

欢迎加入代码随想录知识星球

// 一起抱团取暖

点击进入

本文档是 [代码随想录知识星球](#) 成员: [在线求offer](#), 自己从双非到拿到大厂算法岗的全部知识点总结。

她星球里具体讲了自己的[秋招全过程](#), 和自己的[实习全过程](#)

以下就是算法岗知识点总结:

秋招面试问题记录

预训练语言模型

BERT类

1、bert的基本任务: mask language model 和 next sentence predict

2、BERT 的MASK方式的优缺点?

mask language model的遮盖方式: 选择15%的token进行遮盖, 再选择其中80%进行mask, 10%进行随机替换, 10%不变

优点:

1) 被随机选择15%的词当中以10%的概率用任意词替换去预测正确的词, 相当于文本纠错任务, 为BERT模型赋予了一定的文本纠错能力;

2) 被随机选择15%的词当中以10%的概率保持不变, 缓解了finetune时候与预训练时候输入不匹配的问题(预训练时候输入句子当中有mask, 而finetune时候输入是完整无缺的句子, 即为输入不匹配问题)。

缺点: 针对有两个及两个以上连续字组成的词, 随机mask字割裂了连续字之间的相关性, 使模型不太容易学习到词的语义信息。主要针对这一短板, 因此google此后发表了BERT-WWM, 国内的哈工大联合讯飞发表了中文版的BERT-WWM。

3、Bert的构成: 由12层Transformer Encoder构成

4、Transformer encoder的组成: self-Attention和feed forward组成 中间穿插残差网络

残差网络的作用: 缓解层数加深带来的信息损失, 采用两个线性变换激活函数为relu, 同等层数的前提下残差网络也收敛得更快

5、self-Attention的组成和作用?

- 1) 对每一个token分别与三个权重矩阵相乘，生成q,k,v三个向量，维度是64
- 2) 计算得分，以句子中的第一个词为例，拿输入句子中的每个词的k去点积q1，这个分数决定了在编码第一个词的过程中有多重视句子的其他部分
- 3) 将分数除以键向量维度的平方根（根号dk），这可以使梯度更稳定

为什么要除以根号dk?

假设q和k中的向量都服从高斯分布，即均值为0，方差为1，那可qk点积都的矩阵均值为0，方差为dk，此时若直接做softmax，根据公式，分母是ex的相加，分母较大，会导致softmax的梯度变小，参数更新困难

- 4) softmax将分数归一化，表示每个单词对编码当下位置的贡献
- 5) 用v乘softmax分数，希望关注语义上相关的单词，并弱化不相关的词
- 6) 对第5步的各个位置求和

6、multi-headed attention的作用

多头机制类似于“多通道”特征抽取，self attention通过attention mask动态编码变长序列，解决长距离依赖（无位置偏差）、可并行计算；

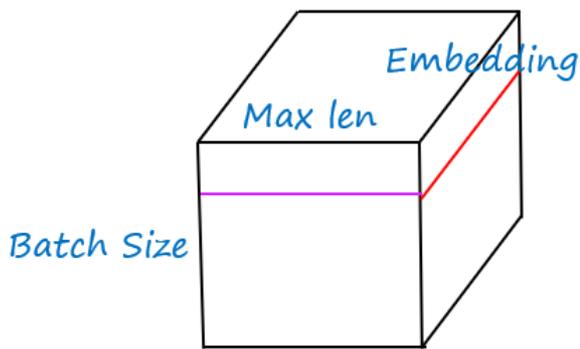
- 多头机制为什么有效？
 - 类似于CNN中通过多通道机制进行特征选择；
 - Transformer中先通过切头（spilt）再分别进行Scaled Dot-Product Attention，可以使进行点积计算的维度d不大（防止梯度消失），同时缩小attention mask矩阵。

7、Feed-Forward Networks

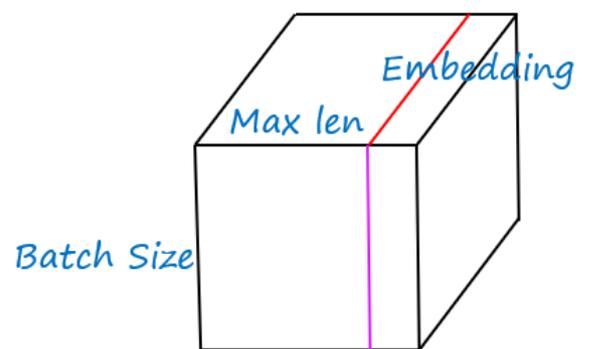
- FFN 将每个位置的Multi-Head Attention结果映射到一个更大维度的特征空间，然后使用ReLU引入非线性进行筛选，最后恢复回原始维度。
- Transformer在抛弃了LSTM结构后，FFN中的ReLU成为了一个主要的提供非线性变换的单元。

8、bert为什么用layer normalization?

Self-Attention的输出会经过Layer Normalization，为什么选择Layer Normalization而不是Batch Normalization? 此时，我们应该先对我们的数据形状有个直观的认识，当一个batch的数据输入模型的时候，形状是长方体如图8.5所示，大小为(batch_size, max_len, embedding)，其中batch_size为batch的批数，max_len为每一批数据的序列最大长度，embedding则为每一个单词或者字的embedding维度大小。而Batch Normalization是对每个Batch的每一列做normalization，相当于是对batch里相同位置的字或者单词embedding做归一化，Layer Normalization是Batch的每一行做normalization，相当于是对每句话的embedding做归一化。显然，LN更加符合我们处理文本的直觉。



Layer Normalization



Batch Normalization

9、BERT基于“字输入”还是“词输入”好？

- 如果基于“词输入”，会出现OOV(Out Of Vocabulary)问题，会增大标签空间，需要利用更多语料去学习标签分布来拟合模型。
- 随着Transformer特征抽取能力，分词不再成为必要，词级别的特征学习可以纳入为内部特征进行表示学习。

10、BERT的三个输入为什么可以相加？

BERT的词嵌入由**符号嵌入** (Token Embedding)、**片段嵌入** (Segmentation Embedding) 和**位置嵌入** (Position Embedding) 合成得到，表示为

$$E_{word} = E_{tok} + E_{seg} + E_{pos}$$

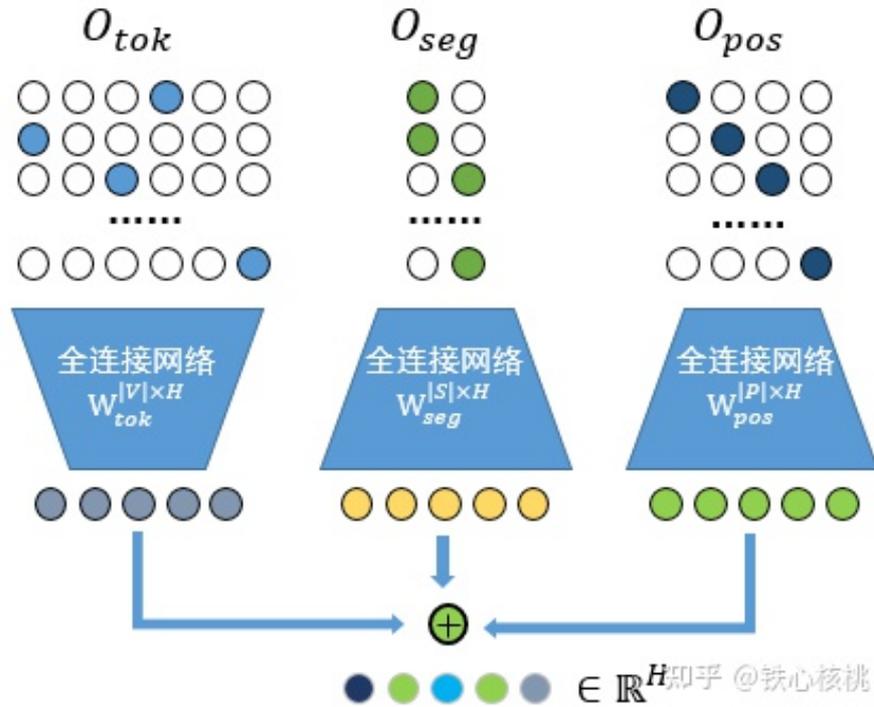
上述三个嵌入分量都可以表达为“独热” (one-hot) 编码表示输入与嵌入矩阵的乘积形式，即

$$E_{word} = O_{tok} W_{tok}^{|V| \times H} + O_{seg} W_{seg}^{|S| \times H} + O_{pos} W_{pos}^{|P| \times H}$$

其中， O_{tok} ：依据符号在词典中位置下标、对输入符号构造的one-hot编码表示； O_{seg} ：依据符号在两个序列中隶属标签（更一般的为符号属性）下标、对输入符号构造的one-hot编码表示； O_{pos} ：以符号在句子位置下标、对输入符号构造的one-hot编码表示； $W_{tok}^{|V| \times H}$ 、 $W_{seg}^{|S| \times H}$ 和 $W_{pos}^{|P| \times H}$ 分别为其对应的待训练嵌入参数矩阵； $|V|$ 、 $|S|$ 和 $|P|$ 分别为词典维度、序列个数（更一般的为符号属性）和最大位置数； $|H|$ 为嵌入维度。下面从三个角度理解合成。

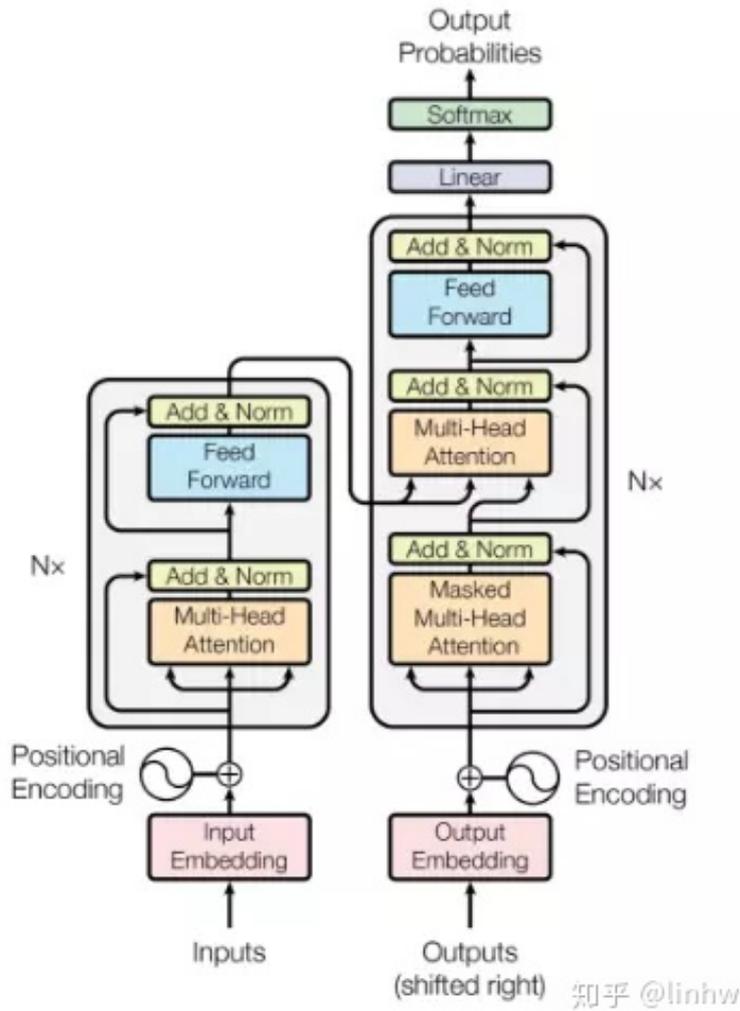
(1) 按照分别过网络再做求和融合的角度理解

三个one-hot编码向量与嵌入矩阵相乘，等价于构造三个以one-hot编码向量作为输入，输入维度分别为 $|V|$ 、 $|S|$ 和 $|P|$ ，输出维度均为 H 的全连接网络。求和即为特征融合。如下图所示



