**深度学习基本问题：**

1. **什么是深度学习？什么是迁移学习？什么是强化学习？**

**机器学习**：定义一系列函数 =>通过训练集筛选出效果好的函数 =>选出效果最好的函数

**深度学习**：定义网络结构 =>通过训练集筛选出效果好的函数 =>选出效果最好的函数

深度学习实际上是学习了样本分布的规律，所以要求测试集和训练集有同分布

**迁移学习**：就是把已学训练好的模型参数迁移到新的模型来帮助新模型训练。考虑到大部分数据或任务是存在相关性的，所以通过迁移学习我们可以将已经学到的模型参数通过某种方式来分享给新模型从而加快并优化模型的学习效率不用像大多数网络那样从零学习

**强化学习**：RL采用的是边获得样例边学习的方式，在获得样例之后更新自己的模型，利用当前的模型来指导下一步的行动，下一步的行动获得reward之后再更新模型，不断迭代重复直到模型收敛。在这个过程中，非常重要的一点在于“在已有当前模型的情况下，如果选择下一步的行动才对完善当前的模型最有利”，这就涉及到了RL中的两个非常重要的概念：探索（exploration）和开发（exploitation），exploration是指选择之前未执行过的actions，从而探索更多的可能性；exploitation是指选择已执行过的actions，从而对已知的actions的模型进行完善

1. **CNN、RNN、LSTM？**

CNN：卷积神经网络，用于图像处理

RNN：循环神经网络，具有记忆的神经网络。隐层的输出被存储到记忆里，记忆可以作为另一个输入

LSTM：Long Short-term Memory具有特别的神经元，4个输入，1个输出



1. **Softmax函数作为输出层的用法？**

**Softmaxt函数的公式是：**

其中ezj是当前神经元输出的zj进行取指得到的值，分母为：当前层所有神经元输出z的取指和，换句话说就是：把一堆实数的值映射到0-1区间，并且使他们的和为1

softmax经常在神经网络中代替sigmoid用于输出层上，通过结果得到的0-1区间的值代表概率来判断谁更可能是符合的输出

softmax直观理解：max是a>b一定取a，没有比别的选择。而在神经网络中有时候我们并不想这样，因为这样会造成分小的那个值的饥饿，我们希望分小的那个值在小概率的情况下仍然会被取到，这个时候我们就用到了softmax。在softmax中值大的会大概率被取到，也经常被取到，取到的次数便多，而值小的也会大概率被取到。而这个概率与值本身和该层各个值有关。

而使用指数的原因：，第一个原因是要模拟max的行为，所以要让大的更大。第二个原因是需要一个可导的函数。让大的更大的原因是让错的更错，这样学习效率更高



1. **深度学习的学习目标？**

最小化所有训练样本的总体损失，得到模型的参数（权值和偏置）

1. **相同参数数量时，宽网络和深度网络哪个好？为什么？**

多层的网络结构表达一些函数时更加简单

1. **深度网络中的Modularization？**

每个模块只解决一个基本问题，并作为模块被后面的分类器继承，这样每个基本分类器都有足够的训练样本，模型可以由少量的数据进行训练

1. **怎么做可以提高使模型在训练数据集上取得好的结果？5个要点**

合适的损失函数、mini-batch、激活函数、自适应的学习率、momentum

1. **为什么softmax作为输出层时，使用交叉熵损失函数比平方损失函数效果好？**

平方损失函数求导时候回陷入饱和区，距离期望值越远，参数更新速度越慢；

交叉熵损失函数不会受到饱和性的影响，而且误差越大，参数更新越快

<https://blog.csdn.net/u014595019/article/details/52562159>

1. **交叉熵损失函数及求导？**

交叉熵损失函数的求导：

交叉熵损失函数为





<https://blog.csdn.net/jasonzzj/article/details/52017438>

1. **Mini-batch是怎么做的？为什么效果更好？**

训练时并不是最小化所用样本的总损失，而是把总样本随机分成若干个batch，用每个batch中的样本训练网络、更新参数。所有的样本都用过一遍，称为一个epoch，每个epoch后对样本进行一次shuffle，在进行下一个epoch

效果更好的原因：每个epoch中进行了多次的更新参数，训练更快；每次训练不同的样本，避免学习到样本之间的相关性，从而发生过拟合

但是，mini-batch更新参数的过程是不稳定的，因为一个batch中的样本可能和总体的分布不同

1. **为什么网络不是越深越好？**

反向传播过程中，靠近输出的层有很大的梯度、学习的速度快；而前面的层梯度小，学习速度很慢（梯度消失）

1. **激活函数都有哪些？**

激活函数的主要作用：提供网络的非线性建模能力，如没有，则只能表达线性映射

类型：分段线性和具有指数形状的非线性函数

Sigmiod、tanh、ReLU、ELU、Maxout

<https://blog.csdn.net/u014595019/article/details/52562159>

1. **激活函数应具备的性质：**

可微性：当优化方法是基于梯度的时候，这个性质是必须的

单调性：当激活函数是单调的时候，单层网络能够保证是凸函数

输出值的范围：当激活函数输出值是有限的时候，基于梯度的优化方法会更加稳定，因为特征的表示受有限权值的影响更显著;当激活函数的输出是无限的时候，模型的训练会更加高效，不过在这种情况小，一般需要更小的learning rate

