**DL常见问题总结**

1. 词向量

前导

0、one-hot缺点：1）各个词之间独立不相关，导致算法的泛化性能差；2）|V|高维度

Word embedding，称之为embedding原因，稠密空间可以理解为维度空间的一个点； 其维度可以理解为一些隐属性。

为了拟合训练数据，**word2vec在相似的词上有相近的词向量。**

**NMT/LM不需要word embedding， 因为其语料库足够大。**

1、上下文与预测词的关系

前面4个词，预测下一个词 =>  语言模型任务

**前面4个词，后面4个词，预测中间词 => 固定窗口大小版本的masked LM:** CBOW

前面一个词，预测后面一个词。

**中间一个词，预测附近(窗口)的一个词: Skip-gram。  // 也因此称为skip gram**

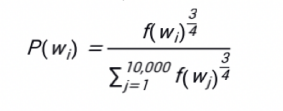
2、Skip gram

负采样技巧

1. 为何使用
2. **本身skip-gram就是构造一个监督问题(跟负采样还没关系)**，学习任务不是为了解决这个监督问题(即目标词预测其附近5-10的附近词，是预测词问题，而非二分类01问题)。 目标词-附近词 or 多个附近词-目标词。 最后都是softmax，分母是词汇表个词的加和。
3. 本质上是，**one-hot \* 词嵌入矩阵 = （词向量) 再接一个softmax单元**， 其中softmax的参数也是词向量，即N \* V 矩阵的一列。【另一个角度理解word2vec】

// softmax(UV)

1. s**oftmax操作很耗时**，1）分层级的softmax（一棵huffman树，每个节点都是二分类器）2）负采样
2. 如何采样，如何使用
   1. 修改训练任务，从预测附近词变为，预测给出词对是否是目标词的附近词，即二分类问题。从softmax模型转为10000(词汇总数)个二分类器。每个分类器只有K+1个正负样本，因此速度是极其快的，而分类器选择LR(相当于softmax的退化)。每次参数更新仅涉及K+1个词向量更新。 //from Andrew NG, 我觉得说明白了。
   2. 因为词不是均匀分布，也不能按照词的分布采样，需要对高频词进行惩罚。  //采样是**处于均匀分布和词的分布之间的分布。 //从词典中按照词频采样。【也许在window中重复】**



c) 正负样本数量，1: K  //小数据集上，K取大些，5-20，反之小些， 2-5。

d) 下采样高频词：以一定概率抛弃，比如a, of, the等词。

Word2vec

1. 理解：
   1. 构造一个浅层（输入是V维度（one-hot vector），映射层是N=300， 输出是V维度, softmax）NN，构造的任务是上下文预测目标词，或者目标词预测上下文，分别对应了CBOW，skip-gram.  //上下文的形式是怎么样的，**固定的窗口中选择**。
   2. **CBOW适合于数据集较小的情况，而Skip-Gram在大型语料中表现更好。/**/ 因此更常用的是Skip-gram，可能跟小数据集不太好选择负样本。
      1. 两个模型本质是浅层NN。  即带一个维度为N的线性映射层，比如N=300（理解1，理解2为softmax模型），即没有非线性的隐藏层。与NNLM的区别。
         1. 输入层： V维度: 1 \* V
         2. 中间层:  N.   W=V \* N
         3. 输出层：V维度. W= N \* V。
      2. CBOW: 连续的词袋模型
         1. 上下文-> 目标词，上下文的序无关，当成是个别词输入。 win\_size=2，则上下文词=4个。
         2. 多个上下文词(one-hot)作为输入，各自乘以W1(embedding矩阵), 然后直接求平均得映射层向量。
         3. 分类模型(V分类)，损失函数是cross-entropy  // 理解2则为softmax损失函数，本质是一个东西。//所以模型超级简单。
      3. Skip-Gram:  目标词-> 上下文，严格说，**目标词到窗口范围内附近的某个词。**
         1. Skip\_window=2, num\_skips=2(K), 表示在上下文=4中选择K=2个作为训练数据。
         2. 模型结构跟CBOW是镜像关系
         3. 输入是one-hot V, 输出是V维度的概率分布
      4. 因为直接对V分类进行softmax开销较大。 //当然也可以采用H-softmax形式，但是耗时上不如下面方法好。
   3. Hierarchical Softmax
      1. 输出层的优化，**利用huffman计算概率值**，时间复杂度从O(V)到logV, 二叉树，叶子节点都是单词，中间节点是参数，左右子树哪个概率大，则走哪边。  // logV个二分类器。
      2. **目标就是找到合适的所有节点的词向量和所有内部节点θ, 使训练样本达到最大似然。**
      3. 既可以用于CBOW，也可以用于skip-gram。
      4. 参数量：叶子节点词向量，以及中间节点的表征Vn。
   4. 在word2vec之前就出现了NNLM, 但是非线性隐藏层没有必要，增加训练复杂度。

**Glove**

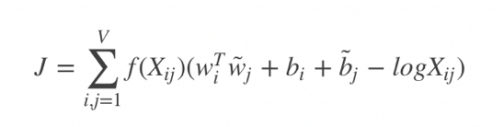
1. Word2vec，利用word周边的上下文信息； // local information
2. Glove不仅考虑local, 还考虑global info， 即co-occurence共现，词的共现频率

根据共现矩阵，认为solid跟ice, steam的关联性比较大。

是一种基于计数的模型，首先会构造一个很大的共现矩阵，就是上述代码中的co-occurrences 矩阵，其shapeshape为[vocab\_size,vocab\_size]，因此我们需要**对其进行降纬，降维后的shape 为[vocab\_size,dim]，**该矩阵的每一行的向量可以看做该单词的表示，我们可以**不断的最小化reconstruction loss来寻找这样一个矩阵。**

**构造函数去拟合共现矩阵的值Xij**，如何构建函数？涉及公式推导。

见第一个Link，讲的比较清楚



<https://blog.csdn.net/mr_tyting/article/details/80180780>

<http://www.fanyeong.com/2018/02/19/glove-in-detail/>

CBOW: 挖空预测中间词（window\_size范围内），V类别

Skip-gram: 两两组合，看是否是同一语境。因为仅仅两两组合，因此需要负样本

Fasttext: 两个工具，一个是词向量的训练(无监督模式，不考虑char n-gram就是word2vec)，另一个是文本分类(CBOW)

1. Based on word2vec
2. CBOW结构中，提出了使用character n-gram, 比如where, <wh, whe, her, ere, er> 3-gram,  + <where>, 然后将n-gram的embedding加和，可以解决了稀有/OOV词汇的问题。

此外，还增加了词粒度的n-gram，来增强词序信息，w1, w2, w3 w12, w23，加和求平均。所以其CBOW的输入是embedding的形式，不同于word2vec的one-hot形式。

1. N-gram量太大，通过hash分桶K个桶，可能不同N-gram共享embedding。
2. 文本分类任务，可以作为下游任务。 //对应于监督模式
3. Word2vec的非(自)监督模式下得到的预训练fasttext模型，可以看成是skip-gram模型，其增加了char-n-gram信息。

常见的思考问题

## **1. What are some of the traditional ways to represent words in numeric vectors?**

word2vec/Glove/fasttext/contextual word embedding，如BERT类/Elmo等

## **2. What is Word Embedding or Word2vec?**

1）传统的One-hot方法存在词之间无联系 & 维度比较大(内存占用比较大)问题，word embedding则用稠密的低维度向量表示word，可以看成是维度空间中的一个点，因此被称为word embedding。

2）word2vec是求word embedding的预训练模型的算法之一。它通过构造一个浅层NN(一个隐藏层，比如300维度)，以**语言模型**（即上下文-目标词）生成式任务为学习目标，得到的词向量。有CBOW & skip-gram模型。后者在大数据上效果较好。

## 3. What are the 2 architectures of Word2vec?

CBOW & skip-gram模型。 两者都是浅层的NN，包含一个隐藏层，输入为V维，输出为V维。两者结构上为镜像关系，但是算法不同。

Skip-gram的负采样方法是构造的是**监督学习问题，通过构建目标词-上下文词对，让模型进行二分类**。本质是一个softmax模型，所以每次更新参数，仅对正负样本的词向量进行更新， 损失函数是交叉熵损失函数/softmax损失函数。其中，每个正样本对应K个负样本。如何选择负样本，属于均匀分布-词分布之间的某种分布，倾向于高频词。**而停用词则高概率下采样**。

CBOW最后也是softmax函数，输出概率分布，但是分母是V维度加和，开销过大，所以使用h-softmax，本质是一个huffman树。

4. How to train Word2vec (Skip-gram)?

**window\_size可以选择2-4**，如果样本大，则K选择2-5，如果样本少，则K选择5-20.

## 5. What are the pros and cons for Word2vec?

优点:

预训练模型，能学习到高质量的词向量，词向量可以给下游使用；

语义相关的的词，其词向量相近；

可以进行类比。

缺点：

**没有考虑词在不同句子中不同含义。**

**不在词表中的词/低频词效果一般，只能通过UNK表示。**

## 6. What is GloVe? How is GloVe different from Word2vec?

全局词向量用于词表征。

考虑了全局词之间的**共现频率**Xij,  f(Xij) 可以平衡词与低频词。

Word2vec是predictive模型，Glove是Count-based模型。

Glove本质是对共现矩阵(V \* V)进行降维，得到V \* D（=300），即学习到每个词向量的低维度表示，这个学习过程是**共现矩阵重构问题**，

所以loss函数是**reconstruction loss，即最小化平方损失。** //降维本质是用低维表达高维数据的大部分信息(方差表示)。

两者都是预训练的词向量模型，可以直接下载使用。

两者都是基于上下文的词共现。

1. word2vec是局部上下文(窗口内)，Glove则是全局上下文。
2. 损失函数，word2vec是交叉熵损失函数。Glove是平方损失函数。
3. Word2vec是predictive模型，Glove是Count-based模型（**重构矩阵拟合的是共现概率**）。