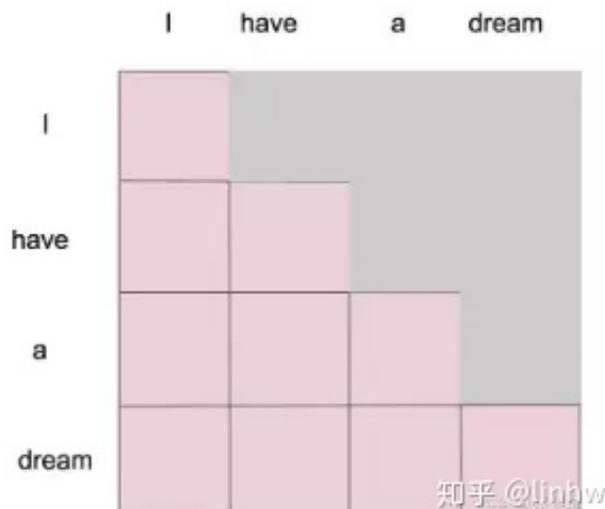
11、bert预训练的batch_size为256，roberta为2K

12、transformer的编码器与解码器



解码器的block由masked multi-head attention和一个encoder-decoder的attention组成。

其中masked multi-head attention部分用于将未来的信息mask掉，因为在生成的时候是无法知道未来的信息的，即当前词无法看到后面的词



Decoder的第二个部分是一个encoder和decoder的attention，这一部分可以看成解码器在用编码器的输出信息来计算当前解码应该输出什么

13、bert预训练任务的损失函数

两个任务是联合学习，可以使得 BERT 学习到的表征既有 token 级别信息，同时也包含了句子级别的语义信息。

bert的损失函数组成：

- 第一部分是来自 Mask-LM 的单词级别分类任务；
- 另一部分是句子级别的分类任务；
- 损失函数

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

 关于NLP那些你不知道的事

注： θ ：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 ：是句子预测任务中在 Encoder 接上的分类器参数；

- 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，所用的损失函数叫做负对数似然函数（且是最小化，等价于最大化对数似然函数），那么具体说来有：

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

 关于NLP那些你不知道的事

- 在第二部分的损失函数中，在句子预测任务中，也是一个分类问题的损失函数：

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

 关于NLP那些你不知道的事

- 两个任务联合学习的损失函数是：

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

 关于NLP那些你不知道的事

Subword

1、BPE获得Subword的步骤如下：GPT/RoBERTa

- 准备足够大的训练语料，并确定期望的Subword词表大小；
- 将单词拆分为成最小单元。比如英文中26个字母加上各种符号，这些作为初始词表；
- 在语料上统计单词内相邻单元对的频数，选取频数最高的单元对合并成新的Subword单元；
- 重复第3步直到达到第1步设定的Subword词表大小或下一个最高频数为1。

2、WordPiece BERT

与BPE算法类似，WordPiece算法也是每次从词表中选出两个子词合并成新的子词。与BPE的最大区别在于，如何选择两个子词进行合并：BPE选择频数最高的相邻子词合并，而WordPiece选择能够提升语言模型概率最大的相邻子词加入词表。

3、ULM

与WordPiece一样，Unigram Language Model(ULM)同样使用语言模型来挑选子词。不同之处在于，BPE和WordPiece算法的词表大小都是从小到大变化，属于增量法。而Unigram Language Model则是减量法,即先初始化一个大词表，根据评估准则不断丢弃词表，直到满足限定条件。ULM算法考虑了句子的不同分词可能，因而能够输出带概率的多个子词分段。

自回归与自编码

单向特征表示的自回归预训练语言模型，统称为单向模型：ELMO、GPT、ULMFiT、SiATL

双向特征表示的自编码预训练语言模型，统称为BERT系列模型：BERT、RoBERTa、ERNIE1.0/ERNIE(THU)

双向特征表示的自回归预训练语言模型：XLNet：将自回归LM方向引入双向语言模型方面

自回归语言模：

优点：文本摘要，机器翻译等，在实际生成内容的时候，就是从左向右的，自回归语言模型天然匹配这个过程

缺点：只能利用上文或者下文的信息，不能同时利用上文和下文的信息

自编码语言模型：

优点：能比较自然地融入双向语言模型，同时看到被预测单词的上文和下文

缺点：主要在输入侧引入[Mask]标记，导致预训练阶段和Fine-tuning阶段不一致的问题，因为Fine-tuning阶段是看不到[Mask]标记的

GPT/ELMO/XLNET/NEZHA

GPT1.0/GPT2.0(OpenAI)

- GPT1.0要点：
 - 采用Transformer进行特征抽取，首次将Transformer应用于预训练语言模型；
 - finetune阶段引入语言模型辅助目标（辅助目标对于大数据集有用，小数据反而有所下降，与SiATL相反），解决finetune过程中的灾难性遗忘；
 - 预训练和finetune一致，统一2阶段框架；
- GPT2.0要点：
 - 没有针对特定模型的精调流程：GPT2.0认为预训练中已包含很多特定任务所需的信息。
 - 生成任务取得很好效果，使用覆盖更广、质量更高的数据；
- 缺点：
 - 依然为单向自回归语言模型，无法获取上下文相关的特征表示；

ELMO(华盛顿大学)

- 要点：
 - 引入双向语言模型，其实是2个单向语言模型（前向和后向）的集成；

- 通过保存预训练好的2层LSTM，通过特征集成或finetune应用于下游任务；
- 缺点：
 - 本质上为自回归语言模型，只能获取单向的特征表示，不能同时获取上下文表示；
 - LSTM不能解决长距离依赖。
- 为什么不能用biLSTM构建双向语言模型？
 - 不能采取2层biLSTM同时进行特征抽取构建双向语言模型，否则会出现**标签泄漏**的问题；因此ELMO前向和后向的LSTM参数独立，共享词向量，独立构建语言模型；

XLNET

- Permutation Language Model(简称PLM,排序模型)；这个可以理解为在自回归LM模式下，如何采取具体手段，来融入双向语言模型。
- 引入了Transformer-XL的主要思路：相对位置编码以及分段RNN机制。处理过长文本
- 加大增加了预训练阶段使用的数据规模

NEZHA

- 相对位置编码，每一层计算隐状态的相互依赖的时候考虑他们之间的相对位置关系
- 全词遮盖
- 训练过程中采用混合精度训练，将训练速度提高到2-3倍，还可以减少模型的空间消耗
- 优化器采用LAMB(Large Batch Optimization for Deep Learning: Training BERT in 76 minutes), LAMB优化器通过一个自适应式的方式为每个参数调整learning rate，能够在Batch Size很大的情况下不损失模型的效果，使得模型训练能够采用很大的Batch Size，进而极大提高训练速度。

Fasttext与CBOW

相同：

- 模型架构相似
- 都用了分层softmax进行优化，其中fasttext频繁出现类别的树形结构的深度要比不频繁出现类别的树形结构的深度要小，这也使得进一步的计算效率更高。

不同：

- fasttext的输入：句子中的每个词和句子的ngram特征，特征向量通过线性变换映射到中间层，中间层再映射到标签。fastText 在预测标签时使用了非线性激活函数，但在中间层不使用非线性激活函数。

cbow的输入：中间词的上下文，与完整句子无关

关于ngram：假设：第n个词出现与前n-1个词相关，而与其他任何词不相关

cbow不考虑词之间的顺序，即没有ngram，fasttext加入ngram

- fasttext是文本分类模型，是有监督模型；可以完成无监督的词向量学习

cbow是训练词向量算法，是无监督模型

RNN与LSTM

1、梯度爆炸与梯度消失

RNN的统一定义为

$$h_t = f(x_t, h_{t-1}; \theta) \quad (1)$$

其中 h_t 是每一步的输出，它由当前输入 x_t 和前一时刻输出 h_{t-1} 共同决定，而 θ 则是可训练参数。在做最基本的分析时，我们可以假设 h_t, x_t, θ 都是一维的，这可以让我们获得最直观的理解，并且其结果对高维情形仍有参考价值。之所以要考虑梯度，是因为我们目前主流的优化器还是梯度下降及其变种，因此要求我们定义模型有一个比较合理的梯度。我们可以求得：

$$\frac{dh_t}{d\theta} = \frac{\partial h_t}{\partial h_{t-1}} \frac{dh_{t-1}}{d\theta} + \frac{\partial h_t}{\partial \theta} \quad (2)$$

可以看到，其实RNN的梯度也是一个RNN，当前时刻梯度 $\frac{dh_t}{d\theta}$ 是前一时刻梯度 $\frac{dh_{t-1}}{d\theta}$ 与当前运算梯度 $\frac{\partial h_t}{\partial \theta}$ 的函数。同时，从上式我们就可以看出，其实梯度消失或者梯度爆炸现象几乎是必然存在的：当 $\left| \frac{\partial h_t}{\partial h_{t-1}} \right| < 1$ 时，意味着历史的梯度信息是衰减的，因此步数多了梯度必然消失（好比 $\lim_{n \rightarrow \infty} 0.9^n \rightarrow 0$ ）；当 $\left| \frac{\partial h_t}{\partial h_{t-1}} \right| > 1$ ，因为这历史的梯度信息逐步增强，因此步数多了梯度必然爆炸（好比 $\lim_{n \rightarrow \infty} 1.1^n \rightarrow \infty$ ）。总不可能一直 $\left| \frac{\partial h_t}{\partial h_{t-1}} \right| = 1$ 吧？当然，也有可能有些时刻大于1，有些时刻小于1，最终稳定在1附近，但这样概率很小，需要很精巧地设计模型才行。

所以步数多了，**梯度消失或爆炸几乎都是不可避免的，我们只能对于有限的步数去缓解这个问题。**

2、梯度消失与梯度爆炸的解决方法

梯度爆炸：设置一个梯度剪切阈值，然后更新梯度的时候，如果梯度超过这个阈值，那么就将其强制限制在这个范围之内。这可以防止梯度爆炸

梯度消失：预训练加微调、使用relu、残差网络、

batchnorm:

问题上。具体来说就是反向传播中，经过每一层的梯度会乘以该层的权重，举个简单例子：正向传

播中 $f_3 = f_2(w^T * x + b)$ ，那么反向传播中， $\frac{\partial f_2}{\partial x} = \frac{\partial f_2}{\partial f_1} w$ ，反向传播式子中有w

的存在，所以 w 的大小影响了梯度的消失和爆炸，batchnorm就是通过对每一层的输出做scale和shift的方法，通过一定的规范化手段，把每层神经网络任意神经元这个输入值的分布强行拉回到接近均值为0方差为1的标准正太分布，即严重偏离的分布强制拉回比较标准的分布，这样使得激活输入值落在非线性函数对输入比较敏感的区域，这样输入的小变化就会导致损失函数较大的变化，使得让梯度变大，避免梯度消失问题产生，而且梯度变大意味着学习收敛速度快，能大大加快训练速度。

3、LSTM有输入门、输出门、遗忘门

遗忘门：用于决定应丢弃或保留的信息

输入门：用于对输入信息进行过滤，选择性的抛弃一些信息，和候选细胞状态共同决定当前细胞状态的更新，决定长期记忆

输出门：用sigmoid层来确定细胞状态的哪个部分将输出出去，ht决定短期记忆

机器学习类

batch_size与学习率

以随机梯度下降为例

$$w_{t+1} = w_t - \eta \frac{1}{n} \sum_{x \in \mathcal{B}} \nabla l(x, w_t).$$

知乎 @龙鹏-言有三

n是批量大小(batchsize), η 是学习率(learning rate)。可知道除了梯度本身，**这两个因子直接决定了模型的权重更新**，从优化本身来看它们是影响模型性能收敛最重要的参数。

学习率直接影响模型的收敛状态，**batchsize则影响模型的泛化性能**，两者又是分子分母的直接关系，相互也可影响，因此这一次来详述它们对模型性能的影响。

学习率 (lr) 直观可以看出lr越大，权重更新的跨度越大，模型参数调整变化越快

- **大的batchsize减少训练时间，提高稳定性。** 这是肯定的，同样的epoch数目，大的batchsize需要的batch数目减少了，所以可以减少训练时间。另一方面，大的batch size梯度的计算更加稳定，因为模型训练曲线会更加平滑。在微调的时候，大的batch size可能会取得更好的结果。
- **大的batchsize泛化能力下降。** 在一定范围内，增加batchsize有助于收敛的稳定性，但是随着batchsize的增加，模型的性能会下降，如下图，来自于文[1]。

评估指标

1、ROC与AUC

ROC要计算FPR和TPR，曲线上的每个点代表不同阈值时的FPR和TPR

在正负样本数量不均衡的时候，比如负样本的数量增加到原来的10倍，那TPR不受影响，FPR的各项也是成比例的增加，并不会太大的变化。因此，在样本不均衡的情况下，同样ROC曲线仍然能较好地评价分类器的性能，这是ROC的一个优良特性，也是为什么一般ROC曲线使用更多的原因。

AUC <https://www.zhihu.com/search?type=content&q=正负样本比例变化auc>

AUC即ROC曲线下的面积，AUC越大，说明分类器越可能把正样本排在前面，衡量的是一种排序的性能。

AUC的概率意义：就是从样本中任意取一个正例和一个负例，正例得分大于负例得分的概率。

所以，AUC的正式计算公式也就有了，如下！（需要牢记）

$$AUC = \frac{\sum_{i \in \text{正样本}} \text{rank}(i) - \frac{M \times (1+M)}{2}}{M \times N}$$

