语言信息），再通过RNN, CNN 或是Attention等深度学习模型去训练下游任务。主要有CBOW和skipgram两种模型。

Word2Vec属于自监督学习(Self-supervised learning)，主要希望能够学习到一种通用的特征表达用于下游任务。其主要方式是通过自己监督自己，比如把一段话里的几个单词去掉，用它的上下文预测缺失的单词，或者将图片的一些部分去掉，依赖周围的信息去预测确实的patch。

10.2.1 CBOW

CBOW是通过上下文预测中心词。通过在一个大的语料库训练，得到一个从输入层到隐含层的权重模型



10.2.2 SkipGram

Skip-gram和CBOW相反，是通过中心词预测上下文，2-gram比较常用。

基于神经网络的语言模型的目标函数通常取为如下的对数似然函数，也就是cross-entropy loss：

$$\mathcal{L} = \sum_{w \in \mathcal{C}} \log p(\text{Context}(w) | w),$$

通过梯度下降法，求得参数更新方式。

CBOW vs SkipGram

Skipgram：数据较少、有较多非常见词的时候使用

CBOW：擅长预测常出现的单词

大部分情况倾向于使用skipgram

在分类数很多的情况下，从隐藏层到输出softmax层的计算量非常大，因为要计算所有词的softmax概率，再去找概率最大的值，速度很慢，word2vec加入了以下训练技巧（模型优化）：从对输入层到隐藏层的映射，没有采取神经网络的线性变换和激活函数的方法，而是采用简单的对所有输入词向量求和并取平均的方法以及hierarchical softmax和负采样。

10.2.3 Hierarchical softmax

从隐藏层到输出的softmax层的计算量做了改进。为了避免计算所有词的softmax概率，word2vec采用了霍夫曼树来代替从隐层到输出层的映射。由于是二叉树，计算量从 V 变成了 $\log_2 V$ ；另外，使用霍夫曼树也可以提高测试的时间，由于使用霍夫曼树的每一个节点都对应着从这个节点到父节点的概率，也就是说节点的概率一定小于它的父节点，用DFS的方式去寻找最大概率的叶节点可以省掉遍历到小概率节点的时间，即高频的词靠近树根，这样高频词需要更少的时间被找到，符合贪心优化思想。

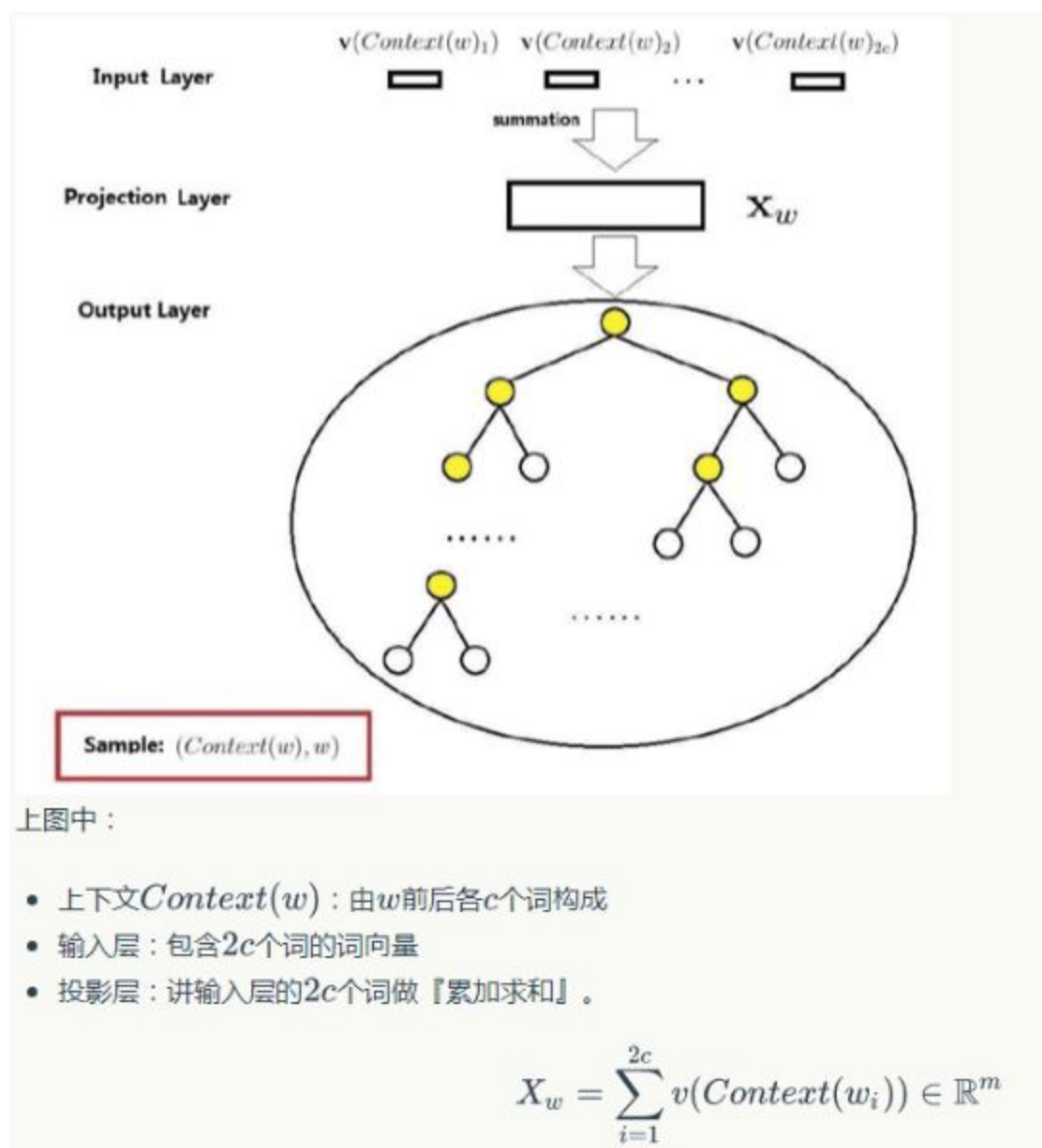
§2.4.2 Huffman 树的构造

给定 n 个权值 $\{w_1, w_2, \dots, w_n\}$ 作为二叉树的 n 个叶子结点，可通过以下算法来构造一颗 Huffman 树。

算法 2.1 (*Huffman* 树构造算法)

- (1) 将 $\{w_1, w_2, \dots, w_n\}$ 看成是有 n 棵树的森林（每棵树仅有一个结点）。
- (2) 在森林中选出两个根结点的权值最小的树合并，作为一棵新树的左、右子树，且新树的根结点权值为其左、右子树根结点权值之和。
- (3) 从森林中删除选取的两棵树，并将新树加入森林。
- (4) 重复 (2)、(3) 步，直到森林中只剩一棵树为止，该树即为所求的 *Huffman* 树。

模型的内部结构：



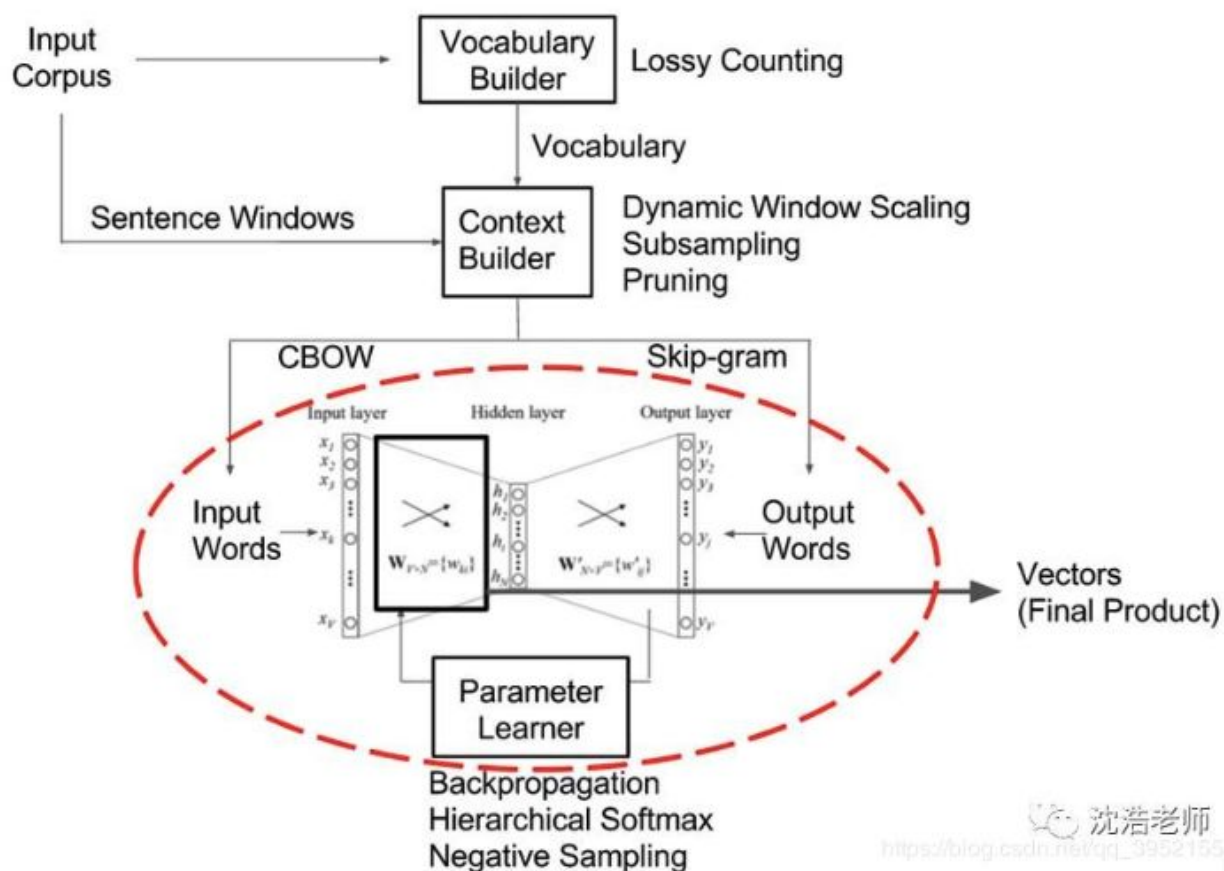
上图中：

- 上下文 $\text{Context}(w)$ ：由 w 前后各 c 个词构成
- 输入层：包含 $2c$ 个词的词向量
- 投影层：将输入层的 $2c$ 个词做『累加求和』。

和之前的RNNLM相比，霍夫曼树的所有内部节点就类似于之前RNN隐藏层的神经元，其中，根节点的词向量对应投影后的词向量，而所有的叶子节点就类似于之前RNN softmax输出层的神经元，叶子节点的个数就是词汇表的大小。在霍夫曼树中，隐藏层到输出层的softmax映射不是一下子完成的，而是沿着霍夫曼树，采用二元逻辑回归，沿着左子树走就编码为1（负类），右子树为正类（0），判断正负类的方式是采用sigmoid函数。

基于霍夫曼树的word2vec的目标在于，找到合适的所有节点的词向量和所有内部节点 θ ，使得训练样本达到最大似然，使用梯度上升法进行计算。

模型主要训练的是如下红色圈内的部分。



10.2.4 Negative sampling

利用Hierarchical softmax的缺点在于，如果训练样本里的中心词是个很生僻的词，那么就得在霍夫曼树中辛苦的往下走很久了，负采样可以不用建立这样一颗复杂的霍夫曼树，将模型变得更简单。

使用negative sampling时，每次从除当前label外的其他label中选择几个作为负样本，作为出现负样本的概率加到损失函数中，用公式可表达为：

$$loss = -\frac{1}{M} \sum_{i=1}^m \left(\log \sigma(u_o^T h_i) + \sum_{j \sim P(w)} [\log \sigma(-u_j^T h_i)] \right)$$