1. **Sigmoid函数的优缺点？**

缺：a. 软饱和性，梯度消失

b. 偏移现象：输出均大于0，不是0均值

优：(1) 具有指数函数形状，在物理意义上最接近生物神经元

(2) (0,1)的输出可以被表示作概率，或用于输入的归一化

1. **ReLU为什么效果好？有什么优缺点？**

优：X>0时能够保持梯度不衰减，从而缓解梯度消失问题

缺：神经元死亡、偏移现象

1. **参数优化学习算法都有哪些？**

SGD算法的缺点：

1. 很难选择合适的学习率 (2) 所有的参数学习率相同 (3) 可能会陷入局部极小点

优化算法学习率的设定思想：

1. 每隔若干个epoch减小学习率 (2) 为不同的参数设置不同的学习率

优化算法：

<https://blog.csdn.net/muyu709287760/article/details/62531509>

1. **为什么要更新学习率？怎么更新？为什么Adagrad算法中梯度越大，学习率越小？**
2. 学习率太大，可能会跳过全局最小；学习率太小，模型训练太慢
3. 开始时，距离目标较远，学习率设置大一点；后面每隔几个epoch减小学习率
4. 梯度大，如学习率大，则容易错过最小点（回忆等高线图）
5. **怎么理解Momentum？**

利用了物理学中动量的思想，通过积累之前的动量(mt−1)来加速当前的梯度。

mt=μ∗mt−1+η∇θJ(θ) θt=θt−1−mt

其中，μ是动量因子，通常被设置为0.9或近似值。

特点：

1. 参数下降初期，加上前一次参数更新值；如果前后2次下降方向一致，乘上较大的μ能够很好的加速。
2. 参数下降中后期，在局部最小值附近来回震荡时，gradient→0gradient→0，μμ使得更新幅度增大，跳出陷阱。
3. 在梯度方向改变时，momentum能够降低参数更新速度，从而减少震荡；在梯度方向相同时，momentum可以加速参数更新， 从而加速收敛。

总而言之，momentum能够加速SGD收敛，抑制震荡。

1. **怎样在测试集上获得好的结果？4个要点**

早停、正则化、Dropout、网络结构

1. **为什么会过拟合？怎么防止？Data augmentation的方法？**
2. 训练数据与测试数据可能不同，而学习目标是由训练数据确定的
3. 数据增强、早停(验证集)、weight decay、dropout、网络结构
4. flip、filter、crop、色彩<https://blog.csdn.net/Yaphat/article/details/54098867>
5. **神经网络中的正则化是怎么实现的？**

Weight Decay：减掉神经元之间的无用的连接

Weight Decay对应着L2正则化

<http://blog.sina.com.cn/s/blog_a89e19440102x1el.html>

1. **Dropout是怎么实现的？反向传播时一样吗？**

通过改变网络本身的结构来实现的

在训练开始时，随机删除一些隐藏层神经元，同时保持输入层与输出层神经元的个数不变，反向传播时只更新剩余神经元之间的权重

为什么减轻过拟合？

随着神经网络模型不断地学习，神经元的权值会与整个网络的上下文相匹配。神经元的权重针对某些特征进行调优，具有一些特殊化。周围的神经元则会依赖于这种特殊化，如果过于特殊化，模型会因为对训练数据过拟合而变得脆弱不堪。

1. **为什么CNN适合图像处理？3个特点**
2. 一些特征比整张图片小很多，不必全览整张图片来发现特征，可以用少量的参数连接小区域；
3. 相同的特征会出现在不同的区域，可以用相同的参数来做同样的检测；
4. 下采样像素不会改变目标，片可以用少量的参数来处理

前两个特点使得我们可以使用卷积操作，第三个特点对应Max Pooling

1. **在全连接之前Feature Map做了什么处理？**

Flatten 把多维矩阵展开为一维线性向量，以保证全连接的实现

1. **图像尺寸n、kernel尺寸k、步长stride、zero pad值、feature map尺寸的关系？**

F = (n – k + 2\*pad) / stride + 1

1. **CNN网络最重要的特点？**

参数共享

**27.Batch Normalization？**

目标：解决internal covariate shift（<https://www.cnblogs.com/bonelee/p/8528722.html>）问题，网络参数的不断改变导致每一层的输入分布也发生变化，而学习的过程又要使每一层适应输入的分布，导致模训练困难

作用：加速训练收敛速度，可以起到一定的正则化作用。可以代替Dropout和LRN层

把越来越偏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，意思是

这样让梯度变大，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。

位置：原文说应该把BN放在激活函数之前，因为Wx+b具有更加一致和非稀疏的分布。

但是也有人做实验表明放在激活函数后面效果更好

如何实现的：

1. 某个层的输出限制在均值为0方差为1的分布会使得网络的表达能力变弱。相当于只做了线性变换
2. 因此给batch normalization层进行一些限制的放松，给它增加两个可学习的参数 β 和 γ，对数据进行缩放和平移，平移参数 β 和缩放参数 γ 是学习出来的。

即每个BN层的γ和β参数都有不同的取值，服从不同的正态分布？

问题：训练时使用的是mini-batch，可以计算均值和方差，但是测试时只有一个数据，BN层怎么做的？

解决方法1：使用所有训练样本的均值和方差(原文的做法)