$\text{rank}(i)$ 表示正样本 i 的排序编号， $M \times N$ 表示随机从正负样本各取一个的情况数。一般考代码题，比如用python写个AUC计算，也是用上述公式原理。

AUC的优点：

- AUC衡量的是一种排序能力，因此特别适合排序类业务；
- AUC对正负样本均衡并不敏感，在样本不均衡的情况下，也可以做出合理的评估。
- 其他指标比如precision, recall, F1，根据区分正负样本阈值的变化会有不同的结果，而AUC不需要手动设定阈值，是一种整体上的衡量方法。

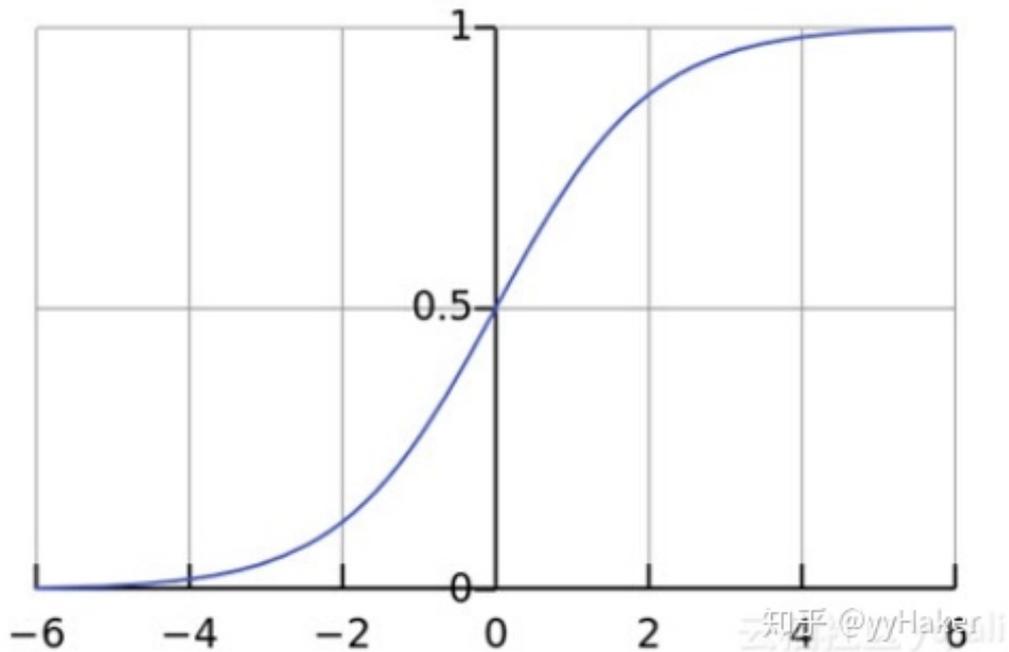
AUC的缺点：

- 忽略了预测的概率值和模型的拟合优度；
- AUC反应了太过笼统的信息。无法反应召回率、精确率等在实际业务中经常关心的指标；
- 它没有给出模型误差的空间分布信息，AUC只关注正负样本之间的排序，并不关心正样本内部，或者负样本内部的排序，这样我们也无法衡量样本对于好坏程度的刻画能力；

常见激活函数：

- sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$



第一点，最明显的就是饱和性，从上图也不难看出其两侧导数逐渐趋近于0，即 $\lim_{x \rightarrow \pm\infty} f'(x) = 0$ 。具体来说，在反向传播的过程中，sigmoid的梯度会包含了一个 $f'(x)$ 因子（sigmoid关于输入的导数），因此一旦输入落入两端的饱和区， $f'(x)$ 就会变得接近于0，导致反向传播的梯度也变得非常小，此时网络参数可能甚至得不到更新，难以有效训练，这种现象称为梯度消失。一般来说，sigmoid网络在5层之内就会产生梯度消失现象。

sigmoid与softmax:

如果模型输出为非互斥类别，且可以同时选择多个类别，则采用Sigmoid函数计算该网络的原始输出值。

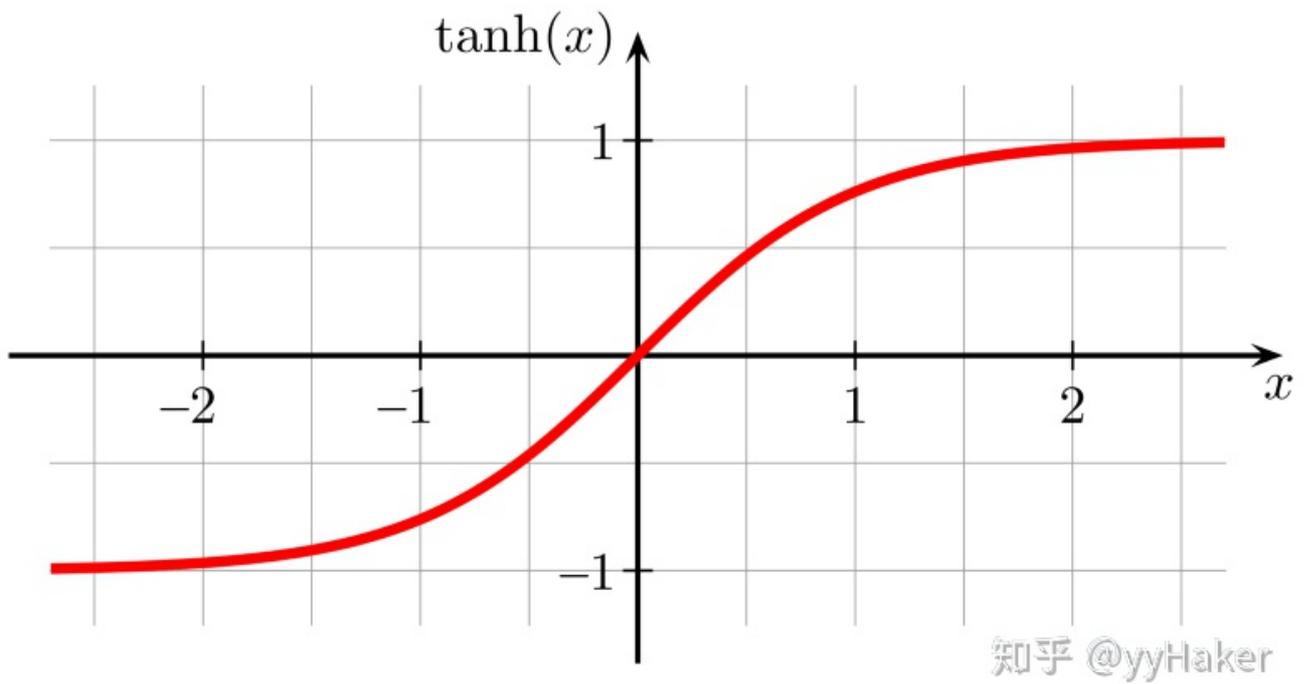
如果模型输出为互斥类别，且只能选择一个类别，则采用Softmax函数计算该网络的原始输出值。

Sigmoid函数可以用来解决多标签问题，Softmax函数用来解决单标签问题。

对于某个分类场景，当Softmax函数能用时，Sigmoid函数一定可以用。

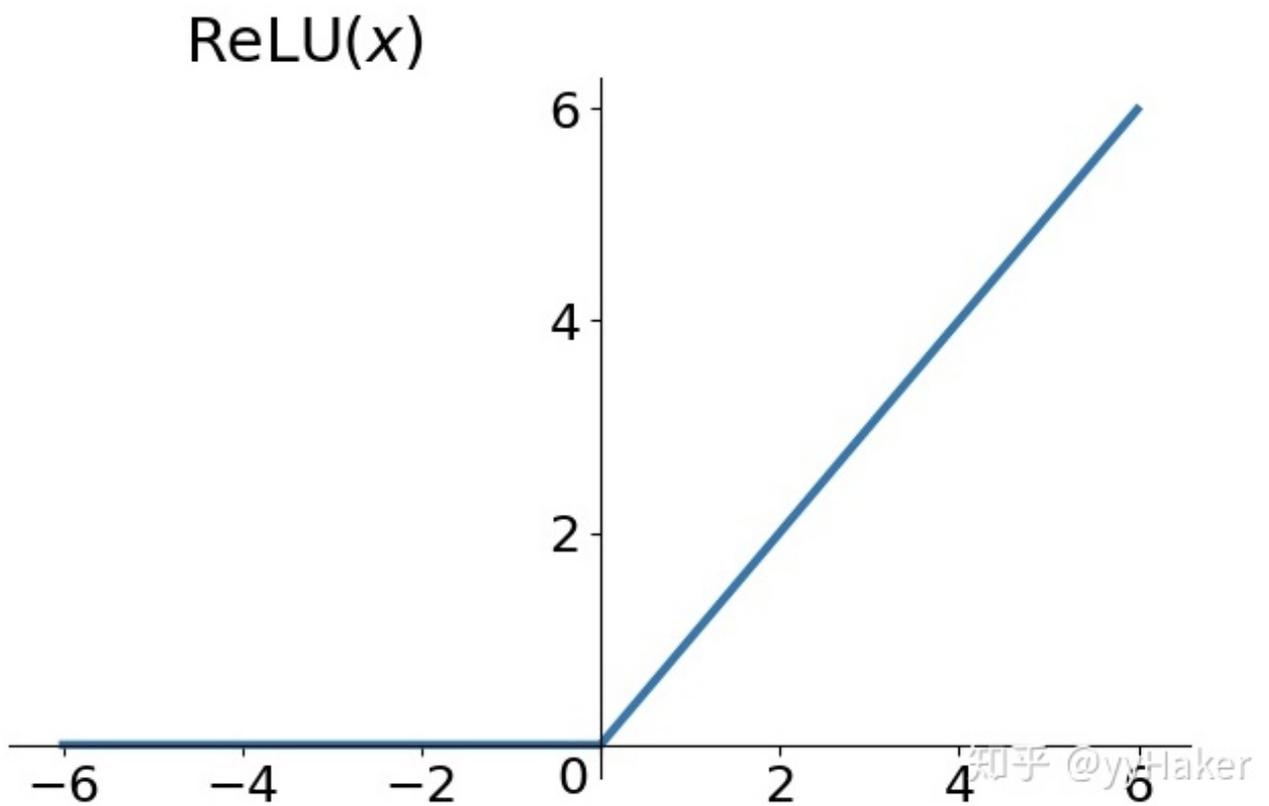
- tanh

$$\tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$



- relu

$$\text{ReLU}(x) = \max(0, x)$$



优点:

- 解决了梯度消失、爆炸的问题
- 计算方便, 计算速度快
- 加速了网络的训练

缺点：

- 由于负数部分恒为0，会导致一些神经元无法激活（可通过设置小学习率部分解决）
- 输出不是以0为中心的

softmax与sigmoid

如何用softmax做多分类和多标签分类

现假设，神经网络模型最后的输出是这样一个向量 z ，就是神经网络最终的全连接的输出。这里假设总共有4个分类

首先用 softmax 将 logits 转换成一个概率分布，然后取概率值最大的作为样本的分类这样看似，`tf.argmax(logits)` 同样可以取得最大的值，也能得到正确的样本分类，这样的话 softmax 似乎作用不大那么 softmax 的主要作用其实是在计算交叉熵上，首先样本集中 y 是一个 one-hot 向量，如果直接将模型输出 logits 和 y 来计算交叉熵，因为 $z=[1,2,3,4]$ ，计算出来的交叉熵肯定很大，这种计算方式不对，而应该将 logits 转换成一个概率分布后再来计算，就是用 `tf.softmax(logits)` 和 y 来计算交叉熵，当然我们也可以直接用 tensorflow 提供的方法 `softmax_cross_entropy_with_logits` 来计算这个方法传入的参数可以直接是 logits，因为这个根据方法的名字可以看到，方法内部会将参数用 softmax 进行处理现在我们取的概率分布中最大的作为最终的分类结果，这是多分类我们也可以取概率的 top 几个，作为最终的多个标签，或者设置一个阈值，并取大于概率阈值的。这就用 softmax 实现了多标签分类

sigmoid激活函数应用于多标签分类

sigmoid 一般不用来做多类分类，而是用来做二分类，它是将一个标量数字转换到 $[0,1]$ 之间，如果大于一个概率阈值（一般是0.5），则认为属于某个类别，否则不属于某个类别。这一属性使得其适合应用于多标签分类之中，在多标签分类中，大多使用 `binary_crossentropy` 损失函数。它是将一个标量数字转换到 $[0,1]$ 之间，如果大于一个概率阈值（一般是0.5），则认为属于某个类别。本质上其实就是针对 logits 中每个分类计算的结果分别作用一个 sigmoid 分类器，分别判定样本是否属于某个类别同样假设，神经网络模型最后的输出是这样一个向量 $z=[1,2,3,4,5,6,7,8,9,10]$ ，就是神经网络最终的全连接的输出。这里假设总共有 10 个分类。通过：

```
tf.nn.sigmoid(logits)
```

sigmoid 应该会将 logits 中每个数字都变成 $[0,1]$ 之间的概率值，假设结果为 $[0.01, 0.05, 0.4, 0.6, 0.3, 0.1, 0.5, 0.4, 0.06, 0.8]$ ，然后设置一个概率阈值，比如 0.3，如果概率值大于 0.3，则判定类别符合，那么该输入样本则会被判定为 类别3、类别4、类别5、类别7 及 类别8。即一个样本具有多个标签。在这里强调一点：将 sigmoid 激活函数应用于多标签分类时，其损失函数应设置为 `binary_crossentropy`。