和霍夫曼树一样，负采样也是采用二元逻辑回归来求解模型参数。其通过似然函数算出loss function并通过随机梯度下降法进行迭代。

可以将以上两种方法当做近似算法。

为何使用负采样？

- 1) 加速了模型的计算，在计算目标函数的时候只需要更新采样的词的权重，而不是更新所有的权重
- 2) 保证了训练效果，因为中心词只和它周围的词有关系，位置离的很远的词没有关系，没必要同时训练更新

那么负采样是如何进行的呢？

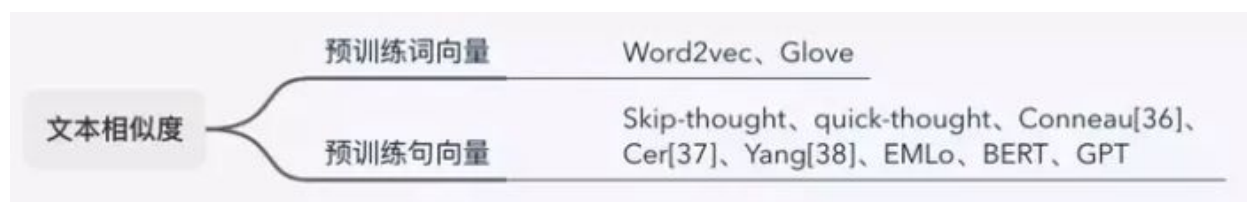
在Negative Sampling中，按照每个词的词频进行随机负采样，词频越大的词，被采样的概率越大。每个词被采样的概率并不是简单的按照词频在总量的占比，而是对词频先取根号，再算占比。这里取根号的目的是降低高频词的采用概率，同时增加低频词的采样概率。

Word2Vec虽然取得了很好的效果，但模型上仍然存在明显的缺陷，比如没有考虑词序，再比如没有考虑全局的统计信息。

Doc2Vec与Word2Vec的CBOW模型类似，也是基于上下文训练词向量，不同的是，Word2Vec只是简单地将一个单词转换为一个向量，而Doc2Vec不仅可以做到这一点，还可以将一个句子或是一个段落中的所有单词汇成一个向量，为了做到这一点，它只是将一个句子标签视为一个特殊的词，并且在这个特殊的词上做了一些处理，因此，这个特殊的词是一个句子的标签。如图所示，词向量作为矩阵W中的列被捕获，而段落向量作为矩阵D中的列被捕获。

如何衡量文本之间的相似度？

深度学习用来处理文本语义相似性任务，主要分为两类：1) 预训练好的词向量word2vec或glove来表示句子，计算向量之间的距离来区分相似性；2) 用预训练模型来表示句子，比如用transformer来获取句子向量，然后用arccos来计算相似距离。



1) Cosine similarity

同一类文章一定是某些主题词用的多，另外一些词用得少。比如金融类的文章，这些词频率就高：股票、利息、债券等；而这些就少：二氧化碳、宇宙、诗歌等。反应在每一篇文章的特征上，如果两篇文章属于同一类，他们的特征特征向量的夹角就可以用来表示对应的文章的接近程度。而两个向量夹角的计算，就要用到余弦定理。

余弦的取值在0-1之间，也就是说夹角在0-90度之间。当两篇文章夹角的余弦等于1时，两个向量的夹角为0，两篇文章属于通同一类，余弦为0时，说明两篇文章完全没有相同的主题词，毫不相关。

对于三角形ABC，如果将两边b和c看作是两个以A为起点的向量，那么：

$$\cos(A) = \frac{\langle b, c \rangle}{|b| \cdot |c|}$$

其中，分母表示向量b和c的长度，分子表示两个向量的内积。假如文章X和文章Y对应的向量分别是 $x_1, x_2, \dots, x_{64000}$ 和 $y_1, y_2, \dots, y_{64000}$ ，那么它们夹角的余弦等于：

$$\cos\theta = \frac{x_1y_1 + x_2y_2 + \dots + x_{64000}y_{64000}}{\sqrt{x_1^2 + x_2^2 + \dots + x_{64000}^2} \cdot \sqrt{y_1^2 + y_2^2 + \dots + y_{64000}^2}}$$

注意，出现在不同位置的词在文本分类的时候的重要性不同。比如，在标题中出现的词对主题贡献远比在正文中的重要。即使在正文中，出现在文章开头和结尾的词也比出现在中间的重要，所以，在NLP的时候，对标题和重要位置的词要进行额外加权，以提高文本分类的准确性。

余弦函数的优化

计算两个向量夹角时，计算量为 $O(|a|+|b|)$ ，假如其中一个向量更长，比如 $|a|>|b|$ ，那么复杂度为 $O(|a|)$ 。如果要比较一篇文章和其他N篇文章的相关性，那么复杂度为 $O(N*|a|)$ ，如果要比较N篇文章之间的两两相关性，复杂度就是 $O(N^2|a|)$ ，这还只是一次迭代，计算量相当大。我们可以从以下几个方面进行优化以降低复杂度：

(1) 首先分母（向量的长度）不需要重复计算，可以在第一次计算后将他们存起来，之后调用即可。

(2) 计算两个向量内积的时候，只需要考虑向量中的非零元素即可。那么计算的复杂度取决于两个向量中，非零元素个数的最小值。

(3) 删除虚词。比如“的”，“和”，“是”以及一些副词、介词。删除虚词后，计算时间可以大大缩短，并且分类的准确信也会增加。因为虚词的权重其实是一种干扰分类正常进行的噪音。

2) SVD

Singular Value Decomposition

由于用余弦定理时要对所有文章做两两计算，并且多次迭代，耗时很长，尤其是文章量大，且词表也很大的时候。这是，我们可以用SVD一次性将所有的文章相关性都算出来。

首先，我们用一个大矩阵A来描述成千上万篇文章和几十上百万词的相关性。在这个矩阵中，每一行对应一篇文章，每一列对应一个词，如果有N个词，M篇文章，那么久是一个如下所示的M*N

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1j} & \cdots & a_{1N} \\ \cdots & & & & \cdots \\ a_{i1} & \cdots & a_{ij} & \cdots & a_{iN} \\ \cdots & & & & \cdots \\ a_{M1} & \cdots & a_{Mj} & \cdots & a_{MN} \end{bmatrix}$$

的矩阵。

其中，第*i*行，第*j*列的元素是字典中第*j*个词在第*i*篇文章中所出现的加权词频（比如TF-IDF值）。SVD就是把这样一个大矩阵，分解成三个小矩阵相乘。

$$\begin{array}{c}
 \boxed{A} \\
 1000\,000 \times 500\,000
 \end{array}
 =
 \begin{array}{c}
 \boxed{X} \\
 1000\,000 \times 100
 \end{array}
 \begin{array}{c}
 \boxed{B} \\
 100 \times 100
 \end{array}
 \begin{array}{c}
 \boxed{Y} \\
 100 \times 500\,000
 \end{array}$$

分解完三个矩阵的元素加起来也不过1.5亿，不到原来的三千分之一，相应的存储量和计算量都会小三个数量级以上。

三个矩阵都有非常清晰的物理意义。X是对词进行分类的一个结果。它的每一行表示一个词，每一列表示一个语义相近的词类，或者称为语义类。这一行的每个非零元素表示这个词在每个语义类中的重要性，数值越大越相关。

Y是对文本分类的结果，它的每一列对应一个文本，每一行对应一个主题。这一列中每个元素表示这篇文本在不同主题中的相关性。

B表示词的类和文章的类之间的相关性。

因此，只要对关联矩阵A进行奇异值分解，就可以同时完成近义词分类和文章的分类，同时还可以得到每个主题和每个词的语义类之间的相关性。

SVD一般分两步进行：

- （1）将A变换成一个双对角矩阵（除了两行双对角元素非零，其他都是零），这个过程的计算量为 $O(MN^2)$ ，假设 $M > N$ 。
- （2）将双对角矩阵变成奇异值分解的三个矩阵。计算量是第一步的零头。

Cosine similarity VS SVD

由于SVD不需要多次迭代，所以比余弦定理计算文本的相似度时间短很多。

但SVD存储量较大，因为需要将整个矩阵都存在内存里，而用余弦定理的聚类则不需要。

因此SVD适合处理超大规模的粗分类。在实际工作中，可以先进行奇异值分解，得到粗分类结果，再利用余弦定理，在粗分类的基础上，进行几次迭代，得到比较精确的结果。

10.3 Fasttext

结构比Word2Vec简单，只有一层隐层以及输出层，所以速度很快。其主要功能有二，训练词向量和文本分类。Fasttext训练出来的词向量对比word2vec的词向量，在很多任务上都有更好的效果，比如语言模型(论文中使用由650个LSTM units组成的RNN)的perplexity降低不少

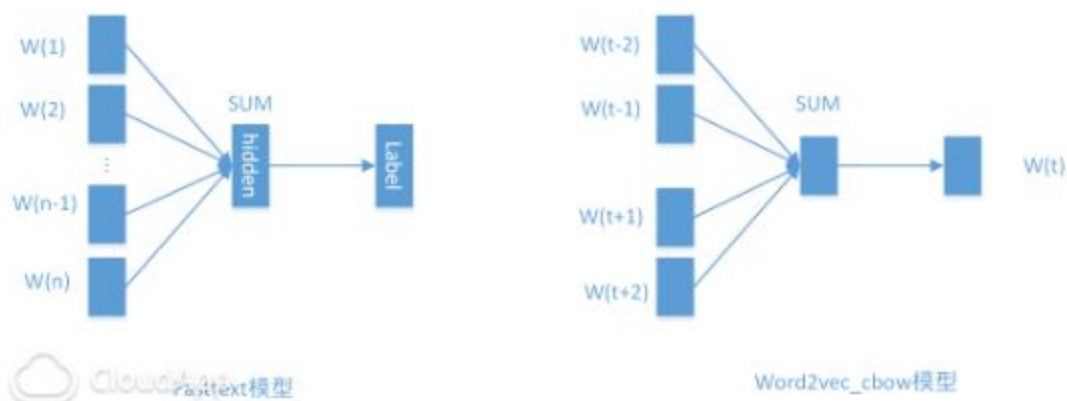
文本分类最简单有效的方式是通过BOW+线性分类器，比如LR或SVM，但是线性分类器不能在特征和类别上共享参数，从而限制了在数据量很小的类别的泛化能力，通常解决方法有将稀疏矩阵分解为低秩矩阵，或采用多层神经网络。

Fasttext相对于Word2Vec来说，增加了subwords特性，从而解决OOV (out of vocabulary)的问题（一般此类V用<UNK>替代）。subwords其实就是一个词的character-level的n-gram。比如单词"hello"，长度至少为3的char-level的ngram有"hel","ell","llo","hell","ello"以及本身"hello"。每个ngram都可以用一个dense的向量 z_g 表示，于是整个单词"hello"就可以表示为：

$$V_{hello} = \sum_{g \in \phi} z_g^T v_c$$

这样丰富了词表示的层次。比方说"english-born"和"china-born"，从单词层面上看，是两个不同的单词，但是如果用char-level的ngram来表示，都有相同的后缀"born"。因此这种表示方法可以学习到当两个词有相同的词缀时，其语义也具有一定的相似性。这种方法对英语等西语来说可能是奏效的，因为英语中很多相同前缀和后缀的单词，语义上确实有所相近。但对于中文来说，这种方法可能会有些问题。比如说，"原来"和"原则"，虽有相同前缀，但意义相去甚远。可能对中文来说，按照偏旁部首等字形的方式拆解可能会更有意义一些。

Subword的属性不仅使fasttext可以解决OOV的问题，对比word2vec，它可以在更小的语料库的训练下达到相同的效果（论文中讨论的是查找同义词的任务）。



Fasttext的网络结构中， $W(1)$ 到 $W(n)$ 表示document中每个词的word embedding表示。文章则可以用所有词的embedding累加后的均值表示，最后从隐层再经过一次的非线性变换得到输出层的label。Fasttext可以用CBOW或是skip-gram进行训练。