解决方法2：基于momentum的指数衰减

running\_mean = momentum \* running\_mean + (1 - momentum) \* sample\_mean

running\_var = momentum \* running\_var + (1 - momentum) \* sample\_var



**28.数据增强怎么做？**

**旋转 | 反射变换(Rotation/reflection):** 随机旋转图像一定角度; 改变图像内容的朝向;

**翻转变换(flip):** 沿着水平或者垂直方向翻转图像;

**缩放变换(zoom):** 按照一定的比例放大或者缩小图像;

**平移变换(shift):** 在图像平面上对图像以一定方式进行平移; 可以采用随机或人为定义的方式指定平移范围和平移步长, 沿水平或竖直方向进行平移. 改变图像内容的位置;

**尺度变换(scale):** 对图像按照指定的尺度因子, 进行放大或缩小; 或者参照SIFT特征提取思想, 利用指定的尺度因子对图像滤波构造尺度空间. 改变图像内容的大小或模糊程度;

**对比度变换(contrast):** 在图像的HSV颜色空间，改变饱和度S和V亮度分量，保持色调H不变. 对每个像素的S和V分量进行指数运算(指数因子在0.25到4之间), 增加光照变化;

**噪声扰动(noise):** 对图像的每个像素RGB进行随机扰动, 常用的噪声模式是椒盐噪声和高斯噪声;

颜色变换(color): 在训练集像素值的RGB颜色空间进行PCA, 得到RGB空间的3个主方向向量,3个特征值, p1, p2, p3, λ1, λ2, λ3.

**29.学习率怎么调整，一般你设置多少？**

学习率设置过小，收敛速度会非常慢，学习率设置过大，则会越过最低点，无法达到最低点。因此选择lr，也就是不断试的过程，基本范围大概就是0.1,0.01,0.001,0.0001这样子，一个数量级一个数量级的尝试就可以了。一般设置0.01或0.001。

**30.迁移学习加L2正则化的好处是什么？**

1. 正则化的目的：防止过拟合！

2. 正则化的本质：约束（限制）要优化的参数。

关于第1点，过拟合指的是给定一堆数据，这堆数据带有噪声，利用模型去拟合这堆数据，可能会把噪声数据也给拟合了，这点很致命，一方面会造成模型比较复杂（想想看，本来一次函数能够拟合的数据，现在由于数据带有噪声，导致要用五次函数来拟合，多复杂！），另一方面，模型的泛化性能太差了（本来是一次函数生成的数据，结果由于噪声的干扰，得到的模型是五次的），遇到了新的数据让你测试，你所得到的过拟合的模型，正确率是很差的。

关于第2点，本来解空间是全部区域，但通过正则化添加了一些约束，使得解空间变小了，甚至在个别正则化方式下，解变得稀疏了。这一点不得不提到一个图，相信我们都经常看到这个图，但貌似还没有一个特别清晰的解释

https://blog.csdn.net/wsj998689aa/article/details/39547771

**31.triplet loss不易收敛怎么训练？**

在有监督的机器学习领域，通常有固定的类别，这时就可以使用基于softmax的交叉熵损失函数进行训练。但有时，类别是一个变量，此时使用triplet loss就能解决问题。在人脸识别，Quora question pair任务中，triplet loss的优势在于细节区分，即当两个输入相似时，triplet loss能够更好地对细节进行建模，相当于加入了两个输入差异性差异的度量，学习到输入的更好表示，从而在上述两个任务中有出色的表现。当然，triplet loss的缺点在于其收敛速度慢，有时不收敛。  
链接：https://www.jianshu.com/p/d41b6447743d  
https://www.cnblogs.com/Alex0111/p/8492471.html

**32.loss出现nan怎么办？**

一、出现梯度爆炸了

1、数据归一化（减均值，除方差，或者加入normalization，例如BN、L2 norm等）；  
2、更换参数初始化方法（对于CNN，一般用xavier或者msra的初始化方法）；  
3、减小学习率、减小batch size；  
4、加入gradient clipping；

二、大致的解决办法就是，在出现Nan值的loss中一般是使用的TensorFlow的log函数，然后计算得到的Nan，一般是输入的值中出现了负数值或者0值，在TensorFlow的官网上的教程中，使用其调试器调试Nan值的出现，也是查到了计算log的传参为0；而解决的办法也很简单，假设传参给log的参数为y，那么在调用log前，进行一次数值剪切，修改调用如下：

loss = tf.log(tf.clip\_by\_value(y,1e-8,1.0))

这样，y的最小值为0的情况就被替换成了一个极小值，1e-8，这样就不会出现Nan值了，StackOverflow上也给出了相同的解决方案。于是，我就采用了上述的解决方案对于log的参数进行数值限制，但是我更加复杂化了这个限制。

<https://blog.csdn.net/qq_32458499/article/details/79468426>

**33.模型压缩你是怎么做的？**

更精细模型的设计，目前的很多网络都具有模块化的设计，在深度和宽度上都很大，这也造成了参数的冗余很多，因此有很多关于模型设计的研究，如SqueezeNet、MobileNet等，使用更加细致、高效的模型设计，能够很大程度的减少模型尺寸，并且也具有不错的性能。

模型裁剪，结构复杂的网络具有非常好的性能，其参数也存在冗余，因此对于已训练好的模型网络，可以寻找一种有效的评判手段，将不重要的connection或者filter进行裁剪来减少模型的冗余。

核的稀疏化，在训练过程中，对权重的更新进行诱导，使其更加稀疏，对于稀疏矩阵，可以使用更加紧致的存储方式，如CSC，但是使用稀疏矩阵操作在硬件平台上运算效率不高，容易受到带宽的影响，因此加速并不明显。