L1正则化和L2正则化分别适用于什么样的场景

如果需要稀疏性就用l1，因为l1的梯度是1或-1，所以每次更新都稳步向0趋近。

一般多用l2因为计算方便“求导置零解方程”，l2只有最好的一条预测线而l1可能有多个最优解。

l1鲁棒性更强对异常值不敏感

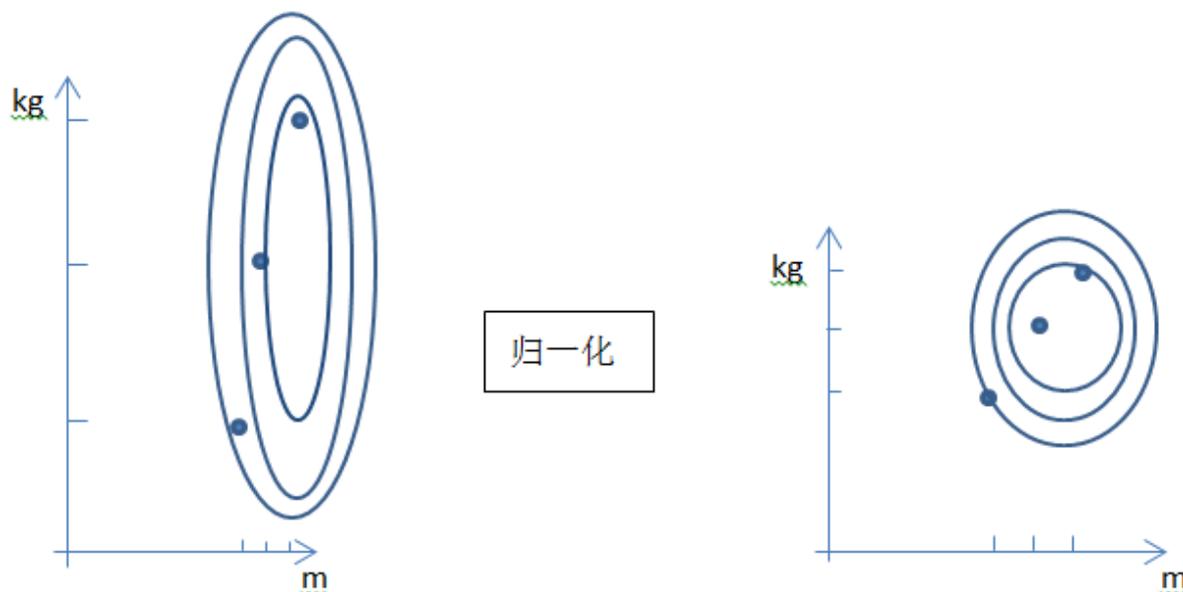
归一化和标准化

归一化：对不同特征维度的伸缩变换的目的是使各个特征维度对目标函数的影响权重是一致的，即使得那些扁平分布的数据伸缩变换成类圆形。这也就改变了原始数据的一个分布。

好处:

- 1 提高迭代求解的收敛速度, 不归一化梯度可能会震荡
- 2 提高迭代求解的精度

再来看数据在身高和体重两个特征的二维分布 (以 m, kg 为单位的这两维)



标准化: 对不同特征维度的伸缩变换的目的是使得不同度量之间的特征具有可比性。同时不改变原始数据的分布。

好处

- 1 使得不同度量之间的特征具有可比性, 对目标函数的影响体现在几何分布上, 而不是数值上
- 2 不改变原始数据的分布

数据在身高维度的分布 (体重类似)



逻辑回归

- 1、逻辑回归假设数据服从伯努利分布, 通过极大化似然函数的方法, 利用梯度下降求解参数, 来达到将数据二分类的目的。
- 2、逻辑回归的第二个假设是: 样本为正的的概率 $p = \frac{1}{1 + e^{-\theta^T x}}$, 逻辑回归即为 $h(x; \theta) = p$
- 3、逻辑回归的损失函数就是 $h(x; \theta)$ 的极大似然函数

$$L_{\theta}(x) = \prod_{i=1}^m h_{\theta}(x^i; \theta)^{y^i} * (1 - h_{\theta}(x^i; \theta))^{1 - y^i}$$

极大似然的核心思想是如果现有样本可以代表总体, 那么极大似然估计就是找到一组参数使得出现现有样本的可能性最大。

4、逻辑回归的求解方法：

- 由于该极大似然函数无法直接求解，我们一般通过对该函数进行梯度下降来不断逼近最优解。在这个地方其实会有个加分的项，考察你对其他优化方法的了解。因为就梯度下降本身来看的话就有随机梯度下降，批梯度下降，small batch 梯度下降三种方式，面试官可能会问这三种方式的优劣以及如何选择最合适的梯度下降方式。
 - 简单来说 批梯度下降会获得全局最优解，缺点是在更新每个参数的时候需要遍历所有的数据，计算量会很大，并且会有很多的冗余计算，导致的结果是当数据量大的时候，每个参数的更新都会很慢。
 - 随机梯度下降是以高方差频繁更新，优点是使得sgd会跳到新的和潜在更好的局部最优解，缺点是使得收敛到局部最优解的过程更加的复杂。
 - 小批量梯度下降结合了sgd和batch gd的优点，每次更新的时候使用n个样本。减少了参数更新的次数，可以达到更加稳定收敛结果，一般在深度学习当中我们采用这种方法。

5、逻辑回归的sigmoid的输出可以表示概率么？

sigmoid 本身不能赋予输出概率意义，而是 cross entropy 这样有概率意义的损失函数，才让输出可以解释成概率的

设在特征空间中，位于 x 附近的训练数据中正例所占的比例为 p ，模型在 x 处给出的输出为 y 。用 cross entropy 作为损失函数，位于 x 附近的这些训练数据，对损失函数的贡献就是：

$$L = -p \log y - (1 - p) \log(1 - y)$$

容易算出，使得 L 最小的 y 就等于 p ，也就是说模型的输出会等于在 x 附近取一个数据，它属于正类的概率。

6、逻辑回归为什么用sigmoid？

由广义线性模型模型做二分类，变量服从伯努利分布，最后推到可以得到sigmoid的形式，得出结论，当因变量服从伯努利分布时，广义线性模型就为逻辑回归。

SVM

1、SVM本质是在求支持向量到超平面的几何间隔的最大值

2、svm适合解决小样本、非线性、高纬度的问题

3、核函数：样本集在低维空间线性不可分时，核函数将原始数据映射到高维空间，或者增加数据维度，使得样本线性可分。

常用核函数：

线性核：不增加数据维度，而是余弦计算内积，提高速度

多项式核：增加多项式特征，提升数据维度，并计算内积

高斯核（默认BRF）：将样本映射到无限维空间，使原来不可分的样本线性可分

4、svm的目标函数（硬间隔）

有两个目标：第一个是使间隔最大化，第二个是使样本正确分类，由此推出目标函数：

$$\text{目标一（使间隔最大化）：} \min_{w,b} \frac{1}{2} \|w\|^2$$

$$\text{目标二（使样本正确分类）：} y_i (w^T x_i + b) \geq 1, i = 1, 2, \dots, m$$

$$\begin{aligned} \text{终极目标：} & \min_{w,b} \frac{1}{2} \|w\|^2 \\ \text{s.t.} & y_i (w^T x_i + b) \geq 1, \forall i \end{aligned}$$

这是一个有约束条件的最优化问题，用拉格朗日函数来解决

上式的拉格朗日函数为：

$$\begin{aligned} \min_{w,b} \max_{\alpha} L(w, b, \alpha) &= \frac{1}{2} \|w\|^2 + \sum_{i=1}^m \alpha_i (1 - y_i (w^T x_i + b)) \\ \text{s.t.} & \alpha_i \geq 0, \forall i \end{aligned}$$

5、svm软间隔

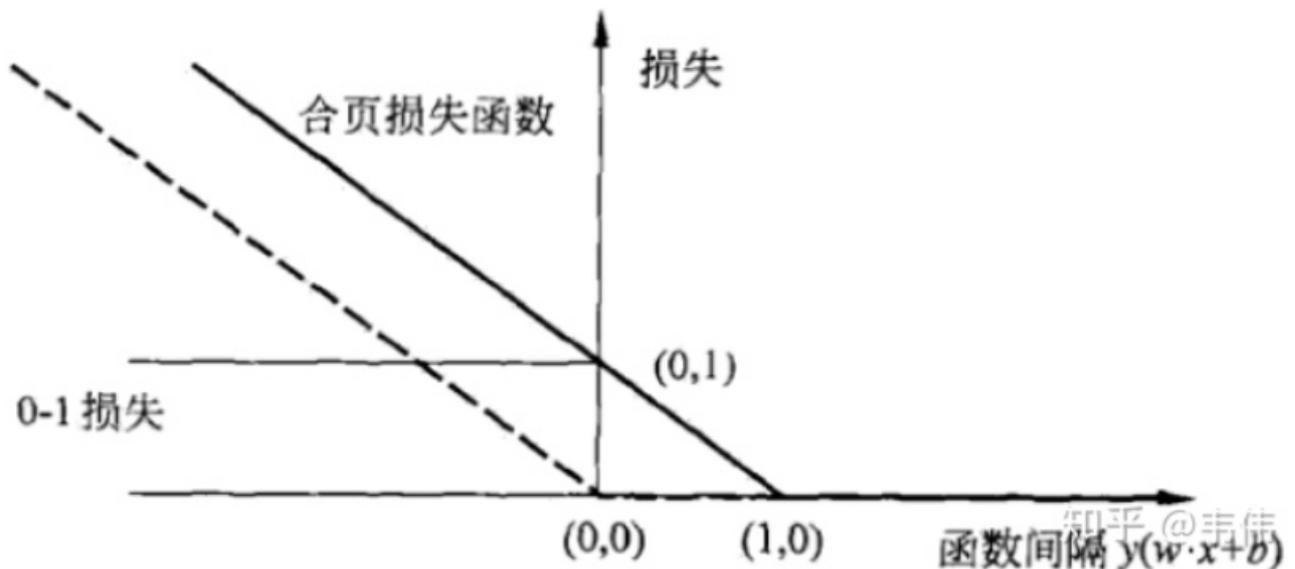
不管直接在原特征空间，还是在映射的高维空间，我们都假设样本是线性可分的。虽然理论上我们总能找到一个高维映射使数据线性可分，但在实际任务中，寻找一个合适的核函数核很困难。此外，由于数据通常有噪声存在，一味追求数据线性可分可能会使模型陷入过拟合，因此，我们放宽对样本的要求，允许少量样本分类错误。这样的想法就意味着对目标函数的改变，之前推导的目标函数里不允许任何错误，并且让间隔最大，现在给之前的目标函数加上一个误差，就相当于允许原先的目标出错，引入松弛变量 $\epsilon_i \geq 0$ ，公式变为：

$$\min_{w,b,\xi} \frac{1}{2} \|w\|^2 + \sum_{i=1}^n \xi_i$$

那么这个松弛变量怎么计算呢，最开始试图用0, 1损失去计算，但0, 1损失函数并不连续，求最值时求导的时候不好求，所以引入合页损失（hinge loss）：

$$l_{hinge}(z) = \max(0, 1 - z)$$

函数图张这样：



理解起来就是，原先约束条件是保证所有样本分类正确， $y_i (w^T x_i + b) \geq 1, \forall i$ ，现在出现错误的时候，一定是这个式子不被满足了，即 $y_i (w^T x_i + b) < 1, \forall i_{\text{错误}}$ ，衡量一下错了多少呢？因为左边一定小于1，那就跟1比较，因为1是边界，所以用1减去 $y_i (w^T x_i + b)$ 来衡量错误了多少，所以目标变为（正确分类的话损失为0，错误的话付出代价）：

$$\min_{w,b} \frac{1}{2} \|w\|^2 + \sum_{i=1}^n \max(0, 1 - y_i (w^T x_i + b))$$

但这个代价需要一个控制的因子，引入 $C > 0$ ，惩罚参数，即：

$$\min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \max(0, 1 - y_i (w^T x_i + b))$$