## 7. What is fastText? How is fastText different from Word2vec?

Based on word2vec， 考虑了character的n-gram信息，解决了OOV问题；学习任务是文本分类问题。

相同点，网络结构类似，以及**H-softmax训练速度优化方法**

不同点：

1. Word2vec无监督(自监督吧)，fasttext有监督；
2. 输入不同，char&词粒度的n-gram的embedding.
3. 学习任务不同

## 8. Why choose fastText over Word2vec?

1）因为考虑ch-n-gram信息**，且n-gram信息是共享的**，**对于低频词 & 不在词表中的词，表现比较好**。

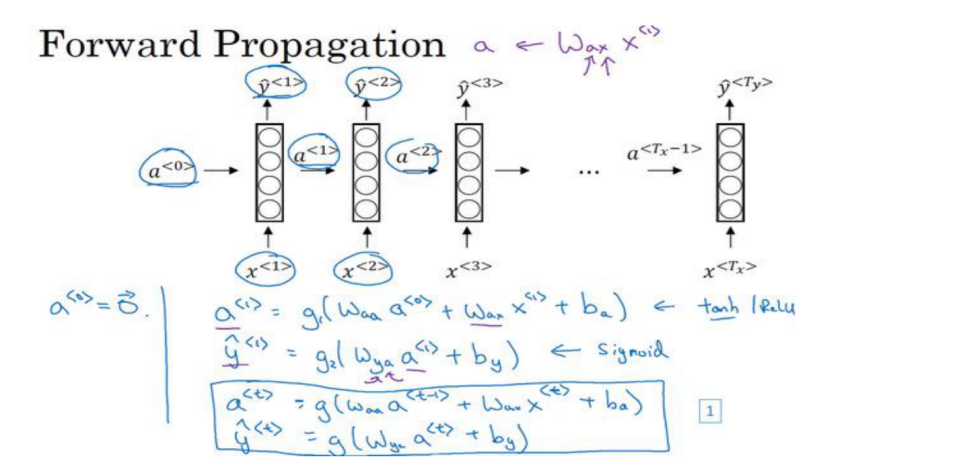
2）为何快呢？

1. **h-softmax**
2. **输出分类任务，类别相比于V维度低很多。**

## 9. How to handle Out-of-Vocabulary words?

## n-gram信息加和

1. CNN & RNN

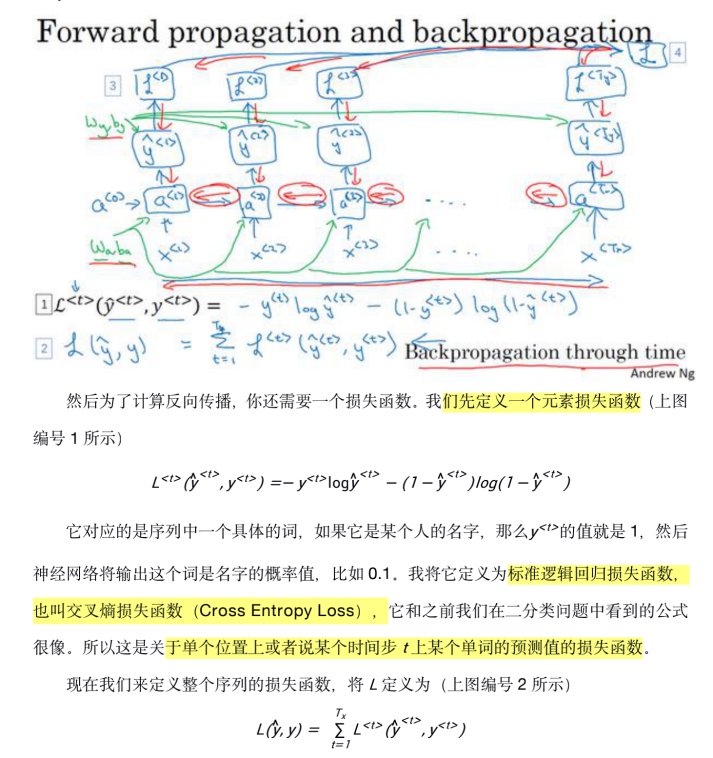


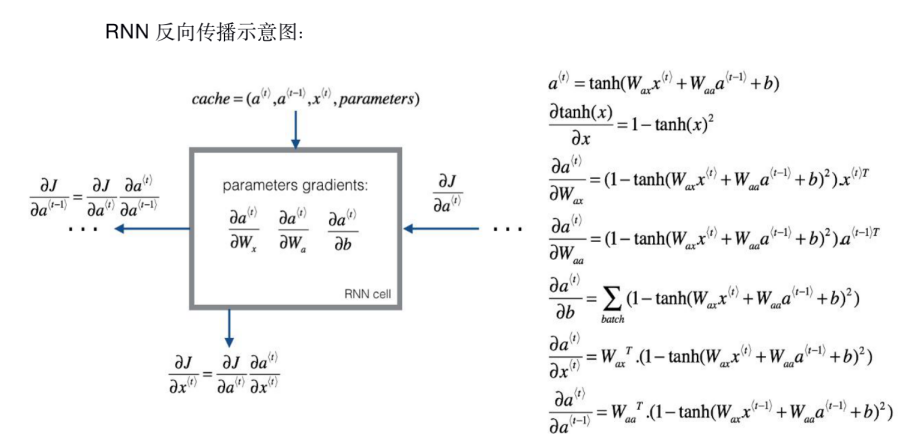
RNN:

1、初始向量a为0向量

2、每个时间步之间参数共享。

3、关于激活函数，根据输出值，如果是二分类，sigmoid函数，如果是多分类，则使用softmax函数；





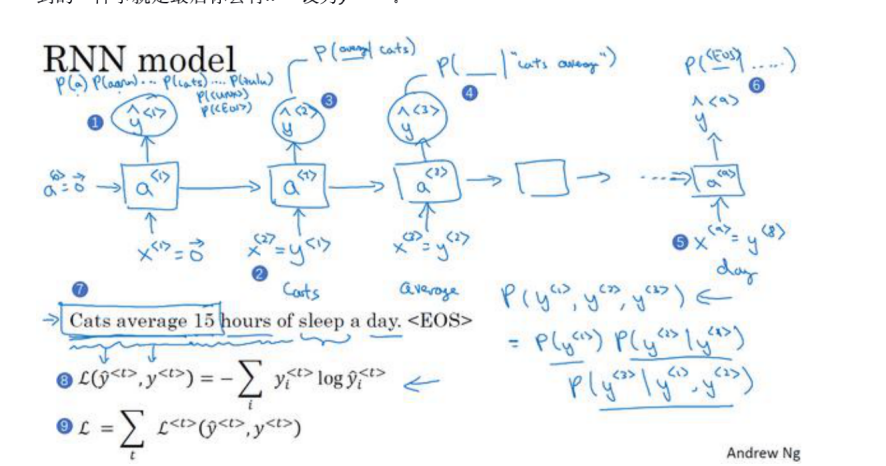
RNN的反向传播

1. 每一个样本的损失函数是各个时间步的损失函数求和;
2. 二分类的损失函数是LR损失函数(log损失函数)，也称之为交叉熵损失函数；
3. 反向传播，因为a(t-1) - > a(t)， 即a(t)是关于a(t-1)的函数，所以求a(t-1)导数，需要链式到a(t), 形成反向传播。

RNN的其他结构

1. 1对1，即标准的神经网络，中间为隐藏层，无关时间步;
2. 1对多 & 多对1, 取时间步最后一个输出;

RNN实现的LM: RNN的一个例子，GPT-2也可以实现之。



1、损失函数是softmax函数【交叉熵损失函数】，每个时间步的损失函数加和作为这个样本最终的损失函数。

2、这个句子的语言模型的概率值为每个时间步的乘积。

P(y1, y2, y3) = P(y1) (y2 | y1) \* P(y3 | y1, y2)

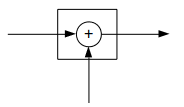
PPL = P(-1/N)

**RNN的缺点**

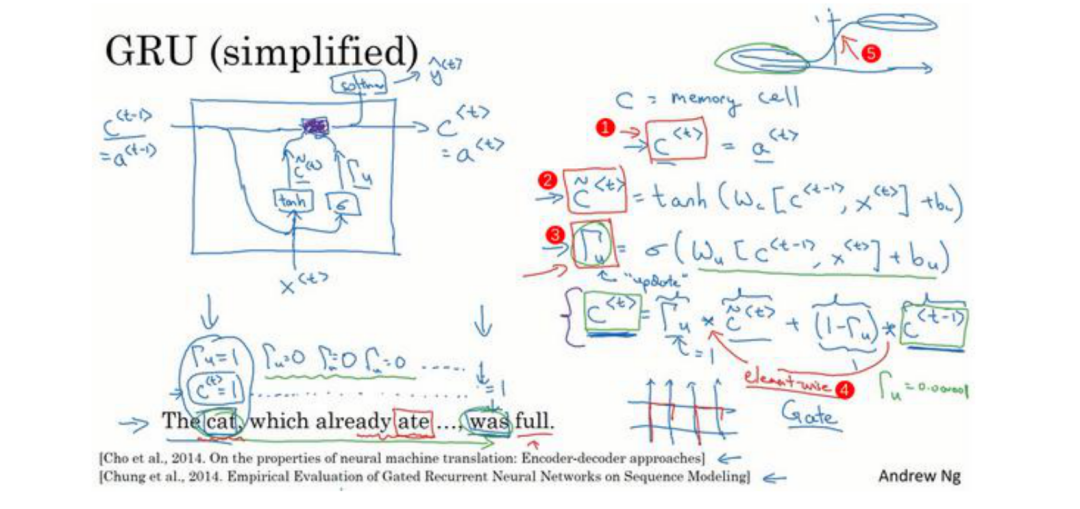
1. 因为很长时间步导致梯度消失，从而无法解决长依赖问题。
   1. 后面时间步的梯度无法传递到前面，导致无法处理长依赖问题
   2. 梯度爆炸可以通过梯度裁剪来缓解。

为啥会导致梯度消失呢？ /因为不同时间步共享矩阵W，梯度计算时，多个时间步有连乘W的现象。反向传播过程中，如果W很小，则会导致梯度消失。

RNN可以抽象为如下结构:

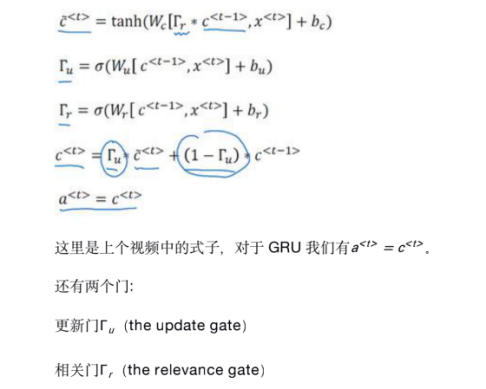


GRU & LSTM



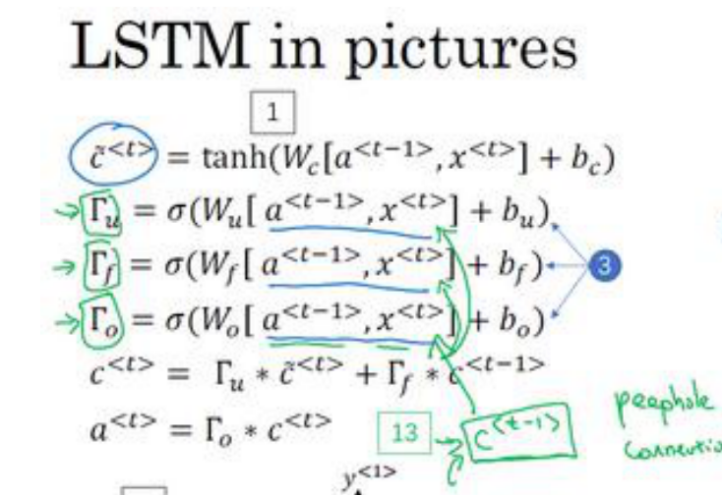
GRU门控循环单元

1. 通过增加一个门控单元，用于控制是否保留之前的信息，还是记忆新的信息，0表示记住之前信息；1表示丢弃之前信息。
2. 记忆细胞c，用于保存之前的信息 //其实相当于之前的a
3. 为啥能缓解梯度消失问题呢？
   1. 因为sigmoid函数做门控，能取值为0，此时的c(t)就等同于c(t-1), 即中间的信息不记住，比如a1等于a10，a10的更新能传导到a1。
4. 正常情况使用tanh函数，门控使用sigmoid函数。
5. 完整的GRU：多一个相关门。且两个门只是W参数不同，形式完全相同。



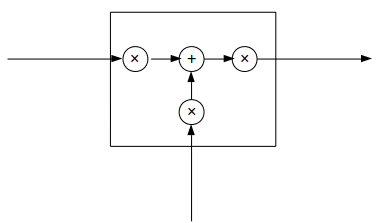
LSTM: 为什么能解决梯度消失问题？

1. 三个门，遗忘门/输入门/输出门，同样W参数不同，形式完全相同。  sigmoid(W(x(t), a(t-1)) +b)  //注意不是c，而是a，这也是不同点。
2. C的更新跟遗忘门以及输入门有关，遗忘c(t-1)，输入c(t)~ 候选c  //比GRU更加通用。
3. state a = 输出门 \* c(t) // 相当于c(t)专门是记忆单元，这与GRU, GRU没有输出门。



注意：1）候选c的计算使用tanh，其他使用sigmoid； 2）a和c不同，c是记忆细胞，而a是hidden state。

LSTM可以抽象为如下结构:



**CNN**

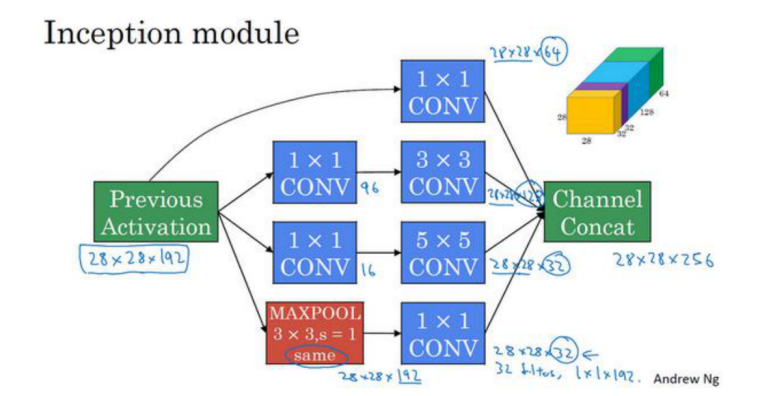
1. 卷积操作
   1. 过滤器/滤波器/filter/卷积核，卷积其实是元素乘积加和。
      1. 不同滤波器可以表示不同边缘的检测
   2. 重要超参数
      1. 核大小，3 \* 3 ，5 \*5  //为何是奇数，因为1）避免非对称的padding 2）缺少中心。
         1. 如果图片是三维的，则是filter也是三维，3 \* 3 \* 3  //最后一个通道需要匹配。  同样是元素积求和。
         2. 如果是文本的，是二维，第二个维度是embedding的大小，第二个维度可以理解为通道。
      2. 卷积核个数
      3. 步长，padding //这两个一般不变
   3. 为何需要padding?  分别对应valid卷积，same 卷积(同等维度输出)
      1. 图像被不断缩小
      2. 图像的边缘信息会被忽略
   4. 激活值是在做卷积之后，加上ReLU，形成的激活值，继续添加后续层。
   5. CNN的参数是卷积核的大小，参数共享，参数减少，避免过拟合。
   6. 池化层作用: 缩小模型大小，提升计算速度，减少冗余信息，提高泛化能力。特点是保持某种不变性。
   7. **CNN两个优势: 避免过拟合**
      1. 权重共享: 整个图片/矩阵共用同一个特征探测器(filter)  //参数共享
      2. 稀疏连接: 卷积抽取的每个特征值是每个filter跟图片的不同部分进行卷积。 比如左上角9个值。
   8. CNN擅长捕捉平移不变性，这个是Pooling贡献的。
2. Res-Net
3. 保持效率: 残差连接的block学习恒等变换， H(X) = X (即新增的层数为0),很容易，即最差的效果也能达到没有新增层的NN(简单点的NN)。
4. 还能学习其他映射的能力。
5. 缓解网络退化，也就是梯度消失问题。

**3、1 \* 1的卷积核**

1. 多个 1\* 1 卷积核，相当于输入做了一次全连接； //**改局部连接为全连接。**
   1. 每个1 \* 1卷积核可以理解为隐层的一个节点，多个理解为隐层这层。
2. **为NN添加非线性函数(ReLU)，减少或者保持，甚至可以提高通道维度**。比如Intception网络。
   1. 减少通道数，可以大大减少计算量。

4、 Inception: 多个不同尺寸的卷积核共同作用。

1. 不需要人为确定使用何种大小filter/是否需要池化，Inception会自行确定这些参数。 //直接堆叠在一起。
2. 多个inception module组成了Inception网络。



1. 迁移学习
   1. 使用pre-trained Model: 视训练样本数而定，此为一般原则，但是也可以都试试。 如果数据较少时，将最后的softmax层替换为自己的softmax层，比如三分类，然后freeze网络的参数，只训练softmax层的参数。
   2. 如果数据较多时，将低层的参数frozen, 将后面几层可训练。 数据越多，freeze的层越少。
   3. 如果数据足够大，将pre-trained model的weight作为初始化，直接重训整个网络。

LeNet/GoogleNet/AlexNet/ResNet/Inception

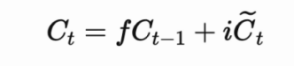
**常见问答**

——————————————————————————

1. 为什么LSTM能解决梯度消失/爆炸问题？【需要再看看，看下百面机器学习】

我的想法:  LSTM有一个记忆单元c, 每个使用候选cell的值(tanh, ct-1,xt-1)  以及上一个cell状态，分别用输入门和遗忘门门控制，如果中间有很多步，遗忘门倾向于1，则之前的信息则被记录下来，a10相当于a1，反向传播也能传递到a1处。 // 即由门可以控制中间段取值为0。

其他解释: <https://weberna.github.io/blog/2017/11/15/LSTM-Vanishing-Gradients.html>

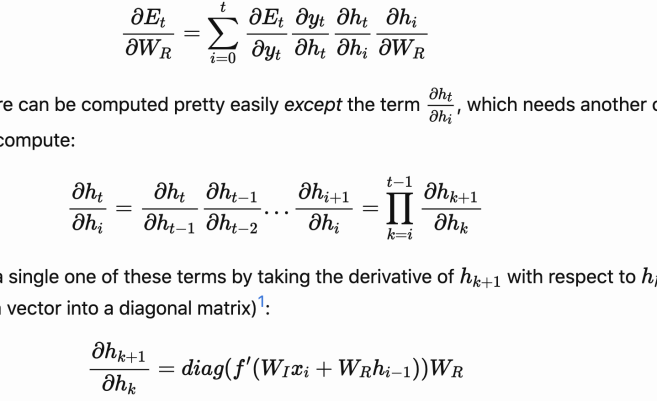
//这就是LSTM的核心  
当前时刻记忆状态对前一时刻记忆状态求偏导的时候，有一项是f遗忘门，**可以通过控制遗忘门，缓解梯度消失**。//公式推导，其实证明了第一点。

1. RNN发生梯度消失的原因是什么？

激活函数是tanh的导数f(z)' = 1 − f(z)f(z)

sigmoid导数f(x) = f(x)(1- f(x))

1. **因为RNN，存在多个时间步的递归，应用链式法则，对参数W求导时，存在多个w的乘积，且激活函数是tanh，其导数是1 − (f(z))2 < 1，所以当w< 1时，梯度消失，W过大，则会梯度爆炸。**



1. GRU与LSTM的区别是什么？

我的想法：GRU只有两个门，分别是更新门，相关门。 通过更新门决定是否遗忘过去的信息，要么遗忘旧信息，记录新信息；要么保留旧信息，丢弃信息。而对于新旧信息的处理，LSTM更加灵活些，通过遗忘和输入门，更好控制新旧信息。

LSTM有三个门，遗忘门和输入门，输出门，

1）对新旧信息的控制上，LSTM比GRU更灵活；

2）LSTM多一个输出门，相当于LSTM专门有记忆单元，而GRU，hidden state即为记忆单元两者功能等同。

3）对于GRU相关门的解释**，提取记忆单元与当前信息的相关部分**，**用于更新候选记忆单元。**

4、RNN中使用ReLU可以解决梯度消失问题吗？

**不可以，因为RNN梯度消失的本质是相同矩阵W(不同时间步共享矩阵)的连乘，而虽然激活函数的导数换成ReLU后始终为1，但是当W很小时候，还是会梯度消失。**

而CNN中是不同的W，随着深度，W1,W2,W3. 因此W之间会相互抵消，而更多受到激活函数影响。

5、解决梯度消失和梯度爆炸有哪些方法？

梯度消失:

1. 激活函数上，选择ReLU/ReLU的变体
2. 网络结构角度上，序列模型可以考虑LSTM/GRU；
3. ResNet残差连接，中间层等价于恒等映射。// Highway，也是连接方式
4. BN

梯度爆炸：还可以是梯度裁剪。

1. Overfitting怎么解决
   1. 数据增强, 增加数据多样性
   2. L1, L2正则化
   3. Dropout
   4. 模型融合, 比如Bagging 和其他集成方法
   5. BN ,batch normalization
   6. Early Stopping(提前终止训练)
   7. 加入噪声
   8. 特征选择(ML)
   9. 模型简单化
2. BERT类

<https://zhuanlan.zhihu.com/p/49271699>

1、Transformer

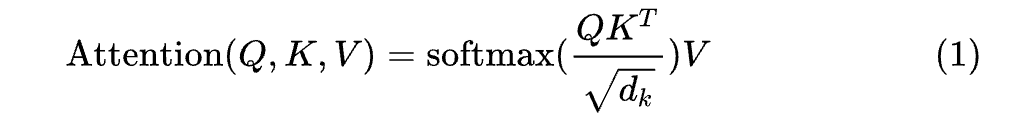
1. 如何做到并行性
   1. self-Attention计算的矩阵操作  //看看下面QKV，都是所有词的。
   2. Multi-head
   3. encoder的多层之间是串行（深度）， RNN不仅深度/广度都做不到并行。
   4. predict是串行，需要预测前一个词，才能计算下一个词QKV； 但是training时候因为是**teach-forcing**且mask self-attention（即后面为0），**因此可以并行计算多个时间步的attention值。**

Attention is all you need

1. RNN的局限性在于1）时间步T依赖T-1步，导致无法并行，RNN的训练速度比较慢；2）长依赖问题的局限性
2. Transformer是为了解决以上两个问题。

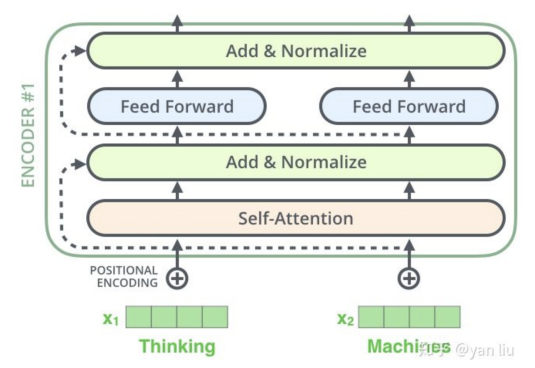
Transformer:

1. Encoder2Decoder结构
   1. Attention的公式： 本质还是一个dot-prodcut attention计算方式



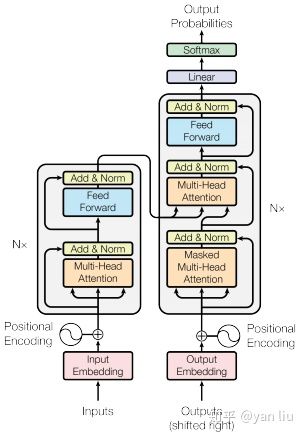
注意: Q=W1 X;  K= W2 X;  V=W3 X

7CE014C0-8A64-43CB-ACAC-91285B4A75B1//激活函数是ReLU

//每个token都有一个feedforward 网络(线性函数和非线性激活函数，参数为W1，W2，其实是矩阵操作)。

* 1. Multi-Head Attention
     1. 与CNN中多个卷积核类似，Multi-Head Attention也是利用多个QKV， 不同矩阵，生成多个Z，**拼接**通过矩阵映射，转换为需要的结果。
  2. Encoder-Decoder Attention
     1. Q=解码器上一个解码输出，K=V=编码器的所有输出。（上游看不到，所以也看不到）

注: 解码器中的self-attention 称之为Masked self-attention，因为解码器step by step过程中，只能看到前面的decode结果，看不到后面的。



* 1. 位置编码
     1. 没有位置编码，Transformer则对位置不敏感，只是功能强大的词袋模型而已。
     2. 位置编码计算公式：1）sin/cos，跟单词位置、单词的维度有关；即直接可以计算出该位置的pos embedding。  //没有训练参数，**不更新， 绝对位置编码；2）相对位置编码，即随机初始化每个固定位置的positional embedding.**
     3. 最终的embedding是word embedding + pos embedding

Transformer优点

1. 并行性好
2. 很好解决了长依赖问题
3. **模型结构设计比较创新，完全抛弃RNN/CNN，仅用attention机制。**

缺点

1. pos embedding 对于单词位置的考虑略弱（相对于RNN这样序列模型)
2. Transformer擅长捕捉全局信息，对于局部信息关注不够，可能需要CNN结合。//？
   1. Transformer的encoder-decoder结构，天然适合机器翻译模型
      1. 其decoder 适合自然语言生成类任务
         1. 但是如果将decoder用于LM任务预训练之后(比如GPT)， 则既可以用作生成类，也可以用于分类等
      2. 其encoder 适合分类/序列标注等词/句子级别的NLP任务
         1. bert可以用于生成类任务吗？ // 不太合适

**关于Transformer中的激活函数：self-attention中的softmax函数，FFNN中的ReLU。**

**decoder的输出接linear线性层 + softmax**

References:

<http://jalammar.github.io/illustrated-transformer/>

**关于BERT** with code // Huggingface

双向Encoder Representation from Transformer

1. 双向Transformer Encoder；对比
2. MLM(15%=80% + 10 % + 10%，担心finetune跟pre-trained差距) + Next sentence prediction
3. BooksCorpus & 英文wikipedia = 33亿单词

batch\_size =  16/32

学习率: 5∗10−5，3∗10−5，2∗10−5

Number of epochs: 3, 4

WordPieces: 而非word

1. ERNIE使用的是单词/实体， 而非单字

两个任务

1. Masked LM(cloze task)
2. NSP: 为了让bert理解两句话的关系

Bert Embedding

1. 最后四层的hidden state效果最好

**关于BERT综述的有意思的结论**

1. BERT在句法方面，对句法结构信息掌握比较弱，能做好POS/句块/SVA等任务，但是无法理解否定以及对混乱输入不敏感；
2. BERT在数字表征比较困难，无法基于通用知识做推理。
3. 大部分attention heads没有编码更多信息，只有小一半的heads有不同结构的pattern。比如垂直pattern(关注CLS, SEP, 标点)、对角pattern(关注自身)，两者结合、关注句法关系（比如主谓宾，修饰词）
   1. 所以调整heads超参效果没有layers个数好
4. BERT低层有更多线性词序信息；中层有句法信息；高层则是面向任务。语法则是贯穿所有层。
   1. 中层更适合迁移到下游任务中； fine-tune时候，高层参数可以设置可变，低层frozen。

5）与下游任务结合时，可以考虑BERT多层的加权和。

References:

<https://jalammar.github.io/illustrated-bert/>

GPT-2

自回归模型

1. **每个新生成的token都将用于下次的input （自回归形式）**

Positional encoding: 是1024个位置，学习到的**pos embedding。 //绝对位置的embedding**

这点上跟BERT一致。//两种位置编码的效果相差无几。

GPT2使用Transformer的decoder时，只有masked multi-self attention + FFNN, 并没有与encoder交互的self attention.

GPT2的输入tokens个数限定在4K，比Transformer的512大得多。

**GPT/GPT2/GPT3区别**

GPT-3

1. **只是放大版的GPT-2，模型比GPT-2大100倍，训练数据比GPT-2大十倍，所以能做到用更少的领域数据、不需要finetune去解决问题，**打破了预训练 + finetune两步走的预训练模型的框架。
2. GPT-3**在一些任务上相对弱（比如涉及两个句子比较**，一个句子是否是另一个句子的paraphrase），主要还是因为GPT-3是一个自回归语言模型，从GPT/GPT-2/GPT3，OpenAI一路执着地遵循单向语言模型训练，虽然天然符合LM的特性，但是同时也丢失一些上下文信息。
3. 虽然各路媒体各种吹捧，理性思考，**但是跟GPT-2其实还是无法真正深度理解文本背后语义，还是简单大量剪切粘贴，以及浅层语义理解，无法做到推理。**

**常见问题**

Transformer题目

1、Attention公式以及其中的Q, K ,V分别指什么？

softmax(QKt/dk开根号)V

Q: query查询词； K: Key关注词的索引，Value为关注词的内容，将词向量通过三个不同矩阵得到不同方式的词表征的向量，作为QKV。 将embedding的维度转变为d维度，加上序列长度n， **QKV的矩阵大小为=n \* d**

2、多注意头的好处？

抓取不同pattern特征，类似多个卷积核，使得表征更加充分。

3、Attention中self-attention的时间复杂度？  //这个问题真好

O(n2 \* d)

Q=K=V= n \* d矩阵。  可以理解为每个词的维度从词向量维度转为了d维度。

点积操作，QK计算时间复杂度为 n \* d \* n = O(n2\* d）,得到n \* n矩阵

softmax操作，时间复杂度为O(n2)

最后.V操作，同样是 n \* n \* d = O(n2 \* d)

因此，最终是O(n2 \* d)

1. 为啥需要根号缩放？ //防止点积过大

其实问了两个问题，

1. 为什么要缩放？

点积过大，softmax函数会push到梯度很小的区域(可以理解为one-hot形式)。

为何如此？

softmax中的指数函数，会将自变量x之间的距离拉大，将x较大时，几乎softmax后的值就接近于1，其他分量则为0。// 在数量级较大时，softmax将几乎全部的概率分布都分配给了最大值对应的标签。

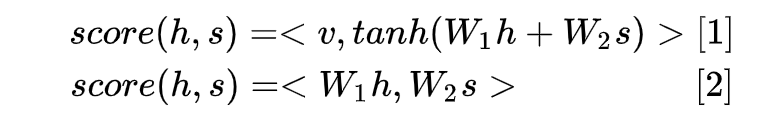
即产生了one-hot的向量输出分布。在这种情况下, softmax的梯度值几乎为0，参数更新缓慢，影响模型收敛。

1. 为何采用dk的根号缩放?