除此之外，量化、Low-rank分解、迁移学习等方法也有很多研究，并在模型压缩中起到了非常好的效果。

原文：https://blog.csdn.net/wspba/article/details/75671573

**34.你认为模型提速最有效的方法是？**

https://blog.csdn.net/QcloudCommunity/article/details/77719498

项目：<https://blog.csdn.net/tsyccnh/article/details/78889838>

**35.准确率、精确率和召回率**

假设我们手上有60个正样本，40个负样本，我们要找出所有的正样本，系统查找出50个，其中只有40个是真正的正样本，计算上述各指标。

TP: 将正类预测为正类数 40

FN: 将正类预测为负类数 20

FP: 将负类预测为正类数 10

TN: 将负类预测为负类数 30

准确率(accuracy) = 预测对的/所有 = (TP+TN)/(TP+FN+FP+TN) = 70%

精确率(precision) = TP/(TP+FP) = 80%

召回率(recall) = TP/(TP+FN) = 2/3

**36.训练集、验证集和测试集**

把数据集随机分为训练集，验证集和测试集，然后用训练集训练模型，用验证集验证模型，根据情况不断调整模型，选择出其中最好的模型，再用训练集和验证集数据训练出一个最终的模型，最后用测试集评估最终的模型

**37.深度学习训练的技巧**

使用一个小数据集可以使训练测试循环快速，因此我们可以快速地进行实验。其次，它生成的模型精度低于使用所有数据。这种低精度通常不是主要问题，因为您可以使用从较小数据集子集中获得的知识对整个数据集进行重新训练。在训练深度学习模型时，这是一个非常有用的技巧，因为在许多情况下，训练数据的数量是巨大的

**38.Python中classmethod修饰符**

classmethod 修饰符对应的函数不需要实例化，不需要 self 参数，但第一个参数需要是表示自身类的 cls 参数，可以来调用类的属性，类的方法，实例化对象等。

**39.Python中\*和\*\*的区别**

Python中，（\*）会把接收到的参数形成一个元组，而（\*\*）则会把接收到的参数存入一个字典

**40.swish激活函数：**

Swish函数先对来说是比较新的一些激活函数，算是由之前的激活函数复合而成出来的。也是由Google提出的，毕竟资力雄厚，承担的起搜索的任务。而且这个算法感觉曝光率还算比较高，就在这里整理一下，同时后面的文章也会再次提到这个函数。

  对前面的激活函数有了一定的基础之后，理解Swish激活就容易很多了，Swish函数的表达式是f(x)=x⋅σ(x)f(x)=x⋅σ(x)，σ(x)σ(x)就是sigmoid函数。因为sigmoid函数的饱和性容易导致梯度消失，借鉴ReLU的效果，当xx非常大的时候，这个时候有f(x)f(x)趋近于xx，但是当x→−∞，则f(x)→0x→−∞，则f(x)→0，函数的大致走势和ReLU比较相似，但是又比ReLU复杂。

**41.geru激活函数：**

gelu（gaussian error linear units）就是我们常说的高斯误差线性单元，它是一种高性能的神经网络激活函数，因为gelu的非线性变化是一种符合预期的随机正则变换方式，公式如下：xP(X≤x)=xΦ(x)(2.1)xP(X≤x)=xΦ(x)(2.1)其中Φ(x)Φ(x)指的是xx的高斯正态分布的累积分布，完整形式如下：xP(X≤x)=x∫x−∞e−(X−μ)22σ22π√σdX(2.2)xP(X≤x)=x∫−∞x​2π

​σe−2σ2(X−μ)2​​dX(2.2)计算结果约为：0.5x(1+tanh[2π−−√(x+0.044715x3)])(2.3)0.5x(1+tanh[π2​(x+0.044715x3)])(2.3)或者可以表示为：xσ(1.702x)(2.4)xσ(1.702x)(2.4)由此可知，概率P(X≤x)P(X≤x)（xx可看成当前神经元的激活值输入）,即XX的高斯正态分布ϕ(X)ϕ(X)的累积分布Φ(x)Φ(x)是随着xx的变化而变化的，当xx增大，Φ(x)Φ(x)增大，当x减小，Φ(x)Φ(x)减小，即当xx越小，在当前激活函数激活的情况下，越有可能激活结果为0，即此时神经元被dropout，而当xx越大越有可能被保留

简历涉及知识点

1.CNN卷积神经网络

<https://blog.csdn.net/qq_25762497/article/details/51052861>

2.RNN循环神经网络

<https://www.jianshu.com/p/498d750f0f7c>

3.GRU神经网络

<https://www.cnblogs.com/Luv-GEM/p/10705967.html>

4.LSTM长短时记忆

<https://www.cnblogs.com/Luv-GEM/p/10705967.html>

5.Word2Vec

<https://www.jianshu.com/p/471d9bfbd72f>

6.GloVe

<https://blog.csdn.net/linchuhai/article/details/97135612>

7.Seq2Seq

<https://www.jianshu.com/p/80436483b13b>

8.Transformer

<https://www.jianshu.com/p/e7d8caa13b21>

9.BERT

<https://zhuanlan.zhihu.com/p/46652512>

10.ALBERT

<https://zhuanlan.zhihu.com/p/87562926>

11.SGD、Momentum、Adagrad、Adadelta、Adam

[https://github.com/NLP-LOVE/ML-NLP/tree/master/Deep%20Learning/15.%20DL%20Optimizer](https://github.com/NLP-LOVE/ML-NLP/tree/master/Deep Learning/15. DL Optimizer)