可以想象， C 越大说明把错误放的越大，说明对错误的容忍度就小，反之亦然。当 C 无穷大时，就变成一点错误都不能容忍，即变成硬间隔。实际应用时我们要合理选取 C ， C 越小越容易欠拟合， C 越大越容易过拟合。

所以软间隔的目标函数为：

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \\ \text{s.t.} \quad & y_i (x_i^T w + b) \geq 1 - \xi_i \\ & \xi_i \geq 0, i = 1, 2, \dots, n \end{aligned}$$

其中：

$$\xi_i = \max(0, 1 - y_i (w^T x_i + b))$$

对比SVM和LR

- LR与SVM的相同点：
 - 都是有监督的分类算法；
 - 如果不考虑核函数，LR和SVM都是线性分类算法。
 - 它们的分类决策面都是线性的。
 - LR和SVM都是判别式模型。
- LR与SVM的不同点：
 - 本质上是loss函数不同，或者说分类的原理不同。
 - SVM是结构风险最小化（经验+正则），LR则是经验风险最小化。
 - SVM只考虑分界面附近的少数点，而LR则考虑所有点。
 - 在解决非线性问题时，SVM可采用核函数的机制，而LR通常不采用核函数的方法。
 - SVM计算复杂，但效果比LR好，适合小数据集；LR计算简单，适合大数据集，可以在线训练。

损失函数

1、交叉熵损失函数

$$H(A, B) = - \sum_i P_A(x_i) \log(P_B(x_i))$$

在分类中，通常用cross entropy作为loss来优化网络，这里讨论一下公式及计算。

1. 交叉熵计算公式

$$CE(\hat{y}, y) = - \sum_{i=0}^{K-1} y_i \log(\hat{y}_i)$$

知乎 @鱼子酱

Notation:

- K : 一共有K个类别;
- \hat{y} : 预测的概率分布 (通常是Softmax的outputs, 是连续的), K 维;
- y : one-hot label, 真实的概率分布 (离散的), K 维。

概括: 总结起来就是每个样本的交叉熵loss等于真实类上的 $-\log(p)$ 。

2. 均方误差损失mean squared error

$$MSE = \frac{1}{n} \sum_i^n (\hat{y}_i - y_i)^2$$

优化器

1. 梯度下降

- 批量梯度下降

在每次更新时用所有样本，要留意，在梯度下降中，对于 θ 的更新，所有的样本都有贡献，也就是参与调整 θ 。其计算得到的是一个标准梯度，对于最优化问题，凸问题，也肯定可以达到一个全局最优。

因而理论上来说一次更新的幅度是比较大的。如果样本不多的情况下，当然是这样收敛的速度会更快啦。但是很多时候，样本很多，更新一次要很久

$$\theta_j := \theta_j + \alpha \sum_{i=1}^m (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)} \quad (\text{for every } j).$$

- 随机梯度下降

在每次更新时用1个样本，可以看到多了随机两个字，随机也就是说我们用样本中的一个例子来近似我所有的样本，来调整 θ ，因而随机梯度下降是会带来一定的问题，因为计算得到的并不是准确的一个梯度，**对于最优优化问题，凸问题**，虽然不是每次迭代得到的损失函数都向着全局最优方向，但是大的整体的方向是向全局最优解的，最终的结果往往是在全局最优解附近。但是相比于批量梯度，这样的方法更快，更快收敛，虽然不是全局最优，但很多时候是我们可以接受的

- 小批量梯度下降

在每次更新时用 b 个样本,其实批量的梯度下降就是一种折中的方法，他用了一些小样本来近似全部的，其本质就是 b 个指不定不太准，那我用个30个50个样本那比随机的要准不少了吧，而且批量的话还是非常可以反映样本的一个分布情况的。在深度学习中，这种方法用的是最多的，因为这个方法收敛也不会很慢，收敛的局部最优也是更多的可以接受！

2、引入动量

SGD方法中的高方差振荡使得网络很难稳定收敛，所以有研究者提出了一种称为动量（Momentum）的技术，通过优化相关方向的训练和弱化无关方向的振荡，来加速SGD训练。换句话说，这种新方法将上个步骤中更新向量的分量 γ 添加到当前更新向量。

在参数更新过程中，其原理类似：

1. 使网络能更优和更稳定的收敛；
2. 减少振荡过程。

当其梯度指向实际移动方向时，动量项 γ 增大；当梯度与实际移动方向相反时， γ 减小。这种方式意味着动量项只对相关样本进行参数更新，减少了不必要的参数更新，从而得到更快且稳定的收敛，也减少了振荡过程。

3、Adagrad

Adagrad方法是通过参数来调整合适的学习率 η ，对稀疏参数进行大幅更新和对频繁参数进行小幅更新。因此，Adagrad方法非常适合处理稀疏数据。

在时间步长中，Adagrad方法基于每个参数计算的过往梯度，为不同参数 θ 设置不同的学习率。

Adagrad方法的主要好处是，不需要手工来调整学习率。大多数参数使用了默认值0.01，且保持不变。

Adagrad方法的主要缺点是，学习率 η 总是在降低和衰减。

4、AdaDelta

解决了Adagrad学习率总是在降低衰减的缺点

AdaDelta方法的另一个优点是，已经不需要设置一个默认的学习率。

5、Adam

在之前的方法中计算了每个参数的对应学习率，但是为什么不计算每个参数的对应动量变化并独立存储呢？这就是Adam算法提出的改良点。

Adam算法即自适应时刻估计方法（Adaptive Moment Estimation），能计算每个参数的**自适应学习率**。同时也**结合了动量**

在实际应用中，Adam方法效果良好。与其他自适应学习率算法相比，其收敛速度更快，学习效果更为有效，而且可以纠正其他优化技术中存在的问题，如学习率消失、收敛过慢或是高方差的参数更新导致损失函数波动较大等问题。

Word2vec

1、负采样样本数量选择

一个单词被选作 negative sample 的概率跟它出现的频次有关，出现频次越高的单词越容易被选作negative words

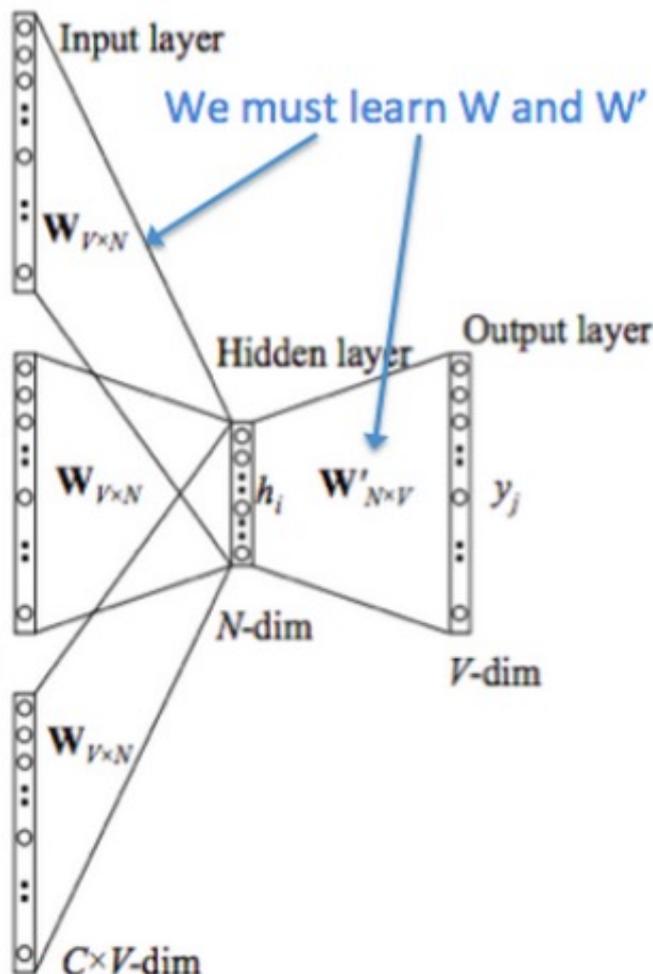
在论文中作者指出指出对于小规模数据集，建议选择 5-20 个 negative words，对于大规模数据集选择 2-5个 negative words.

为什么小规模的数据集要多采样一些？

那肯定咯，数据集规模越大，单词就越多，就上边这个例子，只要A没有挨着的单词都算负采样，就比如有1000个词，与A挨着的只有3个词，那A的负采样就有997个，但如果只有100个词，与A挨着的只有3个词，那A的负采样就有97个，这还是一个词的，那每个词都有那么多的负采样，不得呈指数增长呀

所以小规模的数据集的每个词的负采样可以适当增加一点，规模越大，负采样的个数尽量越少

2、CBOW的过程



- 输入层：上下文单词的onehot. {假设单词向量空间dim为V，上下文单词个数为C}
- 所有onehot分别乘以共享的输入权重矩阵W. {VN矩阵，N为自己设定的数，初始化权重矩阵W} $W^T x$ {NV * V*1}
- 所得的向量 {因为是onehot所以为向量} 相加求平均作为隐层向量, size为1*N.
- 乘以输出权重矩阵W' {N*V}
- 得到向量 {1*V} 激活函数处理得到V-dim概率分布 {PS: 因为是onehot嘛，其中的每一维斗代表着一个单

词}, 概率最大的index所指示的单词为预测出的中间词 (target word)

- 与true label的onehot做比较, 误差越小越好

3、skip-gram过程

第一步: 求 v_c

$v_c = Ww_c$, 这里 W 是 $d \times V$ 的矩阵, w_c 是 $V \times 1$ 的矩阵, 因此 v_c 是 $d \times 1$ 的向量。

直观理解: 矩阵与one-hot向量的内积, 相当于把one-hot向量中索引为1在向量矩阵中对应的那一列提取出来。

第二步: 求 $u_x^T v_c$ 组成的向量

这个向量就是 $W'v_c$, 这里的 W' 是 $V \times d$ 的矩阵, v_c 是 $d \times 1$ 的向量。因此 $u_x^T v_c$ 是 $V \times 1$ 的向量。

直观理解: 下面要说的很重要! 用 W' 与 v_c 相乘, 相当于 v_c 和词汇表中的所有词向量的转置都分别求内积, 得到的结果组成了一个向量!

第三步: 求softmax

这步比较简单, 把得到的相似度矩阵代入softmax公式, 就得到了一个满足概率分布的矩阵。

至此, 我们的目标已经实现: 得到了一个向量。

向量中的数值代表在给定单词的条件下, 其他单词出现的概率! 大功告成!