CBOW vs Fasttext

- 1) 目的不同，Fasttext是用来做文本分类的，虽然中间会产生词向量，但它是个副产物，而CBOW是专门用来训练词向量的工具
- 2) Fasttext的输出层是预测句子的类别标签，而CBOW的输出层是预测中间词
- 3) CBOW是训练词向量的算法，是无监督模型，而fastText是文本分类算法，是有监督模型
- 4) Fasttext的输入层是一个句子的每个词以及句子的ngram特征，而CBOW的输入层是预测中间词的上下文，与完整句子没有关系

和word2vec类似，fasttext本质上也可以看成是一个浅层的神经网络，因此其forward propagation过程可描述如下：

$$h = \frac{1}{n} \sum_{i=1}^n w_i$$

$$z = \text{sigmoid}(W_o h)$$

其中 z 是最后输出层的输入向量， W_o 表示从隐层到输出层的权重。

如果模型的最后我们要给文本分类，那么很自然的选择就是softmax层了，于是损失函数可以定义为：

$$\hat{y} = \text{softmax}(z)$$

$$CE(y, \hat{y}) = - \sum_j y_j \log(\hat{y}_j)$$

$$\text{loss} = \frac{1}{M} \sum_{i=1}^m CE(y_i, \hat{y}_i)$$

为了加快softmax层的计算速度，fasttext也和word2vec一样采用了霍夫曼树和负采样的方法。

10.3.1 N-gram in Fasttext

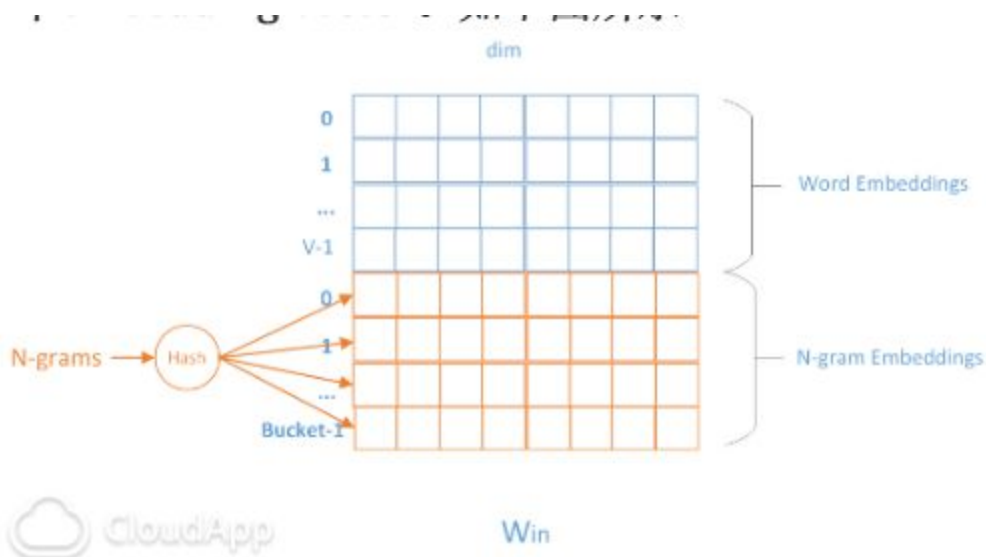
但是通过隐层简单求和取平均值会丢失词顺序的信息。将N-gram加入fasttext就可以避免这个问题。具体做法是把N-gram当成一个词，也用embedding向量来表示，在计算隐层时，把N-gram的embedding向量也加进去求和取平均。举个例子来说，假设某篇文章只有3个词， W_1 ， W_2 ， W_3 ，N-gram的 N 取2， w_1 、 w_2 、 w_3 以及 w_{12} 、 w_{23} 分别表示词 W_1 、 W_2 、 W_3 和bigram W_1 - W_2 ， W_2 - W_3 的embedding向量，那么文章的隐层可表示为：

$$h = \frac{1}{5}(w_1 + w_2 + w_3 + w_{12} + w_{23})$$

通过back-propagation算法，就可以同时学到词的Embedding和n-gram的Embedding了。

10.3.2 Hashbucket

具体实现上，由于n-gram的量远比word大的多，完全存下所有的n-gram也不现实。Fasttext采用了Hash桶的方式，把所有的n-gram都哈希到buckets个桶中，哈希到同一个桶的所有n-gram共享一个embedding vector。如下图所示：



图中Win是Embedding矩阵，每行代表一个word或N-gram的embeddings向量，其中前V行是word embeddings，后Buckets行是n-grams embeddings。每个n-gram经哈希函数哈希到0-bucket-1的位置，得到对应的embedding向量。用哈希的方式既能保证查找时O(1)的效率，又可能把内存消耗控制在O(bucket×dim)范围内。不过这种方法潜在的问题是存在哈希冲突，不同的n-gram可能会共享同一个embedding。如果桶大小取的足够大，这种影响会很小。

10.4 Matrix Factorization

利用语料库构建一个 $W \times C$ 共现矩阵 F ，矩阵每一行代表一个词，每一列是某种上下文表示方法。 W 是词表大小。矩阵每个单元的值可以是二值的表示二者是否共现，可以是未经处理的共现次数，也可以是经过处理后的共现tf-idf值，等等。很多可衡量两个对象之间关联的指标都可以用来作为矩阵中每个单元的值。

由于矩阵每一行的维度大小都等于词表大小，不便计算，所以需要进行降维，降维后的矩阵 f 大小为 $W \times d$ ($d \ll C$)。重点在于如何找出降维函数 $g: F \rightarrow f$

LSA / LDA，这两种方法使用的就是矩阵分解法，矩阵的列都是整篇文档。使用了全局特征。

10.4.1 LDA

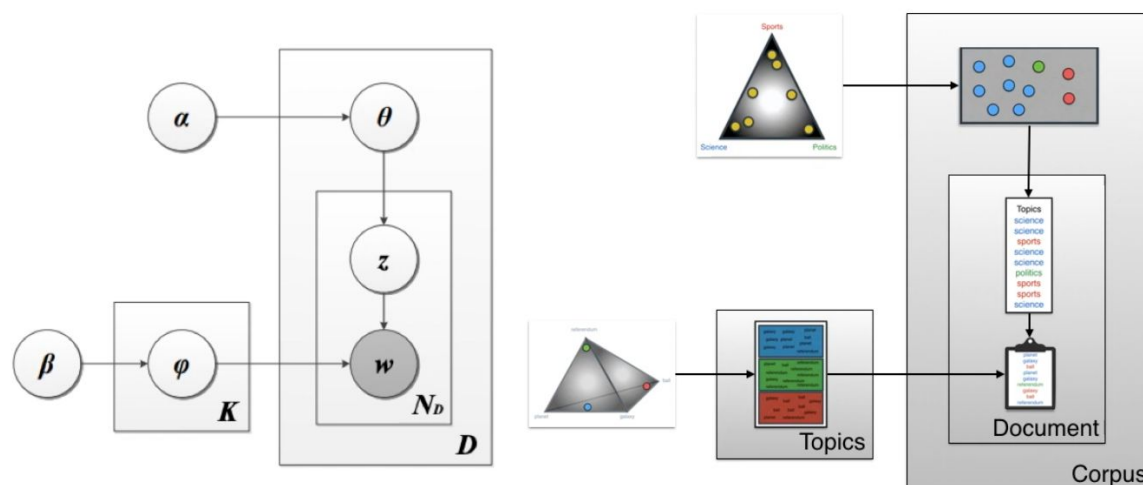
LDA (Latent Dirichlet Allocation) 是一种文档主题生成模型，也称为一个三层贝叶斯概率模型，包含词、主题和文档三层结构。它也是一种典型的词袋模型，即一篇文档由一组词构成，词与词之间没有先后顺序关系。

所谓生成模型，就是说，我们认为一篇文章的每个词都是通过“以一定概率选择了某个主题，并从这个主题中以一定概率选择某个词语”这样一个过程得到。

文档到主题服从多项式分布，主题到词服从多项式分布。LDA是一种非监督机器学习技术，可以用来识别大规模文档集 (document collection) 或语料库 (corpus) 中潜藏的主题信息。它采用了词袋 (bag of words) 的方法，这种方法将每一篇文档视为一个词频向量，从而将文本信息转化为了易于建模的数字信息。但是词袋方法没有考虑词与词之间的顺序，这简化了问题的复杂性，同时也为模型的改进提供了契机。每一篇文档代表了一些主题所构成的一个概率分布，而每一个主题又代表了很多单词所构成的一个概率分布。

利用Gibbs Sampling来训练LDA，LDA最终会输出每篇文档中出现一些重要单词的概率，其中重要单词可以通过TF-IDF等算法求出，或者直接删除stopwords。最后，通过每篇文档中最常出现的单词及其概率，人为参与辨别/总结每篇文档的主题。

Latent Dirichlet Allocation



10.4.2 LSA

Latent Semantic Analysis。可以基于共现矩阵构建词向量，实质上是基于全局预料采用SVD进行矩阵分解，然而SVD计算复杂度高。

10.5 GloVe

Global Vectors for Word Representation

GloVe 结合了LSA矩阵分解技术的全局统计与word2Vec的基于局部语境学习，可以看做是对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。

10.5.1 Word2Vec vs GloVe

两个模型都可以根据词汇的“共现co-occurrence”信息，将词汇编码成一个向量（所谓共现，即语料中词汇一块出现的频率），去学习词之间的关系。两者最直观的区别在于，word2vec是“predictive”的模型，而GloVe是“count-based”的模型。

Predictive的模型的目的在于最小化损失函数以提高预测能力。如Word2vec，根据context预测中间的词汇，要么根据中间的词汇预测context，分别对应了word2vec的两种训练方式cbow和skip-gram。对于word2vec，采用三层神经网络就能训练，最后一层的输出要用一个Huffman树进行词的预测。

Count-based模型，如GloVe，本质上是对共现矩阵进行降维。首先，构建一个词汇的共现矩阵，每一行是一个word，每一列是context。共现矩阵就是计算每个word在每个context出现的频率。由于context是多种词汇的组合，其维度非常大，我们希望像network embedding一样，在context的维度上降维，学习word的低维表示。这一过程可以视为共现矩阵的重构问题，即reconstruction loss。

两个模型在并行化上有一些不同，即GloVe更容易并行化，所以对于较大的训练数据，GloVe更快。

1) word2vec是局部语料库训练的，其特征提取是基于滑窗的；而glove的滑窗是为了构建共现矩阵，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。

2) Word2vec是无监督学习，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。

3) 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。

4) 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。

10.6 BERT

Bidirectional encoder representations from transformers

以上都是静态词向量，其学习出来的一个单词只能对应一个词向量，无法解决一词多义的问题。以下介绍的都是动态词向量，是基于上下文的，同一个词在一个句子中可能意思不同，甚至词性不同，它的词向量也就不相同。

BERT可以用于问答系统、情感分析、垃圾邮件过滤、命名实体识别、文档聚类任务中，作为这些任务的语言模型。目前已经在NLP领域成为了预处理的标配，像VGG, Resnet, Inception在CV领域的角色一样。

不同于ELMo和GPT，BERT的核心思想在于使用了双向transformer用于语言模型，[transformer](#)是一种注意力机制，可以学习到文本中单词之间的上下文关系。Transformer的原形包括两个独立的机制，一个encoder负责接收文本作为输入，一个decoder负责预测任务的结果。而BERT的目标是生成预训练语言模型，所以只需要encoder机制。

Transformer 的 encoder 是一次性读取整个文本序列，而不是从左到右或从右到左地按顺序读取，这个特征使得模型能够基于单词的两侧学习，相当于是一个双向的功能。

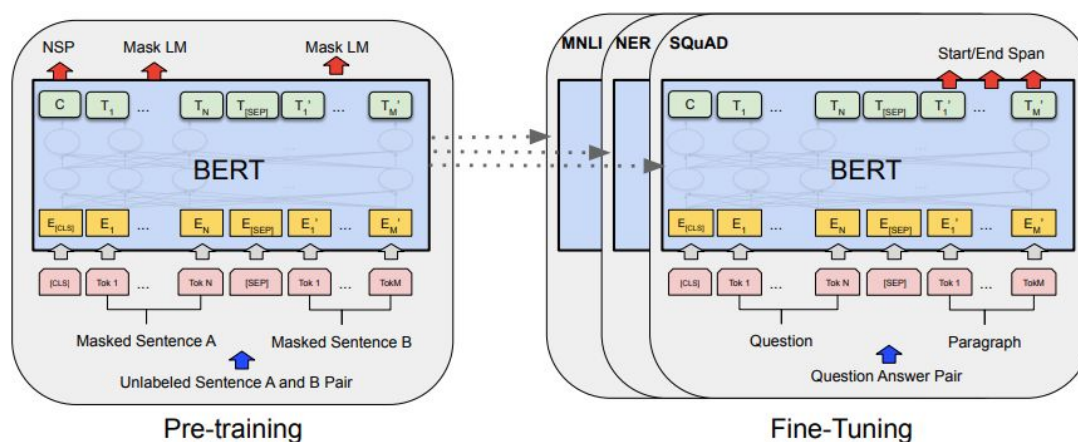


Figure 1: Overall pre-training and fine-tuning procedures for BERT. Apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, and [SEP] is a special separator token (e.g. separating questions/answers).

