

公告

昵称： 大汤姆
园龄： 1年2个月
粉丝： 0
关注： 10
[+加关注](#)

<	2020年7月						>
日	一	二	三	四	五	六	
28	29	30	1	2	3	4	
5	6	7	8	9	10	11	
12	13	14	15	16	17	18	
19	20	21	22	23	24	25	
26	27	28	29	30	31	1	
2	3	4	5	6	7	8	

搜索

找找看

谷歌搜索

常用链接

- 我的随笔
- 我的评论
- 我的参与
- 最新评论
- 我的标签

深度学习 100 题（转）

深度学习 100 题（转）

https://github.com/wizardforcel/data-science-notebook/tree/master/dl/%E6%B7%B1%E5%BA%A6%E5%AD%A6%E4%B9%A0_100_%E9%A2%98

来源：[BAT面试1000题](#)

1、梯度下降算法的正确步骤是什么？

- a.计算预测值和真实值之间的误差
 - b.重复迭代，直至得到网络权重的最佳值
 - c.把输入传入网络，得到输出值
 - d.用随机值初始化权重和偏差
 - e.对每一个产生误差的神经元，调整相应的（权重）值以减小误差
- A.abcd e B.edc b a C.cbaed D.dcaeb

解析：正确答案D，考查知识点-深度学习。

2、已知：

- 大脑是有很多个叫做神经元的东西构成，神经网络是对大脑的简单的数学表达。
- 每一个神经元都有输入、处理函数和输出。
- 神经元组合起来形成了网络，可以拟合任何函数。
- 为了得到最佳的神经网络，我们用梯度下降方法不断更新模型

给定上述关于神经网络的描述，什么情况下神经网络模型被称为深度学习模型？

- A.加入更多层，使神经网络的深度增加
- B.有维度更高的数据
- C.当这是一个图形识别的问题时
- D.以上都不正确

解析：正确答案A，更多层意味着网络更深。没有严格的定义多少层的模型才叫深度模型，目前如果有超过2层的隐层，那么也可以叫做深度模型。

3、训练CNN时，可以对输入进行旋转、平移、缩放等预处理提高模型泛化能力。这么说是，还是不对？

A.对 B.不对

解析：对。如寒sir所说，训练CNN时，可以进行这些操作。当然也不一定是必须的，只是data augmentation扩充数据后，模型有更多数据训练，泛化能力可能会变强。

4、下面哪项操作能实现跟神经网络中Dropout的类似效果？

A.Boosting B.Bagging C.Stacking D.Mapping

解析：正确答案B。Dropout可以认为是一种极端的Bagging，每一个模型都在单独的数据上训练，同时，通过和其他模型对应参数的共享，从而实现模型参数的高度正则化。

5、下列哪一项在神经网络中引入了非线性？

- A.随机梯度下降
- B.修正线性单元（ReLU）
- C.卷积函数
- D.以上都不正确

解析：正确答案B。修正线性单元是非线性的激活函数。

6、CNN的卷积核是单层的还是多层的？

解析：

一般而言，深度卷积网络是一层又一层的。层的本质是特征图，存储输入数据或其中间表示值。一组卷积核则是联系前后两层的网络参数表达体，训练的目标就是每个卷积核的权重参数组。

描述网络模型中某层的厚度，通常用名词通道channel数或者特征图feature map数。不过人们更习惯把作为数据输入的前层的厚度称之为通道数（比如RGB三色图层称为输入通道数为3），把作为卷积输出的后层的厚度称之为特征图数。

卷积核(filter)一般是3D多层的，除了面积参数，比如3x3之外，还有厚度参数H（2D的视为厚度1）。还有一个属性是卷积核的个数N。

卷积核的厚度H，一般等于前层厚度M(输入通道数或feature map数)。特殊情况 $M > H$ 。

卷积核的个数 N ，一般等于后层厚度(后层feature maps数，因为相等所以也用 N 表示)。

卷积核通常从属于后层，为后层提供了各种查看前层特征的视角，这个视角是自动形成的。

卷积核厚度等于1时为2D卷积，对应平面点相乘然后把结果加起来，相当于点积运算；

卷积核厚度大于1时为3D卷积，每片分别平面点求卷积，然后把每片结果加起来，作为3D卷积结果； 1×1 卷积属于3D卷积的一个特例，有厚度无面积，直接把每片单个点乘以权重再相加。

归纳之，卷积的意思就是把一个区域，不管是一维线段，二维方阵，还是三维长方体，全部按照卷积核的维度形状，对应逐点相乘再求和，浓缩成一个标量值也就是降到零维度，作为下一层的一个feature map的一个点的值！

可以比喻一群渔夫坐一个渔船撒网打鱼，鱼塘是多层水域，每层鱼儿不同。

船每次移位一个stride到一个地方，每个渔夫撒一网，得到收获，然后换一个距离stride再撒，如此重复直到遍历鱼塘。

A渔夫盯着鱼的品种，遍历鱼塘后该渔夫描绘了鱼塘的鱼品种分布；

B渔夫盯着鱼的重量，遍历鱼塘后该渔夫描绘了鱼塘的鱼重量分布；

还有 $N-2$ 个渔夫，各自兴趣各干各的；

最后得到 N 个特征图，描述了鱼塘的一切！

2D卷积表示渔夫的网就是带一圈浮标的渔网，只打上面一层水体的鱼；

3D卷积表示渔夫的网是多层嵌套的渔网，上中下层水体的鱼儿都跑不掉；

1×1 卷积可以视为每次移位stride，甩钩钓鱼代替了撒网；

下面解释一下特殊情况的 $M > H$ ：

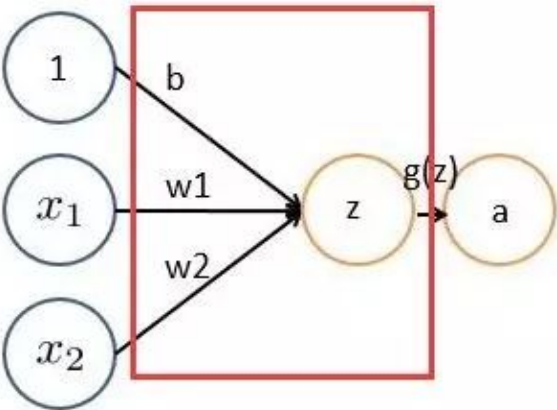
实际上，除了输入数据的通道数比较少之外，中间层的feature map数很多，这样中间层算卷积会累死计算机（鱼塘太深，每层鱼都打，需要的渔网太重了）。所以很多深度卷积网络把全部通道/特征图划分一下，每个卷积核只看其中一部分（渔夫A的渔网只打捞深水段，渔夫B的渔网只打捞浅水段）。这样整个深度网络架构是横向开始分道扬镳了，到最后才又融合。这样看来，很多网络模型的架构不完全是突发奇想，而是被参数计算量逼得。特别是现在需要在移动设备上进行AI应用计算(也叫推断)，模型参数规模必须更小，所以出现很多减少握手规模的卷积形式，现在主流网络架构大都如此。

7、什么是卷积？

解析：

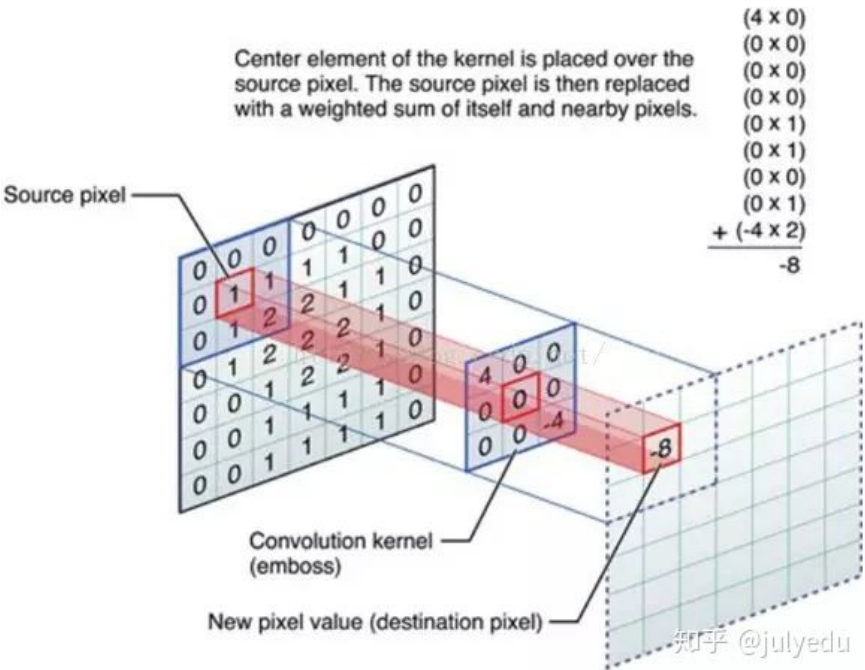
对图像（不同的数据窗口数据）和滤波矩阵（一组固定的权重：因为每个神经元的多个权重固定，所以又可以看做一个恒定的滤波器filter）做内积（逐个元素相乘再求和）的操作就是所谓的『卷积』操作，也是卷积神经网络的名字来源。

非严格意义上讲，下图中红框框起来的部分便可以理解为一个滤波器，即带着一组固定权重的神经元。多个滤波器叠加便成了卷积层。

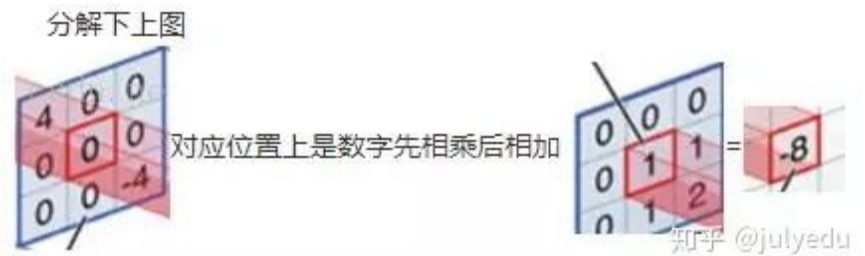


知乎 @julyedu

OK，举个具体的例子。比如下图中，图中左边部分是原始输入数据，图中中间部分是滤波器filter，图中右边是输出的新的二维数据。



知乎 @julyedu



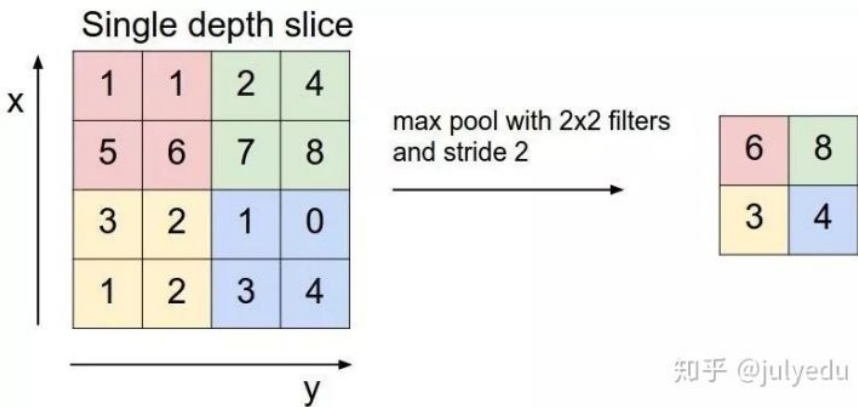
知乎 @julyedu

中间滤波器filter与数据窗口做内积，其具体计算过程则是： $4 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 0 + 0 \times 1 + 0 \times 1 + 0 \times 0 + 0 \times 1 + -4 \times 2 = -8$

8、什么是CNN的池化pool层？

解析：

池化，简言之，即取区域平均或最大，如下图所示（图引自cs231n）



上图所展示的是取区域最大，即上图左边部分中 左上角2x2的矩阵中6最大，右上角2x2的矩阵中8最大，左下角2x2的矩阵中3最大，右下角2x2的矩阵中4最大，所以得到上图右边部分的结果：6 8 3 4。

9、简述下什么是生成对抗网络。

解析：

GAN之所以是对抗的，是因为GAN的内部是竞争关系，一方叫 generator，它的主要工作是生成图片，并且尽量使得其看上去是来自于训练样本的。另一方是 discriminator，其目标是判断输入图片是否属于真实训练样本。

更直白的讲，将 generator 想象成假币制造商，而 discriminator 是警察。generator 目的是尽可能把假币造的跟真的一样，从而能够骗过 discriminator，即生成样本并使它看上去好像来自于真实训练样本一样。

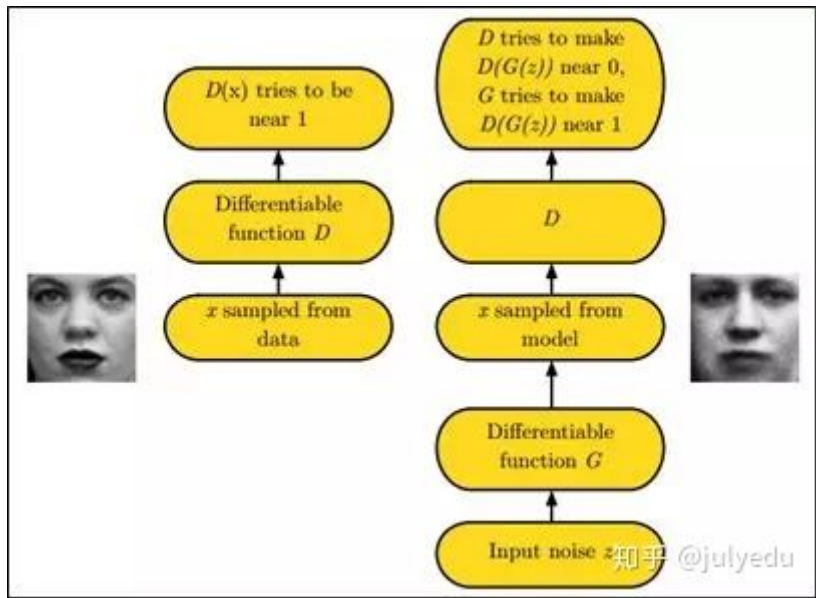
[← 返回](#) 深度学习在计算机视觉领域的...

...

生成对抗网络的一个简单解释如下：假设有两个模型，一个是生成模型

（Generative Model，下文简写为G），一个是判别模型（Discriminative Model，下文简写为D），判别模型(D)的任务就是判断一个实例是真实的还是由模型生成的，生成模型(G)的任务是生成一个实例来骗过判别模型（D），两个模型互相对抗，发展下去就会达到一个平衡，生成模型生成的实例与真实的没有区别，判别模型无法区分自然的还是模型生成的。以赝品商人为例，赝品商人（生成模型）制作出假的毕加索画作来欺骗行家（判别模型D），赝品商人一直提升他的高仿水平来区分行家，行家也一直学习真的假的毕加索画作来提升自己的辨识能力，两个人一直博弈，最后赝品商人高仿的毕加索画作达到了以假乱真的水平，行家最后也很难区分正品和赝品了。下图是

如下图中的左右两个场景：



更多请参见此课程：《生成对抗网络班》（链接：
<https://www.julyedu.com/course/getDetail/83>）

10、学梵高作画的原理是什么？

解析：

这里有篇如何做梵高风格画的实验教程《教你从头到尾利用DL学梵高作画：GTX 1070 cuda 8.0 tensorflow gpu版》（链接：
http://blog.csdn.net/v_july_v/article/details/52658965），至于其原理请看这个视频：NeuralStyle艺术化图片（学梵高作画背后的原理）（链接：<http://www.julyedu.com/video/play/42/523>）。

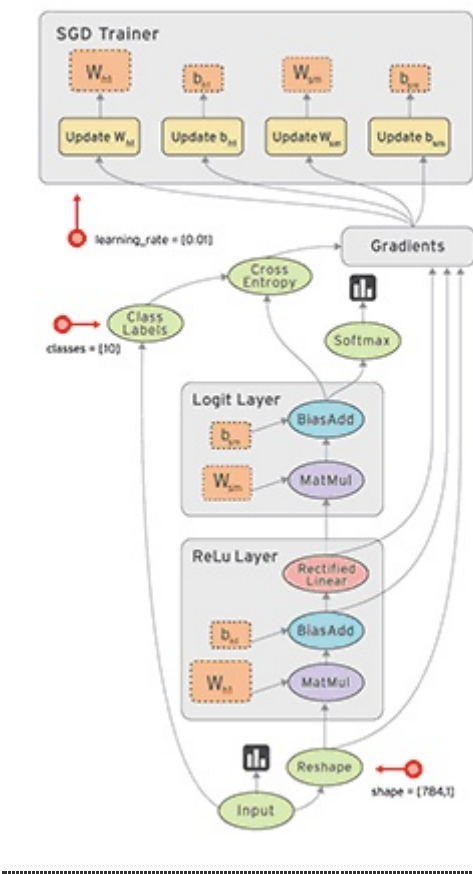
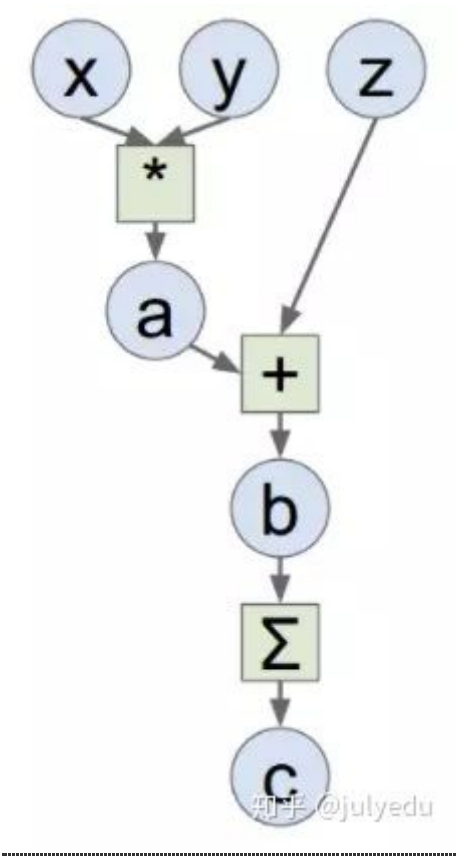
11、请简要介绍下tensorflow的计算图。

解析：

Tensorflow是一个通过计算图的形式来表述计算的编程系统，计算图也叫数据流图，可以把计算图看做是一种有向图，Tensorflow中的每一个节点都是计算图上的一个Tensor，也就是张量，而节点之间的边描述了计算之间的依赖关系(定义时)和数学操作(运算时)。

如下两图表示：

```
a=x*y; b=a+z; c=tf.reduce_sum(b);
```

12、你有哪些deep learning (rnn、cnn) 调参的经验？

解析：

一、参数初始化

下面几种方式,随便选一个,结果基本都差不多。但是一定要做。否则可能会减慢收敛速度,影响收敛结果,甚至造成Nan等一系列问题。

下面的 n_{in} 为网络的输入大小, n_{out} 为网络的输出大小, n 为 n_{in} 或 $(n_{in}+n_{out})*0.5$

Xavier初始法论文:

<http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf>

He初始化论文:

<https://arxiv.org/abs/1502.01852>

uniform均匀分布初始化:

```
w = np.random.uniform(low=-scale, high=scale, size=[n_in,n_out])
```

Xavier初始法, 适用于普通激活函数(tanh, sigmoid): $scale = \sqrt{3/n}$

He初始化, 适用于ReLU: $scale = \sqrt{6/n}$

normal高斯分布初始化: $w = np.random.randn(n_{in}, n_{out}) * stdev$ # stdev为高斯分布的标准差, 均值设为0

Xavier初始法, 适用于普通激活函数 (tanh, sigmoid): $stdev = \sqrt{n}$

He初始化, 适用于ReLU: $stdev = \sqrt{2/n}$

svd初始化: 对RNN有比较好的效果。

二、数据预处理方式

zero-center ,这个挺常用的. $X -= np.mean(X, axis = 0)$ # zero-center
 $X /= np.std(X, axis = 0)$ # normalize

PCA whitening,这个用的比较少.

三、训练技巧

要做梯度归一化,即算出来的梯度除以minibatch size

clip c(梯度裁剪): 限制最大梯度,其实是 $value = \sqrt{w_1^2 + w_2^2 + \dots}$,如果value超过了阈值,就算一个衰减系数,让value的值等于阈值: 5, 10, 15

dropout对小数据防止过拟合有很好的效果,值一般设为0.5,小数据上dropout+sgd在我的大部分实验中, 效果提升都非常明显.因此可能的话, 建议一定要尝试一下。 dropout的位置比较有讲究, 对于RNN,建议放到输入->RNN与RNN->输出的位置.关于RNN如何用dropout,可以参考这篇论文:<http://arxiv.org/abs/1409.2329>

adam, adadelta等,在小数据上,我这里实验的效果不如sgd, sgd收敛速度会慢一些, 但是最终收敛后的结果, 一般都比较好。如果使用sgd的话,可

以选择从1.0或者0.1的学习率开始,隔一段时间,在验证集上检查一下,如果cost没有下降,就对学习率减半. 我看过很多论文都这么搞,我自己实验的结果也很好. 当然,也可以先用ada系列先跑,最后快收敛的时候,更换成sgd继续训练.同样也会有提升.据说adadelta一般在分类问题上效果比较好,adam在生成问题上效果比较好。

除了gate之类的地方,需要把输出限制成0-1之外,尽量不要用sigmoid,可以用tanh或者relu之类的激活函数.1. sigmoid函数在-4到4的区间里,才有较大的梯度.之外的区间,梯度接近0,很容易造成梯度消失问题。2. 输入0均值, sigmoid函数的输出不是0均值的。

rnn的dim和embdding size,一般从128上下开始调整. batch size,一般从128左右开始调整.batch size合适最重要,并不是越大越好。

word2vec初始化,在小数据上,不仅可以有效提高收敛速度,也可以可以提高结果。

四、尽量对数据做shuffle

LSTM 的forget gate的bias,用1.0或者更大的值做初始化,可以取得更好的结果,来自这篇论

文:<http://jmlr.org/proceedings/papers/v37/jozefowicz15.pdf>, 我这里实验设成1.0,可以提高收敛速度.实际使用中,不同的任务,可能需要尝试不同的值.

Batch Normalization据说可以提升效果,不过我没有尝试过,建议作为最后提升模型的手段, 参考论文: Accelerating Deep Network Training by Reducing Internal Covariate Shift

如果你的模型包含全连接层 (MLP), 并且输入和输出大小一样, 可以考虑将MLP替换成Highway Network,我尝试对结果有一点提升, 建议作为最后提升模型的手段, 原理很简单, 就是给输出加了一个gate来控制信息的流动, 详细介绍请参考论文:<http://arxiv.org/abs/1505.00387>

来自@张馨宇的技巧: 一轮加正则, 一轮不加正则, 反复进行。

五、Ensemble

Ensemble是论文刷结果的终极核武器,深度学习中一般有以下几种方式

同样的参数,不同的初始化方式

不同的参数,通过cross-validation,选取最好的几组

同样的参数,模型训练的不同阶段, 即不同迭代次数的模型。

不同的模型,进行线性融合. 例如RNN和传统模型。

13、CNN最成功的应用是在CV, 那为什么NLP和Speech的很多问题也可以用CNN解出来? 为什么AlphaGo里也用了CNN? 这几个

不相关的问题的相似性在哪里？CNN通过什么手段抓住了这个共性？

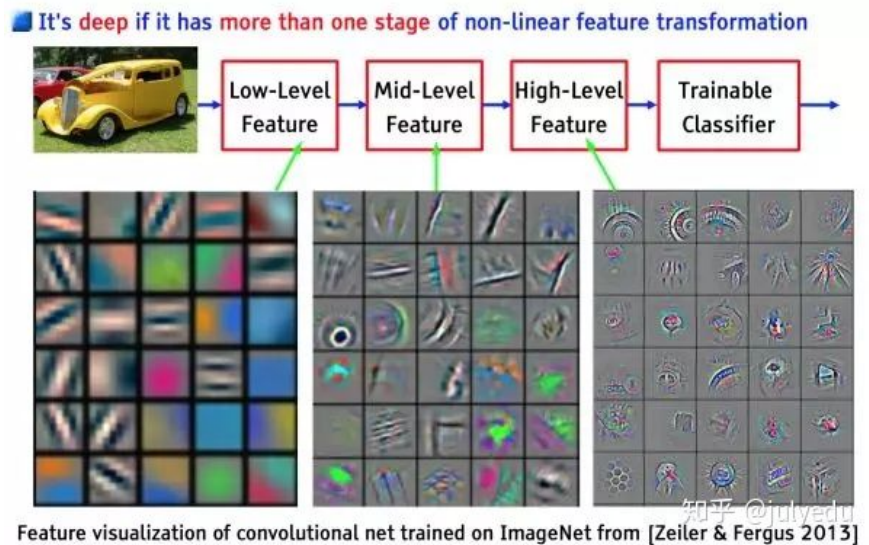
解析：

Deep Learning -Yann LeCun, Yoshua Bengio & Geoffrey Hinton

Learn TensorFlow and deep learning, without a Ph.D.