12.LR

<https://blog.csdn.net/wehung/article/details/88930535>

13.朴素贝叶斯

<https://www.jianshu.com/p/5953923f43f0>

14.决策树

<https://zhuanlan.zhihu.com/p/30059442>

16.随机森林RF

<https://www.cnblogs.com/keye/p/10252134.html>

17.GBDT梯度提升决策树

<https://www.jianshu.com/p/005a4e6ac775>

1. 卷积层和池化层有什么区别？

首先可以从结构上可以看出，卷积之后输出层的维度减小，深度变深。但池化层深度不变。同时池化可以把很多数据用最大值或者平均值代替。目的是降低数据量。降低训练的参数。对于输入层，当其中像素在邻域发生微小位移时，池化层的输出是不变的，从而能提升鲁棒性。而卷积则是把数据通过一个卷积核变化成特征，便于后面的分离。

2. 采用宽卷积的好处有什么？

通过将输入边角的值纳入到滑窗中心进行计算，以便损失更少的信息。

3. 卷积输出的深度与哪个部件的个数相同？

输出深度（通道）与卷积核（过滤器）的个数相等。

4. 激活函数通常放在卷积神经网络的那个操作之后？

通常放在卷积层之后。

5. 为什么激活函数通常都是采用非线性的函数？

如果网络中都采用线性函数的组合，那么线性的组合还是线性，那么使用多次线性组合就等同于使用了一次线性函数。因此采用非线性函数可以来逼近任意函数。

6. 非线性激活函数中sigmod函数存在哪些不足？

Sigmod函数存在饱和状态，尤其是值过大时，当进入饱和状态时，进行梯度下降计算时，很容易出现梯度消失的情况，求导的精确值不能保证。

7. ReLU和SoftPlus激活函数有哪些优势？

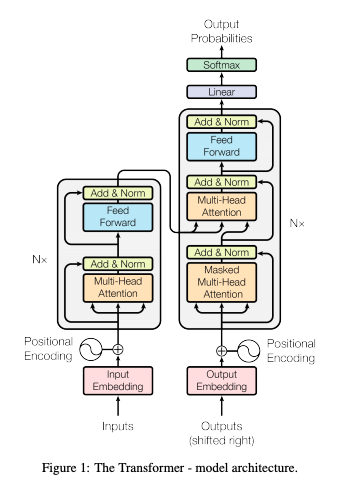
与sigmod相比，不存在指数计算，求导计算量变小，同时缓解了过拟合的情况，一部分输出为0，减少了参数的相互依存

百度地图：路况和ETA

传统机器学习模型在ETA中，比较常用的有线性回归、RF（随机森林）、GBDT（梯度提升决策树）等回归预测类模型。线性模型表达能力较差，需要大量特征工程预先分析出有效的特征；RF通过样本随机和特征随机的方式引入更多的随机性，解决了决策树泛化能力弱的问题；GBDT是通过采用加法模型（即基函数的线性组合），以及不断减小训练过程产生的残差来达到回归的算法

百度面经：

1. 介绍一下transformer模型：



1. transformer相较传统模型RNN有什么好处？

//特征抽取器能力,transformer,rnn,,gru,lstm都是特征抽取器

原始的 RNN 也存在问题，它采取线性序列结构不断从前往后收集输入信息，但这种线性序列结构在反向传播的时候存在优化困难问题，因为反向传播路径太长，容易导致严重的梯度消失或梯度爆炸问题。为了解决这个问题，后来引入了 LSTM 和 GRU 模型，通过增加中间状态信息直接向后传播，以此缓解梯度消失问题，获得了很好的效果，于是很快 LSTM 和 GRU 成为 RNN 的标准模型

NLP 又从图像领域借鉴并引入了 attention 机制（从这两个过程可以看到不同领域的相互技术借鉴与促进作用），叠加网络把层深作深，以及引入 Encoder-Decoder 框架，这些技术进展极大拓展了 RNN 的能力以及应用效果

RNN 本身结构就是个可以接纳不定长输入的由前向后进行信息线性传导的网络结构，而在 LSTM 引入三个门后，对于捕获长距离特征也是非常有效的。所以 RNN 特别适合 NLP 这种线形序列应用场景，这是 RNN 为何在 NLP 界如此流行的根本原因。

RNN 本身的序列依赖结构对于大规模并行计算来说相当之不友好。通俗点说，就是 RNN 很难具备高效的并行计算能力

为什么 RNN 并行计算能力比较差？是什么原因造成的？

T 时刻的隐层状态 St 还依赖 T-1 时刻的隐层状态 S(t-1) 的输出，这是最能体现 RNN 本质特征的一点，RNN 的历史信息是通过这个信息传输渠道往后传输的，示意参考上图。那么为什么 RNN 的并行计算能力不行呢？问题就出在这里。因为 T 时刻的计算依赖 T-1 时刻的隐层计算结果，而 T-1 时刻的计算依赖 T-2 时刻的隐层计算结果…….. 这样就形成了所谓的序列依赖关系。就是说只能先把第 1 时间步的算完，才能算第 2 时间步的结果，这就造成了 RNN 在这个角度上是无法并行计算的，只能老老实实地按着时间步一个单词一个单词往后走