BERT的预训练及其对应的损失函数

和以往采用从左至右或是从右至左的语言模型进行预训练的方法不同，BERT采用了两种非监督学习任务的方式进行预训练，MLM和NSP，在训练BERT的时候，这两个过程是同时训练的，目的是要最小化两种策略的组合损失函数。

MLM是单词级别的分类任务，而NSP是句子级别的分类任务，组合损失函数如下：

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

其中， θ 是BERT中Encoder部分的参数， θ_1 是Mask-LM任务中在Encoder上所接的输出层中的参数， θ_2 则是句子预测任务中在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，因为它是一个词典大小 $|V|$ 上的多分类问题，那么具体说来有：

$$L_1(\theta, \theta_1) = - \sum_{i=1}^M \log p(m = m_i | \theta, \theta_1), m_i \in [1, 2, \dots, |V|]$$

在句子预测任务中，也是一个分类问题的损失函数：

$$L_2(\theta, \theta_2) = - \sum_{j=1}^N \log p(n = n_j | \theta, \theta_2), n_j \in [\text{IsNext}, \text{NotNext}]$$

因此，两个任务联合学习的损失函数是：

$$L(\theta, \theta_1, \theta_2) = - \sum_{i=1}^M \log p(m = m_i | \theta, \theta_1) - \sum_{j=1}^N \log p(n = n_j | \theta, \theta_2)$$

BERT的输入和输出

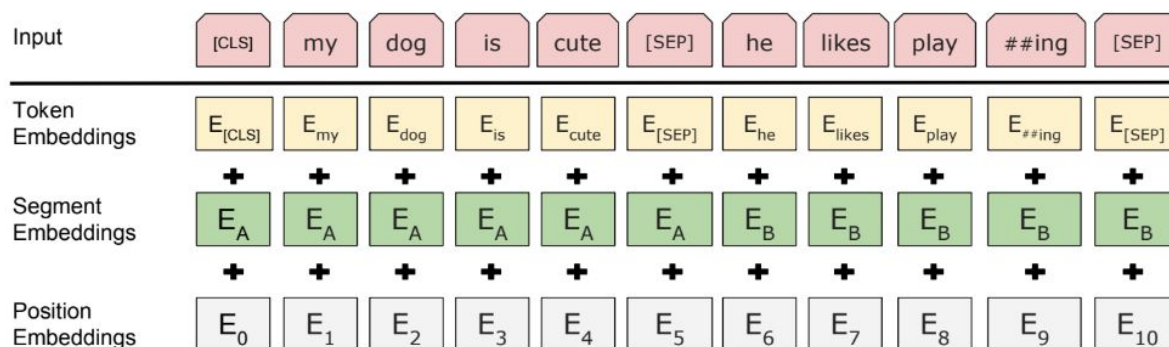
BERT模型的主要输入是文本中各个字/词（或者称为token）的原始词向量，该向量既可以随机初始化，也可以利用word2vec等算法进行预训练以作为初始值；输出是文本中各个token融合了全文语义信息后的向量表示。

注意，BERT除了token embedding外，还采用了两个额外的embedding的形式：

- 1) 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token语义信息相融合，便于处理双句任务，比如判断两个句子间的关系，以及QA等，通过segment embedding区分前后句子。
- 2) 位置向量（position embedding）：由于出现在文本不同位置的token所携带的语义信息存在差异（如“我爱你”和“你爱我”），因此，BERT对不同的token分别附加一个不同的向量作以区分。

另外，在做NSP的时候，会在第一句开头加上Token[CLS]用于标记句子开头，用[SEP]标记句子结束。

具体的结构如下图：



BERT应用于有空格丢失或者单词拼写错误等数据是否还有效

- 1) 对于有空格丢失的数据，需要先做分词处理，比如使用Bi-LSTM+CRF，处理成正常文本后，再输入BERT做下游任务
- 2) 对于有单词拼写错误的数据，如果单词拼写错误少，则不会造成很大影响，因为BERT预训练的语料非常丰富，而且语料也不是很干净，其中一定也有拼写错误的情况；但如果拼错的比例很大，比如达到30%以上，则需要通过人工特征工程的方式，来减少错别字带来的影响。

10.6.2 Masked LM

为了训练一个deep bidirectional representation，我们随机地mask掉一部分输入的token，然后预测这些token，这个过程被称作MLM。和一般的语言模型一样，mask token最后隐层的向量会通过一个softmax函数输出。和denoising autoencoders不同，这个过程只预测masked words而不是重组整个Input。

虽然这个过程可以达到双向预训练的目的，但是一个缺陷在于预训练和预测的数据不匹配了，因为[mask]token在预测过程中不存在。为了解决这个问题，masked tokens不都是直接被[mask]替代，而是：在选中15%的token positions后，80%词它被替换成[mask]，10%次被替换成随机的token，10%次不变。然后通过交叉熵损失函数对maskend token进行预测。

这么做的好处在于：Transformer的encoder并不知道那些单词被随机替代了，所以被迫去依赖于上下文信息预测词汇，并赋予了模型一定的纠错能力，另外，因为只有1.5%(10%*15%)的词被随机替换，随意不会影响模型理解语义的能力。

相较于传统的LM，MLM在每个batch中只预测15%的token，所以模型的收敛需要更多pre-training steps，也就是要花费更长的时间。但加了MLM之后的模型精度提到了很多。

BERT的预训练过程需要大量的数据集，且数据集的大小会影响微调的精度。

MLM vs CBOW

- 1) 相同点：CBOW的核心思想是根据上下文来预测中心词，而BERT本质上也是这么做的，但BERT是给定一个句子，随机MASK 15%的词，然后用BERT来预测这些MASK的词。
- 2) 不同点：在CBOW中，每个单词会成为输入，而BERT不是，因为这么做的话训练数据太大，而且训练时间也很长；其次，对于输入数据部分，CBOW中的输入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；另外，通过CBOW训练出的词向量是静态的，而BERT是动态的。

10.6.3 Next Sentence Prediction (NSP)

NSP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。用下面几个步骤来预测：

- 1) 整个输入序列输入给transformer
- 2) 用一个简单的分类层将[CLS]标记的输出变换为2*1的形状的向量
- 3) 用softmax计算IsNextSequence的概率

对于很多NLP任务来说，了解句子之间的相关性非常重要，比如问答和自然语言解析（natural language inference），而语言模型做不到这一点。

通过在SQuAD，QNLI等多个NLP任务数据上做实验，在不使用NSP的情况下，模型表现差很多。

10.6.4 Fine-tuning based on down-stream tasks

BERT可以用于各种NLP任务，只需要在核心模型中添加一个层。

- 1) 在分类任务中，如情感分析，只需要在transformer的输出层上加一个分类层
- 2) 在问答任务（如SQUAD v1.1）中，问答系统需要接受有关文本序列的question，并需要在序列中标记answer。可以用BERT学习两个标记answer开始和结尾的向量来训练Q&A模型
- 3) 在命名实体识别（NER）中，系统需要接收文本序列，标记文本中的各类实体（人员、组织、日期等）。可以用BERT将每个token的输出向量送到预测NER标签的分类层。

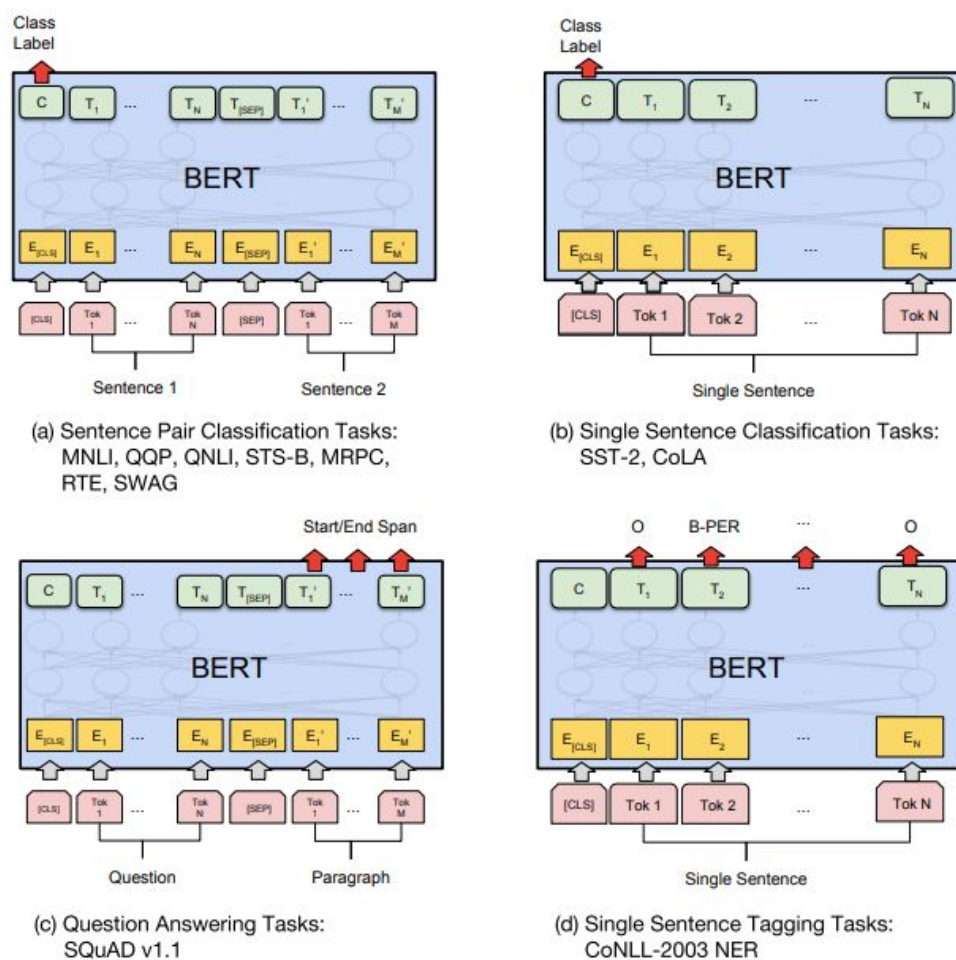


Figure 4: Illustrations of Fine-tuning BERT on Different Tasks.