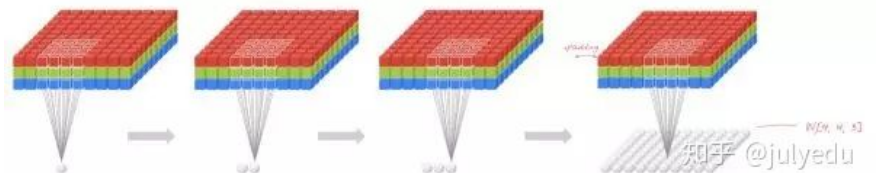
The Unreasonable Effectiveness of Deep Learning -LeCun 16
NIPS Keynote

以上几个不相关问题的相关性在于，都存在局部与整体的关系，由低层次的特征经过组合，组成高层次的特征，并且得到不同特征之间的空间相关性。如下图：低层次的直线 / 曲线等特征，组合成为不同的形状，最后得到汽车的表示。



CNN抓住此共性的手段主要有四个：局部连接 / 权值共享 / 池化操作 / 多层次结构。

局部连接使网络可以提取数据的局部特征；权值共享大大降低了网络的训练难度，一个Filter只提取一个特征，在整个图片（或者语音 / 文本）中进行卷积；池化操作与多层次结构一起，实现了数据的降维，将低层次的局部特征组合成为较高层次的特征，从而对整个图片进行表示。如下图：



上图中，如果每一个点的处理使用相同的Filter，则为全卷积，如果使用不同的Filter，则为Local-Conv。

14、LSTM结构推导，为什么比RNN好？

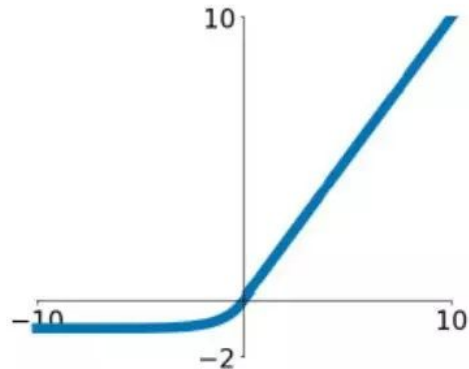
解析：

推导forget gate, input gate, cell state, hidden information等的变化；因为LSTM有进有出且当前的cell informaton是通过input gate控制之后叠加的，RNN是叠乘，因此LSTM可以防止梯度消失或者爆炸。

15、Sigmoid、Tanh、ReLu这三个激活函数有什么缺点或不足，有没改进的激活函数。

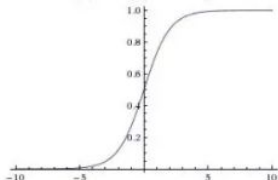
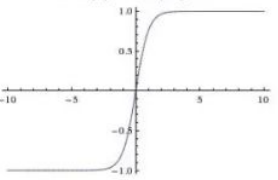
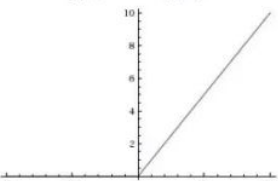
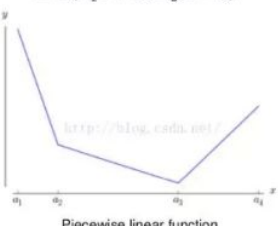
解析：

sigmoid、Tanh、ReLU的缺点在121问题中已有说明，为了解决ReLU的dead cell的情况，发明了Leaky Relu，即在输入小于0时不让输出为0，而是乘以一个较小的系数，从而保证有导数存在。同样的目的，还有一个ELU，函数示意图如下。



$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

还有一个激活函数是Maxout，即使用两套w,b参数，输出较大值。本质上Maxout可以看做Relu的泛化版本，因为如果一套w,b全都是0的话，那么就是普通的ReLU。Maxout可以克服Relu的缺点，但是参数数目翻倍。

激活函数	公式	缺点	优点
Sigmoid	$\sigma(x) = 1/(1 + e^{-x})$ 	1、会有梯度弥散 2、不是关于原点对称 3、计算exp比较耗时	-
Tanh	$\tanh(x) = 2\sigma(2x) - 1$ 	梯度弥散没解决	1、解决了原点对称问题 2、比sigmoid更快
ReLU	$f(x) = \max(0, x)$ 	梯度弥散没完全解决，在（-）部分相当于神经元死亡而且不会复活	1、解决了部分梯度弥散问题 2、收敛速度更快
Leaky ReLU	$f(x) = 1(x < 0)(\alpha x) + 1(x \geq 0)(x)$	-	解决了神经死亡问题
Maxout	$\max(w_1^T x + b_1, w_2^T x + b_2)$  <p>Piecewise linear function</p>	参数比较多，本质上是在输出结果上又增加了一层	克服了ReLU的缺点，比较提倡使用

知乎 @julyedu

资料来源：

1.寒小阳&AntZ，张雨石博客等；

2.萧瑟，

<https://www.zhihu.com/question/41631631/answer/94816420>

；

3.许韩，

<https://zhuanlan.zhihu.com/p/25005808>；

4.《CNN笔记：通俗理解卷积神经网络》，

http://blog.csdn.net/v_july_v/article/details/51812459；

5.我爱大泡泡，

<http://blog.csdn.net/woaidapaopao/article/details/77806273>。

16、为什么引入非线性激励函数？

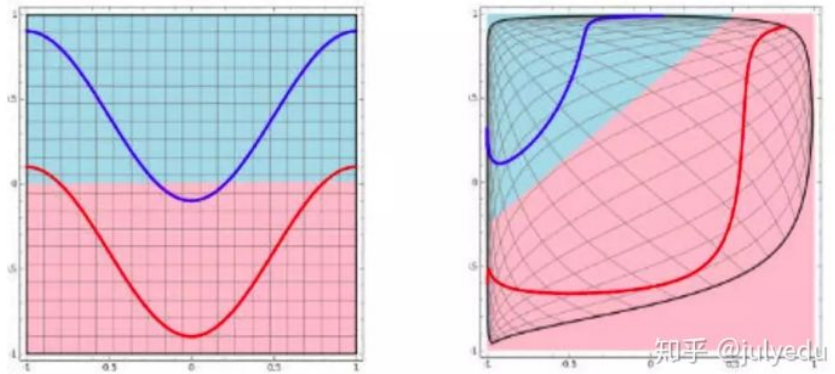
解析：

第一，对于神经网络来说，网络的每一层相当于 $f(wx+b)=f(w'x)$ ，对于线性函数，其实相当于 $f(x)=x$ ，那么在线性激活函数下，每一层相当于用一个矩阵去乘以 x ，那么多层就是反复的用矩阵去乘以输入。根据矩阵的乘

法法则，多个矩阵相乘得到一个大矩阵。所以线性激励函数下，多层网络与一层网络相当。比如，两层的网络 $f(W1 * f(W2x)) = W1W2x = Wx$ 。

第二，非线性变换是深度学习有效的原因之一。原因在于非线性相当于对空间进行变换，变换完成后相当于对问题空间进行简化，原来线性不可解的问题现在变得可以解了。

下图可以很形象的解释这个问题，左图用一根线是无法划分的。经过一系列变换后，就变成线性可解的问题了。



如果不用激励函数（其实相当于激励函数是 $f(x) = x$ ），在这种情况下你每一层输出都是上层输入的线性函数，很容易验证，无论你神经网络有多少层，输出都是输入的线性组合，与没有隐藏层效果相当，这种情况就是最原始的感知机（Perceptron）了。

正因为上面的原因，我们决定引入非线性函数作为激励函数，这样深层神经网络就有意义了（不再是输入的线性组合，可以逼近任意函数）。最早的想法是sigmoid函数或者tanh函数，输出有界，很容易充当下一层输入（以及一些人的生物解释）。

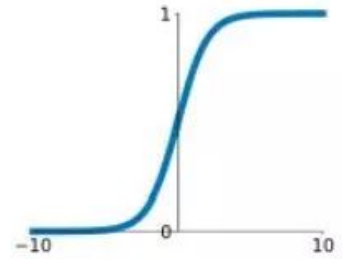
17、请问人工神经网络中为什么ReLu要好过于tanh和sigmoid function?

解析：

先看sigmoid、tanh和RelU的函数图：

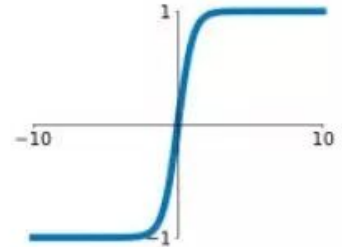
Sigmoid

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



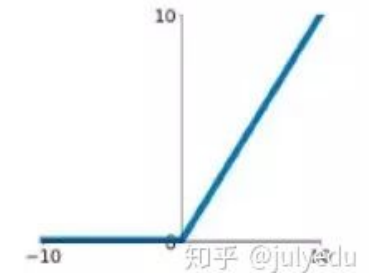
tanh

$$\tanh(x)$$



ReLU

$$\max(0, x)$$



第一，采用sigmoid等函数，算激活函数时（指数运算），计算量大，反向传播求误差梯度时，求导涉及除法和指数运算，计算量相对大，而采用Relu激活函数，整个过程的计算量节省很多。

第二，对于深层网络，sigmoid函数反向传播时，很容易就会出现梯度消失的情况（在sigmoid接近饱和区时，变换太缓慢，导数趋于0，这种情况会造成信息丢失），这种现象称为饱和，从而无法完成深层网络的训练。而ReLU就不会有饱和倾向，不会有特别小的梯度出现。

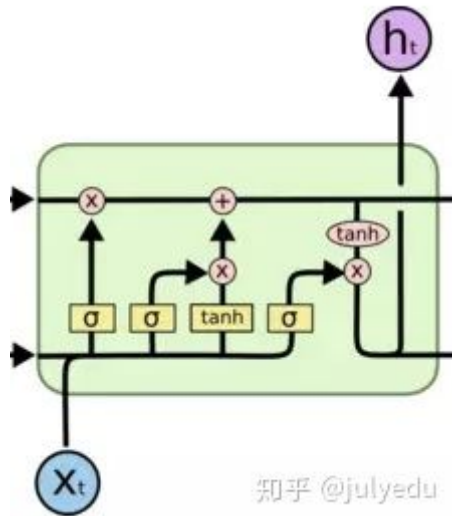
第三，Relu会使一部分神经元的输出为0，这样就造成了网络的稀疏性，并且减少了参数的相互依存关系，缓解了过拟合问题的发生（以及一些人的生物解释balabala）。当然现在也有一些对relu的改进，比如prelu，random relu等，在不同的数据集上会有一些训练速度上或者准确率上的改进，具体的大家可以找相关的paper看。

多加一句，现在主流的做法，会多做一步batch normalization，尽可能保证每一层网络的输入具有相同的分布[1]。而最新的paper[2]，他们在加入bypass connection之后，发现改变batch normalization的位置会有更好的效果。大家有兴趣可以看下。

[1] Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[J]. arXiv preprint arXiv:1502.03167, 2015.

[2] He, Kaiming, et al. "Identity Mappings in Deep Residual Networks." arXiv preprint arXiv:1603.05027 (2016).

18、为什么LSTM模型中既存在sigmoid又存在tanh两种激活函数，而不是选择统一一种sigmoid或者tanh？这样做的目的是什么？



解析：

sigmoid 用在了各种gate上，产生0~1之间的值，这个一般只有sigmoid最直接了。

tanh 用在了状态和输出上，是对数据的处理，这个用其他激活函数或许也可以。

二者目的不一样

另可参见A Critical Review of Recurrent Neural Networks for Sequence Learning的section4.1，说了那两个tanh都可以替换成别的。

19、如何解决RNN梯度爆炸和弥散的问题？

解析：

为了解决梯度爆炸问题，Thomas Mikolov首先提出了一个简单的启发性的解决方案，就是当梯度大于一定阈值的的时候，将它截断为一个较小的数。具体如算法1所述：

算法：当梯度爆炸时截断梯度（伪代码）

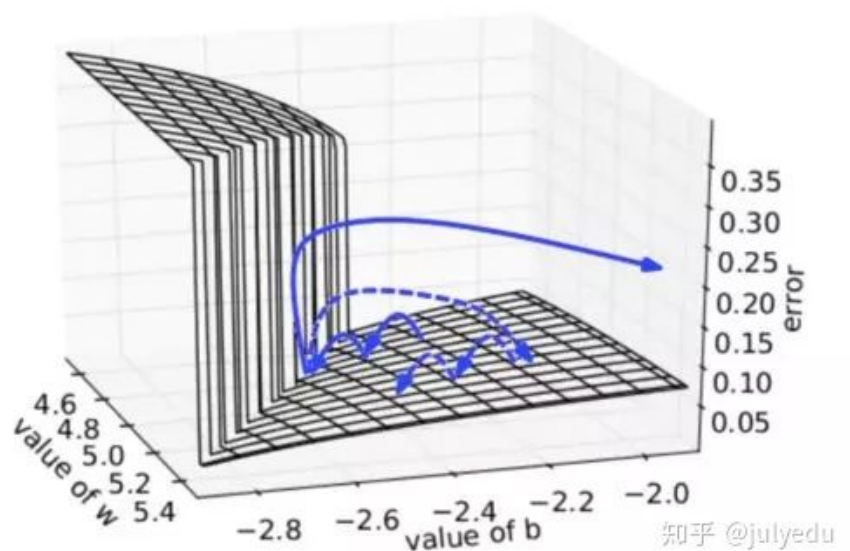
$$\hat{g} \leftarrow \frac{\partial E}{\partial W}$$

if $\|\hat{g}\| \geq \text{threshold}$

$$\hat{g} \leftarrow \frac{\text{threshold}}{\|\hat{g}\|} \hat{g}$$

知乎 @julyedu

下图可视化了梯度截断的效果。它展示了一个小的rnn（其中W为权值矩阵，b为bias项）的决策面。这个模型是一个一小段时间的rnn单元组成；实心箭头表明每步梯度下降的训练过程。当梯度下降过程中，模型的目标函数取得了较高的误差时，梯度将被送到远离决策面的位置。截断模型产生了一个虚线，它将误差梯度拉回到离原始梯度接近的位置。



知乎 @julyedu

梯度爆炸，梯度截断可视化

为了解决梯度弥散的问题，我们介绍了两种方法。第一种方法是将随机初始化

$W^{(hh)}$

改为一个有关联的矩阵初始化。第二种方法是使用ReLU (Rectified Linear Units) 代替sigmoid函数。ReLU的导数不是0就是1.因此，神经元的梯度将始终为1，而不会当梯度传播了一定时间之后变小。

20、什么样的资料集不适合用深度学习？

解析：

(1) 数据集太小，数据样本不足时，深度学习相对其它机器学习算法，没有明显优势。

(2) 数据集没有局部相关特性，目前深度学习表现比较好的领域主要是图像 / 语音 / 自然语言处理等领域，这些领域的一个共性是局部相关性。图像中像素组成物体，语音信号中音位组合成单词，文本数据中单词组合成句子，这些特征元素的组合一旦被打乱，表示的含义同时也被改变。对于没有这样的局部相关性的数据集，不适于使用深度学习算法进行处理。举个例子：预测一个人的健康状况，相关的参数会有年龄、职业、收入、家庭状况等各种元素，将这些元素打乱，并不会影响相关的结果。

21、广义线性模型是怎被应用在深度学习中？

解析：

A Statistical View of Deep Learning (I): Recursive GLMs

深度学习从统计学角度，可以看做递归的广义线性模型。

广义线性模型相对于经典的线性模型($y=wx+b$)，核心在于引入了连接函数 $g(\cdot)$ ，形式变为： $y=g^{-1}(wx+b)$ 。

深度学习时递归的广义线性模型，神经元的激活函数，即为广义线性模型的链接函数。逻辑回归（广义线性模型的一种）的Logistic函数即为神经元激活函数中的Sigmoid函数，很多类似的方法在统计学和神经网络中的名称不一样，容易引起初学者（这里主要指我）的困惑。

下图是一个对照表：

Target Type	Regression	Link	Inv link	Activation
Real	Linear	Identity	Identity	
Binary	Logistic	Logit $\log \frac{\mu}{1-\mu}$	Sigmoid σ $\frac{1}{1+\exp(-\eta)}$	Sigmoid
Binary	Probit	Inv Gauss CDF $\Phi^{-1}(\mu)$	Gauss CDF $\Phi(\eta)$	Probit
Binary	Gumbel	Compl. log-log $\log(-\log(\mu))$	Gumbel CDF $e^{-e^{-x}}$	
Binary	Logistic		Hyperbolic Tangent $\tanh(\eta)$	Tanh
Categorical	Multinomial		Multin. Logit $\frac{\eta_i}{\sum_j \eta_j}$	Softmax
Counts	Poisson	$\log(\mu)$	$\exp(\nu)$	
Counts	Poisson	$\sqrt{\mu}$	ν^2	
Non-neg.	Gamma	Reciprocal $\frac{1}{\mu}$	$\frac{1}{\nu}$	
Sparse	Tobit		max $\max(0; \nu)$	ReLU
Ordered	Ordinal		Cum. Logit $\sigma(\phi_k - \eta)$	

知乎 @julyedu

22、如何解决梯度消失和梯度膨胀？

解析：

(1) 梯度消失：

根据链式法则，如果每一层神经元对上一层的输出的偏导乘上权重结果都小于1的话，那么即使这个结果是0.99，在经过足够多层传播之后，误差对输入层的偏导会趋于0

可以采用ReLU激活函数有效的解决梯度消失的情况，也可以用Batch Normalization解决这个问题。关于深度学习中 Batch Normalization为什么效果好？

(2) 梯度膨胀

根据链式法则，如果每一层神经元对上一层的输出的偏导乘上权重结果都大于1的话，在经过足够多层传播之后，误差对输入层的偏导会趋于无穷大

可以通过激活函数来解决，或用Batch Normalization解决这个问题。

23、简述神经网络的发展历史。

解析：

1949年Hebb提出了神经心理学学习范式——Hebbian学习理论

1952年，IBM的Arthur Samuel写出了西洋棋程序

1957年，Rosenblatt的感知器算法是第二个有着神经系统科学背景的机器学习模型。

3年之后，Widrow因发明Delta学习规则而载入ML史册，该规则马上就很好的应用到了感知器的训练中

感知器的热度在1969被Minsky一盆冷水泼灭了。他提出了著名的XOR问题，论证了感知器在类似XOR问题的线性不可分数据的无力。

尽管BP的思想在70年代就被Linnainmaa以“自动微分的翻转模式”被提出来，但直到1981年才被Werbos应用到多层感知器(MLP)中，NN新的大繁荣。

1991年的Hochreiter和2001年的Hochreiter的工作，都表明在使用BP算法时，NN单元饱和之后会发生梯度损失。又发生停滞。

时间终于走到了当下，随着计算资源的增长和数据量的增长。一个新的NN领域——深度学习出现了。

简言之，MP模型+sgn-->单层感知机（只能线性）+sgn— Minsky 低谷 -->多层感知机+BP+sigmoid--（低谷）-->深度学习+pre-training+ReLU/sigmoid

24、深度学习常用方法。

解析：

全连接DNN（相邻层相互连接、层内无连接）：

AutoEncoder(尽可能还原输入)、Sparse Coding（在AE上加入L1规范）、RBM（解决概率问题）-->特征探测器-->栈式叠加 贪心训练

RBM-->DBN

解决全连接DNN的全连接问题-->CNN

解决全连接DNN的无法对时间序列上变化进行建模的问题-->RNN—解决时间轴上的梯度消失问题——>LSTM

DNN是传统的全连接网络，可以用于广告点击率预估，推荐等。其使用embedding的方式将很多离散的特征编码到神经网络中，可以很大的提升结果。

CNN主要用于计算机视觉(Computer Vision)领域，CNN的出现主要解决了DNN在图像领域中参数过多的问题。同时，CNN特有的卷积、池化、batch normalization、Inception、ResNet、DeepNet等一系列的发展也使得在分类、物体检测、人脸识别、图像分割等众多领域有了长足的进

步。同时，CNN不仅在图像上应用很多，在自然语言处理上也颇有进展，现在已经有基于CNN的语言模型能够达到比LSTM更好的效果。在最新的AlphaZero中，CNN中的ResNet也是两种基本算法之一。

GAN是一种应用在生成模型的训练方法，现在有很多在CV方面的应用，例如图像翻译，图像超清化、图像修复等等。

RNN主要用于自然语言处理(Natural Language Processing)领域，用于处理序列到序列的问题。普通RNN会遇到梯度爆炸和梯度消失的问题。所以现在在NLP领域，一般会使用LSTM模型。在最近的机器翻译领域，Attention作为一种新的手段，也被引入进来。

除了DNN、RNN和CNN外，自动编码器(AutoEncoder)、稀疏编码(Sparse Coding)、深度信念网络(DBM)、限制玻尔兹曼机(RBM)也都有相应的研究。

25、请简述神经网络的发展史。

解析：

sigmoid会饱和，造成梯度消失。于是有了ReLU。

ReLU负半轴是死区，造成梯度变0。于是有了LeakyReLU，PReLU。

强调梯度和权值分布的稳定性，由此有了ELU，以及较新的SELU。

太深了，梯度传不下去，于是有了highway。

干脆连highway的参数都不要，直接变残差，于是有了ResNet。

强行稳定参数的均值和方差，于是有了BatchNorm。

在梯度流中增加噪声，于是有了 Dropout。

RNN梯度不稳定，于是加几个通路和门控，于是有了LSTM。

LSTM简化一下，有了GRU。

GAN的JS散度有问题，会导致梯度消失或无效，于是有了WGAN。

WGAN对梯度的clip有问题，于是有了WGAN-GP。

参考资料：

1.许韩，

<https://www.zhihu.com/question/41233373/answer/145404190>；

2.<https://www.zhihu.com/question/38102762>；

3.SmallisBig，

<http://blog.csdn.net/u010496169/article/details/73550487>；

4.张雨石，现在在应用领域应用的做更多的是DNN，CNN和RNN；

5.SIY.Z，<https://zhuanlan.zhihu.com/p/29435406>。

26、神经网络中激活函数的真正意义？一个激活函数需要具有哪些必要的属性？还有哪些属性是好的属性但不必要的？

解析：

(1) 非线性：即导数不是常数。这个条件是多层神经网络的基础，保证多层网络不退化成单层线性网络。这也是激活函数的意义所在。

(2) 几乎处处可微：可微性保证了在优化中梯度的可计算性。传统的激活函数如sigmoid等满足处处可微。对于分段线性函数比如ReLU，只满足几乎处处可微（即仅在有限个点处不可微）。对于SGD算法来说，由于几乎不可能收敛到梯度接近零的位置，有限的不可微点对于优化结果不会有很大影响[1]。

(3) 计算简单：非线性函数有很多。极端的说，一个多层神经网络也可以作为一个非线性函数，类似于Network In Network[2]中把它当做卷积操作的做法。但激活函数在神经网络前向的计算次数与神经元的个数成正比，因此简单的非线性函数自然更适合作为激活函数。这也是ReLU之流比其它使用Exp等操作的激活函数更受欢迎的其中一个原因。

(4) 非饱和性 (saturation)：饱和指的是在某些区间梯度接近于零（即梯度消失），使得参数无法继续更新的问题。最经典的例子是Sigmoid，它的导数在x为比较大的正值和比较小的负值时都会接近于0。更极端的例子是阶跃函数，由于它在几乎所有位置的梯度都为0，因此处处饱和，无法作为激活函数。ReLU在 $x > 0$ 时导数恒为1，因此对于再大的正值也不会饱和。但同时对于 $x < 0$ ，其梯度恒为0，这时候它也会出现饱和的现象（在这种情况下通常称为dying ReLU）。Leaky ReLU[3]和PReLU[4]的提出正是为了解决这一问题。