假设q, k的每个分量均服从0-1标准正态分布，则点积q.k的分量则服从0-1标准正态分布，求分量求和后的q,k 服从0，dk的正态分布，即正态分布的宽度越大，则更高概率取得较大值，为了控制第一个小问题，把方差稳定到1，做法是将点积除以dk的根号，这样方差则变成了1.

1. 扩展问题: 为何其他attention的softmax不需要呢？

主要的attention为加性attention 和点积attention



加性attention有tanh控制其分布在-1，1之间。而最后一层softmax不需要控制点积过大，是因为最后一层不存在因梯度过小而导致的梯度累积而梯度消失问题等。

5、为何self-attention在长距离序列中如此强大？

因为每个词都能关注到任意距离的词，且有多注意头，关注不同部分。

6、Transformer中encoder和decoder的异同点

相同：都有multi-head self-attention机制，FFNN，ADD & Norm以及层数相同。

不同点：

1. decoder使用了masked self-attention，即后面的单词是不可见的。
2. decoder多一个self-attention计算，q是当前词，k,v是encoder的输出

BERT系列

1、BERT和GPT的异同点

相同点: 都是基于Transformer的预训练模型，训练数据非常大。

不同点:

1. BERT是自编码语言模型，GPT是自回归语言模型。
2. BERT是Transformer的encoder, GPT是decoder。
3. BERT是双向的，GPT是单向自回归的。
4. BERT预训练任务是MLM，自编码语言模型；GPT是自回归语言模型
5. BERT模型中绝对位置embedding和相对位置embedding怎么理解，各自优缺点和使用场景。

绝对位置编码，就是随机初始化位置embedding，然后训练学习；//简单直接，但是无法学习到位置的相对关系，2和1比10和1更近。

相对位置编码，则sin/cos函数直接计算(维度以及位置)，固定不变。//可以学习到位置相对关系，不会引入额外训练参数。 缺点: 计算复杂?1

3、BERT预训练任务有哪些？各自作用是什么？

两个预训练任务，一个token-level的，MLM任务，理解上下文词的embedding信息。

另一个是NSP, 属于句子级别，理解句子级别信息。

4、RoBERTa和AlBERT分别做了哪些修改？

RoBERTa主要4点：1）**训练数据增加了10倍；**2）去掉NSP，增加FULL-SENTENCES机制，句子最大512，跨doc了则使用标识符；3）动态mask机制，不同epoch同一句话mask的位置不同；4）编码方式：RoBERTa使用的是BPE。

ALBERTa主要三点：1）改NSP为SOP，即二分类，两个句子是否调换了顺序  2）不同层之间权重共享，减少了参数；3）embedding矩阵降维。

5、XLNet如何实现permutation LM?

通过Attention掩码，从X的输入单词里面，也就是Ti的上文和下文单词中，随机选择i-1个，放到Ti的上文位置中，把其它单词的输入通过Attention掩码隐藏掉，于是就能够达成我们期望的目标（当然这个所谓放到Ti的上文位置，只是一种形象的说法，其实在内部，就是通过Attention Mask，把其它没有被选到的单词Mask掉，不让它们在预测单词Ti的时候发生作用，如此而已。看着就类似于把这些被选中的单词放到了上文Context\_before的位置了）

是融合了上下文的自回归模型。

6、Attention对比CNN,RNN，分别有哪些点你觉得的优势

1、对于CNN而言， self-attention对于长依赖的解决能力很强，这点是CNN的弱点。

2、对比RNN，Attention提高并行性。

1. BPE和wordpiece之间的区别: 词表综述确定。

BPE选择频数最大的字节对。

wordpiece选择语言模型概率最大的字节对。即wordpiece会训练一个语言模型。

1. 有哪些attention计算？

加性attention, 点积、concat、triliner(Wtanh(q, c q元素积c))

**基础系列1**

1、为何必须在神经网络中引入非线性？

**为了提供网络非线性建模能力，拟合更复杂的函数**；如果不引入的话，那么网络只能拟合线性映射，那么加深网络没有任何意义，W1,W2,W3 X = WX，相当于单层神经网络。

2、ReLU在零点不可导，那么反向传播中怎么处理？

按照零点处导数=0处理。

3、ReLU优缺点

优点:

1. 缓解梯度消失、爆炸问题
2. 求导计算快速
3. 加速网络训练

缺点:

1. 负数恒为0，导致一定神经元激活失效。
2. 输出不是以0为中心。//影响收敛速度。

4、BN解决了什么问题 ? // 以及BN能不能解决梯度消失问题? //直接看后面

6、激活函数有什么作用，常用的激活函数有哪些？

sigmoid：优点0-1概率值；缺点：1）饱和性，两边导数趋于0，导致梯度很小，更新很慢。2）偏移现象，其输出均值不是0。

Tanh: 输出均值为0，**收敛速度快**。缺点是饱和性。

ReLU: 无饱和性缺点，缓解梯度消失；缺点：1）**<0的神经元死亡**；2）均值非0 的偏移现象，影响网络的收敛速度。

Leaky-ReLU

softmax: 和为1的概率分布；大值的饱和性。

7、怎么解决梯度消失问题。

1. 网络结构LSTM /GRU
2. ResNet/highway等残差连接网络结构
3. ReLU等激活函数
4. NN深度不要太深。//
5. BN

8、什么是端到端学习

不需要手动选择特征，输入，经过NN，输出。

9、softmax的原理是什么？有什么作用？

归一化操作，作用是形成概率分布，满足概率定义。

10、logistic 回归公式是什么？

sigmoid(wx+b)

11、pooling层是做什么？

保持某种不变形，比如旋转/平移等。

去掉冗余信息

降低维度。

12、dropout是否用在测试集上

否，因为在训练的时候，已经做了scale，正向反向传播，都需要/keep\_prob，以保证计算下一层的**激活值**期望不变。

13、列举几个梯度下降的方法

**批量、随机梯度下降/mini-batch梯度下降**

牛顿法、拟牛顿法、共轭矩阵

**基础系列2**

1、非线性激活函数

什么时候？隐层最好都使用非线性激活函数，不然隐层没有深度可言，完全可以合并**，而输出层可以使用线性激活函数。//即输出层不使用激活函数，比如transformer最后就是linear + softmax**

2、随机初始化

神经网络w不能用0初始化, b可以为0。 //为什么呢？

1. 因为如果w设置为0，隐藏层的多个隐藏单元因为是计算同样的函数，导致取值相同，反向传播的值也一样，多次迭代都相同，**相当于一个单元的作用**。等价于一个隐藏单元。  而b=0，则不会这样。
2. W随机初始化为较小的值，b可以初始化为0.
   1. **随机初始化为小的值的原因是，如果W很大，导致激活值停留下sigmoid平坦之处，此处的导数很小，导致参数更新的很慢。**

3、正则化

解决过拟合(高方差)的两个手段: 1）正则化；2）更多训练数据

L1正则(绝对值的和)与L2正则(w的平方和)

人们在训练网络时，越来越倾向于使用 L2 正则化。 //为什么呢？

L2正则使得w部分接近于0，部分网络节点失效，减少了网络的复杂度，不容易过拟合。L1绝大部分w为0，更适合特征选择。

3.1 dropout正则

* 1. Dropout在训练阶段以一定概率失活神经元，比如Keep-prob=0.8，则dropout概率为0.2，当前层计算出来的**激活值需要/0.8**， 弥补因为失活结点导致的期望减少，**使得激活值的整体期望不变**。//每个样本(batch\_size)都以一定概率失活网络中的结点，**即每个batch\_size都会学到小规模的NN，防止过拟合。【包含数据的dropout以及网络结构中节点的dropout，因为输入节点也是网络节点。】**
  2. inference时候，则dropout=1，即所有结点参与计算。
  3. dropout**可以产生收缩权重的效果，跟L2正则化类似。 //可以理解为因为结点不确定性，把权重分散在不同w上。**
  4. 不同层(输入层+隐藏层) 可以设置不同的keep-prob，**因为每层的参数个数不同，有些层容易过拟合。**

其他正则化方法

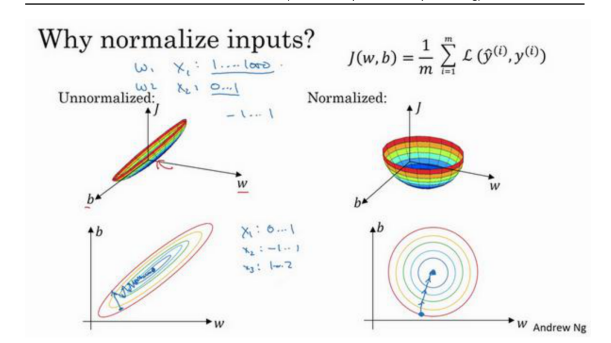
1. 数据增强
2. Early stop
3. BN

4、归一化& Batch Norm

每个特征用均值 & 方差归一化。 //严格上说是标准化，即Norm

为什么需要归一化？

1. 因为特征值间如果相差很大，则参数间相差很大， 那么损失函数曲线，则非常狭长，不够圆滑，这样梯度下降则更加困难，学习率得设置的很小。**反之，归一化后，则可以收敛得更快，加速训练。**
2. 另一个作用: 尤其从神经网络后层之一的角度而言，前层不会左右移动的那么多，因为它们被同样的均值和方差所限制，所以，这会使得后层的学习工作变得更容易些。



**BN发生在激活函数之前，即wx + b之后进行BN应用。 //防止进入激活函数饱和区。**

5、梯度消失与暴涨

随着层数加深，因链式法则，当梯度值过大/过小时，这种影响被放大，比如导致梯度非常小时，后面的梯度都传不到前面几层，称之为梯度消失问题。

6、权重初始化

随机某个很小的数，跟当前层隐结点个数有关。

7、batch\_size影响

batch\_size大好处:  1) 并行高，GPU利用率大，训练加速，收敛更快。 2）batch\_size越大，梯度下降方向越准，引起的训练震荡越小。

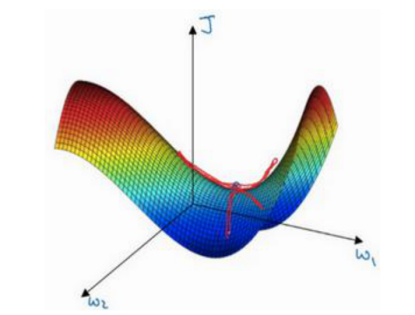
batch\_size大坏处: 1）**容易陷入局部最优，从而导致泛化性不好**。(因为多个batch\_size的下降方向基本变化不大，而小batch size可以理解为加了噪声(防止过拟合)，能走出局部最优) 。