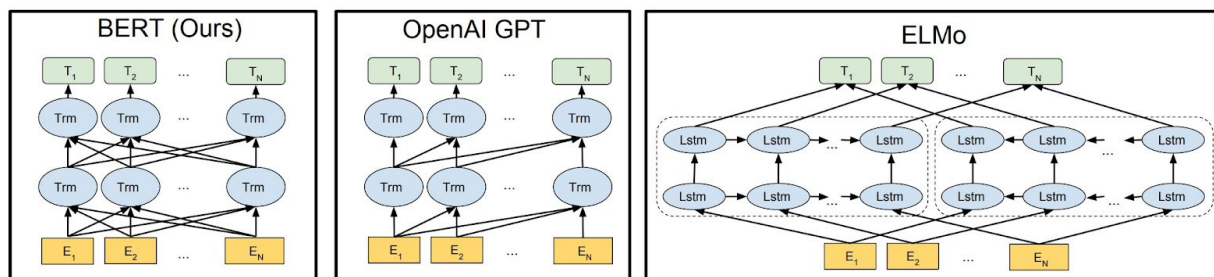
微调的过程中，模型大部分的超参数都和预训练时一样，除了batch size，学习率和training epochs。最优的参数因不同的任务而异。Paper指出如下的一个参数组合在大部分任务中表现都良好:batch size(16,32), learning rate(Adam, 5e-5, 3e-5, 2e-5), Number of epochs(2, 3, 4)。增加训练集的大小对模型表现的影响远小于超参数的选择。所以直接通过exhaustive search对以上参数组合进行试验并选择最好的组合是个不错的方法。

BERT模型Finetune及其改进

1) 基于自己的场景数据，尝试不同的max_sent_len, batch_size, epoch, dropout, warm_up_ratio

- 2) 基于自己的场景数据，尝试BERT加CONV 2D来提取全局信息
- 3) 针对自己的场景数据，尝试BERT的剪裁、蒸馏（也许仅使用一半的layer就可以达到下游所需的精度）
- 4) 针对自己的场景数据，尝试将单分类层改为双分类层

10.6.5 ELMo vs GPT vs BERT



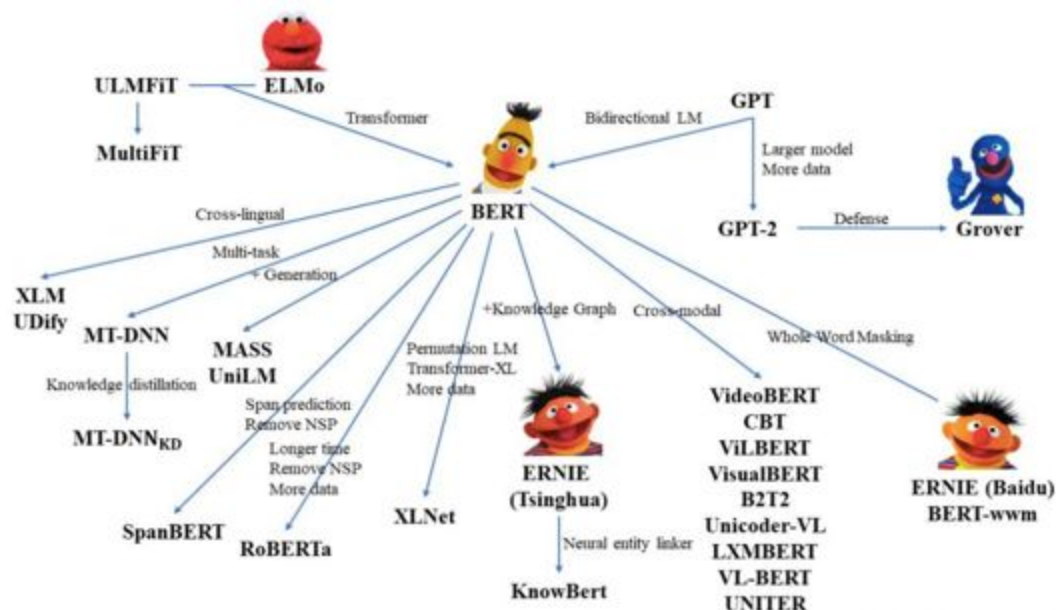
静态词向量无法解决一词多义的问题，而ELMo，GPT和BERT词向量都是基于语言模型的动态词向量，他们三者的区别在于：

- 1) BERT和GPT是fine-tuning的方法，而ELMo是feature-based 方法。
- 2) 特征提取器：ELMo采用LSTM进行提取，GPT和BERT则采用Transformer进行提取。LSTM一般有梯度爆炸或是消失的问题，即便使用了attention也无法全部避免，导致模型无法很好的捕捉长期记忆，模型无法并行化，只会更关注距离较近的词。而Transformer是并行的模型，可以解决以上问题。
- 3) Autoregressive vs Autoencoding：GPT和ELMo都是自回归语言模型，即从左到右或者从右到左得进行预测，在自回归模型中，每个token被预测后都会被放进输入序列来预测下一个输出的token（XLNet和TransformerXL也是自回归模型）。ELMo虽然看上去利用上下文，但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一体化的融合特征方式弱。自回归(autoregressive)LM的缺点在于只能利用上文或者下文的信息，而不能同时利用上下文，优点在于，擅长生成类的任务，比如文本摘要，机器翻译等，因为在生成内容时，一般都是从左到右的。而BERT在生成类问题中，由于面临训练过程和应用过程不一致的问题，反而效果没有autoregressive的LM好。

BERT利用了典型的Denoising autoencoder的思路，MLM的过程就是在输入侧加入噪音，使得模型可以利用上下文进行预测，是一个自编码模型。

4) GPT和BERT的其他差别：预训练的语料库不同；Sentence separator [SEP]和 classifier token [CLS]在GPT中只在微调的过程中引入，而在BERT中预训练过程中也参与了；GPT和BERT预训练时的超参数选择不同

预训练模型家族



BERT存在的问题

- 1) 如何用更小的模型，更少的训练数据
- 2) 可解释性
- 3) 如何加入先验知识，比如把知识图谱加入模型

10.6.6 Optimizations of BERT

BERT最大的问题就是太大，BERT-base包含一亿个参数，BERT-large包含3.4亿个参数，它的训练和推理时间也很长。对以上问题，最直接的方法就是优化神经网络，比较通用的和压缩和加速神经网络的方法如下：

- 1) 架构优化：将原有的架构改进为更快的架构，例如：将RNN替换为Transformer或者CNN；使用较少计算的层等，或从学习率，batch size, epoch数等方面进行优化

- 2) 模型压缩：通常使用量化和修剪来完成，从而能够在架构不变（或者大架构不变）的情况下减少计算总量
- 3) 模型蒸馏：训练一个较小的模型，该模型能够复制原始模型的行为。

BERT模型预训练及其改进

- 1) BERT模型以字为基础，在中文场景中词包含的信息更为广泛

对应改进：使用全词MASK，即MASK的从字替换到词——哈工大版Chinese-WWM

- 2) BERT模型以字为基础，在中文场景中短语或者实体包含的信息更为广泛

对应改进：增加基于短语和实体（人名、位置、组织、产品等）的MASK——ERINE 1.0（百度推出）

- 3) BERT的预训练任务只有两个，为了增大其通用性，增加预处理任务的数量

对应改进：使用多个预处理任务进行训练——ERINE 2.0

- 4) NSP任务在很多下游任务中表现一般

对应改进：使用动态MASK、修改NSP任务及调整BERT参数——Roberta

- 5) BERT模型参数过大，预训练时间过长

对应改进：ALBERT 词嵌入向量的因式分解和层间参数共享

这两种方法可以显著降低BERT的参数数量，而不会严重影响性能，从而提高参数效率。

经实验表明，与BERT-large具有相似配置的ALBERT，其参数能够降低18倍，训练速度提高了1.7倍。

❖ 词嵌入向量的因式分解

- 传统的BERT模型，词向量的维度和模型隐藏层的维度是一致的。
- 由于词向量没有包含上下文信息，而模型隐藏状态向量有包含上下文信息，所以隐藏状态向量应该比词向量包含更多的信息。因此，将两者的维度绑定到一起是不合理的，隐藏状态向量的维度应该远大于词向量的维度。
- 所以将词向量维度降维成E，隐藏状态维度H保持不变，同时训练一个E*H转换矩阵。在词向量和隐藏状态向量交互时，先将词向量乘以转换矩阵升维到隐藏状态的维度。
- 因式分解后，模型需维护一个V*E（词表大小V*词向量维度E）的矩阵，和一个E*H（词向量维度E*隐藏状态维度H）的矩阵。

- ❖ 层间参数共享：传统的BERT有12层，每层有FFN和attention两个子层，模型训练好后，它们的参数是不一样的。在“层间参数共享”方法里，部分层（子层）的参数可以是共享的，这样就节约了存储参数的空间。

经过炼丹实践，得出以下经验：

- 共享FFN层的参数，会导致模型效果下降。
- 共享attention层的参数，模型效果不会下降（词向量维度 $E=128$ 时）或者轻微下降（词向量维度 $E=768$ 时）。
- 我们可以将L层模型分为N组，每组包含M层，进行组内参数的共享。

TinyBERT：采用蒸馏的方法，学习到一个小型的网络

6) BERT在通用场景下效果不错，但在个人场景下效果一般

对应改进：基于内部语料集合进行再次预训练——JDAI-BERT（京东推出）

7) BERT在生成式任务表现一般

对应改进：将BERT模型的DAE变为DAR——XLNET(站在自回归LM的角度，如何引入双向语言模型，如果是站在DAE LM的角度，如何抛掉表面的那个[mask] 标记，让预训练和fine-tuning保持一致)

评估词向量

Intrinsic evaluation metric：通过可视化词向量或者求的词向量之间的距离来判别，rational为同类词的词向量距离应该更小（相似的词总是同时出现）。

外在的评估方法：对词向量的评估主要看它对下游应用的效果，如命名实体识别，情感分析，文本分类等，看对最终的效果有无提升。

11. Down-stream tasks

NLP有四大常见任务

- 1) 序列标注：如NER，语义标注、词性标注、分词等
- 2) 分类任务：如文本分类、情感分析等
- 3) 句对关系判断：如自然语言推理、QA、文本语义相似性等
- 4) 生成式任务：如机器翻译、文本摘要、图像描述等

11.1 Machine Translation

传统的机器翻译需要使用平行语料库：一组文本，每个文本都被翻译成一种或多种不同于原文的其他语言。一般是使用统计模型，比如贝叶斯规则的概率公式，需要大量的手工特征工程，非常复杂。

标准的**神经机器翻译**是一种端到端的神经网络，其中，源语句由编码器的RNN/LSTM/GRU编码，目标词解码器预测。编码器一次读取一个源语句，然后再最后隐藏状态汇总整个源句子。解码器使用反向传播学习这个汇总并返回翻译后的版本。

相比传统的基于统计的机器翻译的方法，神经机器翻译的最大成功在于：

- 1) 端到端训练：NMT中所有参数同时被优化，最大限度的减少网络输出的损耗性能
- 2) 分布式表示的优势：NMT更好地利用单词和短语的相似性
- 3) NMT更好地搜索上下文：源文本和目标文本以此进行更准确的翻译
- 4) 更流利的文本生成：深度学习文本生成质量高于平行语料库。

[RNN](#)的最大问题是梯度消失/爆炸，可以用[LSTM/GRU](#)来替代；加入[Attention机制](#)；改为并行的[Transformer](#)结构

机器翻译在工业界场景的应用主要有同声传译（语音识别）、拍照翻译（OCR）、辅助翻译（人机交互）。

机器翻译主要的方法如下：



机器翻译的困难

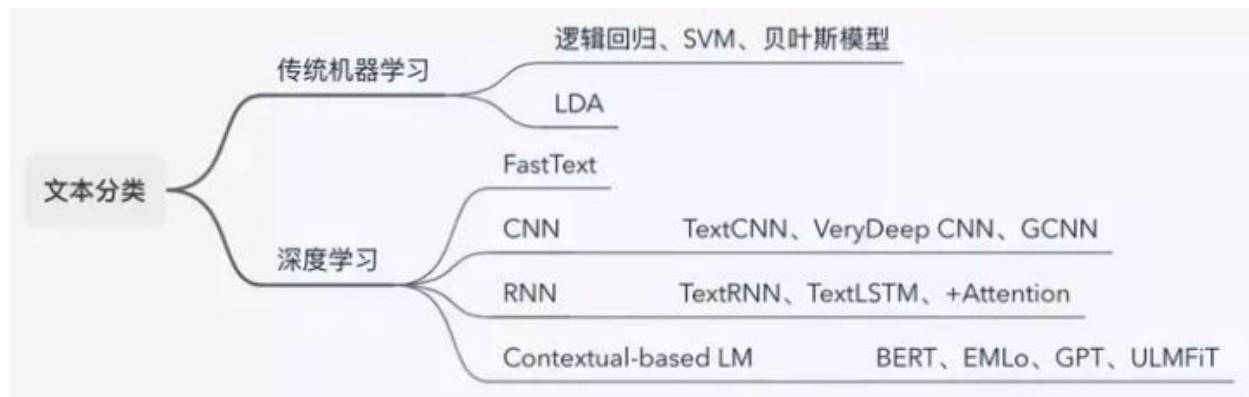
- 1) 歧义和新生词
- 2) 翻译不仅仅是字符串的转换：直接翻译无法解决所有问题。比如青梅竹马直接翻译成英文是“发青的梅子，竹子做的马”
- 3) 翻译的解不唯一：缺乏量化标准，且始终存在人为标准
- 4) 翻译的高度：有些句子意境很高，上升到文学层次了，比如“最是那一低头的温柔”，翻译很难把握到意境。