(5) 单调性 (monotonic)：即导数符号不变。这个性质大部分激活函数都有，除了诸如sin、cos等。个人理解，单调性使得在激活函数处的梯度方向不会经常改变，从而让训练更容易收敛。

(6) 输出范围有限：有限的输出范围使得网络对于一些比较大的输入也会比较稳定，这也是为什么早期的激活函数都以此类函数为主，如Sigmoid、TanH。但这导致了前面提到的梯度消失问题，而且强行让每一层的输出限制到固定范围会限制其表达能力。因此现在这类函数仅用于某些需要特定输出范围的场合，比如概率输出（此时loss函数中的log操作能够抵消其梯度消失的影响[1]）、LSTM里的gate函数。

(7) 接近恒等变换 (identity)：即约等于x。这样的好处是使得输出的幅值不会随着深度的增加而发生显著的增加，从而使网络更为稳定，同时梯度也能够更容易地回传。这个与非线性是有点矛盾的，因此激活函数基本只是部分满足这个条件，比如TanH只在原点附近有线性区（在原点为0且在原点的导数为1），而ReLU只在 $x > 0$ 时为线性。这个性质也让初始化参数范围的推导更为简单[5][4]。额外提一句，这种恒等变换的性质也被

其他一些网络结构设计所借鉴，比如CNN中的ResNet[6]和RNN中的LSTM。

(8) 参数少：大部分激活函数都是没有参数的。像PReLU带单个参数会略微增加网络的大小。还有一个例外是Maxout[7]，尽管本身没有参数，但在同样输出通道数下k路Maxout需要的输入通道数是其它函数的k倍，这意味着神经元数目也需要变为k倍；但如果不考虑维持输出通道数的情况下，该激活函数又能将参数个数减少为原来的k倍。

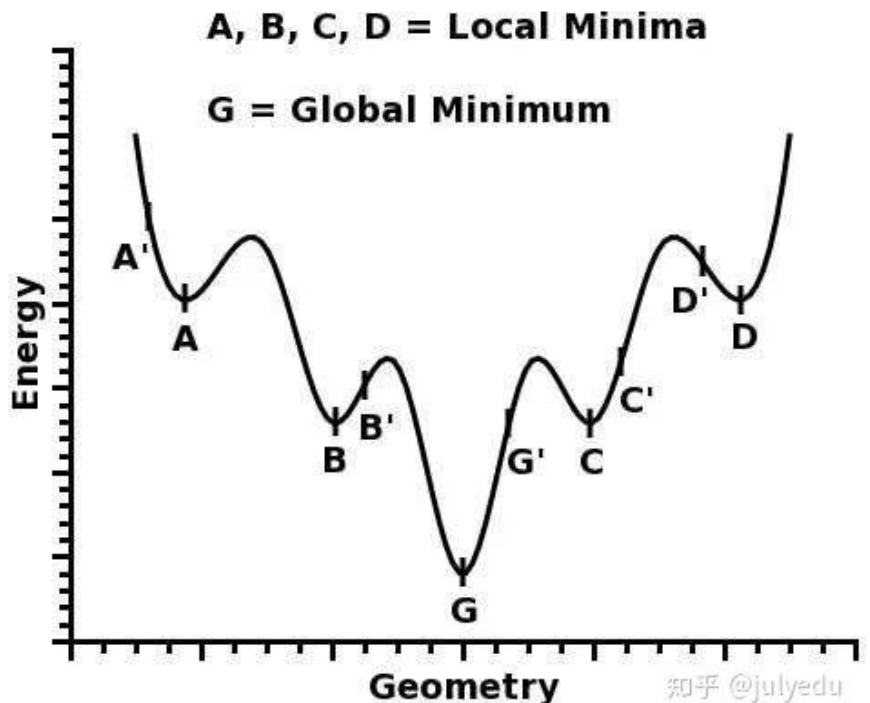
(9) 归一化 (normalization)：这个是最近才出来的概念，对应的激活函数是SELU[8]，主要思想是使样本分布自动归一化到零均值、单位方差的分布，从而稳定训练。在这之前，这种归一化的思想也被用于网络结构的设计，比如Batch Normalization[9]。

27、梯度下降法的神经网络容易收敛到局部最优，为什么应用广泛？

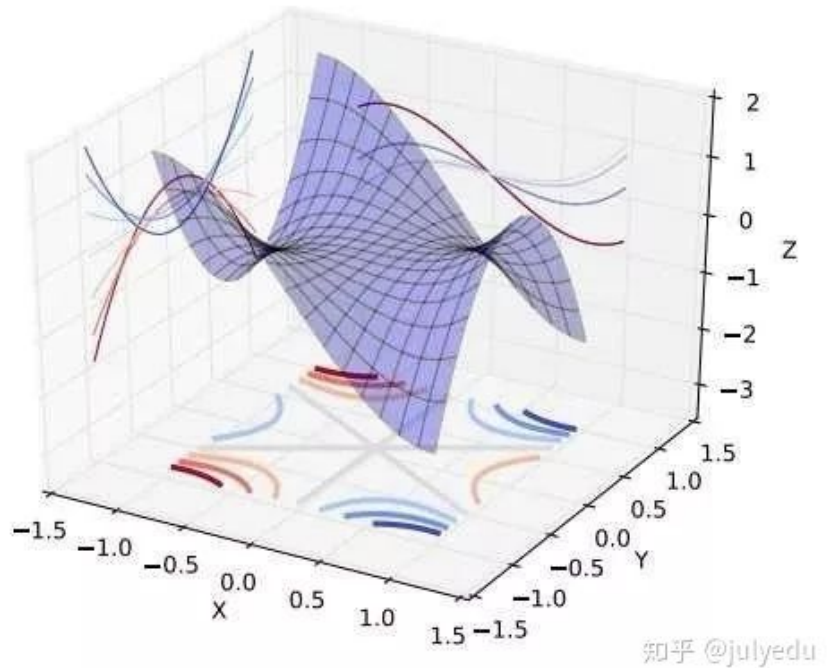
解析：

深度神经网络“容易收敛到局部最优”，很可能是一种想象，实际情况是，我们可能从来没有找到过“局部最优”，更别说全局最优了。

很多人都有一种看法，就是“局部最优是神经网络优化的主要难点”。这来源于一维优化问题的直观想象。在单变量的情形下，优化问题最直观的困难就是有很多局部极值，如



人们直观的印象，高维的时候这样的局部极值会更多，指数级的增加，于是优化到全局最优就更难了。然而单变量到多变量一个重要差异是，单变量的时候，Hessian矩阵只有一个特征值，于是无论这个特征值的符号正负，一个临界点都是局部极值。但是在多变量的时候，Hessian有多个不同的特征值，这时候各个特征值就可能会有更复杂的分布，如有正有负的不定型和有多个退化特征值（零特征值）的半定型



在后两种情况下，是很难找到局部极值的，更别说全局最优了。

现在看来，神经网络的训练的困难主要是鞍点的问题。在实际中，我们很可能也从来没有真的遇到过局部极值。Bengio组这篇文章 [Eigenvalues of the Hessian in Deep Learning \(https://arxiv.org/abs/1611.07476\)](https://arxiv.org/abs/1611.07476) 里面的实验研究给出以下的结论：

- Training stops at a point that has a small gradient. The norm of the gradient is not zero, therefore it does not, technically speaking, converge to a critical point.
- There are still negative eigenvalues even when they are small in magnitude.

另一方面，一个好消息是，即使有局部极值，具有较差的loss的局部极值的吸引域也是很小的Towards Understanding Generalization of Deep Learning: Perspective of Loss Landscapes. (<https://arxiv.org/abs/1706.10239>)

For the landscape of loss function for deep networks, the volume of basin of attraction of good minima dominates over that of poor minima, which guarantees optimization methods with random initialization to converge to good minima.

所以，很可能我们实际上是在“什么也没找到”的情况下就停止了训练，然后拿到测试集上试试，“咦，效果还不错”。

补充说明，这些都是实验研究结果。理论方面，各种假设下，深度神经网络的Landscape 的鞍点数目指数增加，而具有较差loss的局部极值非常少。

28、简单说说CNN常用的几个模型。

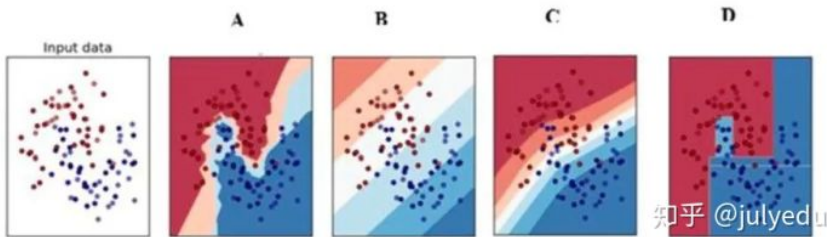
解析：

名称	特点
LeNet5	没啥特点-不过是第一个CNN应该要知道
AlexNet	引入了ReLU和dropout，引入数据增强、池化相互之间有覆盖，三个卷积一个最大池化+三个全连接层
VGGNet	采用1*1和3*3的卷积核以及2*2的最大池化使得层数变得更深。常用VGGNet-16和VGGNet19
Google Inception Net	这个在控制了计算量和参数量的同时，获得了比较好的分类性能，和上面相比有几个大的改进： 1、去除了最后的全连接层，而是用一个全局的平均池化来取代它； 2、引入Inception Module，这是一个4个分支结合的结构。所有的分支都用到了1*1的卷积，这是因为1*1性价比很高，可以用很少的参数达到非线性特征变换。 3、Inception V2第二版将所有的5*5变成2个3*3，而且提出著名的Batch Normalization； 4、Inception V3第三版就更变态了，把较大的二维卷积拆成了两个较小的一维卷积，加速运算、减少过拟合，同时还更改了Inception Module的结构。
微软ResNet残差神经网络 (Residual Neural Network)	1、引入高速公路结构，可以让神经网络变得非常深 2、ResNet第二个版本将ReLU激活函数变成y=x的线性函数

29、为什么很多做人脸的Paper会最后加入一个Local Connected Conv?

解析：

以FaceBook DeepFace 为例：



DeepFace 先进行了两次全卷积 + 一次池化，提取了低层次的边缘 / 纹理等特征。后接了3个Local-Conv层，这里是用Local-Conv的原因是，人脸在不同的区域存在不同的特征（眼睛 / 鼻子 / 嘴的分布位置相对固定），当不存在全局的局部特征分布时，Local-Conv更适合特征的提取。

30、什么是梯度爆炸?

解析：

误差梯度是神经网络训练过程中计算的方向和数量，用于以正确的方向和合适的量更新网络权重。

在深层网络或循环神经网络中，误差梯度可在更新中累积，变成非常大的梯度，然后导致网络权重的大幅更新，并因此使网络变得不稳定。在极端情况下，权重的值变得非常大，以至于溢出，导致 NaN 值。

网络层之间的梯度（值大于 1.0）重复相乘导致的指数级增长会产生梯度爆炸。

31、梯度爆炸会引发什么问题?

解析：

在深度多层感知机网络中，梯度爆炸会引起网络不稳定，最好的结果是无法从训练数据中学习，而最坏的结果是出现无法再更新的 NaN 权重值。

梯度爆炸导致学习过程不稳定。 — 《深度学习》，2016。

在循环神经网络中，梯度爆炸会导致网络不稳定，无法利用训练数据学习，最好的结果是网络无法学习长的输入序列数据。

32、如何确定是否出现梯度爆炸？

解析：

训练过程中出现梯度爆炸会伴随一些细微的信号，如：

模型无法从训练数据中获得更新（如低损失）。

模型不稳定，导致更新过程中的损失出现显著变化。

训练过程中，模型损失变成 NaN。

如果你发现这些问题，那么你需要仔细查看是否出现梯度爆炸问题。

以下是一些稍微明显一点的信号，有助于确认是否出现梯度爆炸问题。

训练过程中模型梯度快速变大。

训练过程中模型权重变成 NaN 值。

训练过程中，每个节点和层的误差梯度值持续超过 1.0。

33、如何修复梯度爆炸问题？

解析：

有很多方法可以解决梯度爆炸问题，本节列举了一些最佳实验方法。

（1）重新设计网络模型

在深度神经网络中，梯度爆炸可以通过重新设计层数更少的网络来解决。

使用更小的批尺寸对网络训练也有好处。

在循环神经网络中，训练过程中在更少的先前时间步上进行更新（沿时间的截断反向传播，truncated Backpropagation through time）可以缓解梯度爆炸问题。

（2）使用 ReLU 激活函数

在深度多层感知机神经网络中，梯度爆炸的发生可能是因为激活函数，如之前很流行的 Sigmoid 和 Tanh 函数。

使用 ReLU 激活函数可以减少梯度爆炸。采用 ReLU 激活函数是最适合隐藏层的新实践。

（3）使用长短期记忆网络

在循环神经网络中，梯度爆炸的发生可能是因为某种网络的训练本身就存在不稳定性，如随时间的反向传播本质上将循环网络转换成深度多层感知

机神经网络。

使用长短期记忆（LSTM）单元和相关的门类型神经元结构可以减少梯度爆炸问题。

采用 LSTM 单元是适合循环神经网络的序列预测的最新最好实践。

（4）使用梯度截断（Gradient Clipping）

在非常深且批尺寸较大的多层感知机网络和输入序列较长的 LSTM 中，仍然有可能出现梯度爆炸。如果梯度爆炸仍然出现，你可以在训练过程中检查和限制梯度的大小。这就是梯度截断。

处理梯度爆炸有一个简单有效的解决方案：如果梯度超过阈值，就截断它们。

——《Neural Network Methods in Natural Language Processing》，2017.

具体来说，检查误差梯度的值是否超过阈值，如果超过，则截断梯度，将梯度设置为阈值。

梯度截断可以一定程度上缓解梯度爆炸问题（梯度截断，即在执行梯度下降步骤之前将梯度设置为阈值）。

——《深度学习》，2016.

在 Keras 深度学习库中，你可以在训练之前设置优化器上的 clipnorm 或 clipvalue 参数，来使用梯度截断。

默认值为 clipnorm=1.0 、clipvalue=0.5。详见：

<https://keras.io/optimizers/>。

（5）使用权重正则化（Weight Regularization）

如果梯度爆炸仍然存在，可以尝试另一种方法，即检查网络权重的大小，并惩罚产生较大权重值的损失函数。该过程被称为权重正则化，通常使用的是 L1 惩罚项（权重绝对值）或 L2 惩罚项（权重平方）。

对循环权重使用 L1 或 L2 惩罚项有助于缓解梯度爆炸。

——On the difficulty of training recurrent neural networks, 2013.

在 Keras 深度学习库中，你可以通过在层上设置 kernel_regularizer 参数和使用 L1 或 L2 正则化项进行权重正则化。

34、LSTM神经网络输入输出究竟是怎样的？

解析：

第一要明确的是神经网络所处理的单位全部都是：向量

下面就解释为什么你会看到训练数据会是矩阵和张量

常规feedforward 输入和输出：矩阵

输入矩阵形状: (n_samples, dim_input)

输出矩阵形状: (n_samples, dim_output)

注: 真正测试/训练的时候, 网络的输入和输出就是向量而已。加入 n_samples 这个维度是为了可以实现一次训练多个样本, 求出平均梯度来更新权重, 这个叫做 Mini-batch gradient descent。如果 n_samples 等于 1, 那么这种更新方式叫做 Stochastic Gradient Descent (SGD)。

Feedforward 的输入输出的本质都是单个向量。

常规 Recurrent (RNN/LSTM/GRU) 输入和输出: 张量

输入张量形状: (time_steps, n_samples, dim_input)

输出张量形状: (time_steps, n_samples, dim_output)

注: 同样是保留了 Mini-batch gradient descent 的训练方式, 但不同之处在于多了 time step 这个维度。

Recurrent 的任意时刻的输入的本质还是单个向量, 只不过是将不同时刻的向量按顺序输入网络。所以你可能更愿意理解为一串向量 a sequence of vectors, 或者是矩阵。

python 代码表示预测的话:

```
import numpy as np
```

```
#当前所累积的hidden_state,若是最初的vector, 则hidden_state全为0
```

```
hidden_state=np.zeros((n_samples, dim_input))
```

```
#print(inputs.shape): (time_steps, n_samples, dim_input)
```

```
outputs = np.zeros((time_steps, n_samples, dim_output))
```

```
for i in range(time_steps):
```

```
#输出当前时刻的输出, 同时更新当前已累积的hidden_state
```

```
outputs[i],hidden_state = RNN.predict(inputs[i],hidden_state)
```

```
#print(outputs.shape): (time_steps, n_samples, dim_output)
```

但需要注意的是, Recurrent nets 的输出也可以是矩阵, 而非三维张量, 取决于你如何设计。

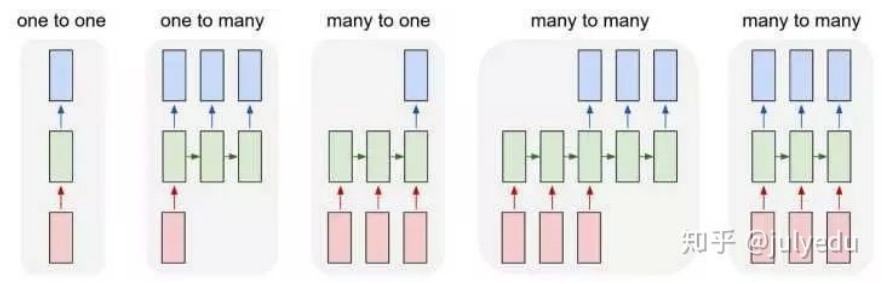
(1) 若想用一串序列去预测另一串序列, 那么输入输出都是张量 (例如语音识别 或 机器翻译 一个中文句子翻译成英文句子 (一个单词算作一个向量), 机器翻译还是个特例, 因为两个序列的长短可能不同, 要用到 seq2seq;

(2) 若想用一串序列去预测一个值, 那么输入是张量, 输出是矩阵 (例如, 情感分析就是用一串单词组成的句子去预测说话人的心情)

Feedforward 能做的是向量对向量的 one-to-one mapping,

Recurrent 将其扩展到了序列对序列 sequence-to-sequence mapping.

但单个向量也可以视为长度为1的序列。所以有下图几种类型：



除了最左侧的one to one是feedforward 能做的，右侧都是Recurrent所扩展的

若还想知道更多

(1) 可以将Recurrent的横向操作视为累积已发生的事情，并且LSTM的memory cell机制会选择记忆或者忘记所累积的信息来预测某个时刻的输出。

(2) 以概率的视角理解的话：就是不断的conditioning on已发生的事情，以此不断缩小sample space

(3) RNN的思想是：current output不仅仅取决于current input，还取决于previous state；可以理解成current output是由current input和previous hidden state两个输入计算而出的。并且每次计算后都会有信息残留于previous hidden state中供下一次计算。

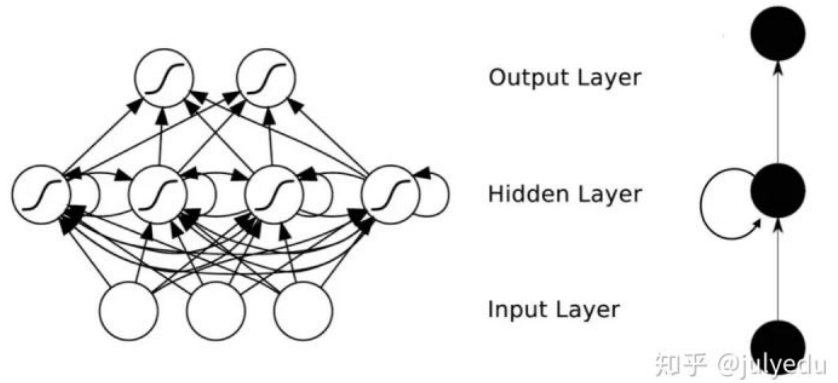
35、什么是RNN？

解析：

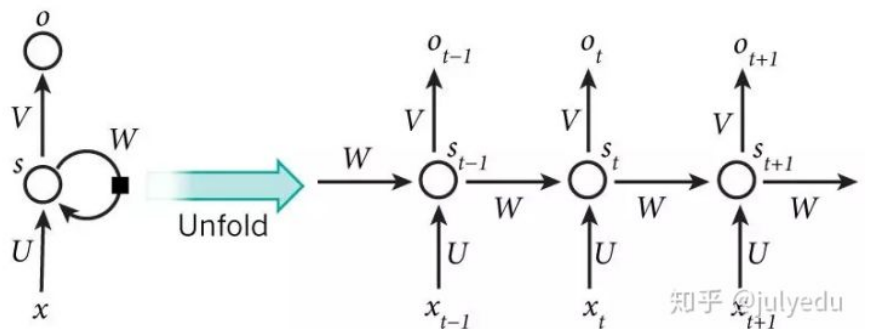
RNNs的目的使用来处理序列数据。在传统的神经网络模型中，是从输入层到隐含层再到输出层，层与层之间是全连接的，每层之间的节点是无连接的。但是这种普通的神经网络对于很多问题却无能为力。例如，你要预测句子的下一个单词是什么，一般需要用前面的单词，因为一个句子中前后单词并不是独立的。

RNNs之所以称为循环神经网络，即一个序列当前的输出与前面的输出也有关。具体的表现形式为网络会对前面的信息进行记忆并应用于当前输出的计算中，即隐藏层之间的节点不再无连接而是有连接的，并且隐藏层的输入不仅包括输入层的输出还包括上一时刻隐藏层的输出。

理论上，RNNs能够对任何长度的序列数据进行处理。但是在实践中，为了降低复杂性往往假设当前的状态只与前面的几个状态相关，下图便是一个典型的RNNs：



RNNs包含输入单元(Input units), 输入集标记为 $\{x_0, x_1, \dots, x_t, x_{t+1}, \dots\}$, 而输出单元(Output units)的输出集则被标记为 $\{y_0, y_1, \dots, y_t, y_{t+1}, \dots\}$ 。RNNs还包含隐藏单元(Hidden units), 我们将其输出集标记为 $\{s_0, s_1, \dots, s_t, s_{t+1}, \dots\}$, 这些隐藏单元完成了最为主要的工作。你会发现, 在图中: 有一条单向流动的信息流是从输入单元到达隐藏单元的, 与此同时另一条单向流动的信息流从隐藏单元到达输出单元。在某些情况下, RNNs会打破后者的限制, 引导信息从输出单元返回隐藏单元, 这些被称为“Back Projections”, 并且隐藏层的输入还包括上一隐藏层的状态, 即隐藏层内的节点可以自连也可以互连。



上图将循环神经网络进行展开成一个全神经网络。例如, 对一个包含5个单词的语句, 那么展开的网络便是一个五层的神经网络, 每一层代表一个单词。对于该网络的计算过程如下:

- (1) x_t 表示第 $t, t=1, 2, 3 \dots$ 步(step)的输入。比如, x_1 为第二个词的 one-hot 向量(根据上图, x_0 为第一个词);
- (2) s_t 为隐藏层的第 t 步的状态, 它是网络的记忆单元。 s_t 根据当前输入层的输出与上一步隐藏层的状态进行计算。 $s_t = f(Ux_t + Ws_{t-1})$, 其中 f 一般是非线性的激活函数, 如tanh或ReLU, 在计算 s_0 时, 即第一个单词的隐藏层状态, 需要用到 s_{-1} , 但是其并不存在, 在实现中一般置为0向量;
- (3) o_t 是第 t 步的输出, 如下个单词的向量表示, $o_t = \text{softmax}(Vs_t)$.