TF-IDF

TF-IDF(Term Frequency-Inverse Document Frequency, 词频-逆文件频率)是一种统计方法, 用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加, 但同时会随着它在语料库中出现的频率成反比下降。

上述引用总结就是, 一个词语在一篇文章中出现次数越多, 同时所有文档中出现次数越少, 越能够代表该文章。这也就是TF-IDF的含义。

TF(Term Frequency, 词频)表示词条在文本中出现的频率

$$TF_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}} \quad (1)$$

其中, $n_{i,j}$ 表示词条 t_i 在文档 d_j 中出现的次数, $TF_{i,j}$ 就是表示词条 t_i 在文档 d_j 中出现的频率。

权重的设计必须满足: 一个词预测主题的能力越强, 权重越大, 反之, 权重越小。

IDF(Inverse Document Frequency, 逆文件频率)**表示关键词的普遍程度

某一特定词语的IDF, 可以由总文件数目除以包含该词语之文件的数目, 再将得到的商取对数得到

$$IDF_i = \log \frac{|D|}{1 + |j : t_i \in d_j|} \quad (2)$$

其中, $|D|$ 表示所有文档的数量, $|j : t_i \in d_j|$ 表示包含词条 t_i 的文档数量, 为什么这里要加 1 呢? 主要是防止包含词条 t_i 的数量为 0 从而导致运算出错的现象发生。

某一特定文件内的高词语频率, 以及该词语在整个文件集合中的低文件频率, 可以产生出高权重的 TF-IDF。因此, TF-IDF 倾向于**过滤掉常见的词语, 保留重要的词语**, 表达为

$$TF-IDF = TF \cdot IDF \quad (3)$$

jieba分词

流程:

1. 依据统计词典 (模型中这部分已经具备, 也可自定义加载) 构建统计词典中词的前缀词典。

前缀词典构造如下, 它是将在统计词典中出现的每一个词的每一个前缀提取出来, 统计词频, 如果某个前缀词在统计词典中没有出现, 词频统计为 0

2. 依据前缀词典对输入的句子进行 DAG (有向无环图) 的构造。

以每个字所在的位置为键值 key, 相应划分的末尾位置构成的列表为 value

3. 使用动态规划的方法在 DAG 上找到一条概率最大路径, 依据此路径进行分词。

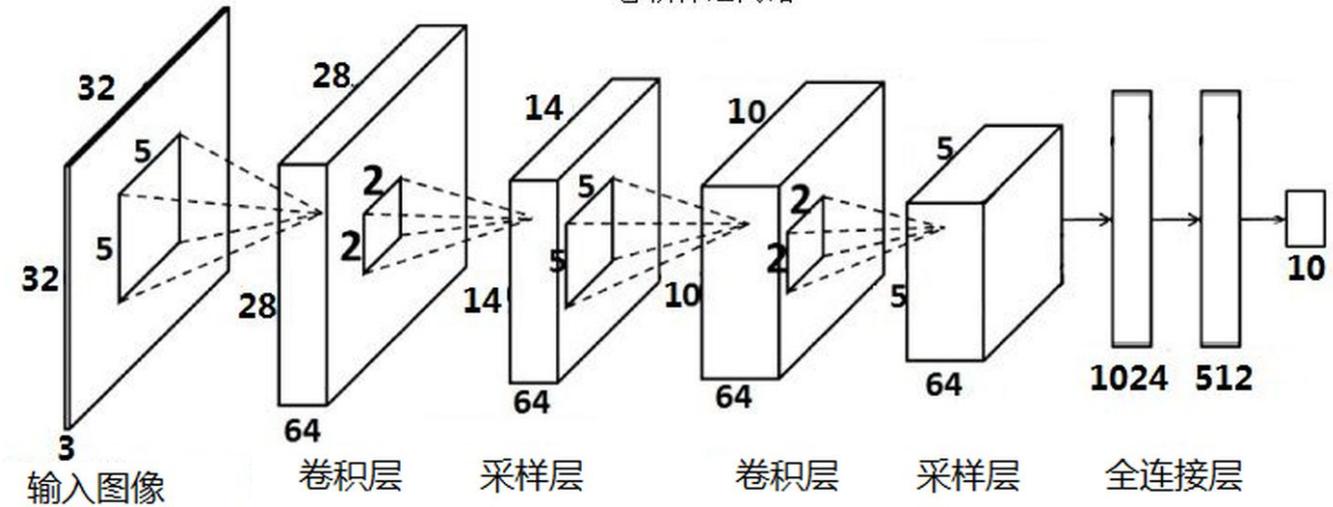
每一个词出现的概率等于该词在前缀里的词频除以所有词的词频之和。如果词频为 0 或是不存在, 当做词频为 1 来处理。

这里会取对数概率, 即在每个词概率的基础上取对数, 一是为了防止下溢, 二后面的概率相乘可以变成相加计算。

4. 对于未收录词 (是指不在统计词典中出现的词), 使用 HMM (隐马尔科夫模型) 模型, 用 Viterbi (维特比) 算法找出最可能出现的隐状态序列。

CNN

卷积神经网络



1、1*1卷积的主要作用有以下几点：

1) **降维 (dimension reductionality)**。比如，一张500 * 500且厚度depth为100 的图片在20个filter上做11的卷积，那么结果的大小为500*500*20。

2) **加入非线性**。卷积层之后经过激励层，1*1的卷积在前一层的学习表示上添加了非线性激励 (non-linear activation)，提升网络的表达能力；

2、各层作用

- 卷积层

1. 提取图像的特征，并且卷积核的权重是可以学习的，由此可以猜测，在高层神经网络中，卷积操作能突破传统滤波器的限制，根据目标函数提取出想要的特征。

2. “局部感知，参数共享”的特点大大降低了网络参数，保证了网络的稀疏性，防止过拟合，之所以可以“参数共享”，是因为样本存在局部相关的特性。

- 激活层

对卷积层的输出结果做一次非线性映射。

如果不用激励函数（其实就相当于激励函数是 $f(x)=x$ ），这种情况下，每一层的输出都是上一层输入的线性函数。容易得出，无论有多少神经网络层，输出都是输入的线性组合，与没有隐层的效果是一样的，这就是最原始的感知机了。

- 池化层（采样层、pooling）

pooling池化的作用则体现在降采样：保留显著特征、降低特征维度，增大kernel的感受野。另外一点值得注意：pooling也可以提供一些旋转不变性。

池化层可对提取到的特征信息进行降维，一方面使特征图变小，简化网络计算复杂度并在一定程度上避免过拟合的出现；一方面进行特征压缩，提取主要特征。

池化层的padding：padding可以认为是扩充图片，在图片周围补充一些像素点，把这些像素点初始化为0。

简历深挖

实体识别部分

RoBERTa+BiLSTM+Attention+CRF

- RoBERTa用于在当前数据集finetune输出WordEmbedding

- 为什么加BiLSTM?

因为BERT使用的是绝对位置编码，相当于一部分削弱了位置信息，在序列标注任务中位置信息是有必要的，而且方向信息也是有必要的，所以我们需要用BiLSTM去学习观测序列上的依赖关系

- Attention的作用

经过BiLSTM的隐向量已包含丰富的上下文特征信息，但这些特征具有相同的权重，在区分实体类别时会造成较大误差，字符之间的相关性利用权重 at 得到体现，通过字符的相关性对语句进行词边界划分

- CRF的作用

用于标签约束，CRF的特征函数是状态矩阵和转移矩阵，状态矩阵来自于BiLSTM的输出，即位置对应标签的分数，转移矩阵来自于CRF层是CRF层的参数。

CRF的学习函数是去学习CRF特征函数的权重

CRF损失函数由两部分组成，真实路径的分数和所有路径的总分数。真实路径的分数应该是所有路径中分数最高的。在训练过程中，BiLSTM-CRF模型的参数值将随着训练过程的迭代不断更新，使得真实路径所占的比值越来越大。

$$\text{\$LossFunction}=\frac{P\{\text{RealPath}\}}{P\{1\}+P\{2\}+\dots+P\{N\}}\text{\$}$$

预测算法采用维特比算法，设定dp去找当前时刻t的最优路径

为什么不用HMM?

HMM的假设：1) 其次马尔科夫假设：HMM的任意时刻t的某个状态只依赖于前一时刻的状态，与其他时刻的状态及观测无关，也与时刻t无关。2) 观测独立假设：任一时刻的观测只依赖于该时刻的马尔科夫链状态，与其他观测状态无关。

crf可以弥补hmm假设，hmm只能局部归一化，而crf可以全局归一化

煤矿实体30972重复，真正用到图谱里的有17690个节点，定义了2500种关系，3113个不重复实体

训练集4086条、验证集1139条、测试集1377条

89.41% 86.37% 87.87%

数据变换

bert层的输出 torch.Size([batch_size,seq_len,hidden_size]) --- [8,128,768]

bilstm层的输出

```
self.birnn = nn.LSTM(input_size=config.hidden_size, hidden_size=rnn_dim, num_layers=1,
bidirectional=True, batch_first=True)
```

```
[8,128,768] --> [8,128,256] shape=(batch_size,seq_length,num_directions*hidden_size)
```

```
sequence_output, _ = self.birnn(sequence_output)
```

全连接层:

```
self.hidden2tag = nn.Linear(out_dim, config.num_labels)
```

CRF:

```
self.crf = CRF(config.num_labels, batch_first=True)
```

其他方案:

BERT-MRC, 基于理解的实体识别, 实现对所有类别进行定义形成query, 对应的answer就是实体, 利用bert的nsp在句子前加入问题。

span的选择

有两种策略, 一种是两个n分类器, n为句子的长度, 来预测开始和结束的索引, 因为softmax分类器会选择最大可能性概率的结果作为实体的开始和结束。这样做的缺陷是在一句话中, 每个问题仅会提取出至多一个实体。另一种方法是, 用两个二分类器, 来预测每一个token是否是一个实体的开始或者结束, 这样的话就可以在一句话中一个问题对应多个实体。

关系抽取

可以简单说一下方案, pipeline式抽取和联合抽取

父类-核心概念 包含

父类-灾害事件 包含

核心概念-系统类 系统

核心概念-人员类 工种

核心概念-设备类 设备

核心概念-参数指标类 参数指标

核心概念-位置类 位置

人员类-灾害事件 引发事故

设备类-灾害事件 引发事故

人员类-设备类 操作

设备类-位置类 安装点

参数指标类-设备类 评价

位置类-系统 安装点

论文: Relational Memory-based Extraction for Relational Triples

主要分为三个模块: 1) RSE(Relation-specific Subject Extraction), 2)RM (Relational Memory) module construction 3) OE (Object Extraction)

解决的问题: 1) 嵌套实体识别 2) 实体对重叠三元组抽取 3) 但实体重叠三元组抽取

bert Embedding后是一个512*768的矩阵, 要对每一个位置进行二分类, 全连接转成512*100, reshape成512*502的矩阵, 对每个token进行二分类, 此时得到可得到两个向量头实体的起始位置和结束位置, 对应实体的语义向量用头实体+尾实体的embedding, 另外假如有50个关系, 关系矩阵随机初始化得到一个512*50的矩阵, 实体标注后, 在关系矩阵对应行找, 如果有1那就代表此实体对应应该关系, 此时已经找到头实体和对应关系, 尾实体和头实体的预测相似, 每次只传入一个头实体, 去找尾实体的开始和结束位置, 则此时便找到头尾实体及其对应关系

Category	NYT		DuIE	
	Train	Test	Train	Test
EPO	9782	978	173,084	1925
SEO	14735	1297	21,639	13331
Inner-NE	63	2	2360	287
Entra-NE	2	0	2367	290
All sentences	56195	5000	173084	21639

Method	NYT			DuIE		
	Prec	Rec	F1	Prec	Rec	F1
* NovelTagging	89.0	55.6	69.3	75.0	38.0	50.4
* GraphRel	82.5	57.9	68.1	41.1	25.8	31.8
* BiTT	89.7	88.0	88.9	75.7	80.6	78.0
* CasRel	89.6	88.8	89.2	81.5	77.8	79.6
Ours	88.2	90.0	89.1	81.5	81.3	81.4

聚类

采用的kmeans的方式，因为一开始只有标注没有类别，想利用这种无监督的方式去看看能不能聚到一起。

kmeans的过程：

1. 首先输入k的值，即我们希望将数据集经过聚类得到k个分组。
2. 从数据集中随机选择k个数据点作为初始大哥（质心，Centroid）
3. 对集合中每一个小弟，计算与每一个大哥的距离（距离的含义后面会讲），离哪个大哥距离近，就跟定哪个大哥。
4. 这时每一个大哥手下都聚集了一票小弟，这时候召开人民代表大会，每一群选出新的大哥（其实是通过算法选出新的质心）。
5. 如果新大哥和老大哥之间的距离小于某一个设置的阈值（表示重新计算的质心的位置变化不大，趋于稳定，或者说收敛），可以认为我们进行的聚类已经达到期望的结果，算法终止。
6. 如果新大哥和老大哥距离变化很大，需要迭代3~5步骤。

KNN k临近值算法

对未知类别属性的数据集中的每个点依次执行以下操作：

- (1) 计算已知类别数据集中的点与当前点之间的距离；
- (2) 按照距离递增次序排序；
- (3) 选取与当前点距离最小的k个点；
- (4) 确定前k个点所在类别的出现频率；
- (5) 返回前k个点出现频率最高的类别作为当前点的预测分类。

分类任务

专利

用IDCNN代替CNN

CNN的缺陷：尽管传统的CNN有明显的计算优势，但是传统的CNN在经过卷积之后，末梢神经元只能得到输入文本的一小部分信息，为了获取上下文信息，需要加入更多的卷积层，导致网络越来越深，参数越来越多，容易发生拟合。

IDCNN的改进：dilated convolutions，中文意思大概是“空洞卷积”。正常CNN的filter，都是作用在输入矩阵一片连续的位置上，不断sliding做卷积，接着通过pooling来整合多尺度的上下文信息，这种方式会损失分辨率。既然网络中加入pooling层会损失信息，降低精度。那么不加pooling层会使感受野变小，学不到全局的特征。如果我们单纯的去掉pooling层、扩大卷积核的话，这样纯粹的扩大卷积核势必导致计算量的增大，此时最好的办法就是Dilated Convolutions（扩张卷积或叫空洞卷积）。具体使用时，dilated width会随着层数的增加而指数增加。这样随着层数的增加，参数数量是线性增加的，而receptive field却是指数增加的，可以很快覆盖到全部的输入数据。在IDCNN中，通过重复使用自膨胀卷积的模块，来达到共享参数的目的以防止过拟合。对于IDCNN的输出也可以使用维特比算法来进行序列标签的预测。