11.2 Sentimental Analysis

情感分析是文本分类的一种。

文本分类模型组成部分的顺序是：文本清理以去除噪声——文本标注以创建更多特征——将基于文本的特征转换为预测器——使用梯度下降学习一个模型——模型调优。

文本分类的方法有如下种，如果是语料语义简单的分类任务，用传统机器学习方法即可，或者用 fasttext；如果是像社交网络数据之类语义丰富的场景，可以考虑深度学习，CNN擅长捕获局部特征，RNN擅长处理时序信息，预训练模型适合场景更多，都可以一试。



目前情感分析的主要研究方法还是基于机器学习的传统算法，比如SVM、信息熵、朴素贝叶斯、CRF等。以上的都是有监督学习算法，但是有监督学习依赖于大量人工标注的数据，系统需要付出很高的标注代价，相反，无监督学习不需要人工标注数据训练模型，是降低标注代价的解决方案，但是它完全依赖算法学习结果，往往效果不佳，而半监督学习则是采取综合利用少量已标注样本和大量未标注样本提高学习性能的机器学习方法，兼顾人工标注成本和学习效果。

不同于传统的词袋模型，在使用神经网络解决NLP问题的时候，通常使用词嵌入的方法来避免数据维灾难和语义鸿沟问题，在将文字转化为向量后用传统机器学习的算法进行分类，即为监督学习的方法。如果文档很大，需要关注long term dependencies，可以使用BERT进行fine-tuning，在最后一层后加一层分类层进行文本分类。

情感分析涉及的问题十分多样，包括很多研究任务，比如情感分析、观点挖掘、观点信息提取、情感挖掘、主观性分析、倾向性分析、情绪分析及评论挖掘等。情感分析的主要方法如下：



11.3 Sequential tagging

序列标注问题是分类问题的推广，其主要应用有如下方面：

1) 命名实体识别指从输入文本中识别出有特定意义或指代性强的实体，是机器翻译、知识图谱、关系抽取、问答系统等的基础。学术上 NER 的命名实体分 3 大类和 7 小类，3 大类指实体类、时间类、数字类，7 小类指人名、地名、组织机构名、时间、日期、货币、百分比。语言具有语法，语料遵循一定的语法结构，所以 CRF、HMM 和 MEMM 等概率图模型被用来分析标签转移概率，包括深度学习模型一般会加上 CRF 层来负责句子级别的标签预测。NER 通常包含两个部分：实体边界识别和实体分类，其中实体边界识别判断一个字符串是否是一个实体，而实体分类将识别出来的实体划分到预先给定的不同类别中。NER 的主要难点在于表达不规律、且缺乏训练语料的开放域命名实体类别（如电影、歌曲名等）。NER 的主要方法如下：



- 2) 词性标注 (Part-of-speech Tagging) 给定切好词的句子，词性标注的目的是为每一个词赋予一个类别，比如名词、动词、形容词等
- 3) 关系抽取指的是检测和识别文本中实体之间的语义关系，并将同一语义关系的表示链接起来的任务。关系抽取的输出通常是一个三元组 (实体1, 关系类别, 实体2)，表示实体1和实体2之间存在特定类别的语义关系。例如，句子“北京是中国的首都、政治中心和文化中心”中表述的关系可以表示为 (中国, 首都, 北京)，(中国, 政治中心, 北京) 和 (中国, 文化中心, 北京)。关系抽取通常包含两个核心模块：关系检测和关系分类，其中关系检测判断两个实体之间是否存在语义关系，而关系分类将存在语义关系的实体对划分到预先指定的类别中。在某些场景和任务下，关系抽取系统也可能包含关系发现模块，其主要目的是发现实体和实体之间存在的语义关系类别。例如，发现人物和公司之间存在雇员、CEO、CTO、创始人、董事长等关系类别。
- 4) 事件抽取：事件抽取指的是从非结构化文本中抽取事件信息，并将其以结构化形式呈现出来的任务。例如，从“毛泽东 1893 年出生于湖南湘潭”这句话中抽取事件{类型：出生，人物：毛泽东，时间：1893年，出生地：湖南湘潭}。事件抽取任务通常包含事件类型识别和事件元素填充两个子任务。事件类型识别判断一句话是否表达了特定类型的事件。事件类型决定了事件表示的模板，不同类型的事件具有不同的模板。例如出生事件的模板是{人物，时间，出生地}，而恐怖袭击事件的模板是{地点，时间，袭击者，受害者，受伤人数,...}。事件元素指组成事件的关键元素，事件元素识别指的是根据所属的事件模板，抽取相应的元素，并为其标上正确元素标签的任务
- 5) 信息集成：实体、关系和事件分别表示了单篇文本中不同粒度的信息。在很多应用中，需要将来自不同数据源、不同文本的信息综合起来进行决策，这就需要研究信息集成技术。目前，信息抽取研究中的信息集成技术主要包括共指消解技术和实体链接技术。共指消解指的是检测同一实体/关系/事件的不同提及，并将其链接在一起的任务，例如，识别“乔布斯是苹果的创始人之一，他经历了苹果公司几十年的起落与兴衰”这句话中的“乔布斯”和“他”指的是同一实体。实体链接的目的是确定实体名所指向的真实世界实体。例如识别上一句话中的“苹果”和“乔布斯”分别指向真实世界中的苹果公司和其CEO史蒂夫·乔布斯。

目前对于NLP中序列标注问题的主要研究方法有概率图模型（HMM, CRF）和神经网络（一般为bi-LSTM+CRF）。

11.3.1 HMM

HMM是概率图模型（生成模型，有向图）的典型代表。概率图模型根据变量间的相关关系一般可以分为贝叶斯网络（变量间存在显性的因果依赖关系，用有向无环图表示）和马尔科夫网（变量间存在相关性但因果关系难以获得，用无向图表示）。HMM在语音识别、NLP、时序数据建模、生物信息（基因序列）、模式识别和故障诊断等多个领域有广泛应用。

马尔科夫模型

马尔科夫模型主要用于描述系统状态间的转移过程，即系统随时间或空间的推移，从每一状态转移至另一状态。在马尔科夫过程中假设系统在 t 时刻的状态只与上一时刻 $t-1$ 有关，而与之之前的状态无关。

在现实生活中对于满足马尔可夫假设的状态随机转移过程，我们均可由马尔可夫模型表示。如天气的变化，产品生产线质量的变化，硬币的抛掷结果（满足二项分布）等等。注意马氏过程描述的是状态间的随机转移过程，其随机性由概率刻画，而对于确定的状态转移过程，如红绿灯的变化一般不用马氏过程描述。

另外对于马尔可夫模型还有一些有趣的结论，如经过相当长的时间后，也即经过多次的状态转移后，最终的状态将收敛于同一结果，而于起始状态无关。

隐马尔科夫模型

HMM表示的是在一个系统中同时存在两条序列，一个是可以直接通过观察得到的显示观察序列，另一条是隐藏的状态转移序列，注意状态转移序列会影响观察序列的结果，因此我们需要通过观察序列推断最可能的状态转移序列，即通过可见事物的变化揭示深藏其后的内在的本质规律。

HMM主要包含五个部分：观察值序列、隐含的状态转移序列、初始状态空间的概率分布、模型中状态的有限集合、状态转移概率矩阵。

三个假设

齐次马尔可夫性假设：在隐含的状态转移序列中，第 t 时刻的状态只与第 $t-1$ 时刻的状态有关，而与之之前的状态无关

不动性假设：状态与具体时间无关

输出独立性假设：观察序列在 t 时刻的取值仅由 t 时刻的隐状态决定而与之之前的隐状态无关。

HMM的三个基本问题

- 1) 评估问题：给定模型和观测序列，计算模型下观测序列输出的概率——前向后向算法
- 2) 解码（预测）问题：已知模型和观测序列，求解对应的状态序列——近似算法（贪心算法）和维比特算法（动态规划求最优解）
- 3) 学习问题：已知观测序列，估计模型参数（状态转移概率矩阵和给定状态下观测值概率分布矩阵）——保姆韦尔奇（即EM）算法和极大似然估计

HMM的评估问题

Forward算法核心是计算 $P(Z_k, x_1 \dots x_k)$ ：某一个时刻的参数和观测值同时发生的联合概率，本质是动态规划算法以降低算法的时间复杂度

Backward算法核心是计算 $P(x_{k+1} \dots x_n | Z_k)$ ：知道参数的情况下的产生某些观测值的条件概率。

再通过贝叶斯定理将F,B算法的公式推导出 $P(Z_k | x)$

前向后向算法的时间复杂度都是 $O(N^2T)$ ， T 是时间序列长度， N 是变量数，如果用暴力穷举法，时间复杂度为 $O(N^M)$ ，其中 M 为观测序列长度。

HMM解码问题

和评估问题相似，它的求解方法也包括暴力穷举法，但考虑时间复杂度，一般实际求解都使用Viterbi算法。它也是一种动态规划算法，同时也是很多NLP任务的解码算法。

动态规划的原理：

- 1) 如果全局最短路径（概率最大路径）在 t 时刻经过某个结点 i ，则我们一定可以找到从 $S_{t=0}$ 结点（开始结点）到 i 结点的最短路径；
- 2) 从 $S_{t=0}$ 结点到 E 结点（末端结点），其最短路径必经过结点 i ；
- 3) 在计算从结点 $S_{t=0}$ 至结点 $i+1$ 的最短路径时，我们只需计算结点 $S_{t=0}$ 到结点 i 的最短路径和结点 i 到结点 $i+1$ 的最短路径即可。

Viterbi算法的时间复杂度为 $O(K^2T)$ ，其中 K 为隐状态的个数， T 为时间序列长度（源码见

https://github.com/fangyiyu/NLP_sourcecode/blob/master/HMM%20and%20Viterbi%20algorithm.ipynb)

HMM参数学习问题

HMM的主要参数为隐状态转移概率矩阵和给定状态下观测值的概率分布。

如果观察序列和隐状态都已知，可利用有监督的学习方法，通过最大似然函数估计参数；如果隐状态未知，此时采用无监督学习的EM算法，估计参数使得 $P(O|\lambda)$ 最大，其中，O为观测状态序列。

EM算法

Expectation Maximization algorithm，期望最大化算法，它是一种非监督学习算法，一般MLE可以直接解决没有隐变量数据样本的问题，而EM则专门通过近似的MLE来求解含有隐变量的概率模型的参数。

EM 算法的每次迭代过程都包含两部:E和M，E（计算期望）是利用对隐藏变量的现有估计值，计算其最大似然估计值；M（最大化）是用在E步上求得的最大似然值来计算参数的值。M步上找到的参数估计值被用于下一个E步计算中，这个过程不断交替进行。

Maximum Likelihood Estimation，极大似然估计，常用来构造目标函数。比如LR，朴素贝叶斯的后验概率等。

似然函数是一种关于统计模型中的参数的函数，表述模型参数中的似然性（概率）。而极大似然就是最大可能的意思。多数情况我们都是通过条件推结果，而最大似然估计是已知结果，然后寻求使得结果出现的可能性最大的条件，以此作为估计值。这么一说，似然函数可以理解为条件概率的逆反。