参考资料:

1. YJango,

<https://www.zhihu.com/question/41949741>

2. 一只鸟的天空,

36、简单说下sigmoid激活函数

解析：

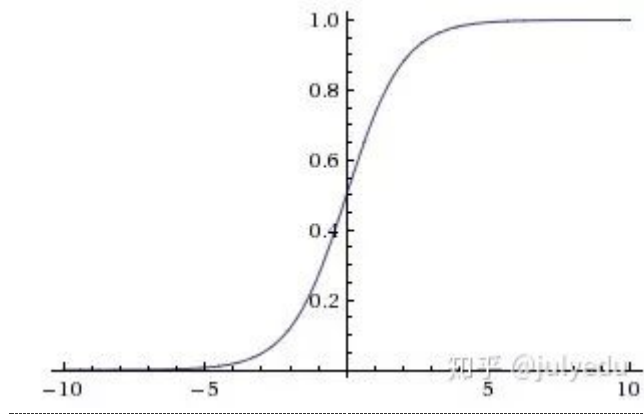
常用的非线性激活函数有sigmoid、tanh、relu等等，前两者sigmoid/tanh比较常见于全连接层，后者relu常见于卷积层。这里先简要介绍下最基础的sigmoid函数（btw，在本博客中SVM那篇文章开头有提过）。

sigmoid的函数表达式如下

$$g(z) = \frac{1}{1 + e^{-z}}$$

其中z是一个线性组合，比如z可以等于： $b + w_1 \cdot x_1 + w_2 \cdot x_2$ 。通过代入很大的正数或很小的负数到g(z)函数中可知，其结果趋近于0或1。

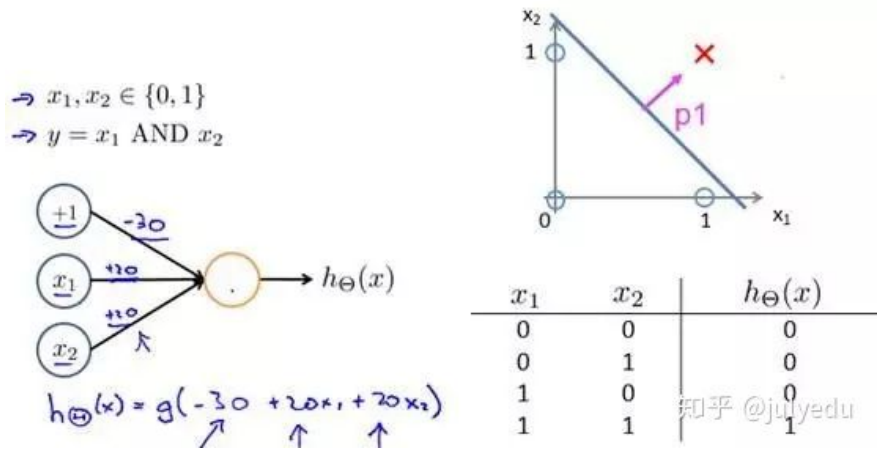
因此，sigmoid函数g(z)的图形表示如下（横轴表示定义域z，纵轴表示值域g(z)）：



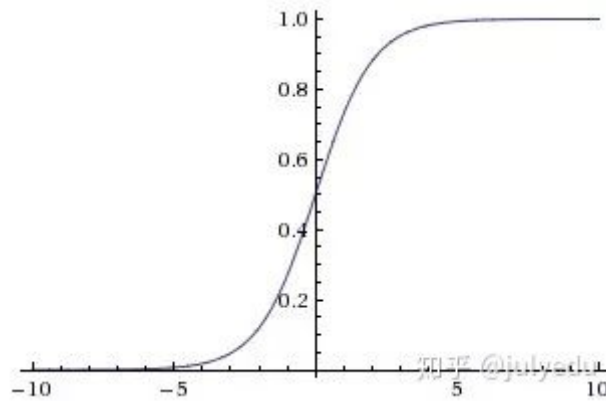
也就是说，sigmoid函数的功能是相当于把一个实数压缩至0到1之间。当z是非常大的正数时，g(z)会趋近于1，而z是非常小的负数时，则g(z)会趋近于0。

压缩至0到1有何用处呢？用处是这样一来便可以把激活函数看作一种“分类的概率”，比如激活函数的输出为0.9的话便可以解释为90%的概率为正样本。

举个例子，如下图（图引自Stanford机器学习公开课）



$z = b + w_1 \cdot x_1 + w_2 \cdot x_2$, 其中 b 为偏置项 假定取 -30 , w_1 、 w_2 都取为 20



如果 $x_1 = 0, x_2 = 0$, 则 $z = -30$, $g(z) = 1 / (1 + e^{-z})$ 趋近于 0 。此外, 从上图 sigmoid 函数的图形上也可以看出, 当 $z = -30$ 的时候, $g(z)$ 的值趋近于 0

如果 $x_1 = 0, x_2 = 1$, 或 $x_1 = 1, x_2 = 0$, 则 $z = b + w_1 \cdot x_1 + w_2 \cdot x_2 = -30 + 20 = -10$, 同样, $g(z)$ 的值趋近于 0

如果 $x_1 = 1, x_2 = 1$, 则 $z = b + w_1 \cdot x_1 + w_2 \cdot x_2 = -30 + 20 \cdot 1 + 20 \cdot 1 = 10$, 此时, $g(z)$ 趋近于 1 。

换言之, 只有 x_1 和 x_2 都取 1 的时候, $g(z) \rightarrow 1$, 判定为正样本; 而当只要 x_1 或 x_2 有一个取 0 的时候, $g(z) \rightarrow 0$, 判定为负样本, 如此达到分类的目的。

综上, sigmoid 函数, 是逻辑斯蒂回归的压缩函数, 它的性质是可以把分隔平面压缩到 $[0, 1]$ 区间一个数 (向量), 在线性分割平面值为 0 时候正好对应 sigmoid 值为 0.5 , 大于 0 对应 sigmoid 值大于 0.5 、小于 0 对应 sigmoid 值小于 0.5 ; 0.5 可以作为分类的阈值; exp 的形式最值求解时候比较方便, 用相乘形式作为 logistic 损失函数, 使得损失函数是凸函数; 不足之处是 sigmoid 函数在 y 趋于 0 或 1 时候有死区, 控制不好在 bp 形式传递 loss 时候容易造成梯度弥散。

37、rcnn、fast-rcnn 和 faster-rcnn 三者的区别是什么

解析：

首先膜拜RBG (Ross B. Girshick) 大神，不仅学术牛，工程也牛，代码健壮，文档详细，clone下来就能跑。断断续续接触detection几个月，将自己所知做个大致梳理，业余级新手，理解不对的地方还请指正。

传统的detection主流方法是DPM(Deformable parts models)，在VOC2007上能到43%的mAP，虽然DPM和CNN看起来差别很大，但RBG大神说“Deformable Part Models are Convolutional Neural Networks” (<http://arxiv.org/abs/1409.5403>)。

CNN流行之后，Szegedy做过将detection问题作为回归问题的尝试 (Deep Neural Networks for Object Detection)，但是效果差强人意，在VOC2007上mAP只有30.5%。既然回归方法效果不好，而CNN在分类问题上效果很好，那么为什么不把detection问题转化为分类问题呢？

RBG的RCNN使用region proposal (具体用的是Selective Search Koen van de Sande: Segmentation as Selective Search for Object Recognition) 来得到有可能得到是object的若干 (大概 10^3 量级) 图像局部区域，然后把这些区域分别输入到CNN中，得到区域的feature，再在feature上加上分类器，判断feature对应的区域是属于具体某类object还是背景。当然，RBG还用了区域对应的feature做了针对boundingbox的回归，用来修正预测的boundingbox的位置。

RCNN在VOC2007上的mAP是58%左右。RCNN存在着重复计算的问题 (proposal的region有几千个，多数都是互相重叠，重叠部分会被多次重复提取feature)，于是RBG借鉴Kaiming He的SPP-net的思路单枪匹马搞出了Fast-RCNN，跟RCNN最大区别就是Fast-RCNN将proposal的region映射到CNN的最后一层conv layer的feature map上，这样一张图片只需要提取一次feature，大大提高了速度，也由于流程的整合以及其他原因，在VOC2007上的mAP也提高到了68%。

探索是无止境的。Fast-RCNN的速度瓶颈在Region proposal上，于是RBG和Kaiming He一帮人将Region proposal也交给CNN来做，提出了Faster-RCNN。Faster-RCNN中的region proposal network实质是一个Fast-RCNN，这个Fast-RCNN输入的region proposal的是固定的 (把一张图片划分成 $n \times n$ 个区域，每个区域给出9个不同ratio和scale的proposal)，输出的是对输入的固定proposal是属于背景还是前景的判断和对齐位置的修正 (regression)。Region proposal network的输出再输入第二个Fast-RCNN做更精细的分类和Boundingbox的位置修正。

Faster-RCNN速度更快了，而且用VGG net作为feature extractor时在VOC2007上mAP能到73%。个人觉得制约RCNN框架内的方法精度提升的瓶颈是将detection问题转化成了对图片局部区域的分类问题后，不能充分利用图片局部object在整个图片中的context信息。

可能RBG也意识到了这一点，所以他最新的一篇文章YOLO (<http://arxiv.org/abs/1506.02640>) 又回到了regression的方法下，这个方法效果很好，在VOC2007上mAP能到63.4%，而且速度非

常快，能达到对视频的实时处理（油管视频：

<https://www.youtube.com/channel/UC7ev3hNVkx4DzZ3LO19oebg>），虽然不如Fast-RCNN，但是比传统的实时方法精度提升了太多，而且我觉得还有提升空间。

38、在神经网络中，有哪些办法防止过拟合？

解析：

缓解过拟合：

- ① Dropout
- ② 加L1/L2正则化
- ③ BatchNormalization
- ④ 网络bagging

39、CNN是什么，CNN关键的层有哪些？

解析：

CNN是卷积神经网络，具体详见此文：

https://blog.csdn.net/v_july_v/article/details/51812459。

其关键层有：

- ① 输入层，对数据去均值，做data augmentation等工作
- ② 卷积层，局部关联抽取feature
- ③ 激活层，非线性变化
- ④ 池化层，下采样
- ⑤ 全连接层，增加模型非线性
- ⑥ 高速通道，快速连接
- ⑦ BN层，缓解梯度弥散

40、GRU是什么？GRU对LSTM做了哪些改动？

解析：

GRU是Gated Recurrent Units，是循环神经网络的一种。

GRU只有两个门（update和reset），LSTM有三个门（forget, input, output），GRU直接将hidden state 传给下一个单元，而LSTM用memory cell 把hidden state 包装起来。

41、请简述应当从哪些方向上思考和解决深度学习中出现的的over fitting问题？

解析：

如果模型的训练效果不好，可先考察以下几个方面是否有可以优化的地方。

(1)选择合适的损失函数 (choosing proper loss)

神经网络的损失函数是非凸的，有多个局部最低点，目标是找到一个可用的最低点。非凸函数是凹凸不平的，但是不同的损失函数凹凸起伏的程度不同，例如下述的平方损失和交叉熵损失，后者起伏更大，且后者更容易找到一个可用的最低点，从而达到优化的目的。

- Square Error (平方损失)
- Cross Entropy (交叉熵损失)

(2)选择合适的Mini-batch size

采用合适的Mini-batch进行学习，使用Mini-batch的方法进行学习，一方面可以减少计算量，一方面有助于跳出局部最优点。因此要使用Mini-batch。更进一步，batch的选择非常重要，batch取太大会陷入局部最小值，batch取太小会抖动厉害，因此要选择一个合适的batch size。

(3)选择合适的激活函数 (New activation function)

使用激活函数把卷积层输出结果做非线性映射，但是要选择合适的激活函数。

- Sigmoid函数是一个平滑函数，且具有连续性和可微性，它的最大优点就是非线性。但该函数的两端很缓，会带来猪队友的问题，易发生学不动的情况，产生梯度弥散。
- ReLU函数是如今设计神经网络时使用最广泛的激活函数，该函数为非线性映射，且简单，可缓解梯度弥散。

(4)选择合适的自适应学习率 (adaptive learning rate)

- 学习率过大，会抖动厉害，导致没有优化提升
- 学习率太小，下降太慢，训练会很慢

(5)使用动量 (Momentum)

在梯度的基础上使用动量，有助于冲出局部最低点。

如果以上五部分都选对了，效果还不好，那就是产生过拟合了，可使如下方法来防止过拟合，分别是

- 1.早停法 (earlyly stoping)。早停法将数据分成训练集和验证集，训练集用来计算梯度、更新权重和阈值，验证集用来估计误差，若训练集误差降低但验证集误差升高，则停止训练，同时返回具有最小验证集误差的连接权和阈值。
- 2.权重衰减 (Weight Decay)。到训练的后后期，通过衰减因子使权重的梯度下降地越来越缓。

- 3.Dropout。Dropout是正则化的一种处理，以一定的概率关闭神经元的通路，阻止信息的传递。由于每次关闭的神经元不同，从而得到不同的网路模型，最终对这些模型进行融合。

- 4.调整网络结构（Network Structure）。

42、神经网络中，是否隐藏层如果具有足够数量的单位，它就可以近似任何连续函数？

解析：

通用逼近性定理指出，一个具有单个隐藏层和标准激活函数的简单前馈神经网络（即多层感知器），如果隐藏层具有足够数量的单位，它就可以近似任何连续函数。让我们在实践中看一下，看看需要多少单位来近似一些特定函数。

方法：我们将在 50 个数据点 (x,y) 上训练一个 1 层神经网络，这些数据点从域 [-1,1] 上的以下函数中绘制，所得拟合的均方误差（mean square error, MSE）。我们将尝试以下函数（你可随时通过更改以下代码来尝试自己的函数。）

$$\begin{aligned} \bullet f(x) &= \sin(12x) \\ \bullet g(x) &= -2x^4 + x^3 + x^2 - 2x + 0 \\ \bullet h(x) &= |x| \end{aligned}$$

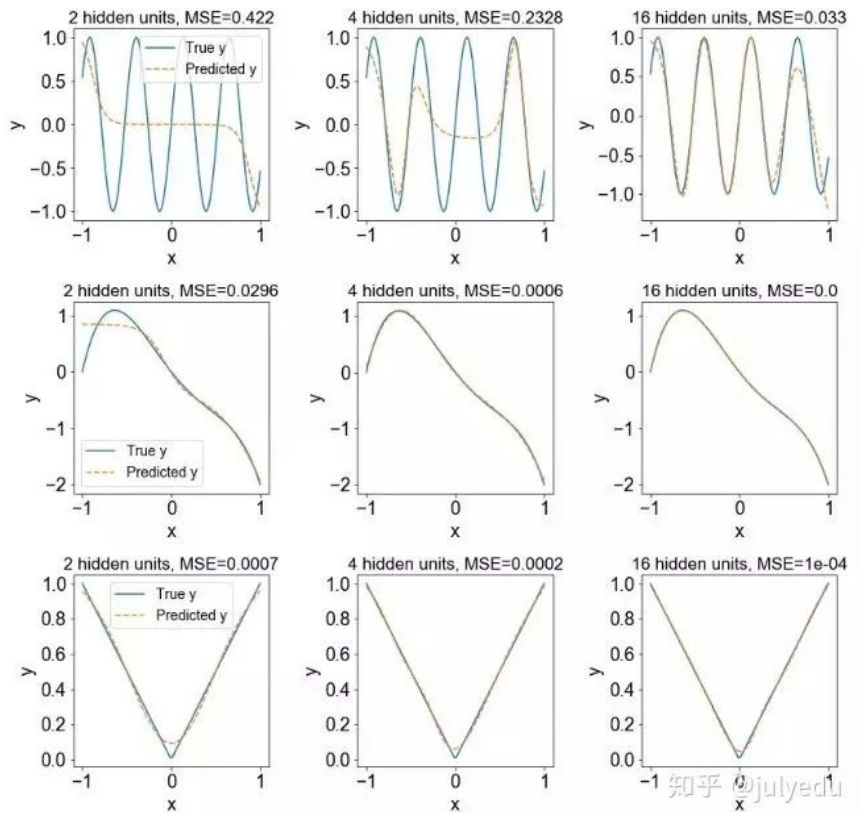
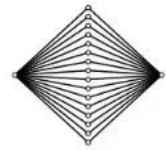
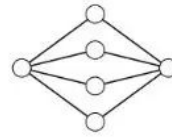
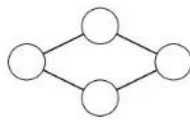
知乎 @julyedu

假设：随着隐藏层中单位的数量增加，所得拟合的正确率（Accuracy）将会增加（误差将会减少）。

```
1 functions = [sin(omega*12),
2             polynomial([0, 2,1,1, 2]),
3             np.abs]
4 n_hidden_units = [2,4,16]
5 experiment = Experiment1()
6
7 f, axs = plt.subplots(4,3,figsize=(12,15))
8 plt.rc("font",family="sans-serif",size=14)
9
10 for i, n_hidden in enumerate(n_hidden_units):
11     ax = axs[0, i]
12     ax.axis('off')
13     draw_neural_net(ax, .15, .9, .1, .85, [1, n_hidden, 1])
14
15 experiment.initialize()
16 plot_idx = 1
17 for i, function in enumerate(functions):
18     for j, n_hidden in enumerate(n_hidden_units):
19         x_values, y_values, y_pred, loss = experiment.run(n_hidden=n_hidden, function=function, verbose=False)
20         ax = axs[i+1, j]
21         ax.plot(x_values, y_values, '-', label='True y')
22         ax.plot(x_values, y_pred, '--', label='Predicted y')
23         ax.set_xlabel('x')
24         ax.set_ylabel('y')
25         if (j==0):
26             ax.legend()
27         ax.set_title(str(n_hidden)+' hidden units, MSE='+str(np.round(loss,4)))
28
29 plt.tight_layout()
30 experiment.conclude()
```

知乎 @julyedu

运行实验所需的时间： 91.595 s



结论：随着隐藏单位数量的增加，训练数据的逼近误差一般会减小。

讨论：尽管通用逼近定理指出，具有足够参数的神经网络可以近似一个真实的分类 / 回归函数，但它并没有说明这些参数是否可以通过随机梯度下降这样的过程来习得。另外，你可能想知道我们是否可以从理论上计算出需要多少神经元才能很好地近似给定的函数。你可参阅论文《NEURAL NETWORKS FOR OPTIMAL APPROXIMATION OF SMOOTH AND ANALYTIC FUNCTIONS》对此的一些讨论。

43、为什么更深的网络更好？

解析：

在实践中，更深的多层感知器（具有超过一个隐藏层）在许多感兴趣的任务上的表现，在很大程度上都胜过浅层感知器。为什么会出现这种情况呢？有人认为，更深的神经网络仅需更少的参数就可以表达许多重要的函数类。

理论上已经表明，表达简单的径向函数和组合函数需要使用浅层网络的指数级大量参数。但深度神经网络则不然。

剧透警告：我打算用实验来验证这些论文，但我不能这样做（这并不会使论文的结果无效——仅仅因为存在一组神经网络参数，并不意味着它们可

以通过随机梯度下降来轻松习得)。

我唯一能做的就是，某种程度上可靠地再现来自论文《Representation Benefits of Deep Feedforward Networks》的唯一结果，这篇论文提出了一系列困难的分类问题，这些问题对更深层的神经网络而言更容易。

方法：该数据集由沿着 x 轴的 16 个等距点组成，每对相邻点都属于相反的类。一种特殊类型的深度神经网络（一种跨层共享权重的神经网络）具有固定数量（152）的参数，但测试了层的不同数量。

假设：随着具有固定数量参数的神经网络中层数的增加，困难的分类问题的正确率将得到提高。

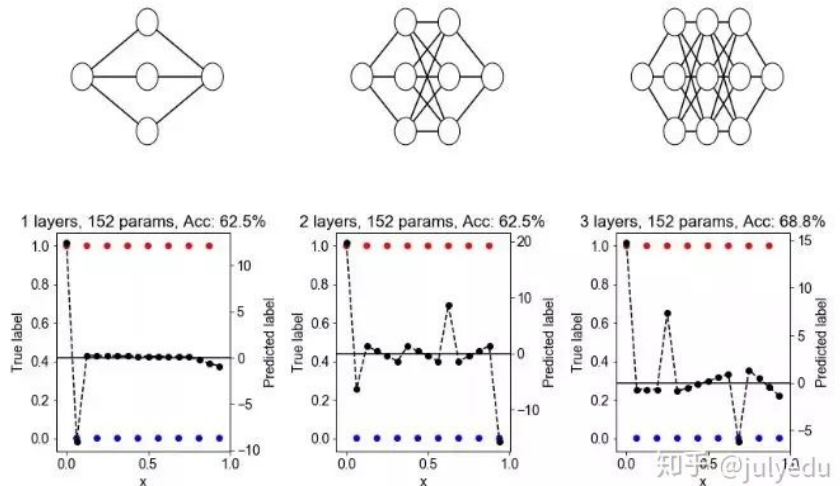
```

1 experiment = Experiment2()
2 experiment.initialize()
3
4 n_hidden_units = [[10],[10,10],[10,10,10]]
5
6 f, axes = plt.subplots(2,3,figsize=(12,8))
7 plt.rc('font',family='sans-serif',size=14)
8
9 for i, n_hidden in enumerate(n_hidden_units):
10     ax = axes[0, i]
11     ax.axis('off')
12     draw_neural_net(ax, [15, 9, 1, .85, 1] + [5]*len(n_hidden) + [1]) #15 hidden units per layer becomes too crowded
13
14 for i, n_hidden in enumerate(n_hidden_units):
15     ax = axes[1, i]
16     x_values, y_values, y_pred, loss, accuracy, n_params = experiment.run(n=16, n_hidden = n_hidden)
17     ax.set_title(str(len(n_hidden))+' layers, '+str(n_params)+' params, Acc: '+str(np.round(100*accuracy,1))+'%')
18     ax.set_xlabel('x')
19     ax.set_ylabel('True label')
20     ax.scatter(x_values, y_values[:,1], markers='o', c=y_values[:,1], cmap='bwr', label='Predicted y')
21     ax2 = ax.twinx()
22     ax2.plot(x_values, y_pred[:,1], y_pred[:,0], 'k--o', label='Predicted y')
23     ax2.axhline(0, color='k', ls='-')
24     ax2.set_ylabel('Predicted label')
25
26 plt.tight_layout()
27 experiment.conclude()

```

知乎 @julyedu

运行实验所需的时间： 28.688 s



此处，红点和蓝点代表属于不同类别的点。黑色的虚线表示最接近神经网络学习的训练数据近似值（若神经网络分配的分数量大于零，则被预测为红点；否则，被预测为蓝点）。零线显示为黑色。

结论：在大多实验中，正确率随深度的增加而增加。

讨论：似乎更深的层允许从输入到输出的学习到的函数出现更多“急弯”。这似乎跟神经网络的轨迹长度有关（即衡量输入沿着固定长度的一维路径变化时，神经网络的输出量是多少）。轨迹长度论文：

<https://arxiv.org/pdf/1606.05336.pdf>

44、更多的数据是否有利于更深的神经网络？

解析：

深度学习和大数据密切相关；通常认为，当数据集的规模大到足够克服过拟合时，深度学习只会比其他技术（如浅层神经网络和随机森林）更有效，并更有利于增强深层网络的表达性。我们在一个非常简单数据集上进行研究，这个数据集由高斯样本混合而成。

方法：数据集由两个 12 维的高斯混合而成，每个高斯生成属于一个类的的数据。两个高斯具有相同的协方差矩阵，但也意味着在第 i 个维度上有 $1/i!i$ 单位。这个想法是基于：有一些维度，允许模型很容易区分不同的类，而其他维度则更为困难，但对区别能力还是有用的。