知识图谱构建方法

- 知识建模：实体定义、关系定义、事件定义等，可以采用传统的rdf、owl等表示方法进行，也可采用现在比较常用的属性图进行，这两者的主要区别就是rdf基本都是通过三元组进行表示，而属性图可以在实体上有属性的概念。
- 知识获取：知识获取根据目标数据类型（包括结构化数据、非结构化数据、半结构化数据）不同采用的技术也有所不同，结构化数据比较容易一些，采用一些ETL工具等等就可以高质量的完成；半结构化数据可以采用爬虫技术+包装器+正则表达式技术进行，非结构化数据主要是通过自然语言处理的技术进行，包括对文本数据中的实体识别、关系识别、事件识别等。
- 知识融合：知识融合又分为模式层的融合以及数据层的融合，模式层的融合主要包括概念、概念的上下位、概念的属性这些统一；数据层的融合主要是将不同数据来源的数据的相同实体的不同表达形式进行融合，包括实体的合并、实体属性与关系的合并等。这一步工作涉及的技术有实体对齐、指代消解等。
- 知识存储：

决策树

1、决策树的生成算法

信息熵越大信息的纯度越低

数据的信息熵：
$$-\sum_{k=1}^K \frac{|C_k|}{|D|} \log_2 \frac{|C_k|}{|D|}$$
 Ck表示集合D中属于第k类样本的样本子集

信息增益=信息熵-条件熵

1) ID3：根据信息增益分类

步骤：

1. 初始化特征集合和数据集合；
2. 计算数据集合信息熵和所有特征的条件熵，选择信息增益最大的特征作为当前决策节点；

- 更新数据集合和特征集合（删除上一步使用的特征，并按照特征值来划分不同分支的数据集合）；
- 重复 2, 3 两步，若子集值包含单一特征，则为分支叶子节点。

缺点：

- ID3 没有剪枝策略，容易过拟合；
- 信息增益准则对可取值数目较多的特征有所偏好，类似“编号”的特征其信息增益接近于 1；
- 只能用于处理离散分布的特征；
- 没有考虑缺失值。

2) C4.5：根据信息增益率划分

剪枝策略：

预剪枝

在节点划分前来确定是否继续增长，及早停止增长的主要方法有：

- 节点内数据样本低于某一阈值；
- 所有节点特征都已分裂；
- 节点划分前准确率比划分后准确率高。

预剪枝不仅可以降低过拟合的风险而且还可以减少训练时间，但另一方面它是基于“贪心”策略，会带来欠拟合风险。

后剪枝

在已经生成的决策树上进行剪枝，从而得到简化版的剪枝决策树。

C4.5 采用的**悲观剪枝方法**，用递归的方式从低往上针对每一个非叶子节点，评估用一个最佳叶子节点去代替这课子树是否有益。如果剪枝后与剪枝前相比其错误率是保持或者下降，则这棵子树就可以被替换掉。C4.5 通过训练数据集上的错误分类数量来估算未知样本上的错误率。

后剪枝决策树的欠拟合风险很小，泛化性能往往优于预剪枝决策树。但同时其训练时间会大的多。

缺点：

- 剪枝策略可以再优化；
- C4.5 用的是多叉树，用二叉树效率更高；
- C4.5 只能用于分类；
- C4.5 使用的熵模型拥有大量耗时的对数运算，连续值还有排序运算；
- C4.5 在构造树的过程中，对数值属性值需要按照其大小进行排序，从中选择一个分割点，所以只适合于能够驻留于内存的数据集，当训练集大得无法在内存容纳时，程序无法运行。

3) CART：根据基尼指数分类，越小越好

$$Gini(D) = \sum_{k=1}^K p_k(1 - p_k) = 1 - \sum_{k=1}^K p_k^2$$

p_k 为 C_k 表示集合D中属于第k类样本的样本子集的概率

CART 在 C4.5 的基础上进行了很多提升。

- C4.5 为多叉树，运算速度慢，CART 为二叉树，运算速度快；
- C4.5 只能分类，CART 既可以分类也可以回归；

- CART 使用 Gini 系数作为变量的不纯度量，减少了大量的对数运算；
- CART 采用代理测试来估计缺失值，而 C4.5 以不同概率划分到不同节点中；
- CART 采用“基于代价复杂度剪枝”方法进行剪枝，而 C4.5 采用悲观剪枝方法。

CART 的一大优势在于：无论训练数据集有多失衡，它都可以将其子集消除不需要建模人员采取其他操作。

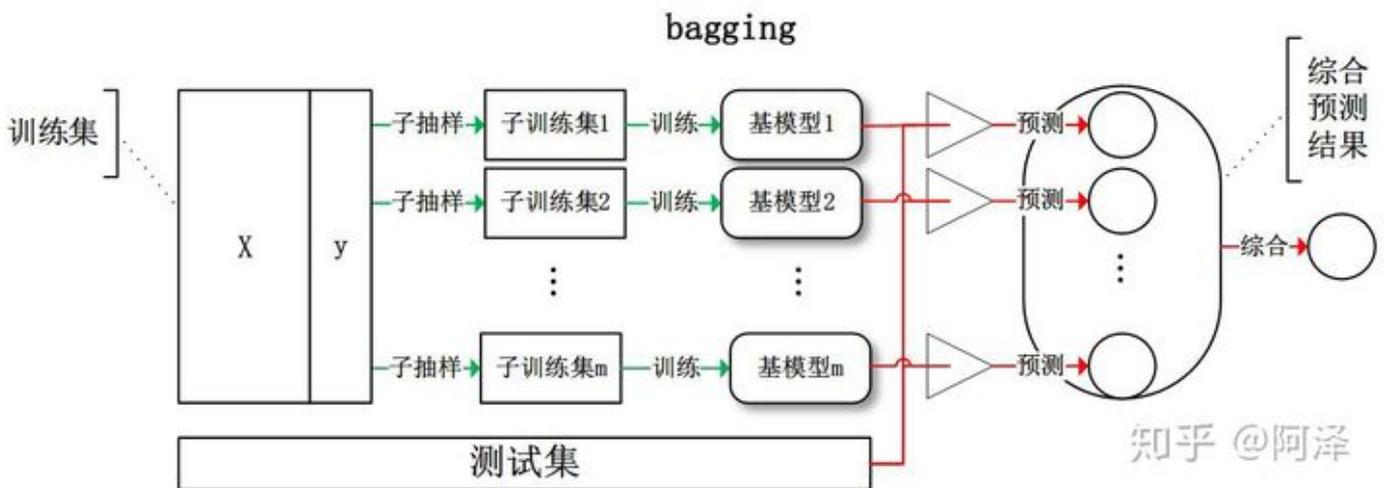
对比：

- **划分标准的差异：**ID3 使用信息增益偏向特征值多的特征，C4.5 使用信息增益率克服信息增益的缺点，偏向于特征值小的特征，CART 使用基尼指数克服 C4.5 需要 \log 的巨大计算量，偏向于特征值较多的特征。
- **使用场景的差异：**ID3 和 C4.5 都只能用于分类问题，CART 可以用于分类和回归问题；ID3 和 C4.5 是多叉树，速度较慢，CART 是二叉树，计算速度很快；
- **样本数据的差异：**ID3 只能处理离散数据且缺失值敏感，C4.5 和 CART 可以处理连续性数据且有多种方式处理缺失值；从样本量考虑的话，小样本建议 C4.5、大样本建议 CART。C4.5 处理过程中需对数据集进行多次扫描排序，处理成本耗时较高，而 CART 本身是一种大样本的统计方法，小样本处理下泛化误差较大；
- **样本特征差异：**ID3 和 C4.5 层级之间只使用一次特征，CART 可多次重复使用特征；
- **剪枝策略的差异：**ID3 没有剪枝策略，C4.5 是通过悲观剪枝策略来修正树的准确性，而 CART 是通过代价复杂度剪枝。

2、集成学习

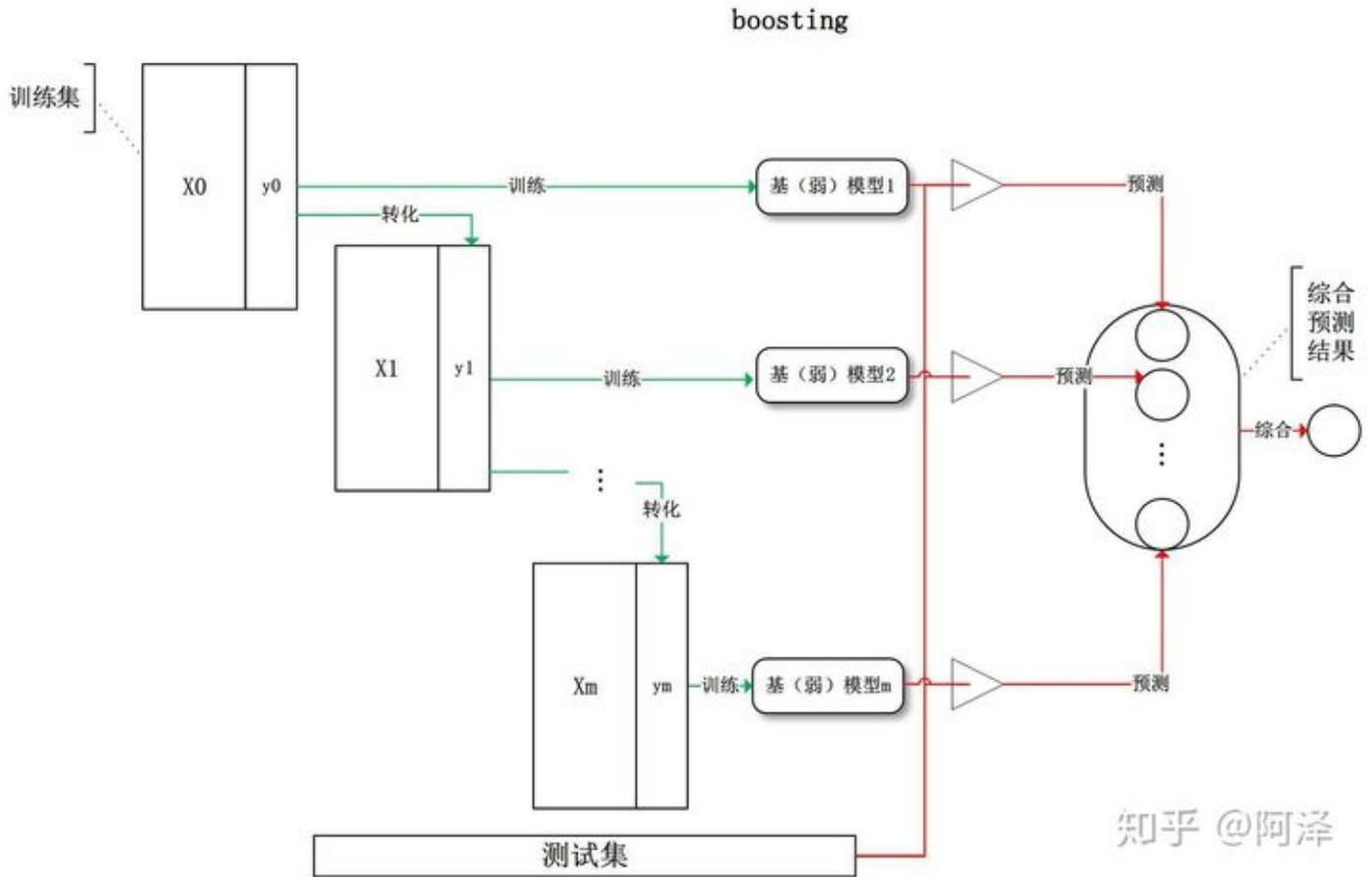
1) bagging(bootstrap aggregating)

每个基学习器都会对训练集进行有放回抽样得到子训练集，每个基学习器基于不同子训练集进行训练，并综合所有基学习器的预测值得到最终的预测结果。Bagging 常用的综合方法是投票法，票数最多的类别为预测类别。