Transformer 做法跟 CNN 是类似的，一般设定输入的最大长度，如果句子没那么长，则用 Padding 填充，这样整个模型输入起码看起来是定长的了。另外，NLP 句子中单词之间的相对位置是包含很多信息的，上面提过，RNN 因为结构就是线性序列的，所以天然会将位置信息编码进模型；而 CNN 的卷积层其实也是保留了位置相对信息的，所以什么也不做问题也不大。但是对于 Transformer 来说，为了能够保留输入句子单词之间的相对位置信息，必须要做点什么。为啥它必须要做点什么呢？因为输入的第一层网络是 Muli-head self attention 层，我们知道，Self attention 会让当前输入单词和句子中任意单词发生关系，然后集成到一个 embedding 向量里，但是当所有信息到了 embedding 后，位置信息并没有被编码进去。所以，Transformer 不像 RNN 或 CNN，必须明确的在输入端将 Positon 信息编码，Transformer 是用位置函数来进行位置编码的，而 Bert 等模型则给每个单词一个 Position embedding，将单词 embedding 和单词对应的 position embedding 加起来形成单词的输入 embedding

NLP 句子中长距离依赖特征的问题，Self attention 天然就能解决这个问题，因为在集成信息的时候，当前单词和句子中任意单词都发生了联系，所以一步到位就把这个事情做掉了。不像 RNN 需要通过隐层节点序列往后传，也不像 CNN 需要通过增加网络深度来捕获远距离特征，Transformer 在这点上明显方案是相对简单直观的

**语义特征提取能力；长距离特征捕获能力；任务综合特征抽取能力；并行计算能力及运行效率：**

从**语义特征提取能力**来说，目前实验支持如下结论：Transformer 在这方面的能力非常显著地超过 RNN 和 CNN（在考察语义类能力的任务 WSD 中，Transformer 超过 RNN 和 CNN 大约 4-8 个绝对百分点），RNN 和 CNN 两者能力差不太多

在**长距离特征捕获能力**方面，目前在特定的长距离特征捕获能力测试任务中（主语 - 谓语一致性检测，比如 we……..are…），实验支持如下结论：原生 CNN 特征抽取器在这方面极为显著地弱于 RNN 和 Transformer，Transformer 微弱优于 RNN 模型 (尤其在主语谓语距离小于 13 时)，能力由强到弱排序为 Transformer>RNN>>CNN; 但在比较远的距离上（主语谓语距离大于 13），RNN 微弱优于 Transformer，所以综合看，可以认为Transformer 和 RNN 在这方面能力差不太多，而 CNN 则显著弱于前两者。对于 Transformer 来说，Multi-head attention 的 head 数量严重影响 NLP 任务中 Long-range 特征捕获能力：结论是 head 越多越有利于捕获 long-range 特征，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

**任务综合特征抽取能力：**机器翻译基本上是对 NLP 各项处理能力综合要求最高的任务之一，要想获得高质量的翻译结果，对于两种语言的词法，句法，语义，上下文处理能力，长距离特征捕获等等更方面都需要考虑进来才行。这是为何看到很多比较工作是在机器翻译上作出的，这里给个背后原因的解释，以避免被质疑任务单一，没有说服力的问题

先给出一个机器翻译任务方面的证据，仍然是 why Self attention 论文的结论，对比实验结果数据参考上图。在两个机器翻译任务中，可以看到，翻译质量指标 BLEU 证明了如下结论：Transformer 综合能力要明显强于 RNN 和 CNN（你要知道，技术发展到现在阶段，BLEU 绝对值提升 1 个点是很难的事情），而 RNN 和 CNN 看上去表现基本相当，貌似 CNN 表现略好一些

关于三个特征抽取器的**并行计算能力**，其实我们在前文分述三个模型的时候都大致提过，在此仅做个归纳，结论如下：

RNN 在并行计算方面有严重缺陷，这是它本身的序列依赖特性导致的，所谓成也萧何败也萧何，它的这个线形序列依赖性非常符合解决 NLP 任务，这也是为何 RNN 一引入到 NLP 就很快流行起来的原因，但是也正是这个线形序列依赖特性，导致它在并行计算方面要想获得质的飞跃，看起来困难重重，近乎是不太可能完成的任务。

而对于 CNN 和 Transformer 来说，因为它们不存在网络中间状态不同时间步输入的依赖关系，所以可以非常方便及自由地做并行计算改造，这个也好理解。

所以归纳一下的话，可以认为并行计算能力由高到低排序如下：Transformer 和 CNN 差不多，都远远远远强于RNN

<https://blog.csdn.net/ningyanggege/article/details/89707196>

3、Transformer里的Self-Attention作用是什么，有什么优势

我对attention的理解是一个寻址机制，和数据结构里面的map是一样的，key-value结构。其实attention的形态是非常多的，有self-attention，mult-head attention等。我理解的attention都是把该位置的vec与其他所有位置的vec进行向量乘积所得到的一组参数。再说通俗点类似于把每个特征跟其他所有特征进行一次全连接。那么qkv其实类似于把一个特征跟其他所有特征进行3次全连接，然后分别根据数值特征得到qkv。该qkv里面放着这个特征值与其他特征的交互信息

我们先看一下论文中提到的放缩点积attention（scaled dot-Product attention）。对比我在前面背景知识里提到的attention的一般形式，其实scaled dot-Product attention就是我们常用的使用点积进行相似度计算的attention，（这里可以查看我的另一篇文章https://blog.csdn.net/qq\_38343151/article/details/102632649）只是多除了一个（为K的维度）起到调节作用，使得内积不至于太大