假设：随着数据集大小的增加，所有技术方法的测试正确率都会提高，但深度模型的正确率会比非深度模型的正确率要高。我们进一步预计非深度学习技术的正确率将更快地饱和。

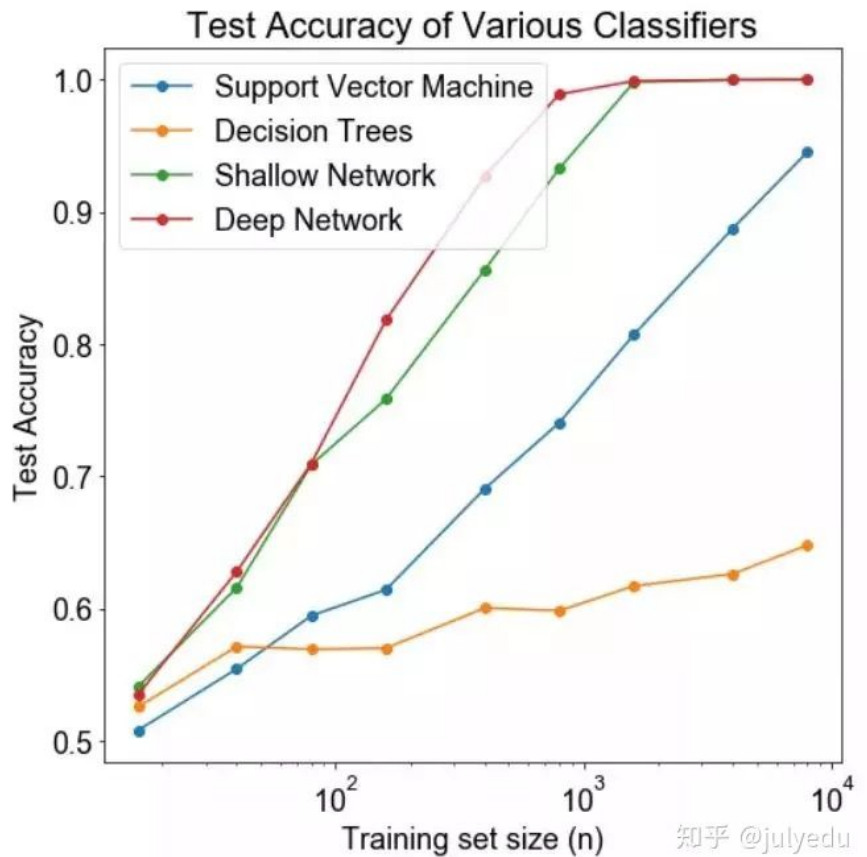
```

1 from sklearn.neural_network import MLPClassifier
2 from sklearn.tree import DecisionTreeClassifier
3 from sklearn.naive_bayes import GaussianNB
4 from sklearn.svm import SVC
5
6 ns = [20, 50, 100, 200, 500, 1000, 2000, 5000, 10000]
7 test_size=0.2; train_size=1-test_size
8
9 class_seps = [1/(1-i) for i in range(12)]
10
11 classifiers = [SVC(),
12               DecisionTreeClassifier(),
13               MLPClassifier(),
14               MLPClassifier(hidden_layer_sizes=(100,100)),
15               ]
16
17 classifier_names = ['Support Vector Machine',
18                   'Decision Trees',
19                   'Shallow Network',
20                   'Deep Network']
21
22 experiment = Experiment3()
23 experiment.initialize()
24
25 plt.figure(figsize=[8,8]); plt.rc("font",family="sans-serif",size=18)
26 accuracies = experiment.run(ns=ns,classifiers=classifiers,class_seps=class_seps,d=12,ifers=3, test_size=test_size)
27 plt.title('Test Accuracy of Various Classifiers')
28 plt.plot(train_size*np.array(ns), np.mean(accuracies,axis=2), '-o')
29 plt.xlabel('Training set size (n)')
30 plt.ylabel('Test Accuracy')
31 plt.xscale('log')
32 plt.legend(classifier_names)
33
34 experiment.conclude()

```

知乎 @julyedu

运行实验所需的时间： 138.239 s



结论：神经网络在数据集大小方面上表现始终优于 SVM 和随机森林。随着数据集大小的增加，性能上的差距也随之增加，至少在神经网络的正确率开始饱和之前，这表明神经网络更有效地利用了不断增加的数据集。然而，如果有足够的数据，即使是 SVM 也会有可观的正确率。深度网络比浅层网络的表现更好。

讨论：虽然增加的数据集大小确实会像我们预计的那样有利于神经网络。但有趣的是，在相对较小的数据集上，神经网络已经比其他技术表现得更好。似乎 2 层网络并没有显著的过拟合，即使我们预计某些特征（如 6-12 特征，信号水平低）导致网络过拟合。同样有趣的是，SVM 看上去似乎有足够的数据来接近于 1.0。

45、不平衡数据是否会摧毁神经网络？

解析：

当数据集不平衡时（如一个类的样本比另一个类还多），那么神经网络可能就无法学会如何区分这些类。在这个实验中，我们探讨这一情况是否存在。同时我们还探讨了过采样是否可以减轻问题带来的影响，这是一种流行的补救措施，该措施使用少数类中抽样替换的样本。

方法：我们生成两个二维的结果（结果未在这里显示，表明相同的结果适用于更高维）高斯，每个产生属于一个类别的数据。两个高斯具有相同的协方差矩阵，但它们的意思是在第 i 个维度上相距 $1/i^{1/i}$ 单位。每个训练数据集由 1,200 个数据点组成，但我们将类别不平衡从 1:1 变为 1:99。测试数据集以 1:1 的比例保持固定，以便于性能比较，并由 300 个点组成。我们还会在每种情况下显示决策边界。

假设：我们预计测试正确率会随着类别不平衡的增加而降低，但我们预计过采样可以缓解这个问题。

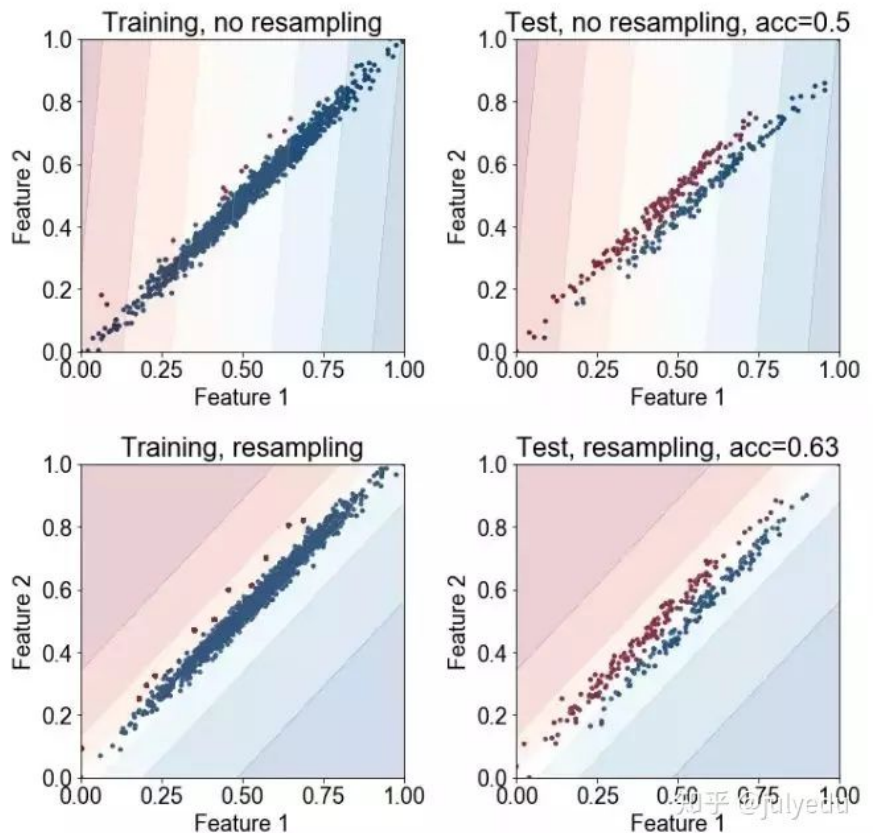
```

1 ratios = [1, 2, 5, 10, 20, 50, 100]
2 experiment = Experiment4()
3 experiment.initialize()
4
5 plt.figure(figsize=[8,6]); plt.rc("font",family="sans-serif",size=18)
6
7 accuracies, saved_covs = experiment.run(ratios=ratios,
8                                         d=2,
9                                         iters=5,
10                                        resample=False)
11 plt.plot(ratios, np.mean(accuracies,axis=1), "-o")
12 plt.fill_between(ratios, np.mean(accuracies,axis=1) - np.std(accuracies,axis=1), np.mean(accuracies,axis=1) + np.std(accuracies,axis=1), alpha=0.3)
13
14 accuracies, _ = experiment.run(ratios=ratios,
15                                d=2,
16                                iters=5,
17                                resample=True,
18                                load_covs=saved_covs)
19
20 plt.plot(ratios, np.mean(accuracies,axis=1), "-o")
21 plt.fill_between(ratios, np.mean(accuracies,axis=1) - np.std(accuracies,axis=1), np.mean(accuracies,axis=1) + np.std(accuracies,axis=1), alpha=0.3)
22
23 plt.title('Accuracy on Imbalanced Datasets')
24 plt.xlabel('Ratio (X:1)')
25 plt.ylabel('Test Accuracy')
26 plt.xscale('log')
27 plt.legend(['Without resampling', 'With resampling'])
28
29 plt.figure(figsize=[10,5]); plt.rc("font",family="sans-serif",size=18)
30
31 accuracies, saved_covs, X_train, y_train, X_test, y_test, y_pred, grid_pred = experiment.run(ratios=[100],
32                                                  d=2,
33                                                  iters=1,
34                                                  resample=False,
35                                                  classify_grid=True)
36
37 plt.subplot(1,2,1)
38 plot_decision_boundary(X_train, y_train, grid_pred)
39 plt.title('Training, no resampling')
40
41 plt.subplot(1,2,2)
42 plot_decision_boundary(X_test, y_test, grid_pred)
43 plt.title('Test, no resampling, acc=' + str(round(accuracies[0][0],2)))
44 plt.tight_layout()
45
46 accuracies, _, X_train, y_train, X_test, y_test, y_pred, grid_pred = experiment.run(ratios=[100],
47                                                  d=2,
48                                                  iters=1,
49                                                  resample=True,
50                                                  classify_grid=True,
51                                                  load_covs=saved_covs)
52
53 plt.subplot(1,2,1)
54 plot_decision_boundary(X_train, y_train, grid_pred)
55 plt.title('Training, resampling')
56
57 plt.subplot(1,2,2)
58 plot_decision_boundary(X_test, y_test, grid_pred)
59 plt.title('Test, resampling, acc=' + str(round(accuracies[0][0],2)))
60 plt.tight_layout()
61
62 experiment.conclude()

```

知乎 @julyedu

运行实验所需的时间： 392.157 s



最下面的四张图显示了连同训练点（左）或测试点（右）绘制的决策边界的数量。第一行显示没有重采样法的结果，底部显示了使用重采样法的结果。

结论： 研究表明，类的不平衡无疑地降低了分类的正确率。重采样法可以显著提高性能。

讨论： 重采样法对提高分类正确率有显著的影响，这可能有点让人惊讶了，因为它并没有将分类器展示少数类中的新训练的样本。但该图显示，重采样法足以“助推（nudge）”或将决策边界推向正确的方向。在重采样法不是有效的情况下，那么可能需要复合方式来合成新的训练样本，以提高正确率。

46、你如何判断一个神经网络是记忆还是泛化？

解析：

具有许多参数的神经网络具有记忆大量训练样本的能力。那么，神经网络是仅仅记忆训练样本（然后简单地根据最相似的训练点对测试点进行分类），还是它们实际上是在提取模式并进行归纳？这有什么不同吗？

人们认为存在不同之处的一个原因是，神经网络学习随机分配标签不同于它学习重复标签的速度。这是 Arpit 等人在论文中使用的策略之一。让我们看看是否有所区别？

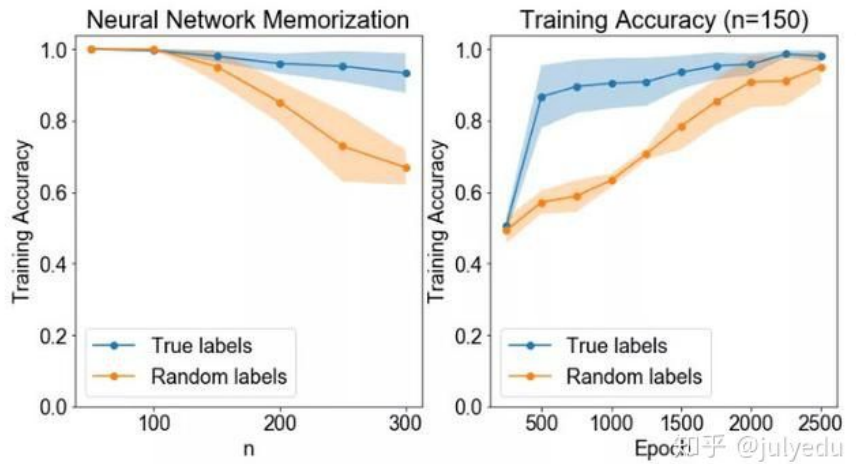
方法： 首先我们生成一个 6 维高斯混合，并随机分配它们的标签。我们测量训练数据的正确率，以增加数据集的大小，了解神经网络的记忆能力。然后，我们选择一个神经网络能力范围之内的数据集大小，来记忆并观察训练过程中神经网络与真实标签之间是否存在本质上的差异。特别是，我们观察每个轮数的正确率，来确定神经网络是真正学到真正的标签，还是随机标签。

假设： 我们预计，对随机标签而言，训练应该耗费更长的时间。而真正标签则不然。

```
1  m = [10, 100, 150, 200, 250, 300]
2  experiment = Experiment(5)
3  experiment.initialize()
4
5  plt.rc("font", family="sans-serif", size=18)
6  fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))
7
8  accuracies = experiment.run(m, hidden_layer_size=[100, 200, 300], return_accuracy_per_epoch=True)
9  ax1.plot(m, np.mean(accuracies[:, :, 0]), "-o")
10 ax1.fill_between(m, np.mean(accuracies[:, :, 0]), ax1.get_yaxis().get_ticks()[0], alpha=0.3)
11 ax1.set_ylabel("0.5")
12
13 steps = np.linspace(100, 200, 10)
14 ax2.plot(steps, np.mean(accuracies[:, :, 1]), "-o")
15 ax2.fill_between(steps, np.mean(accuracies[:, :, 1]), ax2.get_yaxis().get_ticks()[0], alpha=0.3)
16 ax2.set_ylabel("0.5")
17
18 accuracies = experiment.run(m, hidden_layer_size=[100, 200, 300], randomize=True, return_accuracy_per_epoch=True)
19 ax1.plot(m, np.mean(accuracies[:, :, 0]), "-o")
20 ax1.fill_between(m, np.mean(accuracies[:, :, 0]), ax1.get_yaxis().get_ticks()[0], alpha=0.3)
21 ax1.legend(["True Labels", "Random Labels"])
22 ax1.set_ylabel("0.5")
23 ax1.set_xlabel("Training Accuracy")
24 ax1.set_title("Neural Network Memorization")
25
26 ax2.plot(steps, np.mean(accuracies[:, :, 1]), "-o")
27 ax2.fill_between(steps, np.mean(accuracies[:, :, 1]), ax2.get_yaxis().get_ticks()[0], alpha=0.3)
28 ax2.legend(["True Labels", "Random Labels"])
29 ax2.set_xlabel("Epoch")
30 ax2.set_ylabel("Training Accuracy")
31 ax2.set_title("Training Accuracy (s=150)")
32
33 experiment.conclude()
```

知乎 @julyedu

运行实验所需的时间： 432.275 s



结论：神经网络的记忆能力约为 150 个训练点。但即便如此，神经网络也需要更长的时间来学习随机标签，而不是真实值（ground truth）标签。

讨论：这个结果并不令人感到意外。我们希望真正的标签能够更快的学到，如果一个神经网络学会正确地分类一个特定的数据点时，它也将学会分类其他类似的数据点——如果标签是有意义的，但前提它们不是随机的！

47、无监督降维提供的是帮助还是摧毁？

解析：

当处理非常高维的数据时，神经网络可能难以学习正确的分类边界。在这些情况下，可以考虑在将数据传递到神经网络之前进行无监督的降维。这种做法提供的是帮助还是摧毁呢？

方法：我们生成两个10维高斯混合。高斯具有相同的协方差矩阵，但在每个维度上都有一个由 1 隔开的均值。然后，我们在数据中添加“虚拟维度”，这些特征对于两种类型的高斯都是非常低的随机值，因此对分类来说没有用处。

然后，我们将结果数据乘以一个随机旋转矩阵来混淆虚拟维度。小型数据集大小 ($n=100$) 使神经网络难以学习分类边界。因此，我们将数据 PCA 为更小的维数，并查看分类正确率是否提高。

假设：我们预计 PCA 将会有所帮助，因为变异最多的方向（可能）与最有利于分类的方向相一致。

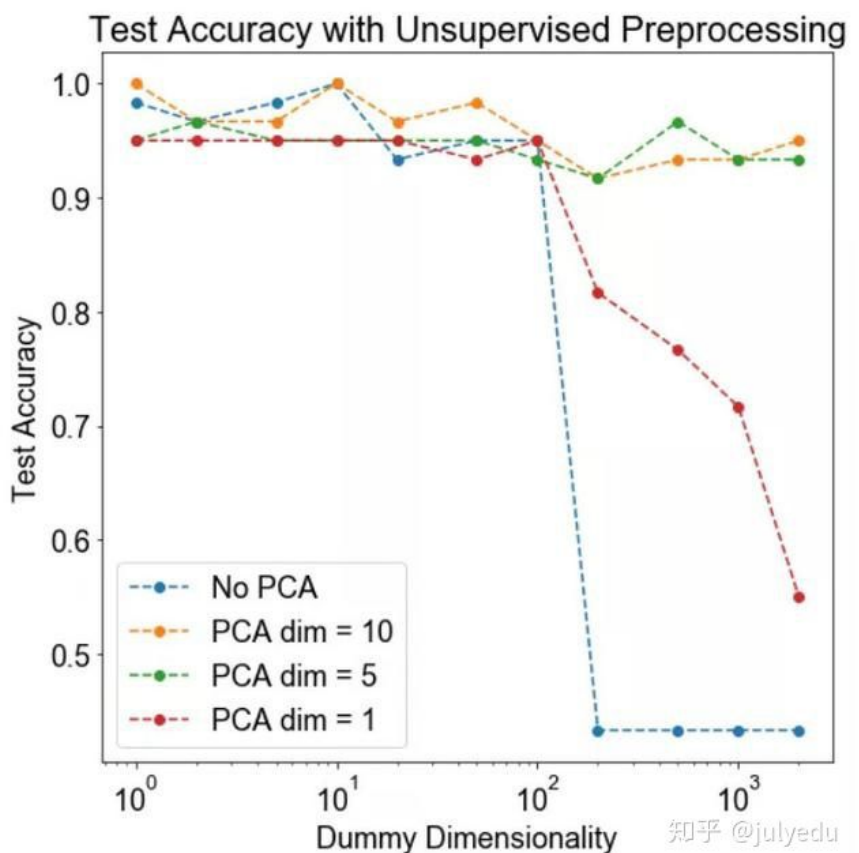
```

1 experiment = Experiment6()
2 experiment.initialize()
3
4 dummy_dims = [1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000]
5 pca_dims = [None, 10, 5, 1]
6
7 accuracies = experiment.run(dummy_dims=dummy_dims,
8                             noise_level=1,
9                             pca_dims=pca_dims)
10
11 plt.figure(figsize=[8,8]); plt.rc("font",family="sans-serif",size=18)
12 plt.plot(dummy_dims, np.mean(accuracies,axis=0), '--o')
13
14 plt.title('Test Accuracy with Unsupervised Preprocessing')
15 plt.xlabel('Dummy Dimensionality')
16 plt.ylabel('Test Accuracy')
17 plt.xscale('log')
18 plt.legend(['No PCA', 'PCA dim = 10', 'PCA dim = 5', 'PCA dim = 1'])
19
20 experiment.conclude()

```

知乎 @julyedu

运行实验所需的时间： 182.938 s



结论：当维度非常大时，无监督的 PCA 步骤可以显著改善下游分类。

讨论：我们观察到一个有趣的阈值行为。当维数超过 100 时（有趣的是，这数字是数据集中数据点的数量——这值得进一步探讨），分类的质量会有显著的下降。在这些情况下，5~10 维的 PCA 可显著地改善下游分类。

48、是否可以将任何非线性作为激活函数？

解析：

在通过具有超出典型 ReLU() 和 tanh() 的特殊激活函数的神经网络获得小幅提高的研究，已有多篇论文报道。我们并非试图开发专门的激活函数，而是简单地询问它是否可能在神经网络中使用任何旧的非线性函数？

方法：我们生成著名的二维卫星数据集，并训练一个具有两个隐藏层的神经网络来学习对数据集进行分类。我们尝试了六种不同的激活函数。

- $f(x) = \text{ReLU}(x)$
- $f(x) = \tanh(x)$
- $f(x) = x^2$
- $f(x) = \sin(x)$
- $f(x) = \text{sign}(x)$
- $f(x) = x$

知乎 @julyedu

假设：我们预计恒等函数执行很差（因为直到最后一个 softmax 层，网络仍然保持相当的线性）。我们可能会进一步期望标准的激活函数能够发挥最好的效果。

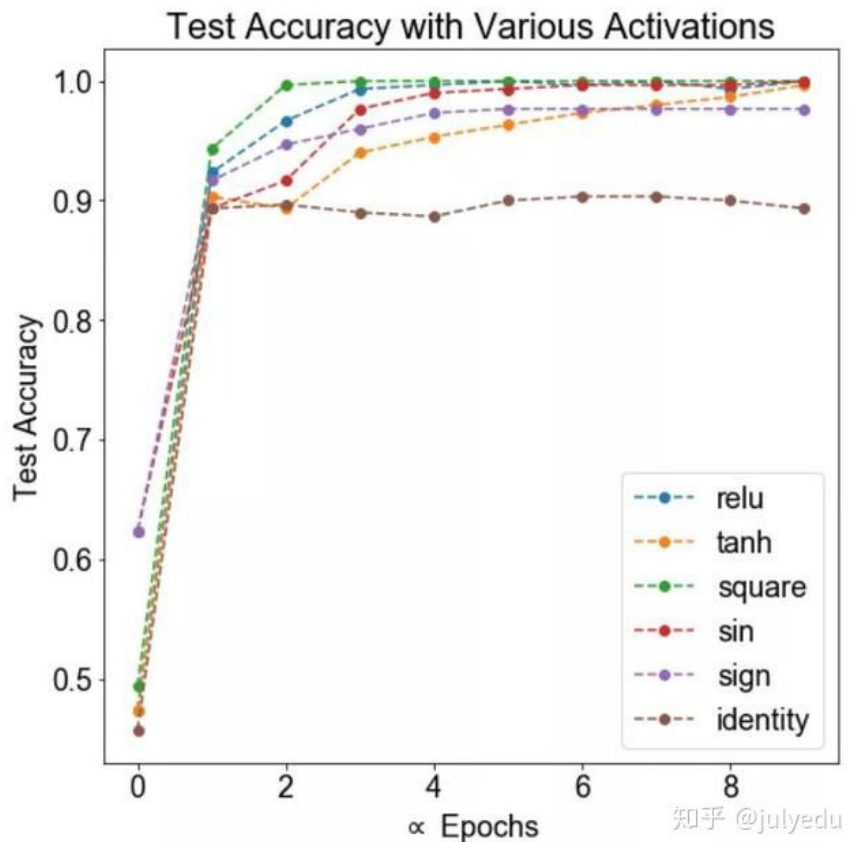
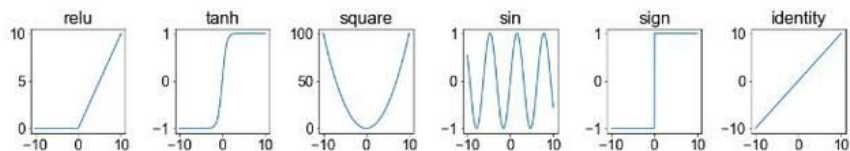
```

1 experiment = Experiment7()
2 experiment.initialize()
3
4 activations = [tf.nn.relu, tf.tanh, tf.square, tf.sin, tf.sign, tf.identity]
5 x = np.linspace(-10,10,1000)
6
7 accuracies = experiment.run(activations=activations,
8                             iters=3)
9
10
11 fig, axs = plt.subplots(1, 6, figsize=[15,3])
12 sess = tf.Session()
13 for i, ax in enumerate(axs):
14     ax.plot(x, sess.run(activations[i](x)))
15     ax.set_title(activations[i].__name__)
16 plt.tight_layout()
17
18 plt.figure(figsize=[8,8]); plt.rc("font",family="sans-serif",size=18)
19 plt.plot(np.mean(accuracies,axis=0), '--o')
20
21 plt.title('Test Accuracy with Various Activations')
22 plt.xlabel(r'$\propto$ Epochs')
23 plt.ylabel('Test Accuracy')
24 plt.legend([a.__name__ for a in activations])
25
26 experiment.conclude()

```

知乎 @julyedu

运行实验所需的时间: 22.745 s



结论：除去 $\text{sign}(x)$ 外，所有的非线性激活函数对分类任务都是非常有效的。

讨论：结果有些令人吃惊，因为所有函数都同样有效。事实上，像 x^2 这样的对称激活函数表现得和 ReLUs 一样好！从这个实验中，我们应该谨慎地推断出太多的原因。