2)boosting

基模型的训练是有顺序的，每个基模型都会在前一个基模型学习的基础上进行学习，最终综合所有基模型的预测值产生最终的预测结果，用的比较多的综合方式为加权法



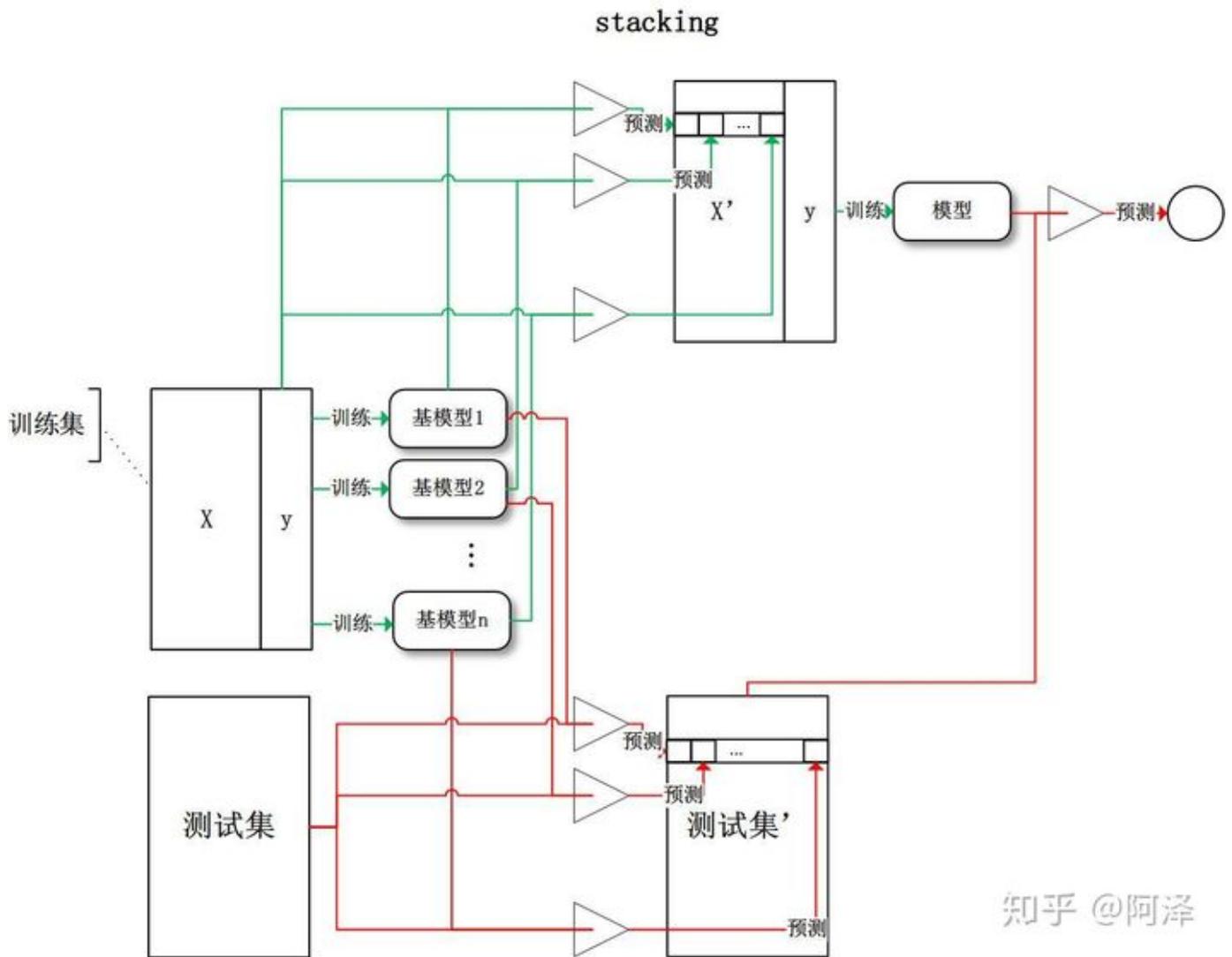
3) Stacking

并通过训练一个元模型来组合它们，然后基于这些弱模型返回的多个预测结果形成新的训练集，通过新的训练集去训练组合模型得到新模型去预测。其中用了k-折交叉训练

假设我们想要拟合由 L 个弱学习器组成的 stacking 集成模型。我们必须遵循以下步骤：

- 将训练数据分为两组
- 选择 L 个弱学习器，用它们拟合第一组数据
- 使 L 个学习器中的每个学习器对第二组数据中的观测数据进行预测
- 在第二组数据上拟合元模型，使用弱学习器做出的预测作为输入

在前面的步骤中，我们将数据集一分为二，因为对用于训练弱学习器的数据的预测与元模型的训练不相关。因此，将数据集分成两部分的一个明显缺点是，我们只有一半的数据用于训练基础模型，另一半数据用于训练元模型。



知乎 @阿泽

为什么集成学习会好于单个学习器呢？

1. 训练样本可能无法选择出最好的单个学习器，由于没法选择出最好的学习器，所以干脆结合起来一起用；
2. 假设能找到最好的学习器，但由于算法运算的限制无法找到最优解，只能找到次优解，采用集成学习可以弥补算法的不足；
3. 可能算法无法得到最优解，而集成学习能够得到近似解。比如说最优解是一条对角线，而单个决策树得到的结果只能是平行于坐标轴的，但是集成学习可以去拟合这条对角线。

Bagging 和 **Stacking** 中的基模型为强模型（偏差低，方差高），而**Boosting** 中的基模型为弱模型（偏差高，方差低）。

3、随机森林 = Bagging+CART决策树

步骤：

1. 随机选择样本（放回抽样）；
2. 随机选择特征；
3. 构建决策树；
4. 随机森林投票（平均）。

随机采样由于引入了两种采样方法保证了随机性，所以每棵树都是最大可能的进行生长就算不剪枝也不会出现过拟合。

优点

1. 在数据集上表现良好，相对于其他算法有较大的优势
2. 易于并行化，在大数据集上有很大的优势；
3. 能够处理高维度数据，不用做特征选择。

4、Adaboost

前一个基本分类器分错的样本会得到加强，加权后的全体样本再次被用来训练下一个基本分类器。同时，在每一轮中加入一个新的弱分类器，直到达到某个预定的足够小的错误率或达到预先指定的最大迭代次数。

Adaboost 迭代算法有三步：

1. 初始化训练样本的权值分布，每个样本具有相同权重；
2. 训练弱分类器，如果样本分类正确，则在构造下一个训练集中，它的权值就会被降低；反之提高。用更新过的样本集去训练下一个分类器；
3. 将所有弱分类组合成强分类器，各个弱分类器的训练过程结束后，加大分类误差率小的弱分类器的权重，降低分类误差率大的弱分类器的权重。

优点

1. 分类精度高；
2. 可以用各种回归分类模型来构建弱学习器，非常灵活；
3. 不容易发生过拟合。

缺点

1. 对异常点敏感，异常点会获得较高权重

5、GDBT = Gradient boosting + CART 决策树

GDBT 由三个概念组成：Regression Decision Tree（即 DT）、Gradient Boosting（即 GB），和 Shrinkage（一个重要演变）

- 1) 回归树：GDBT 中的树都是回归树
- 2) 梯度迭代：GDBT 的每一步残差计算其实变相地增大了被分错样本的权重，而对与分对样本的权重趋于 0，这样后面的树就能专注于那些被分错的样本
- 3) 缩减：不直接用残差修复误差，而是只修复一点点，把大步切成小步。

优点

1. 可以自动进行特征组合，拟合非线性数据；
2. 可以灵活处理各种类型的数据。

缺点

1. 对异常点敏感。

与Adaboost的对比

相同：

1. 都是 Boosting 家族成员，使用弱分类器；
2. 都使用前向分布算法；

不同:

1. **迭代思路不同**: Adaboost 是通过提升错分数据点的权重来弥补模型的不足 (利用错分样本), 而 GBDT 是通过算梯度来弥补模型的不足 (利用残差);
2. **损失函数不同**: AdaBoost 采用的是指数损失, GBDT 使用的是绝对损失或者 Huber 损失函数;

6、XgBoost

XGBoost 是大规模并行 boosting tree 的工具, 它是目前最快最好的开源 boosting tree 工具包

优点

1. **精度更高**: GBDT 只用到一阶泰勒展开, 而 XGBoost 对损失函数进行了二阶泰勒展开。XGBoost 引入二阶导一方面是为了增加精度, 另一方面也是为了更好地自定义损失函数, 二阶泰勒展开可以近似大量损失函数;
2. **灵活性更强**: GBDT 以 CART 作为基分类器, XGBoost 不仅支持 CART 还支持线性分类器, (使用线性分类器的 XGBoost 相当于带 L1 和 L2 正则化项的逻辑斯蒂回归 (分类问题) 或者线性回归 (回归问题))。此外, XGBoost 工具支持自定义损失函数, 只需函数支持一阶和二阶求导;
3. **正则化**: XGBoost 在目标函数中加入了正则项, 用于控制模型的复杂度。正则项里包含了树的叶子节点个数、叶子节点权重的 L2 范式。正则项降低了模型的方差, 使学习出来的模型更加简单, 有助于防止过拟合;
4. **Shrinkage (缩减)**: 相当于学习速率。XGBoost 在进行完一次迭代后, 会将叶子节点的权重乘上该系数, 主要是为了削弱每棵树的影响, 让后面有更大的学习空间;
5. **列抽样**: XGBoost 借鉴了随机森林的做法, 支持列抽样, 不仅能降低过拟合, 还能减少计算;
6. **缺失值处理**: XGBoost 采用的稀疏感知算法极大的加快了节点分裂的速度;
7. **可以并行化操作**: 块结构可以很好的支持并行计算。

缺点

1. 虽然利用预排序和近似算法可以降低寻找最佳分裂点的计算量, 但在节点分裂过程中仍需要遍历数据集;
2. 预排序过程的空间复杂度过高, 不仅需要存储特征值, 还需要存储特征对应样本的梯度统计值的索引, 相当于消耗了两倍的内存。

SQL

```
#!/bin/bash
begin_dt=20210419
end_dt=20210509
hive -e "select uid,count(distinct mid) as send_num
from universal_material_db
where dt>=${begin_dt} and dt<=${end_dt}
group by uid" > uid_send_num
hive -e "select uid,count(distinct mid) as pt_num
from universal_material_db
where dt>=${begin_dt} and dt<=${end_dt} and pic_num>=1
group by uid" > uid_pt_num
hive -e "add file deal_pic_ocr.py;
select transform(uid,mid,son_data,dt) using 'python deal_pic_ocr.py' as
(uid,mid,pic_num,high_ocr)
from universal_material_db
where dt>=${begin_dt} and dt<=${end_dt}" > uid_pic_ocr
```

```
hive -e "select a.uid,c.tag_id,count(distinct a.rootmid) from
(select distinct uid,rootmid from mds_bhv_pubblog where dt>=20201201 and dt<=20210107
and is_transmit=1) a
join
(select distinct mid from new_mblog_mannual_mark_result where deleted=0 and
to_id_type=2 and dubious=0 and dislike=0 and risk=0 and low_quality=0 and dt>=20201201
and dt<=20201230) b
on a.rootmid=b.mid
join
(select distinct mid,tag_id from mblog_newtags_recall_history_new where dt>=20201201
and dt<=20210107 and tag_id like '1042015:newTagCategory%') c
on a.rootmid=c.mid group by a.uid,c.tag_id" > fenzi_firstag
```

计算机基础

1、进程间的通信方式：

管道 (pipe)：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。

有名管道 (named pipe)：有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。

消息队列 (message queue)：消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

信号 (sinal)：信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

信号量 (semaphore)：信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

共享内存 (shared memory)：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号量，配合使用，来实现进程间的同步和通信。

套接字 (socket)：套接口也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同及其间的进程通信。

2、数据库事务acid

ACID，是指[数据库管理系统 \(DBMS\)](#) 在写入或更新资料的过程中，为保证[事务 \(transaction\)](#) 是正确可靠的，所必须具备的四个特性：[原子性 \(atomicity\)](#)，或称不可分割性)、[一致性 \(consistency\)](#)、[隔离性 \(isolation\)](#)，又称独立性)、[持久性 \(durability\)](#)

3、



4、进程与线程

进程：是cpu调度的基本单位，是系统进行资源分配和运行调度的独立单位；进程在启动时都会最先产生一个线程，这个线程被称为主线程，然后再有主线程创建其他子线程

线程：是操作系统中进行运算调度的最小单位，包含在进程中，是进程中的实际运作单位；一个进程可以有多个线程，每条线程可以同时执行不同的任务

python中的gil (Global Interpreter Lock 的缩写)。顾名思义，就是 Python 解释器的一个全局锁，Python 的多线程在同一时刻只能有一个线程在运行。多线程情况下就是线程不停地在抢锁，抢得头破血流。

5、python变量

不可变类型：字符串、数值型、布尔型

可变类型：列表、字典、集合

集合是set(), 元组也是不可变对象

6、三次握手

欢迎加入代码随想录知识星球

// 一起抱团取暖

[点击进入](#)