

目录

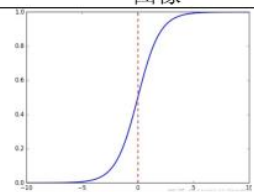
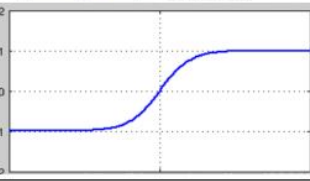
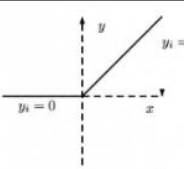
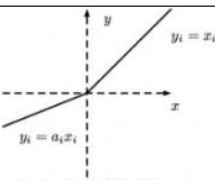
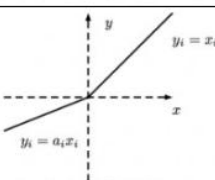
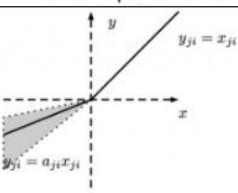
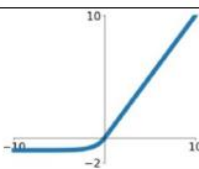
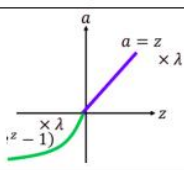
1.神经网络	4
1.1 各个激活函数的优缺点?	4
1.2 为什么 ReLU 常用于神经网络的激活函数?	5
1.3 梯度消失和梯度爆炸的解决方案? 梯度爆炸引发的问题?	6
1.4 如何确定是否出现梯度爆炸?	6
1.5 神经网络中有哪些正则化技术?	6
1.6 批量归一化(BN) 如何实现? 作用?	7
1.7 神经网络中权值共享的理解?	7
1.8 对 fine-tuning(微调模型)的理解? 为什么要修改最后几层神经网络权值?	7
1.9 什么是 Dropout? 为什么有用? 它是如何工作的?	7
1.10 如何选择 dropout 的概率?	7
1.11 什么是 Adam? Adam 和 SGD 之间的主要区别是什么?	8
1.12 为什么 Momentum 可以加速训练?	8
1.13 什么时候使用 Adam 和 SGD?	9
1.14 batch size 和 epoch 的平衡	9
1.15 SGD 每步做什么, 为什么能 online learning?	9
1.16 学习率太大(太小)时会发生什么? 如何设置学习率?	9
1.17 神经网络为什么不用拟牛顿法而是用梯度下降?	10
1.18 BN 和 Dropout 在训练和测试时的差别?	10
1.19 若网络初始化为 0 的话有什么问题?	10
1.20 sigmoid 和 softmax 的区别? softmax 的公式?	10
1.21 改进的 softmax 损失函数有哪些?	11
1.22 深度学习调参有哪些技巧?	11
1.23 神经网络调参, 要往哪些方向想?	12
1.24 深度学习训练中是否有必要使用 L1 获得稀疏解?	12
1.25 神经网络数据预处理方法有哪些? 中心化/零均值, 归一化	12
1.26 如何初始化神经网络的权重? 神经网络怎样进行参数初始化?	12
1.27 为什么构建深度学习模型需要使用 GPU?	12
1.28 前馈神经网络(FNN),递归神经网络(RNN)和 CNN 区别?	12
1.29 神经网络可以解决哪些问题?	12
1.30 如何提高小型网络的精度?	13
1.31 列举你所知道的神经网络中使用的损失函数	13
1.32 什么是鞍点问题? 梯度为 0, 海森矩阵不定的点, 不是极值点。	13
1.33 网络设计中, 为什么卷积核设计尺寸都是奇数?	13
2.CNN	13
2.1 卷积神经网络的结构	13
2.2 Keras 搭建 CNN	14
2.2 经典网络分类	15
2.2.1 LeNet	15
2.2.2 AlexNet	15
2.2.3 VGG	16
2.2.4 Inception(GoogLeNet)	16
2.2.5 ResNet	16
2.2.6 DenseNet	17
2.3 卷积层有哪些基本参数?	17

2.4 如何计算卷积层的输出的大小?	17
2.5 如何计算卷积层参数数量?	17
2.6 有哪些池化方法?	17
2.7 1*1 卷积的作用?	18
2.8 卷积层和池化层有什么区别?	18
2.9 卷积核是否一定越大越好?	18
2.10 卷积在图像中有什么直观作用?	18
2.11 CNN 中空洞卷积的作用是什么?	18
2.12 怎样才能减少卷积层参数数量?	18
2.13 在进行卷积操作时, 必须同时考虑通道和区域吗?	19
2.14 采用宽卷积, 窄卷积的好处有什么?	19
2.15 介绍反卷积(转置卷积)	19
2.16 如何提高卷积神经网络的泛化能力?	19
2.17 卷积神经网络在 NLP 与 CV 领域应用的区别?	20
2.18 全连接、局部连接、全卷积与局部卷积的区别?	20
2.19 卷积层和全连接层的区别?	21
2.20 Max pooling 如何工作? 还有其他池化技术吗?	21
2.21 卷积神经网络的优点? 为什么用小卷积核?	21
2.22 CNN 拆成 3x1 1x3 的优点?	21
2.23 BN、LN、IN、GN 和 SN 的区别?	21
3.RNN	22
3.1 RNNs 训练和传统 ANN 训练异同点?	22
3.2 为什么 RNN 训练的时候 Loss 波动很大?	22
3.3 RNN 中为什么会出现梯度消失?	22
3.4 如何解决 RNN 中的梯度消失问题?	23
3.5 CNN VS RNN	23
3.6 Keras 搭建 RNN	23
4.LSTM	24
4.1 LSTM 结构推导, 为什么比 RNN 好?	25
4.2 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?	25
4.3 LSTM 中为什么经常是两层双向 LSTM?	25
4.4 RNN 扩展改进	25
4.4.1 Bidirectional RNNs	25
4.4.2 CNN-LSTMs	25
4.4.3 Bidirectional LSTMs	25
4.4.4 GRU	25
4.5 LSTM、RNN、GRU 区别?	26
4.6 LSTM 是如何实现长短期记忆功能的?	26
4.7 LSTM 的原理、写 LSTM 的公式、手推 LSTM 的梯度反向传播	26
5.反向传播	26
5.1 什么是反向传播?	26
5.2 反向传播是如何工作的?	27
5.3 为什么需要反向传播?	27
5.4 手推 BP	27
6.GAN	29
6.1 生成器	29