极大似然估计的核心关键是由于一些情况，样本太多，无法得出分布的参数值，可以采样小样本后，利用极大似然估计获取假设中分布的参数值。

极大似然估计中采样需要满足一个重要的假设，即所有的采样都是独立同分布的。

似然函数举例：

假设我们要统计10万人中的男女身高分布，由于10万数量巨大，要怎么做呢？对于这个问题，可以通过数学建模解决：

- 1) 我们采用随机抽样，从10万人中抽取100个男生，100个女生，然后依次统计他们的身高。组成样本集 X ， $X=\{x_1, x_2, \dots, x_n\}$ ，其中 x_i 表示抽到第 i 个人的身高， $N=100$ ，表示抽到的样本数。
- 2) 假设男生身高服从正态分布 $N(\mu_1, \sigma_1^2)$ ，女生身高服从另一个正态分布 $N(\mu_2, \sigma_2^2)$ 。
- 3) 但是这两个分布的均值和方差都不知道，设为未知参数 $\theta = [\mu, \sigma]^T$
- 4) 现在要通过极大似然法，通过这100个男生或者100个女生的身高结果，即样本集 X 来估计这两个正态分布的未知参数 θ ，问题定义相当于知道 X ，求 θ ，换言之就是求 $p(\theta | X)$ 。

由于抽到每个男生的概率都服从高斯分布并且相互独立，所以抽中他们的联合概率是连乘的，用下式表示：

$$L(\theta) = L(x_1, \dots, x_n; \theta) = \prod_{i=1}^n p(x_i; \theta), \theta \in \Theta.$$

那么从10万个人中抽到这100个人，而不是其他人，就说明在10万人当中，这100个人出现的概率最大，这个概率就是上面这个似然函数，那么怎样的 θ 可以让 $L(\theta)$ 最大呢？

为了便于求解，可以定义对数似然函数，将其变成连加的：

$$H(\theta) = \ln L(\theta) = \ln \prod_{i=1}^n p(x_i; \theta) = \sum_{i=1}^n \ln p(x_i; \theta)$$

对于求解函数的极值，直接想法就是求导，让导数为0，那么解这个方程得到的 θ 就是了（前提是函数连续可微），但如果 θ 是包含多个参数的向量，就求函数对所有参数的偏导，也就是梯度，我们有 n 个未知的参数，就会得到 n 个方程，求出方程组的解，就是似然函数的极值了。

求MLE估计值的一般步骤：

- 1) 写出似然函数;
- 2) 对似然函数取对数, 并整理 ;
- 3) 求导数, 令导数为0, 得到似然方程 ;
- 5) 解似然方程, 得到的参数即为所求。

在以上的例子中, 如果在抽取的200人中, 我们不知道哪些是男生哪些是女生, 也就是性别是个隐变量, 那么这时的求解概率分布参数就要用EM算法。

EM算法的思想

- 1) 给 θ 自主规定个初值
 - 2) 根据给定观测数据和当前参数 θ , 求未观测数据 z 的条件概率分布的期望
 - 3) 上一步的在已经求出来了, 于是根据极大似然估计求最优的 θ
 - 4) 因为第二步和第三步的结果可能不是最优的, 所以重复第二步和第三步, 知道收敛。
- 以上步骤中第二步被称作E步, 第三步被称作M步, 从而不断E、M。

也就是说, EM算法是一种从不完整数据或有数据丢失的数据集 (存在隐含变量) 中求解概率模型参数的最大似然估计方法。

EM (expectation maximization) 算法和Gradient Descent算法系出同源, 都是通过自身的迭代更新求得使得目标函数达到最优解的模型系数。EM是这一类特定的算法, 而GD可以算是其中一个特例, 因为它用抛物线逼近来得到结果, 而用其他曲线的逼近方案, 基本都被称作EM算法。注意EM算法对初值敏感, 无法保证找到全局最优解。

而在NLP问题中的目标函数涉及概率, 在0-1之间, 而GD无法保证非负数的约束, 这就导致GD无法在NLP中使用。

HMM的局限性

- 1) 马尔科夫性 (有限历史性) : 实际上在NLP领域的文本数据, 很多词语都是长依赖的
- 2) 齐次性 : 序列不同位置的状态转移矩阵可能会有所变化, 即位置信息会影响预测结果

- 3) 观测独立性：观测值和观测值（字与字）之间是有相关性的
- 4) 单向图：只与前序状态有关，和后续状态无关。在NLP任务中，上下文的信息都是必须的。

HMM在序列标记中的应用

1) 分词。分词问题主要包括隐马模型解码问题和评估问题。解码问题即根据隐马模型直接进行分词，此时观察序列即为词序列而状态序列即为分词结果。而评估问题则主要用于分词出现多种可能时，求解观察序列中概率最高的结果，此时观察序列则为多个分词结果，而状态序列则为句子序列。评估问题主要用于分词消歧。

2) 词性标注。词性标注相当于HMM的解码问题，即根据观察序列（词序列）下，概率最大的标注序列（词性）。

3) 短语识别、语音识别

11.3.2 Belief Networks

马尔科夫链的一个假设是每个状态值取决于前面有限个状态。但现实生活中，很多事物相互的关系并不能用一条链串起来，它们之间的关系可能是交叉的、错综复杂的。比如

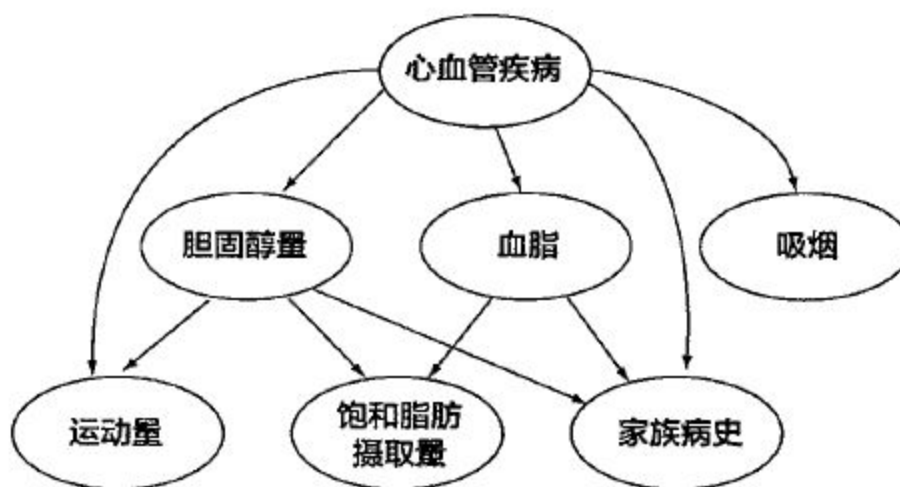


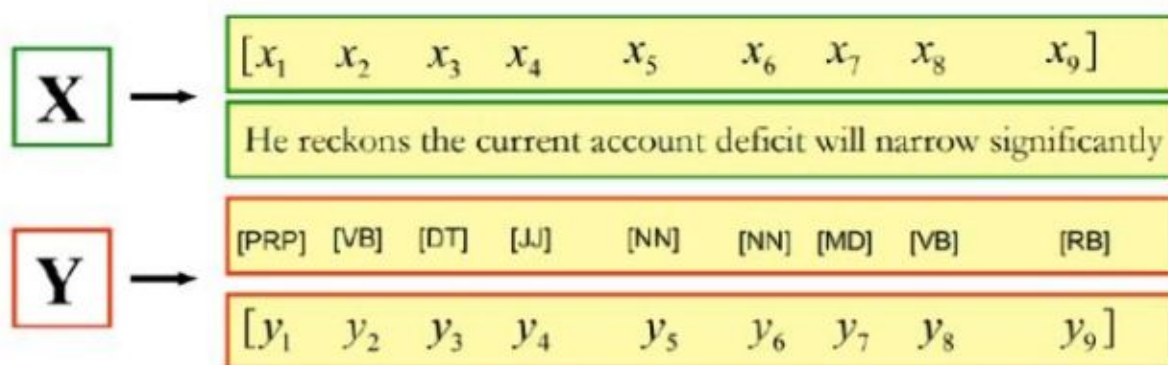
图 24.1 描述心血管疾病和成因的一个简单的贝叶斯网络

在以上的有向图中，假设马尔科夫假设成立，即每个状态只与它直接相连的状态有关，而与它间接相连的状态没有直接关系，那么它就是贝叶斯网络。在网络中每个节点的概率，都可以用贝叶斯公式进行计算。

贝叶斯网络是一个加权的有向图，是马尔科夫链的扩展。其克服了马尔科夫链那种机械的线性约束，它可以把任何有关联的时间统一到它的框架下。因此，其应用很广，包括文本分类、概念抽取，甚至生物统计、图像处理和博弈论等。

11.3.3 CRF

CRF 是一个序列化标注算法 (sequence labeling algorithm)，接收一个输入序列如 $X = (x_1, x_2, \dots, x_n)$ 并且输出目标序列 $Y = (y_1, y_2, \dots, y_n)$ ，也能被看作是一种seq2seq模型。这里使用大写 X, Y 表示序列。例如，在词性标注任务中，输入序列为一串单词，输出序列就是相应的词性。



CRF常用来做词性标注、组块分析、命名实体识别、句法分析等任务。

随机场可看成是一组随机变量的集合。这些随机变量间可能存在某种相互依赖的关系，当我们为每一个位置的随机变量根据某种分布随机赋予相应空间一个值后，其全体就是随机场。

我们将满足马尔科夫性的随机场称为马尔科夫随机场（MRF），若给定的MRF中每一个随机变量均对应一个观察值，此时需要根据观察值集合确定MRF分布（条件分布），则将这个MRF分布称为CRF。

CRF是由输入对输出进行预测的判别模型。通常，判别式模型 *discriminative model* 计算条件概率，而生成式模型 *generative model* 计算联合概率分布。极大似然法（MLE）是CRF的学习方法。

HMM vs CRF

- 1) 他们都利用了图的知识，但CRF利用的是马尔科夫随机场（无向图），而HMM利用的是贝叶斯网络（有向图）
- 2) CRF也有概率计算问题、学习问题和预测问题，大致计算方法和HMM相似，但是没有利用EM算法进行学习，而是MLE
- 3) CRF是判别模型，HMM是生成模型

11.3.4 Bi-LSTM + CRF

简单的LSTM的优点是能够通过双向的设置学习到观测序列（输入的子）之间的依赖，在训练过程中，LSTM可以根据目标（比如NER）自动提取观测序列的特征，但缺点是无法学习到状态序列（输出的标注）之间的关系，相反，CRF的优点是能对隐含状态建模，学习状态序列的特点，但缺点是需要手动提取序列特征，所以，在LSTM后再加一层CRF，可以获得两者的优点。