8、学习率衰减

先大后小，随着epoch数减小。前期学习快点，后期慢慢逼近极小值。

9、局部最优问题

如果网络很大，参数很多，损失函数被定义在一个高维空间，那么很可能不太会发生局部最优情况，**因为想让所有参数往同一方向可能性不大**，更可能是达到鞍点，也就是会遇到一段平滑区，怎么走出平滑区，可以借助其他优化算法，比如**Adam**等。



10、偏差与方差

前者反应训练误差，后者反应泛化性，即在测试集上的误差。

如果算法存在高偏差，那么准备更多训练数据有用不？ // 没有用，高偏差说明模型学习能力不够，可以用更大的模型，高方差才需要更多训练数据。

11、softmax 回归

对应LR，其是多分类，是LR多分类的扩展。

1、交叉熵，又叫相对熵。

softmax损失函数/log损失函数/交叉熵的关系

LR中log损失函数是二分类的情况下， L(w) = -ylogf(x)-(1-y)log(1-f(x))

softmax损失函数就是交叉熵函数，适用于多分类情况，当然也包括了二分类情况

为啥选用交叉熵，而不是平方损失函数？

**平方损失函数的导数，跟f’(x)有关，对于sigmoid函数而言，其导数大部分时间都接近于0，导致梯度下降更新参数的时候，会变得非常慢；**

**交叉熵，没有这个问题，因为其导数跟误差有关，所以误差越大，更新的幅度越大。**

**交叉熵本身特性，当输出与label比较接近时，loss=0。**

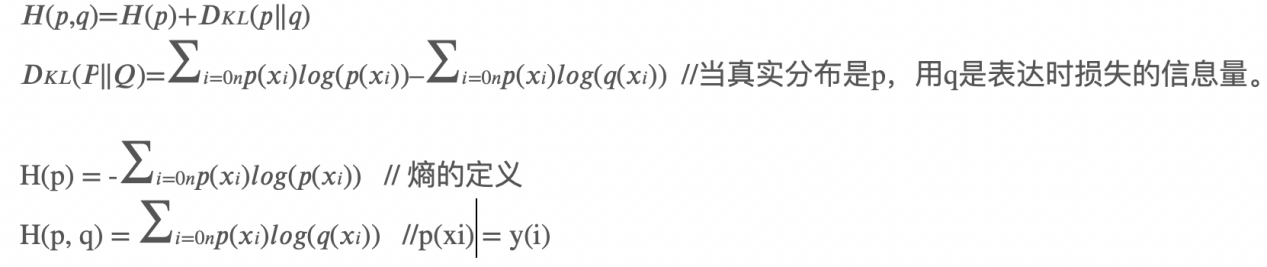
2、交叉熵与KL散度

熵的定义: H(X) = - sum(P(x) \* logP(x)) //关于logP(x)的期望。

H(X) ~ [0, logN]   /**/暗示了编码整个信息(概率分布)所需要的最小编码位数**，比如8个，需要3维。

交叉熵H(P, Q): 衡量了两个概率分布之间的距离，其两分布的差异来自KL散度表达的差异。

KL散度，也是描述两个概率分布的差异。



所以两者的关系是，**交叉熵= 真实分布p + 两个分布p,q之间的离散度**(因为熵计算有负号)，由于真实分布保持不变，所以优化交叉熵，也是优化KL离散度，优化方向完全一样，只是取值不同，交叉熵也大些。从公式角度看，KL散度多一项，多出的一项是不变的值，所以分类模型直接用交叉熵即可。

**其他预训练模型**

**RoBERTa**: BERT的加强版，from facebook & 华盛顿大学，可以理解为BERT++模型。

1. **训练语料增加了10倍**，原BERT有16G，包括BookCorpus & Eng-wiki; 此外，RoBERTa还包括News, Stories等。//注意，模型结构没有差异。
2. 预训练任务上
   1. RoBERTa是动态mask，即句子在模型进行训练前才随机mask
      1. 静态mask，先mask好，然后输入不同epoch用，即不同epoch可能用同一份（相同句子相同mask位置）数据。
      2. 动态mask， 每个epoch时候，再对样本进行随机mask
   2. 取消NSP任务，**增加FULL-SENTENCES机制**
      1. 比如最大长度=512，即选择连续句子，使之填满512，如果跨文章了，则加上特殊分隔符。
3. 其他差别
   1. 超参数： batch\_size更大=8K
   2. **编码方式：RoBERTa使用的是BPE，BERT使用wordpiece，GPT使用BPE。**

**ALBERT：Google**

更轻量级的BERT，参数变小，内存空间变小，效果更好。 // 参数个数少了9倍

Factorized Embedding Parameterization。  V \* E 词表维度太大了，这**个矩阵经过因式分解，可以得到更小的参数。**

跨层的参数共享：不同层之间可以共享参数。 比BERT-LARGE小18倍。 //每个层的参数一样。

改NSP为SOP，即二分类，**两个句子是否调换了顺序。**

**DistillBERT**

知识蒸馏:

1. Loss分为distill\_loss,  train\_loss，前者是学生预测输出分布拟合老师的分布。后者是具体任务分布，比如BERT上的MLM。
2. softmax-temperature， 通过控制T使得输出分布更平滑，增强泛化性

Triple loss

1. 作者发现增加cosine embedding loss**能使得学生和老师的hidden state vector的方向上对齐** //BERT contextual embedding输出的cosine作为loss.

Student model

1. The token-type embeddings  and the pooler  are removed
2. The number of layers is reduced by a factor of 2
3. Init student from the teacher by taking one layer out of two.
4. Gradient accumulation(up to 4K per batch) ，dynamic masking，without NSP objective

效果：97% 效果，60% faster, 40% size

**DistillRoBERTa**

1. It follows the same training procedure as [DistilBERT](https://huggingface.co/distilbert-base-uncased).
2. The model has 6 layers, 768 dimension and 12 heads, totalizing 82M parameters (compared to 125M parameters for RoBERTa-base). On average DistilRoBERTa is twice as fast as Roberta-base。
3. RoBERTa: DistilRoBERTa reaches 95% of RoBERTa-base's performance on GLUE while being twice faster and **35% smaller.**

**XLNet**

自回归语言模型

1. 从左向右预测/或者从右向左预测，比如GPT，而ELMO本质是两个自回归的语言模型。
2. 优点: 天然适合生成类任务；缺点：上下文只能利用一侧的信息。

自编码语言模型

1. AutoEncoder LM。即在输入侧通过**mask一些单词，加入噪声**。比如BERT。即DAE(Denoising Autoencoder)
2. 缺点pre-train跟finetune阶段看到的的不一致，finetune阶段没有mask标记。

XLNet虽然也是从左到右预测，但是，context\_before已经融合了context\_after的内容，怎么做的呢？

1. Permutation Language Model 作为训练任务。 13 => 24
2. 在Transformer内部，通过Attention掩码，从X的输入单词里面，也就是Ti的上文和下文单词中，随机选择i-1个，放到Ti的上文位置中，把其它单词的输入通过Attention掩码隐藏掉，于是就能够达成我们期望的目标（当然这个所谓放到Ti的上文位置，只是一种形象的说法，其实在内部，就是通过Attention Mask，把其它没有被选到的单词Mask掉，不让它们在预测单词Ti的时候发生作用，如此而已。看着就类似于把这些被选中的单词放到了上文Context\_before的位置了）
3. **本质是融合了上下文的自回归语言模型: 在生成类问题上比较nice**
4. 引入了Transformer-XL的主要思路：**相对位置编码以及分段RNN机制**。实践已经证明这两点对于长文档任务是很有帮助的。
   1. **解决超长序列的依赖问题**。比如1000长度分10段，首先取第一个段进行计算，然后把得到的结果的**隐藏层的值进行缓存**，第二个段计算的过程中，把缓存的值拼接起来再进行计算。
   2. 引入相对位置编码，用变量表示是否同一段中的i, j。

**ELECTRA**: 斯坦福博士生 & Google Brain

生成模型用于生成任意样本替换一些tokens， 判别模型用于判断each token是否是被替换的。

Replace token detection Task比MLM 更好，因为定义在所有的输入token上，而不是仅在mask掉的token。

为啥这个好呢？因为训练的数据更多。

1. These masked language modeling (MLM) approaches incur a substantial compute cost because the network only learns from 15% of the tokens per example.

解决了BERT pre-train与fine-tune不一致问题

而MLM 则可以看成是generator，去生成MASK的词

RTD本质上是训练一个判别器。

对于每个位置，generator会根据softmax生成特定的词（可能跟原词一致，或者不一致），是否一致交给discrminator（也是softmax分类）

1. generator看成是MLM， 首先类似BERT， mask 15%, 然后generator预测出词，代替MASK标记。作为输入。
   1. generator的训练目标是最大似然
2. 因为最后loss是两者相加，所以是Jointly learning在pre-train阶段。
3. fine-tune阶段，则是直接在discriminator上进行fine-tune

High light

ELECTRA: Replace Tokens and then Detect it。  // 看成是分类任务

而BERT目标是MLM，随机MASK一些词，然后生成他们；  // 生成任务

因此，有两点主要不同。这两大点不同带来的一些优缺点如下：

ELECTRA优点

1. 判别器的难度提升，从而学习更好的表示
   1. 由Replace Tokens 阶段带来的福利。
      1. 相对于Random select and Replace tokens,  使用generator的优势在于generator会先做一遍过滤，过滤掉简单好预测的词，把难token交给判别器，从而让判别器学习到更好的表示。
2. 效率提升
   1. Detect it阶段的福利
   2. 计算效率提升
      1. 相对于BERT 生成V词表大小类别，二分类更快。
   3. 判别器对所有的token进行预测而非仅MASK词预测(BERT)，在信息利用上更好（因为可以计算所有token的loss），加快了模型收敛。
      1. Fix了BERT pretrain与fine-tune不一致问题。