多头attention（Multi-head attention）结构如下图，Query，Key，Value首先进过一个线性变换，然后输入到放缩点积attention，注意这里要做h次，其实也就是所谓的多头，每一次算一个头。而且每次Q，K，V进行线性变换的参数W是不一样的。然后将h次的放缩点积attention结果进行拼接，再进行一次线性变换得到的值作为多头attention的结果。可以看到，google提出来的多头attention的不同之处在于进行了h次计算而不仅仅算一次，论文中说到这样的好处是可以允许模型在不同的表示子空间里学习到相关的信息

整个的Encoder结构里包含6层，每一层里面有两层。分别是一层self-attention层和一层全连接层。需要注意的是，这里的self-attention并不是只有一层。模型中使用的是multi-head-Attention。其实就是多个self-attention，可以把每个self-attention理解为一个head，多个self-attention自然就是多头了

4、产生梯度消失的原因有哪些

概念：在深度神经网络中的梯度是不稳定的，在靠近输入层的隐藏层中或会消失，或会爆炸。这种不稳定性才是深度神经网络中基于梯度学习的根本问题。

产生梯度不稳定的根本原因：前面层上的梯度是来自后面层上梯度的乘积。当存在过多的层时，就会出现梯度不稳定场景，比如梯度消失和梯度爆炸。

划重点：梯度消失和梯度爆炸属于梯度不稳定的范畴

梯度消失：（1）隐藏层的层数过多；（2）采用了不合适的激活函数(更容易产生梯度消失，但是也有可能产生梯度爆炸)

梯度爆炸：（1）隐藏层的层数过多；（2）权重的初始化值过大

总结：从深层网络角度来讲，不同的层学习的速度差异很大，表现为网络中靠近输出的层学习的情况很好，靠近输入的层学习的很慢，有时甚至训练了很久，前几层的权值和刚开始随机初始化的值差不多。因此，梯度消失、爆炸，其根本原因在于反向传播训练法则，属于先天不足

梯度消失和梯度爆炸问题都是因为网络太深，网络权值更新不稳定造成的，本质上是因为梯度反向传播中的连乘效应。对于更普遍的梯度消失问题，可以考虑一下三种方案解决：

（1）用ReLU、Leaky-ReLU、P-ReLU、R-ReLU、Maxout等替代sigmoid函数。(几种激活函数的比较见我的博客)

（2）用Batch Normalization。(对于Batch Normalization的理解可以见我的博客)

（3）LSTM的结构设计也可以改善RNN中的梯度消失问题

5、学习率是否也会导致模型不收敛？那么学习率应该怎么选？

<https://blog.csdn.net/ytusdc/article/details/107738749>

loss对a的函数为二次函数，Loss对a的导数为一次函数，导数大小取决输入数据x/y的大小，当学习率太高/输入数据太大，会导致导数非常大，直接跳到二次函数对称轴另一边，BN应该可以很大程度上解决这个问题，其他的就是改小学习率。。。凭经验。。

learning rate 减小10倍有时候意味网络训练非常慢

对于这种情况建议用二分法尝试。0.1~0.0001.不同模型不同任务最优的lr都不一样

只要你一直在train总会收敛（rp问题跑飞了不算）。反而不收敛一般是由于样本的信息量太大导致网络不足以fit住整个样本空间。样本少只可能带来过拟合的问题，你看下你的training set上的loss收敛了吗？如果只是validate set上不收敛那就说明overfitting了，这时候就要考虑各种anti-overfit的trick了，比如dropout，SGD，增大minibatch的数量，减少fc层的节点数量，momentum，finetune等

train loss 不断下降，test loss不断下降，说明网络仍在学习;

train loss 不断下降，test loss趋于不变，说明网络过拟合;

train loss 趋于不变，test loss不断下降，说明数据集100%有问题;

train loss 趋于不变，test loss趋于不变，说明学习遇到瓶颈，需要减小学习率或批量数目;

train loss 不断上升，test loss不断上升，说明网络结构设计不当，训练超参数设置不当，数据集经过清洗等问题

6、你在这里都提及了SPP，请问SPP具体是指什么？

不做科普，我回答的核心是：**Maxpooling是固定感受野动态输出，而SPP是动态感受野，固定输出，这对下游任务的支撑更加鲁棒**

7、你提到了优化方法，那么介绍下SGD和Adam的区别？

<https://zhuanlan.zhihu.com/p/32230623>

SGD最大的缺点是下降速度慢，而且可能会在沟壑的两边持续震荡，停留在一个局部最优点

谈到这里，Adam和Nadam的出现就很自然而然了——它们是前述方法的集大成者。我们看到，SGD-M在SGD基础上增加了一阶动量，AdaGrad和AdaDelta在SGD基础上增加了二阶动量。把一阶动量和二阶动量都用起来，就是Adam了——Adaptive + Momentum。

SGD的一阶动量：

加上AdaDelta的二阶动量：

优化算法里最常见的两个超参数 就都在这里了，前者控制一阶动量，后者控制二阶动量

<https://www.cnblogs.com/wuliytTaotao/p/11101652.html>

前两行：

这是对梯度和梯度的平方进行滑动平均，使得每次的更新都和历史值相关

这是对初期滑动平均偏差较大的一个修正，叫做 bias correction，当 t越来越大时，都趋近于 1，这时 bias correction 的任务也就完成了