49、批大小如何影响测试正确率？

解析：

方法：我们生成两个 12 维高斯混合。高斯具有相同的协方差矩阵，但在每个维度上都有一个由 1 隔开的均值。该数据集由 500 个高斯组成，其中 400 个用于训练，100 个用于测试。我们在这个数据集上训练一个神经网络，使用不同的批大小，从 1 到 400。我们测量了之后的正确率。

假设：我们期望较大的批大小会增加正确率（较少的噪声梯度更新），在一定程度上，测试的正确率将会下降。我们预计随着批大小的增加，运行时间应有所下降。

运行实验所需的时间： 293.145 s

结论：正如我们预期那样，运行时间确实随着批大小的增加而下降。然而，这导致了测试正确率的妥协，因为测试正确率随着批大小的增加而单调递减。

讨论：这很有趣，但这与普遍的观点不一致，严格来说，即中等规模的批大小更适用于训练。这可能是由于我们没有调整不同批大小的学习率。因为更大的批大小运行速度更快。总体而言，对批大小的最佳折衷似乎是为 64 的批大小。

50、损失函数重要吗？

解析：

对于分类任务，通常使用交叉熵损失函数。如果我们像通常在回归任务中那样使用均方差，结果会怎么样？我们选择哪一个会很重要么？

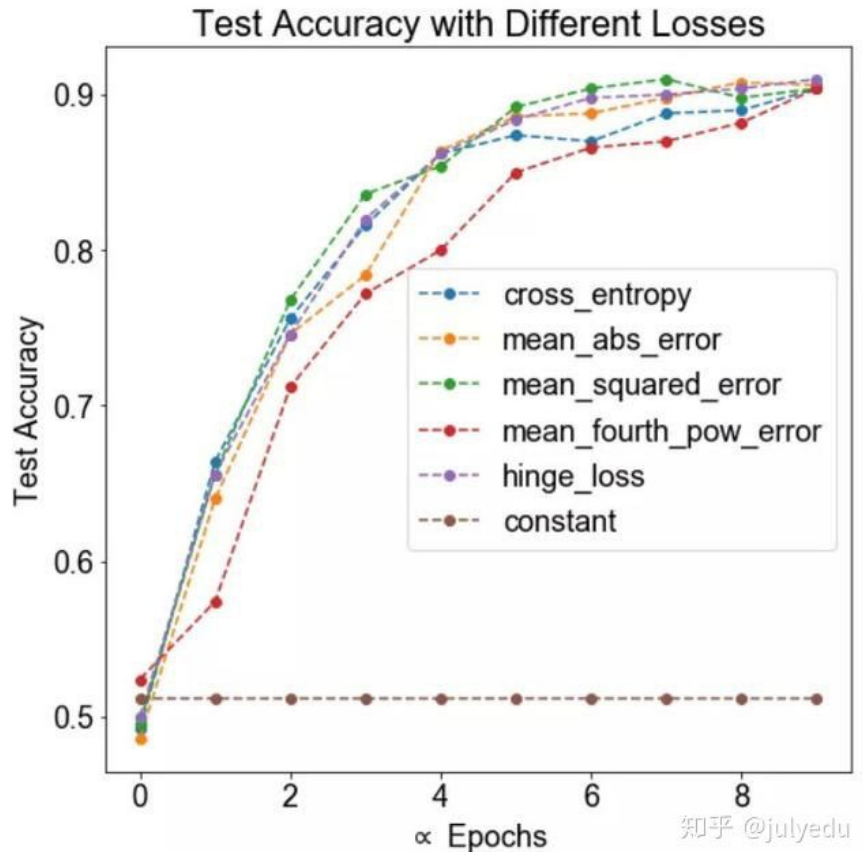
方法：我们生成两个 12 维高斯混合。高斯具有相同的协方差矩阵，但在每个维度上都有一个由 1 隔开的均值。该数据集由 500 个高斯组成，其中 400 个用于训练，100 个用于测试。我们使用几种不同的函数在这个数据集上训练一个神经网络，以确定最终正确率是否存在系统差异。作为阴性对照，包括一个不变的损失函数。

假设：我们预计交叉熵损失函数作为分类任务的标准损失函数，表现最好，同时我们预计其他损失函数表现不佳。

```
1 experiment = Experiment9()
2 experiment.initialize()
3
4 loss_functions = ['cross_entropy',
5                  'mean_abs_error',
6                  'mean_squared_error',
7                  'mean_fourth_pow_error',
8                  'hinge_loss',
9                  'constant']
10
11 accuracies = experiment.run(loss_functions=loss_functions,
12                             iters=5)
13
14 plt.figure(figsize=[8,8]); plt.rc("font",family="sans-serif",size=18)
15 plt.plot(np.mean(accuracies,axis=0), '--o')
16
17 plt.title('Test Accuracy with Different Losses')
18 plt.xlabel(r'$\propto$ Epochs')
19 plt.ylabel('Test Accuracy')
20 plt.legend(loss_functions)
21
22 experiment.conclude()
```

知乎 @julyedu

运行实验所需的时间： 36.652 s



结论：除去阴性对照外，所有的损失都有类似的表现。损失函数是标签与逻辑之间的区别，提升到四次幂，其性能要比其他差一些。

讨论：损失函数的选择对最终结果没有实质影响，这也许不足为奇，因为这些损失函数非常相似。

51、初始化如何影响训练？

解析：

方法：我们生成两个 12 维高斯混合。高斯具有相同的协方差矩阵，但在每个维度都有一个由 1 隔开的均值。该数据集由 500 个高斯组成，其中 400 个用于训练，100 个用于测试。我们在这个神经网络中初始化权重值，看哪一个具有最好的训练性能。

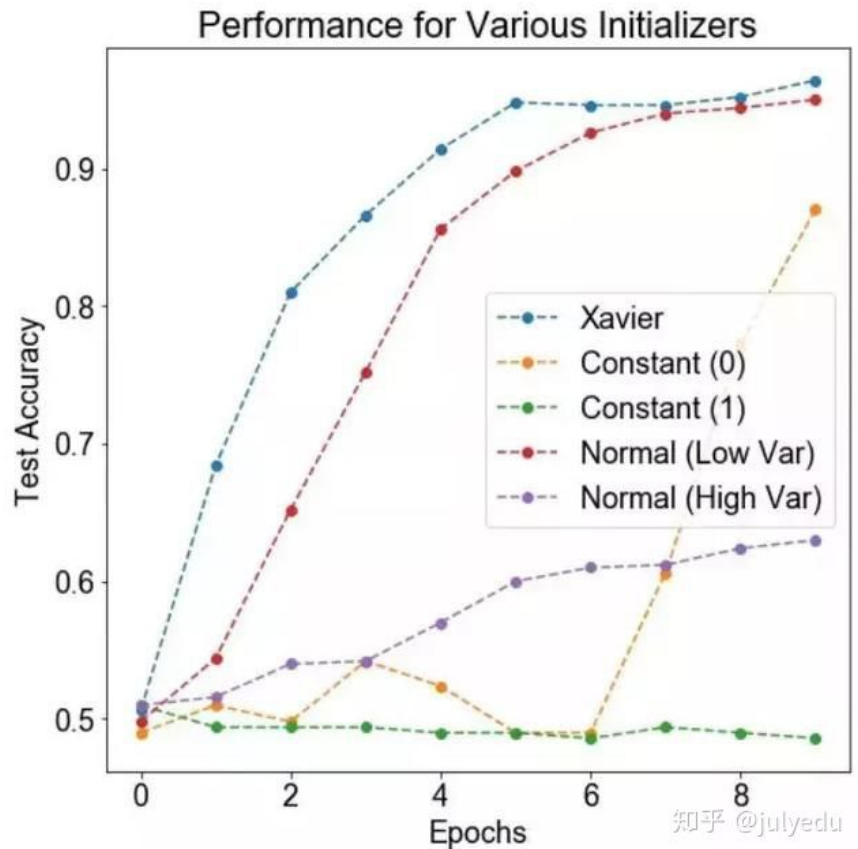
假设：我们期望 Xavier 损失具有最好的性能（它是 tensorflow 中使用的默认值），而其他方法性能不佳（尤其是不断的初始化）。

```

1 experiment = Experiment10()
2 experiment.initialize()
3
4 initializers = [tf.contrib.layers.xavier_initializer(),
5                 tf.zeros_initializer(),
6                 tf.ones_initializer(),
7                 tf.random_normal_initializer(0,0.01),
8                 tf.random_normal_initializer(0,1),
9                 ]
10
11 accuracies = experiment.run(initializers=initializers,
12                             iters=5)
13
14 plt.figure(figsize=[8,8]); plt.rc("font",family="sans-serif",size=18)
15 plt.plot(np.mean(accuracies,axis=0), '--o')
16
17 plt.title('Performance for Various Initializers')
18 plt.xlabel('Epochs')
19 plt.ylabel('Test Accuracy')
20 plt.legend(['Xavier', 'Constant (0)', 'Constant (1)', 'Normal (Low Var)', 'Normal (High Var)'])
21
22 experiment.conclude()

```

运行实验所需的时间： 34.137 s



结论：Xavier 和高斯（具有较低的方差）初始化会得到很好的训练。有趣的是，常数 0 的初始化最终导致训练，而其他初始化并不会。

讨论：Xavire 初始化提供了最好的性能，这并不奇怪。标准偏差小的高斯也适用（但不像 Xavire 那样好）。如果方差变得太大，那么训练速度就会变得较慢，这可能是因为神经网络的大部分输出都发生了爆炸。

52、不同层的权重是否以不同的速度收敛？

解析：

我们的第一个问题是，不同层的权重是否以不同的速度收敛。

方法：我们生成两个 12 维高斯混合。高斯具有相同的协方差矩阵，但每个维度上都有一个由 1 隔开的均值。该数据集由 500 个高斯组成，其中 400 个用于训练，100 个用于测试。我们在这个数据集上训练一个带有 3 个隐藏层（将导致 4 层权重，包括从输入到）第一层的权重）的神经网络，我们在训练过程中绘制每层 50 个权重值。我们通过绘制两个轮数之间的权重的差分来衡量收敛性。

假设：我们期望后一层的权重会更快地收敛，因为它们在整个网络中进行反向传播时，后期阶段的变化会被放大。

运行实验所需的时间： 3.924 s

结论：我们发现后一层的权重比前一层收敛得更快。

讨论：看上去第三层的权重是几乎单调地收敛到它们的最终值，而且这一过程非常快。至于前几层权重的收敛模式，比较复杂，似乎需要更长的时间才能解决。

53、正则化如何影响权重？

解析：

方法：我们生成两个 12 维高斯混合。高斯具有相同的协方差矩阵，但在每个维度上都有一个由 1 隔开的均值。该数据集由 500 个高斯组成，其中 400 个用于训练，100 个用于测试。我们在这个数据集上训练一个具有 2 个隐藏层的神经网络，并在整个训练过程中绘制 50 个权重值。

然后我们在损失函数中包含 L1 或 L2 正则项之后重复这一过程。我们研究这样是否会影响权重的收敛。我们还绘制了正确率的图像，并确定它在正则化的情况下是否发生了显著的变化。

假设：我们预计在正则化的情况下，权重的大小会降低。在 L1 正则化的情况下，我们可能会得到稀疏的权重。如果正则化强度很高，我们就会预计正确率下降，但是正确率实际上可能会随轻度正则化而上升。

运行实验所需的时间： 17.761 s

结论：我们注意到正则化确实降低了权重的大小，在强 L1 正则化的情况下导致了稀疏性。对正确率带来什么样的影响尚未清楚。

讨论：从我们所选的 50 个权重的样本可以清晰地看出，正则化对训练过程中习得的权重有着显著的影响。我们在 L1 正则化的情况下能够获得一定程度的稀疏性，虽然看起来有较大的正则化强度，这就导致正确率的折衷。而 L2 正则化不会导致稀疏性，它只有更小幅度的权重。同时，对正确率似乎没有什么有害的影响。

54、什么是fine-tuning？

解析：

在实践中，由于数据集不够大，很少有人从头开始训练网络。常见的做法是使用预训练的网络（例如在ImageNet上训练的分类1000类的网络）来重新fine-tuning（也叫微调），或者当做特征提取器。

以下是常见的两类迁移学习场景：

1 卷积网络当做特征提取器。使用在ImageNet上预训练的网络，去掉最后的全连接层，剩余部分当做特征提取器（例如AlexNet在最后分类器前，是4096维的特征向量）。这样提取的特征叫做CNN codes。得到这样的特征后，可以使用线性分类器（Linear SVM、Softmax等）来分类图像。

2 Fine-tuning卷积网络。替换掉网络的输入层（数据），使用新的数据继续训练。Fine-tune时可以选择fine-tune全部层或部分层。通常，前面的层提取的是图像的通用特征（generic features）（例如边缘检测，色

彩检测)，这些特征对许多任务都有用。后面的层提取的是与特定类别有关的特征，因此fine-tune时常常只需要Fine-tuning后面的层。

预训练模型

在ImageNet上训练一个网络，即使使用多GPU也要花费很长时间。因此人们通常共享他们预训练好的网络，这样有利于其他人再去使用。例如，Caffe有预训练好的网络地址Model Zoo。

何时以及如何Fine-tune

决定如何使用迁移学习的因素有很多，这是最重要的只有两个：新数据集的大小、以及新数据和原数据集的相似程度。有一点一定记住：网络前几层学到的是通用特征，后面几层学到的是与类别相关的特征。这里有使用的四个场景：

- 1、新数据集比较小且和原数据集相似。因为新数据集比较小，如果fine-tune可能会过拟合；又因为新旧数据集类似，我们期望他们高层特征类似，可以使用预训练网络当做特征提取器，用提取的特征训练线性分类器。
- 2、新数据集大且和原数据集相似。因为新数据集足够大，可以fine-tune整个网络。
- 3、新数据集小且和原数据集不相似。新数据集小，最好不要fine-tune，和原数据集不类似，最好也不使用高层特征。这时可是使用前面层的特征来训练SVM分类器。
- 4、新数据集大且和原数据集不相似。因为新数据集足够大，可以重新训练。但是实践中fine-tune预训练模型还是有益的。新数据集足够大，可以fine-tune整个网络。

实践建议

预训练模型的限制。使用预训练模型，受限于其网络架构。例如，你不能随意从预训练模型取出卷积层。但是因为参数共享，可以输入任意大小图像；卷积层和池化层对输入数据大小没有要求（只要步长stride fit），其输出大小和属于大小相关；全连接层对输入大小没有要求，输出大小固定。

学习率。与重新训练相比，fine-tune要使用更小的学习率。因为训练好的网络模型权重已经平滑，我们不希望太快扭曲（distort）它们（尤其是当随机初始化线性分类器来分类预训练模型提取的特征时）。

55、请简单解释下目标检测中的这个IOU评价函数（intersection-over-union）

解析：

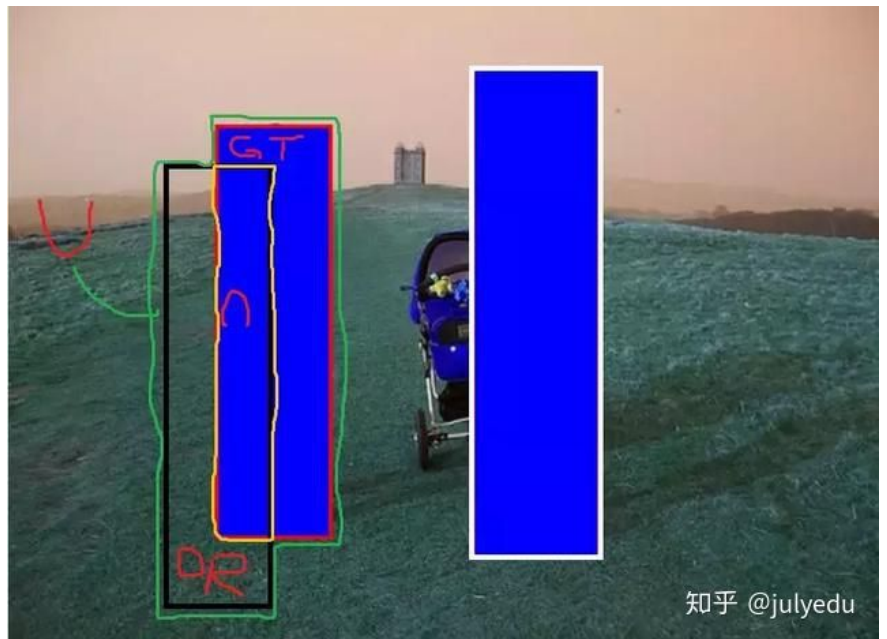
在目标检测的评价体系中，有一个参数叫做 IoU，简单来讲就是模型产生的目标窗口和原来标记窗口的交叠率。

具体我们可以简单的理解为：即检测结果DetectionResult与真实值Ground Truth的交集比上它们的并集，即为检测的准确率 IoU：

举个例子，下面是一张原图



然后我们对其做下目标检测，其DR = DetectionResult, GT = GroundTruth。



黄色边框框起来的是：

$DR \cap GT$

绿色框框起来的是：

$DR \cup GT$

不难看出，最理想的情况就是DR与GT完全重合，即IoU = 1。

56题

什么是边框回归Bounding-Box regression, 以及为什么要做、怎么做
解析:

这个问题可以牵扯出不少问题, 比如

为什么要边框回归?

什么是边框回归?

边框回归怎么做的?

边框回归为什么宽高, 坐标会设计这种形式?

为什么边框回归只能微调, 在离真实值Ground Truth近的时候才能生效?

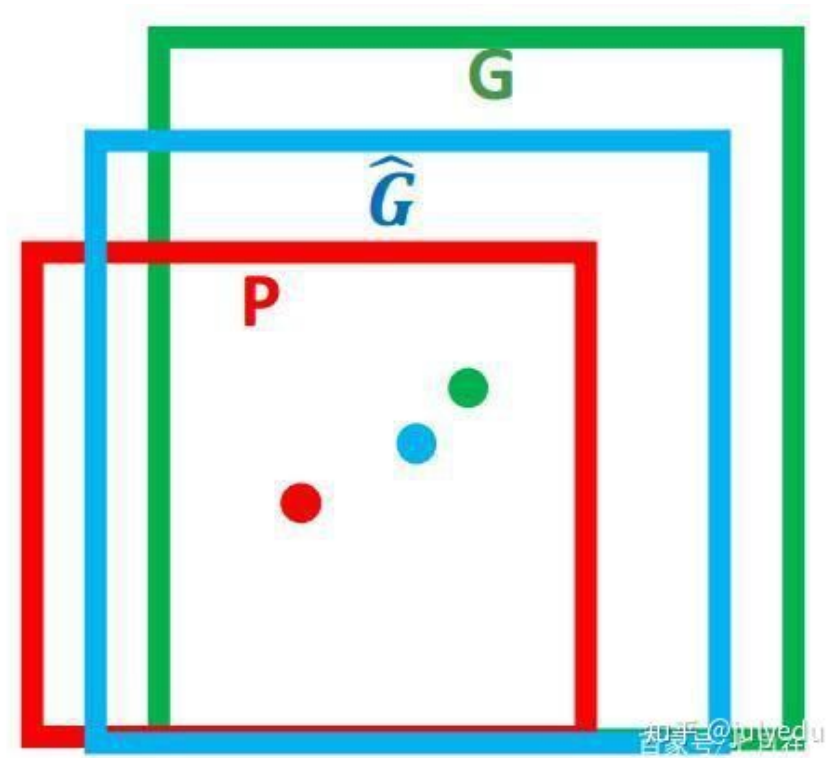
如图1所示, 绿色的框表示真实值Ground Truth, 红色的框为Selective Search提取的候选区域/框Region Proposal。那么即便红色的框被分类器识别为飞机, 但是由于红色的框定位不准($\text{IoU} < 0.5$), 这张图也相当于没有正确的检测出飞机。



如果我们能对红色的框进行微调fine-tuning, 使得经过微调后的窗口跟Ground Truth 更接近, 这样岂不是定位会更准确。而Bounding-box regression 就是用来微调这个窗口的。

边框回归是什么?

对于窗口一般使用四维向量 (x, y, w, h) 来表示, 分别表示窗口的中心点坐标和宽高。对于图2, 红色的框 P 代表原始的Proposal, 绿色的框 G 代表目标的 Ground Truth, 我们的目标是寻找一种关系使得输入原始的窗口 P 经过映射得到一个跟真实窗口 G 更接近的回归窗口 G^{\wedge} 。



所以，边框回归的目的即是：给定 (P_x, P_y, P_w, P_h) 寻找一种映射 f ，使得
 $f(P_x, P_y, P_w, P_h) = (G_x^{\wedge}, G_y^{\wedge}, G_w^{\wedge}, G_h^{\wedge})$ 并且
 $(G_x^{\wedge}, G_y^{\wedge}, G_w^{\wedge}, G_h^{\wedge}) \approx (G_x, G_y, G_w, G_h)$

边框回归怎么做的？

那么经过何种变换才能从图2中的窗口 P 变为窗口 G^{\wedge} 呢？ 比较简单的思路就是：平移+尺度放缩

先做平移 $(\Delta x, \Delta y)$ ， $\Delta x = P_w dx(P), \Delta y = P_h dy(P)$ 这是R-CNN论文的：

$$G^{\wedge}_x = P_w dx(P) + P_x, (1)$$

$$G^{\wedge}_y = P_h dy(P) + P_y, (2)$$

然后再做尺度缩放 (S_w, S_h) ， $S_w = \exp(dw(P)), S_h = \exp(dh(P))$ ，对应论文中：

$$G^{\wedge}_w = P_w \exp(dw(P)), (3)$$

$$G^{\wedge}_h = P_h \exp(dh(P)), (4)$$

观察(1)-(4)我们发现，边框回归学习就是 $dx(P), dy(P), dw(P), dh(P)$ 这四个变换。

下一步就是设计算法那得到这四个映射。

线性回归就是给定输入的特征向量 X ，学习一组参数 W ，使得经过线性回归后的值跟真实值 Y (Ground Truth) 非常接近。即 $Y \approx WX$ 。那么 Bounding-box 中我们的输入以及输出分别是什么呢？

Input:

RegionProposal $\rightarrow P = (P_x, P_y, P_w, P_h)$ 这个是什么？ 输入就是这四个数值吗？ 其实真正的输入是这个窗口对应的 CNN 特征，也就是 R-CNN 中的

Pool5 feature (特征向量)。(注:训练阶段输入还包括 Ground Truth, 也就是下边提到的 $t^*=(tx,ty,tw,th)$)

Output:

需要进行的平移变换和尺度缩放 $dx(P),dy(P),dw(P),dh(P)$, 或者说是 $\Delta x,\Delta y,Sw,Sh$ 。我们的最终输出不应该是 Ground Truth 吗? 是的, 但是有了这四个变换我们就可以直接得到 Ground Truth。

这里还有个问题, 根据(1)~(4)我们可以知道, P 经过 $dx(P),dy(P),dw(P),dh(P)$ 得到的并不是真实值 G , 而是预测值 G^{\wedge} 。的确, 这四个值应该是经过 Ground Truth 和 Proposal 计算得到的真正需要的平移量 (tx,ty) 和尺度缩放 (tw,th) 。

这也就是 R-CNN 中的(6)~(9):

$$tx=(Gx-Px)/Pw,(6)$$

$$ty=(Gy-Py)/Ph,(7)$$

$$tw=\log(Gw/Pw),(8)$$

$$th=\log(Gh/Ph),(9)$$

那么目标函数可以表示为 $d^*(P)=w^T\Phi_5(P)$, $\Phi_5(P)$ 是输入 Proposal 的特征向量, w^* 是要学习的参数 (*表示 x,y,w,h , 也就是每一个变换对应一个目标函数), $d^*(P)$ 是得到的预测值。

我们要让预测值跟真实值 $t^*=(tx,ty,tw,th)$ 差距最小, 得到损失函数为:

$$Loss=\sum_i N(t_i^*-w^T\phi_5(P_i))^2$$

函数优化目标为:

$$W^*=\operatorname{argmin}_W \sum_i N(t_i^*-w^T\phi_5(P_i))^2 + \lambda ||w^*||^2$$

利用梯度下降法或者最小二乘法就可以得到 w^* 。

57题

请阐述下Selective Search的主要思想

解析:

- 1 使用一种过分割手段, 将图像分割成小区域 (1k~2k 个)
- 2 查看现有小区域, 按照合并规则合并可能性最高的相邻两个区域。重复直到整张图像合并成一个区域位置
- 3 输出所有曾经存在过的区域, 所谓候选区域

其中合并规则如下: 优先合并以下四种区域:

- ①颜色 (颜色直方图) 相近的
- ②纹理 (梯度直方图) 相近的

③合并后总面积小的：保证合并操作的尺度较为均匀，避免一个大区域陆续“吃掉”其他小区域（例：设有区域a-b-④c-d-e-f-g-h。较好的合并方式是：ab-cd-ef-gh -> abcd-efgh -> abcdefgh。不好的合并方法是：ab-c-d-e-f-g-h -> abcd-e-f-g-h -> abcdef-gh -> abcdefgh）

合并后，总面积在其BBOX中所占比例大的：保证合并后形状规则。

上述四条规则只涉及区域的颜色直方图、梯度直方图、面积和位置。合并后的区域特征可以直接由子区域特征计算而来，速度较快。

58题

什么是非极大值抑制（NMS）？

解析：

R-CNN会从一张图片中找出n个可能是物体的矩形框，然后为每个矩形框为做类别分类概率：

就像上面的图片一样，定位一个车辆，最后算法就找出了一堆的方框，我们需要判别哪些矩形框是没用的。非极大值抑制的方法是：先假设有6个矩形框，根据分类器的类别分类概率做排序，假设从小到大属于车辆的概率分别为A、B、C、D、E、F。

(1)从最大概率矩形框F开始，分别判断A~E与F的重叠度IOU是否大于某个设定的阈值；

(2)假设B、D与F的重叠度超过阈值，那么就扔掉B、D；并标记第一个矩形框F，是我们保留下来的。

(3)从剩下的矩形框A、C、E中，选择概率最大的E，然后判断E与A、C的重叠度，重叠度大于一定的阈值，那么就扔掉；并标记E是我们保留下来的第二个矩形框。

就这样一直重复，找到所有被保留下来的矩形框。

非极大值抑制（NMS）顾名思义就是抑制不是极大值的元素，搜索局部的极大值。这个局部代表的是一个邻域，邻域有两个参数可变，一是邻域的维数，二是邻域的大小。这里不讨论通用的NMS算法，而是用于在目标检测中用于提取分数最高的窗口的。

例如在行人检测中，滑动窗口经提取特征，经分类器分类识别后，每个窗口都会得到一个分数。但是滑动窗口会导致很多窗口与其他窗口存在包含或者大部分交叉的情况。这时就需要用到NMS来选取那些邻域里分数最高（是行人的概率最大），并且抑制那些分数低的窗口。

59题

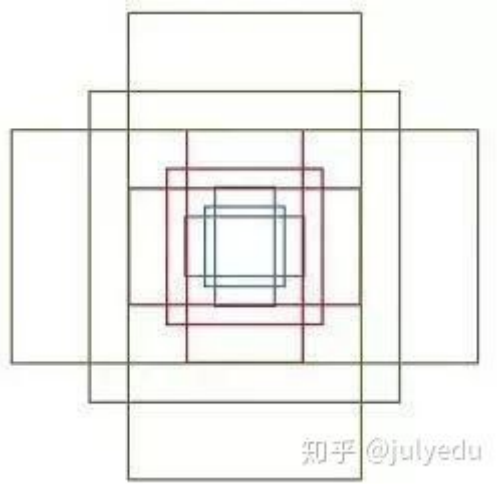
什么是深度学习中的anchor？

解析：

当我们使用一个3*3的卷积核，在最后一个feature map上滑动，当滑动到特征图的某一个位置时，以当前滑动窗口中心为中心映射回原图的一个

区域(注意 feature map 上的一个点是可以映射到原图的一个区域的，相当于感受野起的作用)，以原图上这个区域的中心对应一个尺度和长宽比，就是一个anchor了。

fast rcnn 使用3种尺度和3种长宽比（1:1; 1:2; 2:1），则在每一个滑动位置就有 $3*3 = 9$ 个anchor。



60题

CNN的特点以及优势

解析：

CNN使用范围是具有局部空间相关性的数据，比如图像，自然语言，语音

局部连接：可以提取局部特征。

权值共享：减少参数数量，因此降低训练难度（空间、时间消耗都少了）。可以完全共享，也可以局部共享（比如对人脸，眼睛鼻子嘴由于位置和样式相对固定，可以用和脸部不一样的卷积核）

降维：通过池化或卷积stride实现。

多层次结构：将低层次的局部特征组合成为较高层次的特征。不同层级的特征可以对应不同任务。

61题

深度学习中有什么加快收敛/降低训练难度的方法？

解析：

瓶颈结构

残差

学习率、步长、动量

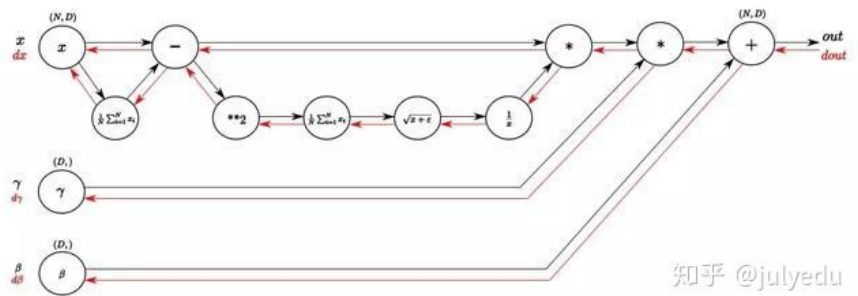
优化方法

预训练

62题

请简单说下计算流图的前向和反向传播

解析：



知乎 @julyedu

63题

请写出链式法则并证明

解析：

链式法则或链锁定则（英语：chain rule），是求复合函数导数的一个法则。设 f 和 g 为两个关于 x 的可导函数，则复合函数

$$(f \circ g)(x)$$

的导数

$$(f \circ g)'(x)$$

为

$$(f \circ g)'(x) = f'(g(x))g'(x)$$

以下是一个简单的例子

求函数 $f(x) = (x^2 + 1)^3$ 的导数。设 $g(x) = x^2 + 1$, $h(g) = g^3 \Rightarrow h(g(x)) = g(x)^3$.

$$f(x) = h(g(x)) \quad || \vdash |f'(x)| = h'(g(x))g'(x) | = 3(g(x))^2(2x) | = 3(x^2 + 1)^2(2x) \quad | \vdash | = 6x(x^2 + 1)^2. \quad ||$$

求函数 $\arctan \sin x$ 的导数。

$$\frac{d}{dx} \arctan x = \frac{1}{1+x^2}$$

$$\frac{d}{dx} \arctan f(x) = \frac{f'(x)}{1+f^2(x)}$$

$$\frac{d}{dx} \arctan \sin x = \frac{\cos x}{1+\sin^2 x}$$

知乎 @julyedu

以下的简单的一个证明

设 f 和 g 为函数， x 为常数，使得 f 在 $g(x)$ 可导，且 g 在 x 可导。根据可导的定义，

$$g(x + \delta) - g(x) = \delta g'(x) + \epsilon(\delta)\delta, \text{ 其中当 } \delta \rightarrow 0 \text{ 时, } \epsilon(\delta) \rightarrow 0.$$

同理，

$$f(g(x) + \alpha) - f(g(x)) = \alpha f'(g(x)) + \eta(\alpha)\alpha, \text{ 其中当 } \alpha \rightarrow 0 \text{ 时, } \eta(\alpha) \rightarrow 0.$$

现在

$$\begin{aligned} f(g(x + \delta)) - f(g(x)) &= f(g(x) + \delta g'(x) + \epsilon(\delta)\delta) - f(g(x)) \\ &= \alpha_\delta f'(g(x)) + \eta(\alpha_\delta)\alpha_\delta \end{aligned}$$

其中 $\alpha_\delta = \delta g'(x) + \epsilon(\delta)\delta$. 注意到当 $\delta \rightarrow 0$ 时, $\frac{\alpha_\delta}{\delta} \rightarrow g'(x)$ 及 $\alpha_\delta \rightarrow 0$, 因此 $\eta(\alpha_\delta) \rightarrow 0$. 因此

$$\frac{f(g(x + \delta)) - f(g(x))}{\delta} \rightarrow g'(x)f'(g(x)).$$

知乎 @julyedu

64题

请写出Batch Normalization的计算方法及其应用

解析：

机器学习流程简介

1) 一次性设置 (One time setup)

- 激活函数 (Activation functions)

- 数据预处理 (Data Preprocessing)

- 权重初始化 (Weight Initialization)

- 正则化 (Regularization: 避免过拟合的一种技术)

- 梯度检查 (Gradient checking)

2) 动态训练 (Training dynamics)

- 跟踪学习过程 (Babysitting the learning process)

- 参数更新 (Parameter updates)

- 超级参数优化 (Hyperparameter optimization)

- 批量归一化 (Batch Normalization简称BN, 其中, Normalization是数据标准化或归一化、规范化, Batch可以理解为批量, 加起来就是批量标准化。解决在训练过程中中间层数据分布发生改变的问题, 以防止梯度消失或爆炸、加快训练速度)

3) 评估 (Evaluation)

- 模型组合 (Model ensembles)

(训练多个独立的模型, 测试时, 取这些模型结果的平均值)

为什么输入数据需要归一化 (Normalized Data), 或者说, 归一化后有什么好处呢?

原因在于神经网络学习过程本质就是为了学习数据分布, 一旦训练数据与测试数据的分布不同, 那么网络的泛化能力也大大降低, 所以需要使用输入数据归一化方法, 使训练数据与测试数据的分布相同。

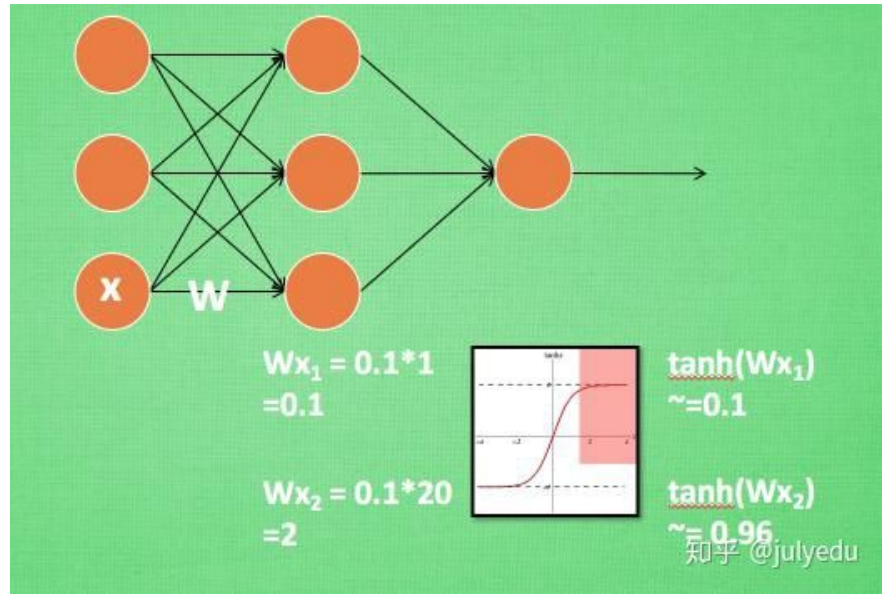
另外一方面, 加之神经网络训练时一旦网络某一层的输入数据的分布发生改变, 那么这一层网络就需要去适应学习这个新的数据分布, 所以如果训练过程中, 训练数据的分布一直在发生变化, 那么将会影响网络的训练速度。

为了让训练深度网络简单高效, 研究者提出了随机梯度下降法 (SGD), 但是它有个毛病, 就是需要我们人为的去选择参数, 比如学习率、参数初始化、权重衰减系数、Drop out比例等。这些参数的选择对训练结果至关重要, 以至于我们很多时间都浪费在这些的调参上。

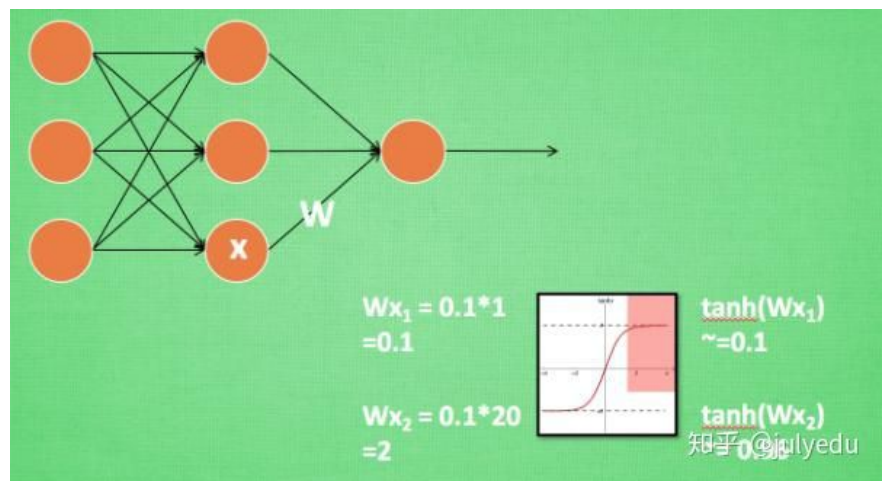
举个例子，比如某个神经元 $x = 1$ ，某个 Weights 的初始值为 0.1，这样后一层神经元计算结果就是 $Wx = 0.1 * 1 = 0.1$ ；

如果 $x = 20$ ，这样 $Wx = 0.1 * 20 = 2$ 。现在还不能看出什么问题，但是，当我们加上一层激励函数，激活这个 Wx 值的时候，问题就来了。

如果使用像 \tanh 的激励函数， Wx 的激活值就变成了 ~ 0.1 和 ~ 1 ，接近于 1 的部分已经处在了激励函数的饱和阶段，也就是如果 x 无论再怎么扩大， \tanh 激励函数输出值也还是接近 1。



换句话说，神经网络在初始阶段已经不对那些比较大的 x 特征范围敏感了。这样很糟糕，想象我轻轻拍自己的感觉和重重打自己的感觉居然没什么差别，这就证明我的感官系统失效了。当然我们是可以利用之前提到的对数据做 normalization 预处理，使得输入的 x 变化范围不会太大，让输入值经过激励函数的敏感部分。但刚刚这个不敏感问题不仅仅发生在神经网络的输入层，而且在隐藏层中也经常会发生。



既然 x 换到了隐藏层当中，我们能不能对隐藏层的输入结果进行像之前那样的 normalization 处理呢？答案是可以的，因为大牛们发明了一种技术，叫做 batch normalization，正是处理这种情况。

Batch Normalization 由 Google 提出在这篇论文中《Batch Normalization Accelerating Deep Network Training by Reducing

Internal Covariate Shift》提出。

与激活函数层、卷积层、全连接层、池化层一样，BN(Batch Normalization)也属于网络的一层。

BN的本质原理：在网络的每一层输入的时候，又插入了一个归一化层，也就是先做一个归一化处理（归一化至：均值0、方差为1），然后再进入网络的下一层。不过归一化层可不像我们想象的那么简单，它是一个可学习、有参数（ γ 、 β ）的网络层。

归一化公式：

$$\text{一层有d维输入: } \mathbf{X} = (x^{(1)} \dots x^{(d)})$$

$$\text{归一化每一维: } \hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

以下是Normalization过程（引用Google论文中的解释）：

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots x_m\}$;	
Parameters to be learned: γ, β	
Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$	
$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i$	// mini-batch mean
$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2$	// mini-batch variance
$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$	// normalize
$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i)$	// scale and shift

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch. 知乎 @julyedu

输入：输入数据 $x_1 \dots x_m$ （这些数据是准备进入激活函数的数据）

计算过程中可以看到，

- 1.求数据均值；
- 2.求数据方差；
- 3.数据进行标准化（个人认为称作正态化也可以）
- 4.训练参数 γ, β
- 5.输出 y 通过 γ 与 β 的线性变换得到新的值

How to BN?

怎样学BN的参数就是经典的chain rule。

在正向传播的时候，通过可学习的 γ 与 β 参数求出新的分布值

在反向传播的时候，通过链式求导方式，修正 γ 与 β 以及相关权值

$$\begin{aligned}\frac{\partial \ell}{\partial \hat{x}_i} &= \frac{\partial \ell}{\partial y_i} \cdot \gamma \\ \frac{\partial \ell}{\partial \sigma_B^2} &= \sum_{i=1}^m \frac{\partial \ell}{\partial \hat{x}_i} \cdot (x_i - \mu_B) \cdot \frac{-1}{2} (\sigma_B^2 + \epsilon)^{-3/2} \\ \frac{\partial \ell}{\partial \mu_B} &= \left(\sum_{i=1}^m \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{-1}{\sqrt{\sigma_B^2 + \epsilon}} \right) + \frac{\partial \ell}{\partial \sigma_B^2} \cdot \frac{\sum_{i=1}^m -2(x_i - \mu_B)}{m} \\ \frac{\partial \ell}{\partial x_i} &= \frac{\partial \ell}{\partial \hat{x}_i} \cdot \frac{1}{\sqrt{\sigma_B^2 + \epsilon}} + \frac{\partial \ell}{\partial \sigma_B^2} \cdot \frac{2(x_i - \mu_B)}{m} + \frac{\partial \ell}{\partial \mu_B} \cdot \frac{1}{m} \\ \frac{\partial \ell}{\partial \gamma} &= \sum_{i=1}^m \frac{\partial \ell}{\partial y_i} \cdot \hat{x}_i \\ \frac{\partial \ell}{\partial \beta} &= \sum_{i=1}^m \frac{\partial \ell}{\partial y_i}\end{aligned}$$

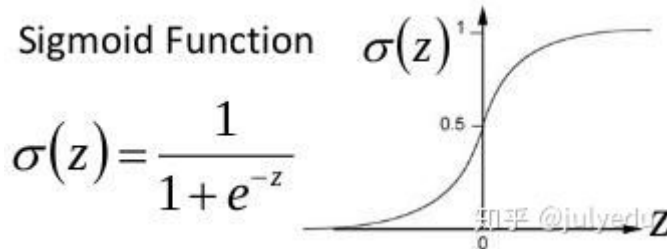
知乎 @julyedu

Why is BN?

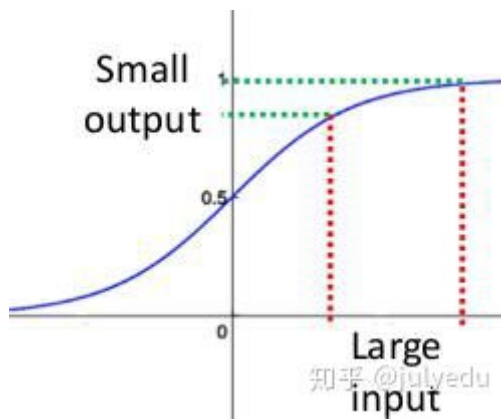
因为BN保证每一层的输入分布稳定，这一点本身可以使得训练加速，而且另一方面它也可以帮助减少梯度消失和梯度爆炸的现象。

梯度消失

关于梯度消失，以sigmoid函数为例子，sigmoid函数使得输出在[0,1]之间。

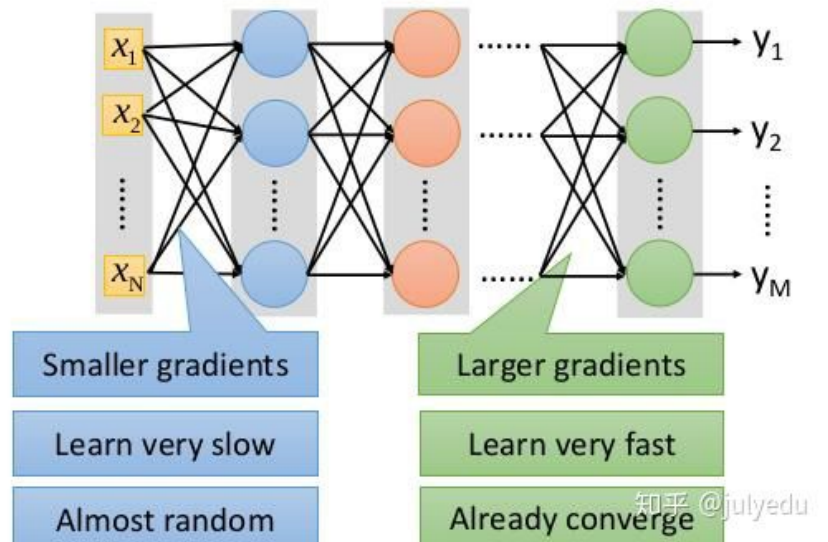


事实上x到了一定大小，经过sigmoid函数的输出范围就很小了，参考下图



如果输入很大，其对应的斜率就很小，我们知道，其斜率（梯度）在反向传播中是权值学习速率。所以就会出现如下的问题

Vanishing Gradient Problem



在深度网络中，如果网络的激活输出很大，其梯度就很小，学习速率就很慢。假设每层学习梯度都小于最大值0.25，网络有 n 层，因为链式求导的原因，第一层的梯度小于0.25的 n 次方，所以学习速率就慢，对于最后一层只需对自身求导1次，梯度就大，学习速率就快。

这会造成影响是在一个很大的深度网络中，浅层基本不学习，权值变化小，后面几层一直在学习，结果就是，后面几层基本可以表示整个网络，失去了深度的意义。

梯度爆炸

关于梯度爆炸，根据链式求导法，第一层偏移量的梯度=激活层斜率 $1 \times$ 权值 $1 \times$ 激活层斜率 $2 \times \dots \times$ 激活层斜率 $(n-1) \times$ 权值 $(n-1) \times$ 激活层斜率 n 。假如激活层斜率均为最大值0.25，所有层的权值为100，这样梯度就会指数增加。

65题

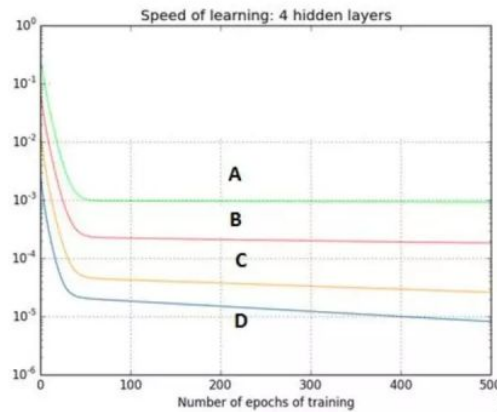
神经网络中会用到批量梯度下降（BGD）吗？为什么用随机梯度下降（SGD）？

解析：

- 1) 一般不用BGD
- 2) a. BGD每次需要用到全量数据，计算量太大
- b. 引入随机因素，即便陷入局部极小，梯度也可能不为0，这样就有机会跳出局部极小继续搜索（可以作为跳出局部极小的一种方式，但也可能跳出全局最小。还有解决局部极小的方式：多组参数初始化、使用模拟退火技术）

66、下图是一个利用sigmoid函数作为激活函数的含四个隐藏层的神经网络训练的梯度下

降图。这个神经网络遇到了梯度消失的问题。下面哪个叙述是正确的？



- A、第一隐藏层对应D，第二隐藏层对应C，第三隐藏层对应B，第四隐藏层对应A
- B、第一隐藏层对应A，第二隐藏层对应C，第三隐藏层对应B，第四隐藏层对应D
- C、第一隐藏层对应A，第二隐藏层对应B，第三隐藏层对应C，第四隐藏层对应D
- D、第一隐藏层对应B，第二隐藏层对应D，第三隐藏层对应C，第四隐藏层对应A

正确答案是：A

解析：

由于反向传播算法进入起始层，学习能力降低，这就是梯度消失。换言之，梯度消失是梯度在前向传播中逐渐减为0，按照图标题所说，四条曲线是4个隐藏层的学习曲线，那么第一层梯度最高(损失函数曲线下降明显)，最后一层梯度几乎为零(损失函数曲线变成平直线)。所以D是第一层，A是最后一层。

67、考虑某个具体问题，你可能只有少量数据来解决这个问题。不过幸运的是你有一个类似问题已经预先训练好的神经网络。可以用下面哪种方法来利用这个预先训练好的网络？

- A、把除了最后一层外所有的层都冻结，重新训练最后一层
- B、对新数据重新训练整个模型
- C、只对最后几层进行调参(fine tune)
- D、对每一层模型进行评估，选择其中的少数来用

正确答案是：C

解析：

如果有个预先训练好的神经网络，就相当于网络各参数有个很靠谱的先验代替随机初始化。若新的少量数据来自于先前训练数据(或者先前训练数据量很好地描述了数据分布，而新数据采样自完全相同的分布)，则冻结前面所有层而重新训练最后一层即可；但一般情况下，新数据分布跟先前训练集分布有所偏差，所以先验网络不足以完全拟合新数据时，可以冻结大部分前层网络，只对最后几层进行训练调参(这也称之为fine tune)。

68、在选择神经网络的深度时，下面哪些参数需要考虑？

- 1 神经网络的类型(如MLP,CNN)
- 2 输入数据
- 3 计算能力(硬件和软件能力决定)
- 4 学习速率
- 5 映射的输出函数

- A、 1,2,4,5
- B、 2,3,4,5
- C、 都需要考虑
- D、 1,3,4,5

正确答案是：C

解析：

所有上述因素对于选择神经网络模型的深度都是重要的。特征抽取所需分层越多，输入数据维度越高，映射的输出函数非线性越复杂，所需深度就越深。另外为了达到最佳效果，增加深度所带来的参数量增加，也需要考虑硬件计算能力和学习速率以设计合理的训练时间。

69、当数据过大以至于无法在RAM中同时处理时，哪种梯度下降方法更加有效？

- A、 随机梯度下降法(Stochastic Gradient Descent)
- B、 不知道
- C、 整批梯度下降法(Full Batch Gradient Descent)
- D、 都不是

正确答案是：A

解析：

梯度下降法分随机梯度下降(每次用一个样本)、小批量梯度下降法(每次用一小批样本算出总损失，因而反向传播的梯度折中)、全批量梯度下降法则

一次性使用全部样本。这三个方法, 对于全体样本的损失函数曲面来说, 梯度指向一个比一个准确. 但是在工程应用中, 受到内存/磁盘IO的吞吐性能制约, 若要最小化梯度下降的实际运算时间, 需要在梯度方向准确性和数据传输性能之间取得最好的平衡. 所以, 对于数据过大以至于无法在RAM中同时处理时, RAM每次只能装一个样本, 那么只能选随机梯度下降法。

70、当在卷积神经网络中加入池化层(pooling layer)时, 变换的不变性会被保留, 是吗?

