**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