学习率为，每轮的学习率不再保持不变，在一轮中，每个参数的学习率也不一样了，这是因为 除以了每个参数轮梯度均方和的平方根

那么问题来了，Adam 做不做学习率衰减呢？

我个人会选择做学习率衰减。（仅供参考吧。）在初始学习率设置较大的时候，做学习率衰减比不做要好；而当初始学习率设置就比较小的时候，做学习率衰减似乎有点多余，但从 validation set 上的效果看，做了学习率衰减还是可以有丁点提升的。

8、分类时，样本不均衡问题如何解决

<https://blog.csdn.net/login_sonata/article/details/54290402>

1，扩充数据集

首先想到能否获得更多数据，尤其是小类（该类样本数据极少）的数据，更多的数据往往能得到更多的分布信息

2，对数据集进行重采样

过采样（over-sampling），对小类的数据样本进行过采样来增加小类的数据样本个数，即采样的个数大于该类样本的个数。

欠采样（under-sampling），对大类的数据样本进行欠采样来减少大类的数据样本个数，即采样的个数少于该类样本的个数。

采样算法容易实现，效果也不错，但可能增大模型的偏差（Bias），因为放大或者缩小某些样本的影响相当于改变了原数据集的分布。对不同的类别也要采取不同的采样比例，但一般不会是1:1，因为与现实情况相差甚远，压缩大类的数据是个不错的选择。

3，人造数据

一种简单的产生人造数据的方法是：在该类下所有样本的每个属性特征的取值空间中随机选取一个组成新的样本，即属性值随机采样。此方法多用于小类中的样本，不过它可能破坏原属性的线性关系。如在图像中，对一幅图像进行扭曲得到另一幅图像，即改变了原图像的某些特征值，但是该方法可能会产生现实中不存在的样本。

4，改变分类算法

1. 使用代价函数时，可以增加小类样本的权值，降低大类样本的权值
2. 可以把小类样本作为异常点(outliers)，把问题转化为异常点检测问题(anomaly detection)

5，尝试其它评价指标

我们已经知道了“准确度(Accuracy)”这个评价指标在数据不均衡的情况下有时是无效的。因此在类别不均衡分类任务中，需要使用更有说服力的评价指标来对分类器进行评价。

9、CNN在NLP领域的应用有什么优势和弊端

1.并行计算，加快训练速度

2.cnn可以解决“相似的词语在这样的模型中不能共享参数权重等，这样就会导致相似词无法获得交互信息

3.有效避免参数爆炸问题”

10、transformer中采用三角函数positional embedding的好处：



2. 如果是学习到的positional embedding，（个人认为，没看论文）会像词向量一样受限于词典大小。也就是只能学习到“位置2对应的向量是(1,1,1,2)”这样的表示。所以用三角公式明显不受序列长度的限制，也就是可以对 比所遇到序列的更长的序列进行表示

3. BERT[6] 都没有选择使用 Positional Encoding 的方式生成位置表示，而是采取了所谓的“learned and fixed”的可学习的 Position embedding ，也就是去训练一个嵌入矩阵，大小为L \* d

我自己的实验结果也是使用两种embedding的方法最终结果差不多，所以果断的抛弃了迷雾重重不好解释的sinusoidal position encoding，后续实验都是用直接可学习得positional embedding。

不知道会不会是这个原因呢。

**毕竟两种方法都能取得一样的效果，其中一种方法明显的直观简单，所有大家自然的选择简单的方法，复杂的方法自然就被慢慢遗忘在一边了~~~**

**再啰嗦一句，**sinusoidal position encoding可以无限扩展到任意长度的优点，在实际使用上并没有特别的好处，毕竟过长的数据还是都被截断了的

11、transformer的有点？

1. Total computational complexity per layer （每层计算复杂度较少）

2. Amount of computation that can be parallelized, as mesured by the minimum number of sequential operations required

作者用最小的序列化运算来测量可以被并行化的计算。也就是说对于某个序列x\_1, x\_2, …, x\_n ，self-attention可以直接计算 x\_i, x\_j 的点乘结果，而rnn就必须按照顺序从 x\_1 计算到 x\_n

3. Path length between long-range dependencies in the network

这里Path length指的是要计算一个序列长度为n的信息要经过的路径长度。cnn需要增加卷积层数来扩大视野，rnn需要从1到n逐个进行计算，而self-attention只需要一步矩阵计算就可以。所以也可以看出，self-attention可以比rnn更好地解决长时依赖问题。当然如果计算量太大，比如序列长度n>序列维度d这种情况，也可以用窗口限制self-attention的计算数量

4. 从作者在附录中给出的栗子可以看出，self-attention模型更可解释，attention结果的分布表明了该模型学习到了一些语法和语义信息

缺点：

实践上：有些rnn轻易可以解决的问题transformer没做到，比如复制string，尤其是碰到比训练时的sequence更长的时

理论上：transformers非computationally universal（图灵完备），（我认为）因为无法实现“while”循环

损失函数种类，差异，作用，适用情景？

激活函数种类，差异，作用， 适用情景？

crf和gpt论文

什么是消融研究？

*消融研究”通常用于神经网络，尤其是相对复杂的神经网络，如R-CNN。我们的想法是通过删除部分网络并研究网络的性能来了解网络*

LC220，JZ50，ACWING70，labuladong

niuke mianjing

zhihu mianjing