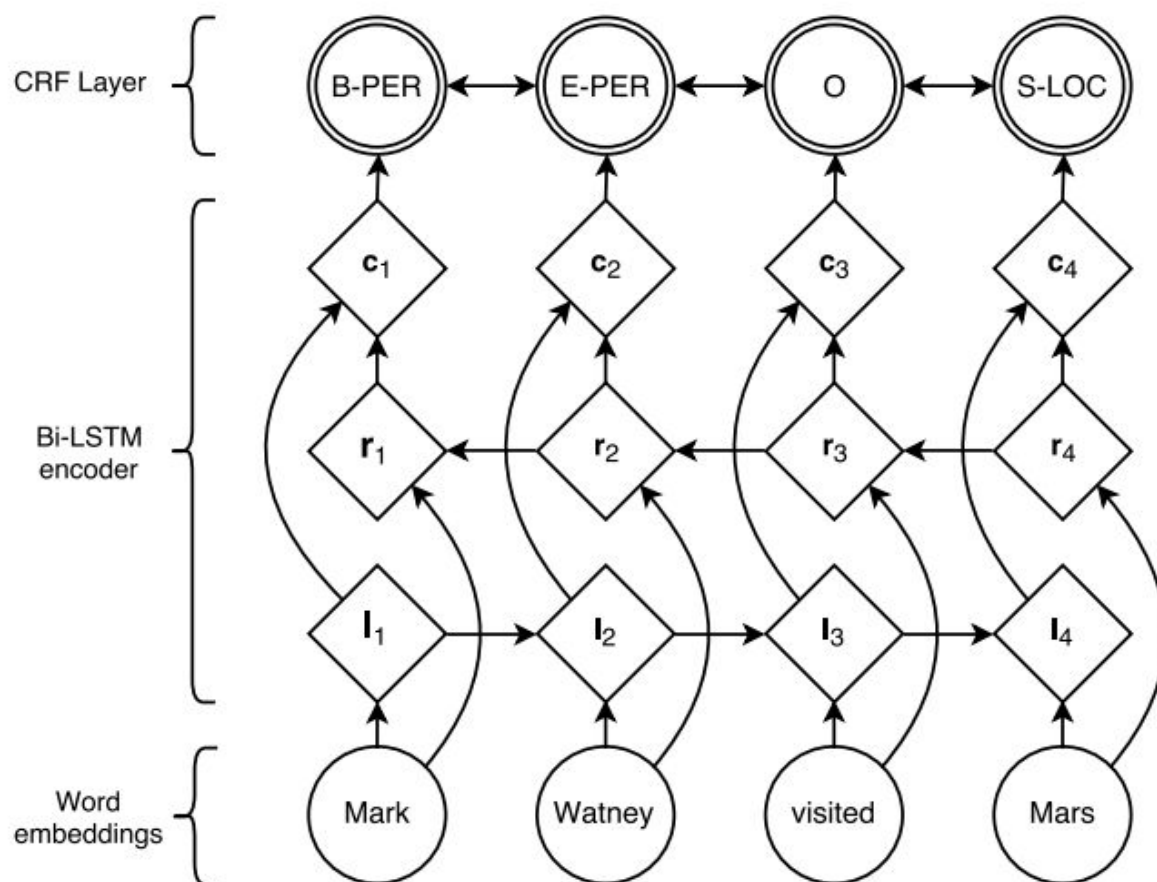




图18. 神经网络序列标注模型架构

11.4 Language Modeling

给定前一个单词去预测下一个单词。应用场景包含智能键盘、电子邮件自动回复、拼写自动改正等。

语言建模是很多技术发展的核心，如：

- 1) 词嵌入：word2vec的目标是简化语言模型
- 2) seq2seq：通过一次预测一个单词生成一个输出序列
- 3) 预训练语言模型：使用语言模型的表述进行迁移学习，如BERT

11.4.1 N-gram

某个词的出现依赖于其他若干个词；我们获得的信息越多，预测越准确。N-gram是一种语言模型，语言模型是一个基于概率的判别模型，它的输入是一句话（单词的顺序序列），输出的是这句话的概率，即这些单词的联合概率。

概率计算

假设我们有一个由 n 个词组成的句子 $S = (w_1, w_2, \dots, w_n)$ ，如何衡量它的概率呢？让我们假设，每一个单词 w_i 都要依赖于从第一个单词 w_1 到它之前一个单词 w_{i-1} 的影响：

$$p(S) = p(w_1 w_2 \dots w_n) = p(w_1) p(w_2 | w_1) \dots p(w_n | w_{n-1} \dots w_2 w_1)$$

但这个方法有两个缺陷：

- 1) 参数空间过大，the last term in the formula的参数有 $O(N)$ 个；
- 2) 数据稀疏性严重：词同时出现的情况可能没有，组合阶数高的时候尤其明显。

为了解决第一问题，引入了**马尔科夫假设**(Markov Assumption)：一个词的出现仅与它之前的若干个词有关。

$$p(w_1 \dots w_n) = \prod p(w_i | w_{i-1} \dots w_1) \approx \prod p(w_i | w_{i-1} \dots w_{i-N+1})$$

- 如果一个词的出现仅依赖于它前面出现的一个词，那么我们就称之为 **Bi-gram**：

$$p(S) = p(w_1 w_2 \dots w_n) = p(w_1) p(w_2 | w_1) \dots p(w_n | w_{n-1})$$

- 如果一个词的出现仅依赖于它前面出现的两个词，那么我们就称之为 **Tri-gram**：

$$p(S) = p(w_1 w_2 \dots w_n) = p(w_1) p(w_2 | w_1) \dots p(w_n | w_{n-1} w_{n-2})$$

N-gram的 N 可以取很高，然而现实中一般 bi-gram 和 tri-gram 就够用了。

那么，如何计算其中的每一项条件概率 $p(w_n|w_{n-1} \cdots w_2 w_1)$ 呢？答案是**极大似然估计 (Maximum Likelihood Estimation, MLE) **，说人话就是数频数：

$$p(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

$$p(w_n|w_{n-1}w_{n-2}) = \frac{C(w_{n-2}w_{n-1}w_n)}{C(w_{n-2}w_{n-1})}$$

$$p(w_n|w_{n-1} \cdots w_2 w_1) = \frac{C(w_1 w_2 \cdots w_n)}{C(w_1 w_2 \cdots w_{n-1})}$$

N 的取值一般都比较小，主要有两个原因：

1) N元模型的空间复杂度几乎是N的指数函数，即 $O(|V|^{**N})$ ，这里|V|是一种语言词典的词汇量，一般在几万到几十万个。而使用N元模型的时间复杂度，也几乎是一个指数函数， $O(|V|^{** (N-1)})$ 。因此，N不能很大。当N从1到2，再从2到3时，模型的效果上升显著，而从3到4时，提升的效果就不显著了，然而资源的耗费增加却非常快。

2) 并不是更高阶的模型就能覆盖所有的语言现象。在自然语言中，上下文之间的相关性可能跨度非常大，甚至可能从一个段落到另一个段落。因此，即便模型的阶数再高，对这种情况也无可奈何，这就是**马尔科夫假设的局限性**。这时就要采用其他一些长程的依赖性(long term dependencies)来解决这个问题。

通常会在句子的开头添加N-1个<s>，在句子结尾加一个</s>以便于句子概率的计算。

N-gram的应用

1) 词性标注

N-gram可以实现词性标注。例如“爱”这个词，它既可以作为动词使用，也可以作为名词使用。不失一般性，假设我们需要匹配一句话中“爱”的词性。



匹配：“龙龙 爱 中区饭堂”

<https://blog.csdn.net/songbinxu>

我们可以将词性标注看成一个多分类问题，按照Bi-gram计算每一个词性概率：

$$p(\text{词性}_i | \text{"龙龙"的词性, "爱"}) = \frac{\text{前面是“名词”的“爱”作为词性}_i \text{的出现次数}}{\text{前面是“名词”的“爱”的出现次数}}$$

选取概率更大的词性（比如动词）作为这句话中“爱”字的词性。

2) 垃圾短信分类

文章开头提到的那个垃圾短信分类问题，我们可以用N-gram来解决。在朴素贝叶斯的基础上，稍微对条件概率做一点改动即可。

$$p(\text{垃圾短信} | \text{"在家日赚百万"}) \propto p(\text{垃圾邮件}) p(\text{"在家日赚百万"} | \text{垃圾短信})$$

条件概率不再是各词语之间独立：

$$\begin{aligned} p(\text{"在家日赚百万"} | J) &= p(\text{"在", "家", "日", "赚", "百", "万"} | J) \\ &= p(\text{"在"} | J) \times p(\text{"家"} | \text{"在"}, J) \times p(\text{"日"} | \text{"家"}, J) \times \\ &\quad p(\text{"赚"} | \text{"日"}, J) \times p(\text{"百"} | \text{"赚"}, J) \times p(\text{"万"} | \text{"百"}, J) \end{aligned}$$

垃圾短信分类问题可以总结为以下三个步骤：

- 步骤一：给短信的每个句子断句。
- 步骤二：用N-gram判断每个句子是否垃圾短信中的敏感句子。
- 步骤三：若敏感句子个数超过一定阈值，认为整个邮件是垃圾短信。

3) 分词器

在NLP中，分词的效果很大程度上影响着模型的性能，因此分词甚至可以说是最重要的工程。用N-gram可以实现一个简单的分词器（Tokenizer）。同样地，将分词理解为多分类问题： X 表示有待分词的句子， Y_i 表示该句子的一个分词方案。

$$X = \text{"我爱深度学习"}$$

$$\begin{aligned} Y_1 &= \{\text{"我"}, \text{"爱深"}, \text{"度学习"}\} \\ Y_2 &= \{\text{"我爱"}, \text{"深"}, \text{"度学"}, \text{"习"}\} \\ Y_3 &= \{\text{"我"}, \text{"爱"}, \text{"深度学习"}\} \end{aligned}$$

$$\begin{aligned} p(Y_1) &= p(\text{我})p(\text{爱深}|\text{我})p(\text{度学习}|\text{爱深}) \\ p(Y_2) &= p(\text{我爱})p(\text{深}|\text{我爱})p(\text{度学}|\text{深})p(\text{习}|\text{度学}) \\ p(Y_3) &= p(\text{我})p(\text{爱}|\text{我})p(\text{深度学习}|\text{爱}) \end{aligned}$$

三个概率中，“我爱”可能在语料库中比较常见，因此 $p(\text{爱}|\text{我})$ 会比较大，然而“我爱深”这样的组合比较少见，于是 $p(\text{爱深}|\text{我})$ 和 $p(\text{深}|\text{我爱})$ 都比较小，导致 $p(Y_3)$ 比 $p(Y_1)$ 和 $p(Y_2)$ 都大，因此第三种分词方案最佳。

这里面有个实现技巧，如果穷举所有可能的分词方法并计算每种可能下出现的概率，那么计算量是相当大的，因此，可以把它看成是一个动态规划问题，并利用为Viterbi算法快速找出最佳分词。

4) 机器翻译和语音识别

机器翻译

同一句话，可能有多种翻译方式，它们的区别仅在于单词的组合顺序，这时候使用N-gram分别计算各种情况的概率，选最大的那个即可。

“泽龙上电视了”	on Zelong appeared TV.		P=0.01
	appeared Zelong on TV.	tri-gram	P=0.02
	Zelong appeared on TV.		P=0.12

<https://blog.csdn.net/young000>

语音识别

同一种发音，可能被解析成不同的句子，然而其中有一种更符合语法规则。

ni xian zai zai gan shen me	“你西安再敢什么？”		P=0.01
	“你现在在干什么？”	tri-gram	P=0.15

<https://blog.csdn.net/young000>

针对数据稀疏问题，用数据平滑来解决。

对于没有看见的事件，我们不能认为他们发生的概率为0，因此我们从概率的总量中，分配一个很小的比例给予这些没有看见的事件。这样一来，看见的那些事件的概率总和就小于1了。在自然语言处理中，一般对出现次数超过某个阈值的词，频率不下调，只对出现次数低于这个阈值的词，频率下调，下调得到的频率总和给未出现的词。

N-gram的训练是很挑数据集的，你要训练一个问答系统，那就要用问答的语料库来训练，要训练一个金融分析系统，就要用类似于华尔街日报这样的语料库来训练。

Perplexity

信息熵是不确定性的衡量，所以可以直接用于衡量统计语言模型的好坏。因为有了上下文的条件，对于高阶的语言模型，应该用条件熵。如果再考虑到从训练语料和真实应用的文本中得到的概率函数有偏差，就要引入交叉熵的概念。

模型复杂度(Perplexity)，可以直接用来衡量语言模型的好坏。它表示在给定上下文的条件下，句子中每个位置平均可以选择的单词数量。一个模型的复杂度越小，每个位置的词就越确定，模型越好。一个语言模型的复杂度定义为： $PPL(P, Q) = 2^{H(P, Q)}$ ，或 $PP(W) = P(s_1, s_2, \dots, s_m)^{-1/m}$ ，其中W是包含m个句子的测试集， s_i 是测试集中的第i个句子，以</s>结尾，m为整个测试集中包含的所有字的个数，包含</s>，但不包含<s>。除此之外，它还可以用来衡量用交叉熵作为损失函数的生成任务的表现，比如机器翻译、语音识别和open-domain dialogue.

由于语言模型被大量用作NLP的预训练模型，所以经常用下游任务的表现来衡量语言模型的好坏，但语言模型复杂度的降低并不能保证它在下游任务的表现更好。

11.4.2 [BERT](#)