**预训练模型**

1. Finetune阶段有两种选择，一种frozen浅层不动，高层则随机初始化(因为高层是跟任务相关的，中低层跟词义、句法信息相关)；另一种预训练模型的参数被初始化，但是是可微调的。 //看训练数据量级而定
   1. 不管数据多少，至少可以做参数初始化。
   2. 底层一般是通用特征，而高层一般是跟任务相关特征。 这里多层CNN的特征抽取展现的层级性，跟BERT底层偏词汇，中层偏句法，高层任务相关基本上一致。

2、ELMO: Feature-based Pre-Training

预训练任务: 根据单词的上下文去预测单词的语言模型任务。

网络结构: word embedding + 双向双层的RNN。  // 最后将每层的embedding 加权和 作为下游任务的embedding 特征。 权重是学习参数，任务参数是超参数。

1. 第一层RNN表示表达了句法信息；第二层RNN表示表达了语义信息。

缺点:

1. LSTM的特征抽取能力远弱于Transformer
2. elmo的双向拼接(两个方向的RNN，最后拼接)，对于BERT的方式，比较弱。

CNN的最大优点是易于做并行计算，所以速度快，但是在捕获NLP的序列关系尤其是长距离特征方面天然有缺陷，不是做不到而是做不好，目前也有很多改进模型，但是特别成功的不多。

NLP的四大任务

**序列标注**: 分词/POS/NER/dependency parse

**文本分类**

**句子关系判断:** 是否蕴含/QA, 离题检测。 // 它的特点是给定两个句子，模型判断出两个句子是否具备某种语义关系；

**自然语言生成。**

其他: Dialog

BERT的MASK挖空任务跟CBOW确实非常相似，都是完形填空式的任务。

BERT的NSP为了下游的句子关系判断任务，多任务的学习

BERT本身没有太多创新，借鉴了Elmo/GPT/CBOW，是各个技术的集大成者，主要效果非常好以及非常适合适配到下游任务。

特点: Transformer特征抽取器 + 双向考虑上下文 + 适配任务能力强，效果好。

其他

其实图像领域最早由HighwayNet/Resnet等导致模型革命的skip connection的原始思路就是从LSTM的隐层传递机制借鉴来的。所以为啥LSTM能**缓解梯度消失**问题。

CNN应用于NLP:

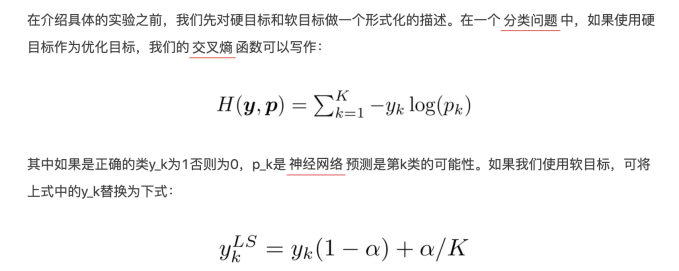
可能遇到的问题，

1. 对于如何捕获长距离依赖，通过加深CNN层既可以解决。
2. 位置信息如何编码进去，本身CNN在线性序列的的窗口移动保留了相对位置信息。思考: 如果是在所有CNN层最后加入Pooling，也是选择最强的特征，这个最强特征已经是考虑了长依赖情况下的特征了。完全不折损相对位置信息。**但是，在CNN层与层中间不要加pooling。**

在远距离特征捕获能力方面**，Transformer和RNN能力相近**，而CNN在这方面则显著弱于前两者。

**其他问题补充**

1. label smoothing: 将之前hard target转为soft target，从而提高泛化性。



偏差=0.1(默认值)，K是分类个数，则类别从[0, 1] = > [0.05, 0.95] //对于类别之间有关联比较友好。

2、优化器: 见下面总结

3、序列标注的其他算法

HMM, CRF, Bi-LSTM CRF

4、HMM与CRF的联系与区别

联系: 都是图模型; 都可以用于序列标注任务

区别:

1. HMM是生成模型; CRF是判别模型;
2. HMM有两个假设(**一阶马尔科夫假设**和观测独立性假设，但是实际中序列标注的词的上下文是有关联的), CRF打破了这两个假设(通过特征函数)。
3. HMM有向图，CRF无向图。

5、样本不平衡的解决方法？

**1）上采样和下采样；2）修改权重（修改损失函数）；3）集成方法：bagging，类似随机森林、自助采样；4）多任务联合学习(联想到gector)。**

1. 文本分类方法

Fasttext, CNN, TextCNN, Transformer, RNN, 即所有的特征提取器 + softmax都可以，BERT类。

7、知识蒸馏

核心思想是因为好模型的目标不是拟合训练数据，而是学习如何泛化到新的数据。

loss中加入与教师输出分布的KL散度（也可以是logits的平方损失），学习教师的soft target，可以提高泛化性，学习教师权重越大/或者T越大，泛化性越强。

Distilled Bi-LSTM. 迁移到bi-LSTM。

BERT-PKD: 教师模型的中间层提取知识。 //中间层是基础知识，通识。

DistillBERT: 加上cosine embedding loss。

TinyBERT: 蒸馏了注意力矩阵，以及embedding。 //logits的平方损失

1. **关于BN 以及LN的详解**  // 从以下四个角度整理

为什么需要Norm?

深度模型中，因为每层的参数更新导致后面层的输入数据分布变化**，这种变化差异随着层数累积被放大**，导致高层学习即受限数据分布的影响又受限本身参数更新的影响，导致训练难度增大。

1. 后层需要不断适应新的输入数据的分布，降低学习速度；
2. 前层输入的增大/减小，导致可能后层落入sigmoid 饱和区，学习缓慢。

希望是数据同分布。

Norm的抽象公式是什么?

f( g \* (x - u) / o + b ) //注意此处x为wx+b的线性值，进入激活层之前的值。

u, o //平移参数，缩放参数。 标准分布

g, b //再平移，再缩放参数

BN & LN分别是什么?

BN: 针对每个神经元，同batch\_size中的所有样本的值，计算均值和标准差。所以受限batch\_size大小影响。**希望batch\_size中的样本跟总体样本分布尽可能一致，所以batch\_size大小有要求，小batch\_size效果一般，且shuffle数据比较重要(保证同分布)。**

Batch\_s**ize中分布跟总体分布的一点差异，可以理解为引入噪声，增强泛化性。**

缺点：不适用于RNN

LN: 是针对每一层Layer，同层所有输入(特征)进行累计，计算均值和标准差。所以不受限batch\_size。

缺点: **如果同层不同特征之间数值意义不接近，则LN标准化的意义不大。**

Norm为何有效?

Norm中的权重伸缩不变性 & 数据伸缩不变性。

1）即放大缩小权重，其norm值一样，避免了W过大过小，反向传播中链式法则梯度累积导致的梯度消失/梯度暴炸问题；//**梯度消失问题**。

2）同样norm后的值避免走入如sigmoid的饱和区，加快了模型收敛，训练速度提升。

**下层权重越大时，梯度则越小，因为分子都是norm后的值（一样**），避免了参数震荡，整体参数变化比较稳定，相当于实现了参数正则化的效果。

问两次伸缩变化，第一、也不是同分布，第二、两次会不会相当于一次呢？

1. 不是同分布，是**为了保证模型的表达能力**不因标准化而下降。因为1）可以增强低层的网络学习能力，2）如果仅是0-1标准分布，sigmoid进入非饱和区，线性的学习能力，降低了NN表达能力，所以希望能学习到**非线性能力**。
2. 两次是否相当于一次，第一次学习取决于低层NN的复杂性，而第二次学习g,b解耦了低层NN，**很容易通过梯度下降学习到。**

牛客面经基础系列

1、SGD,Momentum,Adagard,Adam原理

SGD为随机梯度下降，每一次迭代计算数据集的mini-batch的梯度，然后对参数进行更新。

//相当于batch内的梯度累积，然后对参数更新；而梯度累积技巧，则是跨不同batch，梯度累积之后再一起更新。

缺点: **learning rate对所有参数都是一致，且learning rate的选择影响比较大，太大(训练震荡)太小(收敛很慢)**

Momentum参考了物理中动量的概念,前几次的梯度也会参与到当前的计算中,但是前几轮的梯度叠加在当前计算中会有一定的衰减。

优势: 可以使得梯度方向不变的维度上速度变快，梯度方向有所改变的维度上的更新速度变慢，这样就可以加快收敛并减小震荡。

Adagard在训练的过程中可以自动变更学习的速率,设置一个全局的学习率,**而实际的学习率与以往的参数模和的开方成反比**。 可以对低频的参数做较大的更新，对高频的做较小的更新，也因此，对于稀疏的数据它的表现很好，很好地提高了 SGD 的鲁棒性。

缺点: 分母会不断积累，这样学习率就会收缩并最终会变得非常小。

RMSprop 是一种自适应学习率方法。

RMSprop都是为了解决 Adagrad 学习率急剧下降问题的。

RMSprop使用的**是指数加权平均，旨在消除梯度下降中的摆动**，与Momentum的效果一样，某一维度的导数比较大，则指数加权平均就大，某一维度的导数比较小，则其指数加权平均就小，这样就保证了各维度导数都在一个量级，进而减少了摆动。允许使用一个更大的学习率η）

Adam：另一种计算每个参数的自适应学习率的方法。相当于 RMSprop + Momentum

(利用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率, 在经过偏置的校正后,每一次迭代后的学习率都有个确定的范围,使得参数较为平稳。)

实践表明，Adam比其他适应性学习方法效果要好。

总结

1. AUC的意义与公式

AUC: 在ROC曲线下的面积。

ROC曲线的横轴和纵轴分别是:

FPRate:

TPRate: **正例的recall。** = TP / TP + FN. // T/F表示是否预测准确; P/N表示预测类别为1/0, 即都是针对预测而言。混淆矩阵。