- A、不知道
- B、看情况
- C、是
- D、否

正确答案是: C

解析:

池化算法比如取最大值/取平均值等, 都是输入数据旋转后结果不变, 所以多层叠加后也有这种不变性。

71、深度学习是当前很热门的机器学习算法, 在深度学习中, 涉及到大量的矩阵相乘, 现在需要计算三个稠密矩阵 A, B, C 的乘积 ABC , 假设三个矩阵的尺寸分别为 $m \times n, n \times p, p \times q$, 且 $m < n < p < q$, 以下计算顺序效率最高的是 ()

- A、 $(AB)C$
- B、 $AC(B)$
- C、 $A(BC)$
- D、所以效率都相同

正确答案是: A

解析:

首先, 根据简单的矩阵知识, 因为 $A*B$, A 的列数必须和 B 的行数相等。因此, 可以排除 B 选项,

然后, 再看 A、C 选项。在 A 选项中, $m \times n$ 的矩阵 A 和 $n \times p$ 的矩阵 B 的乘积, 得到 $m \times p$ 的矩阵 $A*B$, 而 $A*B$ 的每个元素需要 n 次乘法和 $n-1$ 次加法, 忽略加法, 共需要 $m \times n \times p$ 次乘法运算。同样情况分析 $A*B$ 之后再乘以 C 时的情况, 共需要 $m \times p \times q$ 次乘法运算。因此, A 选项 $(AB)C$

需要的乘法次数是 $m \times n \times p + m \times p \times q$ 。同理分析，C 选项 A (BC) 需要的乘法次数是 $n \times p \times q + m \times n \times q$ 。

由于 $m \times n \times p$

72、输入图片大小为 200×200 ，依次经过一层卷积 (kernel size 5×5 , padding 1, stride 2)，pooling (kernel size 3×3 , padding 0, stride 1)，又一层卷积 (kernel size 3×3 , padding 1, stride 1) 之后，输出特征图大小为

- A、 95
- B、 96
- C、 97
- D、 98

正确答案是：C

解析：

首先我们应该知道卷积或者池化后大小的计算公式，其中，padding指的是向外扩展的边缘大小，而stride则是步长，即每次移动的长度。

这样一来就容易多了，首先长宽一般大，所以我们只需要计算一个维度即可，这样，经过第一次卷积后的大小为：本题 $(200 - 5 + 2 \times 1) / 2 + 1$ 为 99.5，取99

经过第一次池化后的大小为： $(99 - 3) / 1 + 1$ 为97

经过第二次卷积后的大小为： $(97 - 3 + 2 \times 1) / 1 + 1$ 为97

73、基于二次准则函数的H-K算法较之于感知器算法的优点是()？

- A、 计算量小
- B、 可以判别问题是否线性可分
- C、 其解完全适用于非线性可分的情况

正确答案是：B

解析：

HK算法思想很朴实,就是在最小均方误差准则下求得权矢量.

他相对于感知器算法的优点在于,他适用于线性可分和非线性可分得情况,对于线性可分的情况,给出最优权矢量,对于非线性可分得情况,能够判别出来,以退出迭代过程。

来源：@刘炫320，链接：

<http://blog.csdn.net/column/details/16442.html>

74、在一个神经网络中，知道每一个神经元的权重和偏差是最重要的一步。如果知道了神经元准确的权重和偏差，便可以近似任何函数，但怎么获知每个神经元的权重和偏移呢？

- A、搜索每个可能的权重和偏差组合，直到得到最佳值
- B、赋予一个初始值，然后检查跟最佳值的差值，不断迭代调整权重
- C、随机赋值，听天由命
- D、以上都不正确的

正确答案是：B

解析：

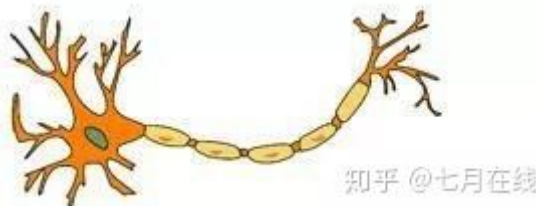
答案：（B）

选项B是对梯度下降的描述。

75、神经网络模型（Neural Network）因受人类大脑的启发而得名



神经网络由许多神经元（Neuron）组成，每个神经元接受一个输入，对输入进行处理后给出一个输出，如下图所示。请问下列关于神经元的描述中，哪一项是正确的？



- A、每个神经元可以有一个输入和一个输出
- B、每个神经元可以有多个输入和一个输出

- C、 每个神经元可以有一个输入和多个输出
- D、 每个神经元可以有多个输入和多个输出
- E、 上述都正确

正确答案是： E

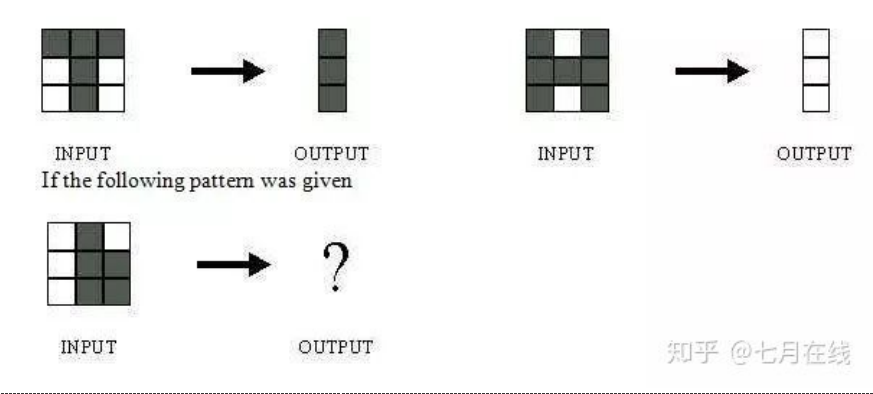
解析：

答案： （E）

每个神经元可以有一个或多个输入， 和一个或多个输出。

76题

下图所示的网络用于训练识别字符H和T， 如下所示



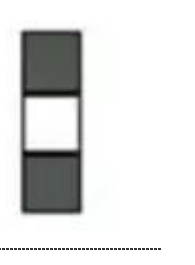
A、



B、



C、



D、可能是A或B，取决于神经网络的权重设置

正确答案是：D

解析：

不知道神经网络的权重和偏差是什么，则无法判定它将会给出什么样的输出。

77题

如果我们用了一个过大的学习速率会发生什么？

A、神经网络会收敛

B、不好说

C、都不对

D、神经网络不会收敛

正确答案是：D

解析

学习率过大，会使得迭代时，越过最低点。

78题

在一个神经网络中，下面哪种方法可以用来处理过拟合？

A、Dropout

B、分批归一化(Batch Normalization)

C、正则化(regularization)

D、都可以

正确答案是：D

解析：

都可以。对于选项C，分批归一化处理过拟合的原理，是因为同一个数据在不同批中被归一化后的值会有差别，相当于做了data augmentatio。

79题

批规范化(Batch Normalization)的好处都有啥？

A、让每一层的输入的范围都大致固定

B、它将权重的归一化平均值和标准差

C、它是一种非常有效的反向传播(BP)方法

D、这些均不是

正确答案是：A

80题

下列哪个神经网络结构会发生权重共享？

- A、卷积神经网络
- B、循环神经网络
- C、全连接神经网络
- D、选项A和B

正确答案是：D

81、下列哪个函数不可以做激活函数？

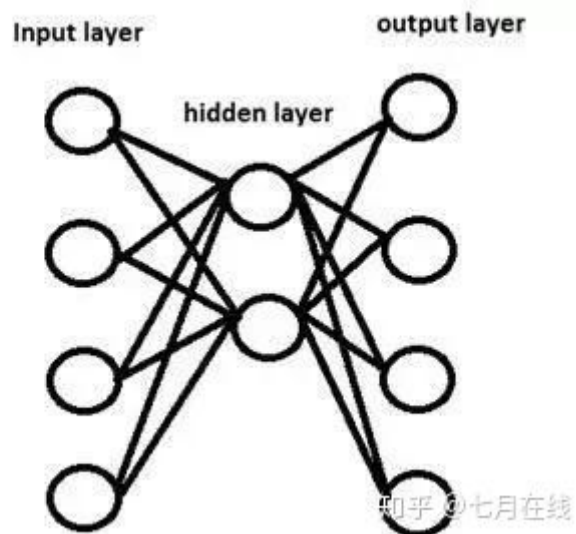
- A、 $y = \tanh(x)$
- B、 $y = \sin(x)$
- C、 $y = \max(x, 0)$
- D、 $y = 2x$

正确答案是：D

解析：

线性函数不能作为激活函数。

82、假设我们有一个如下图所示的隐藏层。隐藏层在这个网络中起到了一定的降维作用。假如现在我们用另一种维度下降的方法，比如说主成分分析法(PCA)来替代这个隐藏层。



那么，这两者的输出效果是一样的吗？

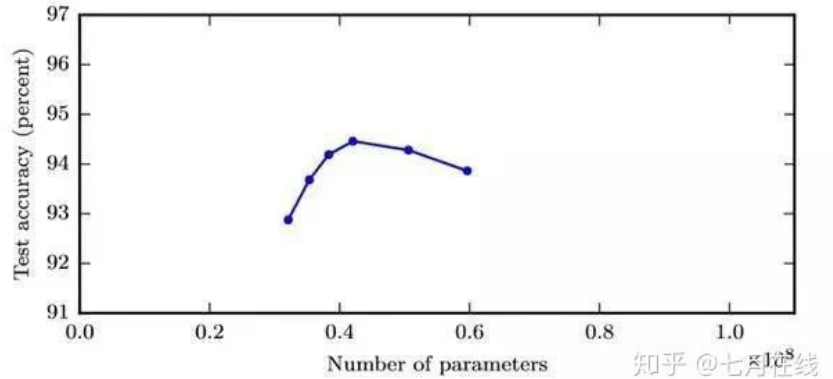
- A、是
- B、否

正确答案是：B

解析：

PCA 提取的是数据分布方差比较大的方向，隐藏层可以提取有预测能力的特征

83、下图显示了训练过的3层卷积神经网络准确度，与参数数量(特征核的数量)的关系。



从图中趋势可见，如果增加神经网络的宽度，精确度会增加到一个特定阈值后，便开始降低。造成这一现象的可能原因是什么？

- A、即使增加卷积核的数量，只有少部分的核会被用作预测
- B、当卷积核数量增加时，神经网络的预测能力（Power）会降低
- C、当卷积核数量增加时，导致过拟合
- D、以上都不正确

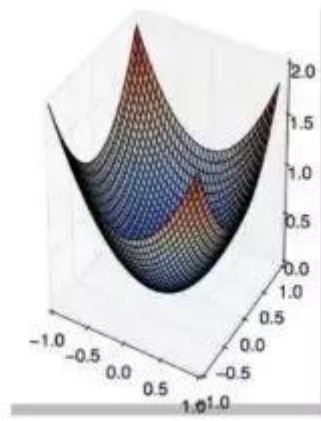
正确答案是：C

解析：

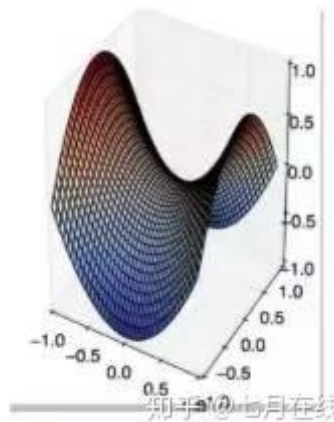
网络规模过大时，就可能学到数据中的噪声，导致过拟合

84、在下面哪种情况下，一阶梯度下降不一定正确工作（可能会卡住）？

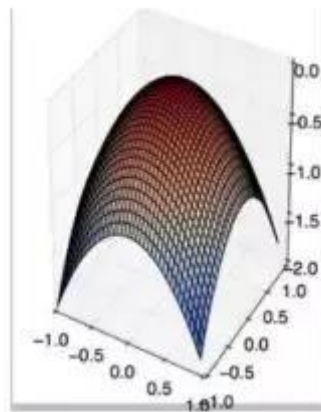
- A、



B、



C、



正确答案是：B

解析：

这是鞍点

(Saddle <https://www.analyticsvidhya.com/blog/2017/01/must-know-questions-deep-learning/>).

85、假设你需要调整超参数来最小化代价函数 (cost function) ， 会使用下列哪项技术？

- A、穷举搜索
- B、随机搜索
- C、Bayesian优化
- D、都可以

正确答案是：D

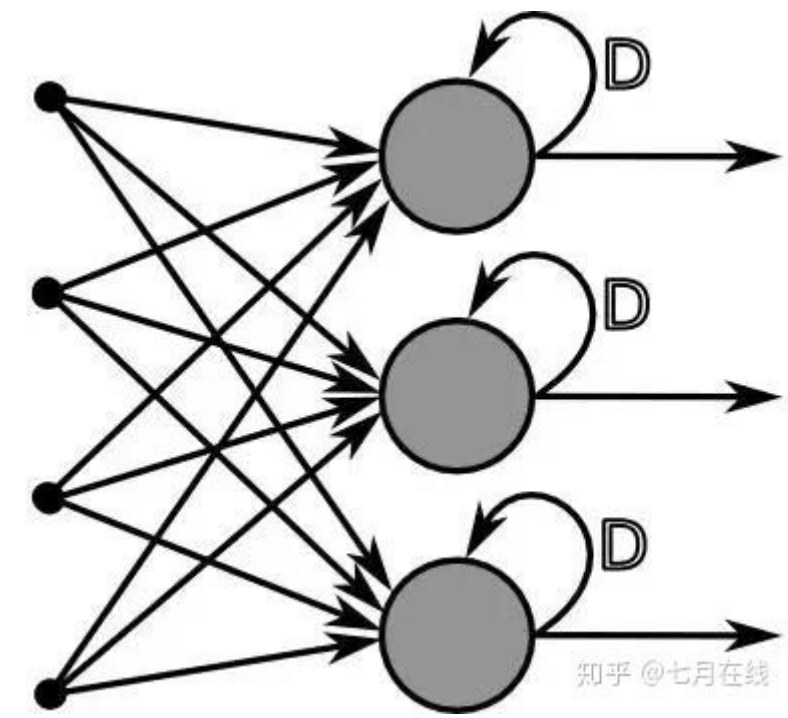
86、在感知机中 (Perceptron) 的任务顺序是什么？

- 1、随机初始化感知机的权重
- 2、去到数据集的下一批 (batch)
- 3、如果预测值和输出不一致，则调整权重
- 4、对一个输入样本，计算输出值

- A、 1, 2, 3, 4
- B、 4, 3, 2, 1
- C、 3, 1, 2, 4
- D、 1, 4, 3, 2

正确答案是：D

87、构建一个神经网络，将前一层的输出和它自身作为输入。



下列哪一种架构有反馈连接？

- A、循环神经网络

- B、卷积神经网络
- C、限制玻尔兹曼机
- D、都不是

正确答案是：A

88、如果增加多层感知机（Multilayer Perceptron）的隐藏层层数，分类误差便会减小。这种陈述正确还是错误？

- A、正确
- B、错误

正确答案是：B

解析：

并不总是正确。层数增加可能导致过拟合，从而可能引起错误增加。

89、下列哪项关于模型能力（model capacity）的描述是正确的？（指神经网络模型能拟合复杂函数的能力）

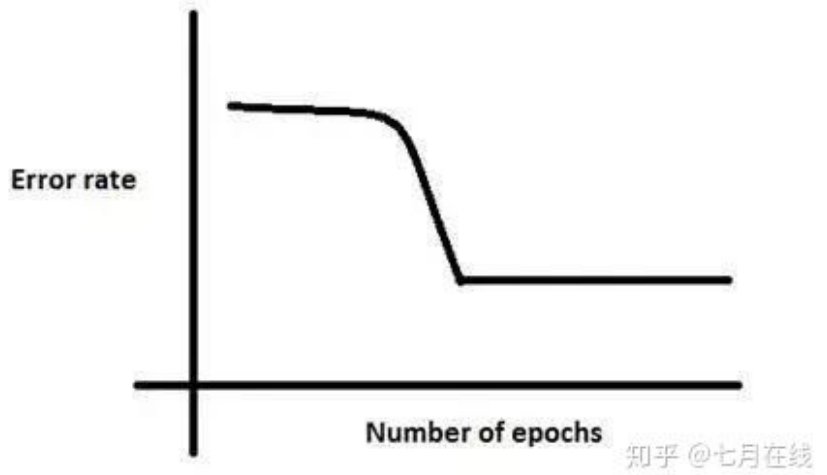
- A、隐藏层层数增加，模型能力增加
- B、Dropout的比例增加，模型能力增加
- C、学习率增加，模型能力增加
- D、都不正确

正确答案是：A

解析：

A是对的，其它选项不确定

90、在训练神经网络时，损失函数(loss)在最初的几个epochs时没有下降，可能的原因是？



- A、学习率(learning rate)太低
- B、正则参数太高
- C、陷入局部最小值
- D、以上都有可能

正确答案是：D

91、深度学习与机器学习算法之间的区别在于，后者过程中无需进行特征提取工作，也就是说，我们建议在深度学习过程之前要首先完成特征提取的工作。这种说法是：

- A、正确的
- B、错误的

正确答案是：B

解析：

正好相反，深度学习可以自行完成特征提取过程而机器学习需要人工来处理特征内容。

92、下列哪一项属于特征学习算法 (representation learning algorithm) ？

- A、K近邻算法
- B、随机森林
- C、神经网络
- D、都不属于

正确答案是：C

解析：

神经网络会将数据转化为更适合解决目标问题的形式，我们把这种过程叫做特征学习。

93、下列哪些项所描述的相关技术是错误的？

- A、AdaGrad使用的是一阶差分(first order differentiation)
- B、L-BFGS使用的是二阶差分(second order differentiation)
- C、AdaGrad使用的是二阶差分

正确答案是： C

94、提升卷积核(convolutional kernel)的大小会显著提升卷积神经网络的性能，这种说法是

- A、正确的
- B、错误的

正确答案是： B

解析：

卷积核的大小是一个超参数(hyperparameter)，也就意味着改变它既有可能提高亦有可能降低模型的表现。

95、阅读以下文字：

假设我们拥有一个已完成训练的、用来解决车辆检测问题的深度神经网络模型，训练所用的数据集由汽车和卡车的照片构成，而训练目标是检测出每种车辆的名称（车辆共有10种类型）。现在想要使用这个模型来解决另外一个问题，问题数据集中仅包含一种车（福特野马）而目标变为定位车辆在照片中的位置。

- A、除去神经网络中的最后一层，冻结所有层然后重新训练
- B、对神经网络中的最后几层进行微调，同时将最后一层（分类层）更改为回归层
- C、使用新的数据集重新训练模型
- D、所有答案均不对

正确答案是： B

96、假设你有5个大小为7x7、边界值为0的卷积核，同时卷积神经网络第一层的深度为1。此时如果你向这一层传入一个维度为

224x224x3的数据，那么神经网络下一层所接收到的数据维度是多少？

- A、218x218x5
- B、217x217x8
- C、217x217x3
- D、220x220x5

正确答案是：A

97、假设我们有一个使用ReLU激活函数 (ReLU activation function) 的神经网络，假如我们把ReLU激活替换为线性激活，那么这个神经网络能够模拟出同或函数 (XNOR function) 吗？

- A、可以
- B、不好说
- C、不一定
- D、不能

正确答案是：D

解析：

使用ReLU激活函数的神经网络是能够模拟出同或函数的。

但如果ReLU激活函数被线性函数所替代之后，神经网络将失去模拟非线性函数的能力。

98、考虑以下问题：

假设我们有一个5层的神经网络，这个神经网络在使用一个4GB显存显卡时需要花费3个小时来完成训练。而在测试过程中，单个数据需要花费2秒的时间。如果我们现在把架构变换一下，当评分是0.2和0.3时，分别在第2层和第4层添加Dropout，那么新架构的测试所用时间会变为多少？

- A、少于2s
- B、大于2s
- C、仍是2s
- D、说不准

正确答案是：C

解析：

在架构中添加Dropout这一改动仅会影响训练过程，而并不影响测试过程。

99、下列的哪种方法可以用来降低深度学习模型的过拟合问题？

- 1 增加更多的数据
- 2 使用数据扩增技术(data augmentation)
- 3 使用归纳性更好的架构
- 4 正规化数据
- 5 降低架构的复杂度

- A、 1 4 5
- B、 1 2 3
- C、 1 3 4 5
- D、 所有项目都有用

正确答案是：D

解析：


上面所有的技术都会对降低过拟合有所帮助。

100、混沌度(Perplexity)是一种常见的应用在使用深度学习处理NLP问题过程中的评估技术，关于混沌度，哪种说法是正确的？

- A、混沌度没什么影响
- B、混沌度越低越好
- C、混沌度越高越好
- D、混沌度对于结果的影响不一定

正确答案是： B



 大汤姆
关注 - 10
粉丝 - 0
[+加关注](#)

0

0

posted @ 2019-05-20 15:56 大汤姆 阅读(1043) 评论(0) 编辑 收藏

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论，请 [登录](#) 或 [注册](#)，[访问](#) 网站首页。

【推荐】了解你才能更懂你，博客园首发问卷调查，助力社区新升级

【推荐】超50万行VC++源码：大型组态工控、电力仿真CAD与GIS源码库

【推荐】如何打造一支战斗力爆棚的技术团队？



相关博文：

- AI面试必备/深度学习100问1-50题答案解析
 - Pandas100题
 - python 100题
 - 转：深度学习课程及深度学习公开课资源整合
 - 算法100题54
- » 更多推荐...

最新 IT 新闻：

- 消失的“体育大年”中，大平台的日子也不好过
 - 攻入腾讯腹地 字节跳动开始长视频暗战
 - “断舍离”经验丰富的百度 爱奇艺是下一个吗？
 - 摇摆的西瓜视频：抢人之战继续 长短视频皆涉足
 - 瑞幸退市后的三种命运
- » 更多新闻...

Copyright © 2020 大汤姆

Powered by .NET Core on Kubernetes