FPRate: **负例的虚警。**= FP / FP + TN。

这两个指标，不一定非要这两个指标，也可以是正例的recall & 负例的recall

AUC最小值=0.5 //相当于分类器随机猜。

需要正例的recall够大，而负例的虚警够小。 //

如何绘制ROC曲线？

依次使用所有预测值作为阈值，得到一系列TPRate，FPRate，描点，求面积，即可得到AUC。

AUC的优点: 同时考虑分类器对于正例和负例的分类能力，在样本不平衡情况下，也能对分类器做合理的评价。

但AUC有一个更优雅的，概率学上的意义：随机选取一个正例和一个负例，分类器给正例的打分大于分类器给负例的打分的概率。

2、概率题

连续两次正面硬币停止，实验次数的期望。

N(k)为连续k次正面朝上的试验次数。 //状态定义

k=1时，是一个几何分布，E(N(k)) = 1 / p = 2。

E(N(k)) = p \*( E(N(k - 1)) + 1) + (1 - p)( E(N(k - 1)) + 1 + E(N(k))) //递推公式

相当于以连续K次正面的期望作为状态。

假设递归信任K-1次，那么下一次(当前步)时，要么正面p，要么反面(1-p)。//分步用加

如果是正面，则结束; + 1次 //因为是期望值，则用+ 次数。

如果是反面，则当前步以及之前步，全部作废(但是其值还是保留)，需要继续连续K次正面。

关于项目中一些问题

语法流畅性、语义相同性 & 多样性

流畅性通过语言模型保证，perplexity困惑度。

1. NBest的多样性如何改进?

Greedy search: 句子不通顺

Beam search: k越大，计算复杂度高，句子通顺，但是安全&通用，并没有多样性，且倾向于短句子。

以下两种方法可以提升多样性:

Pure sampling, 按照softmax后的概率分布进行采样。

Top-n sampling, 按照top-n的词中按概率分布采样。

而加入softmax temperature，可以增加**流畅度，平庸度或者新奇度**。

BLEU：基于模型预测与标准答案的n-gram的overlap， 1-4-gram的指数平均。**惩罚短句子**。更关注precision。

ROUGUE, 同上，但是是分1-4-gram计算，且不惩罚短句，更关注recall。

1. 引申，seq2seq模型可能有哪些问题，如何解决？

unk问题，copy/point generation

可以作为难点解答。

5、nmt可能有哪些问题，如何改进?

两个问题，一个是teacher forcing带来的exposure bias暴露偏差，误差累积的问题。

解决方案：以一定概率从模型预测和ground-truth中选择一个单词作为当前步的输入(有点类似于unk的解决)

第二个问题是过度correction，即必须跟ground-truth一致，但是其实不同词也许是合理的。

解决: 最大似然函数(可以解决吗？) 或者label smoothing

1. BERT中FFNN的作用，扩展到Transformer中的所有核心部分的作用

1) 关于attention

加法attention是使用单隐层前向网络来计算相关函数V.tanh(w1h1+w2h2)。两种方式在理论上复杂度是相似的，由于dot-product可以使用高度优化的矩阵代码实现，所以它在**应用中更快**，空间使用也更加有效。

尽管当dk值比较小的时候，两种机制性能相似，但dk变大时候，加法attention不需要缩放，比dot product attention表现更好。我们认为，当dk很大的时候，点积大幅增大，将softmax函数推向具有极小梯度的区域。为了避免这个影响，我们使用dk来缩放点积。//速度快选择了缩放点积的attention.

mask的attention 设置为负无穷，softmax值则为0. 指数函数曲线。

1. FFNN

两种理解，1种是非线性激活函数ReLU的全连接层 + 线性全连接层；另一种理解是卷积核大小是1 \* 1的两个卷积操作。

作用有两个：1）非线性增强模型的表征能力(非线性建模能力，拟合更复杂的函数)；2）维度的变换，特征空间的转换。【卷积核为1的两个作用】

1. positional encoding位置编码

相对位置编码：sin/cos(位置，维度两个参数)

因为对于任意固定的偏移项k, 都能由PEpos +k = PE pos线性表示。 //表达了位置之间的相对关系。

绝对位置编码：随机初始化，随着模型训练而更新参数。

1. 三种正则化手段

Dropout/label smoothing/layer norm

1. 模型预测单词是使用线性映射和softmax两层来实现的。
2. 两种mask

Padding mask: 对齐的padding设置为负无穷，softmax后为0。//参考指数函数。

Sequence mask

1. multi-head = 8, model dim=512,其实是d=64每个self-attention
2. ResNet残差连接作用

- 缓解梯度消失，除了学习恒等变换话，还能学习到其他映射的能力，增强模型表达能力； 防止信息在流动过程中的折损。

1. Layer Norm: 均值为0，方差为1标准化，当然有一个缩放，每个维度（多个结点）上每个样本都做norm。 相比于BN，BN则是同一个batch内对每个节点做norm。
2. XLNet解决了什么问题？

解决了BERT以及GPT的缺点。

1. 未来NLP的方向思考

1）更强的特征抽取器；2）更强大的训练方式； 3）多任务的融合=>QA.

1. word2vec中h-softmax的loss以及负采样的loss

H-softmax中: 高频词路径短。每个叶子节点是单词。

**每个词的概率=词到根路径上所有节点的概率乘积。**

能看出，每个中间节点相当于是LR分类器，参数为w, 输入为映射层的词向量结果。

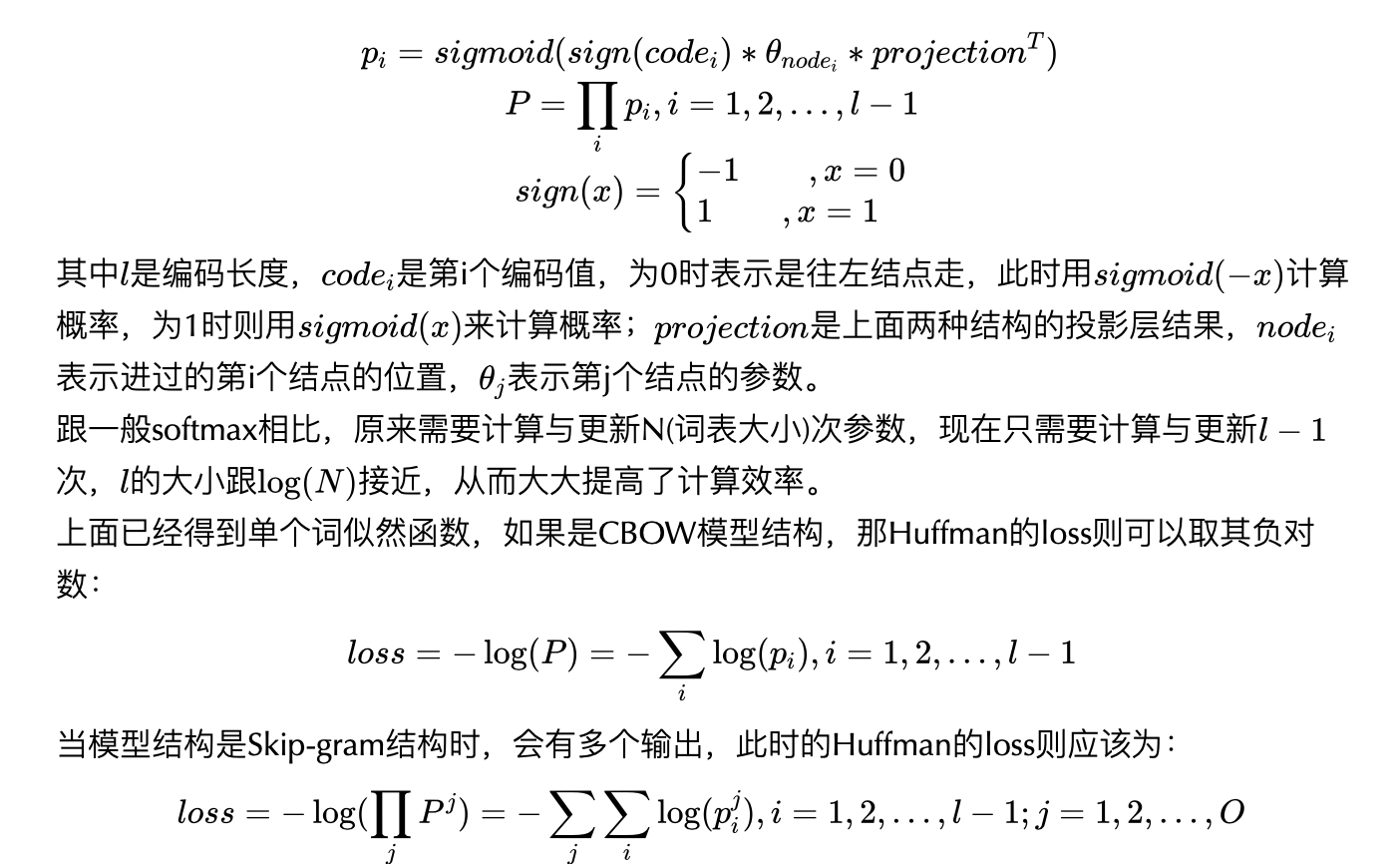
H-softmax的loss是单词的似然函数的负对数。即log损失函数。

如果skip-gram，可以看成多分类，则是交叉熵损失函数。 两个加和

H-softmax使用的就转为分层二分类逻辑斯蒂回归，从根节点出发，分为负类则向左子树走，分为正类则向右子树走，分类的依据来源于每个非叶子节点上所带有的内部节点向量 & 映射的特征。

负采样方法中: 构造二分类，正负比例是1：K

则loss**：K+1个样本之和，而每个样本的loss则是log损失函数。(其实并不是对K+1求softmax)**



扩展

1. 阅读理解中是如何确定answer范围，span-answer, 多选项answer。

QANet如何做

P1, P2 起始位置 & 结束位置，是不同attention block的attention vec的softmax。看成是分类任务。loss是两个位置的交叉熵损失函数的加和。

BERT-based如何做

- passage + question， CLS + SEP 分隔

- 为每一个token分别作为答案开头和结尾的logit值，再经过softmax层之后就得到了相应的概率值。 max(start + end)的logits, 且start < end.

如何判断为no answer, start & end均指向CLS

以上是span-answer类

如果是多项选择题，则是分类问题，concat, softmax即可。

1. NLP有哪些方向

序列标注

文本分类

情感分析

机器翻译

问答

机器阅读理解

Dialog

1. BERT如何做序列标注

BERT + FC layer + softmax

BERT + FC layer + CRF

1. stacking ensemble方法
2. 在生成类问题上，如何处理数据不平衡，low resource 以及跨语言问题。
3. 如何将工作中遇到的问题，或者任务可能遇到的问题，提炼为一套解决方案。//在经验上的积累，技术的沉淀。

比如n-best的多样性问题，low resource问题等。

1. 看下CopyNet。