6.2 判别器	30
6.3 训练技巧	30
7.超参数调整	31
7.1 神经网络中包含哪些超参数?	31
7.2 为什么要进行超参数调优?	31
7.3 超参数的重要性顺序	31
7.4 极端批样本数量下, 如何训练网络?	31
7.5 合理使用预训练网络	32
7.5.1 什么是微调 (fine-tune)	32
7.5.2 微调有哪些不同方法?	32
7.5.3 微调先冻结底层, 训练顶层的原因?	32
7.5.4 不同的数据集特性下如何微调?	33
7.5.5 目标检测中使用预训练模型的优劣?	33
7.5.6 目标检测中如何从零开始训练(train from scratch)?	33
7.6 自动化超参数搜索方法有哪些?	33

1.神经网络

1.1 各个激活函数的优缺点？

激活函数	公式	图像	优点	缺点
Sigmoid	$\sigma(x) = \frac{1}{1 + e^{-x}}$ $\sigma'(x) = \sigma(x)[1 - \sigma(x)]$		1. Sigmoid函数的输出在 (0, 1) 之间，输出范围有限，优化稳定，可以用作输出层。 2. 连续函数，便于求导。	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	$a = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ $\tanh(x) = 2\text{sigmoid}(2x) - 1$		1. 解决了原点对称问题 2. 比sigmoid更快	具有软饱和性，从而造成 梯度消失 ，在两边一样有趋近于 0 的情况。
ReLU	$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}$ $f(x) = \max(0, x)$		1. 收敛速度比 s型 和 tanh 快 2. 在x>0区域上，不会出现梯度饱和、梯度消失的问题。 3. 计算复杂度低，只要一个阈值就可以得到激活值。	梯度消失 没有完全解决 ，在x<0时，梯度为0，相当于神经元死亡而且不会复活
Leaky ReLU	$y_i = \begin{cases} x_i & \text{if } x_i \geq 0 \\ \frac{x_i}{a_i} & \text{if } x_i < 0, \end{cases}$		解决了神经死亡问题；	
PReLU (参数化修正线性单元)	$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{if } x < 0 \end{cases}$			
RReLU (随机纠正线性单元)	$y_{ji} = \begin{cases} x_{ji} & \text{if } x_{ji} \geq 0 \\ a_{ji}x_{ji} & \text{if } x_{ji} < 0, \end{cases}$ $a_{ji} \sim U(l, u), l < u \text{ and } l, u \in [0, 1)$			
ELU (指数线性单元)	$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases}$		其中 α 是一个可调整的参数，它控制着ELU负值部分在何时饱和。 右侧线性部分使得ELU能够 缓解梯度消失 ； 左侧软饱和能够让ELU对输入变化或噪声更鲁棒； ELU的输出均值接近于零，所以收敛速度更快。	
SELU (缩放指数线性单元)	$\text{selu}(x) = \lambda \begin{cases} x \\ \alpha e^x - \alpha \end{cases}$		selu的正半轴大于1，在方差过小的时候可以让它增大，同时防止了梯度消失。这样激活函数就有一个不动点，网络深了以后每一层的输出都是均值为0方差为1	
Maxout			1. Maxout的拟合能力非常强，可以拟合任意的凸函数。 2. Maxout具有ReLU的所有优点，线性、不饱和性。	参数比较多，本质上是在输出结果上又增加了一层。

1.2 为什么 ReLU 常用于神经网络的激活函数？

1. 在前向传播和反向传播过程中，ReLU 相比于 Sigmoid 等激活函数**计算量小**；
2. **避免梯度消失问题**。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问题（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），从而无法完成深层网络的训练。
3. 可以**缓解过拟合问题**的发生。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。

4.相比 Sigmoid 型函数，ReLU 函数有助于随机梯度下降方法收敛。

为什么需要激活功能？

激活函数是用来加入非线性因素的，因为线性模型的表达能力不够。

1.3 梯度消失和梯度爆炸的解决方案？梯度爆炸引发的问题？

梯度消失：靠近输出层的 hidden layer 梯度大，参数更新快，所以很快就会收敛；而靠近输入层的 hidden layer 梯度小，参数更新慢，几乎就和初始状态一样，随机分布。
另一种解释：当反向传播进行很多层的时候，由于每一层都对前一层梯度乘以了一个小数，因此越往前传递，梯度就会越小，训练越慢。

梯度爆炸：前面 layer 的梯度通过训练变大，而后面 layer 的梯度指数级增大。

- ①在深度多层感知机(MLP)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。
- ②在 RNN 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。

	梯度消失	梯度爆炸
原因	隐藏层的层数过多	
	采用了不合适的激活函数	权重的初始化值过大
解决方案	1. 换用Relu、LeakyRelu、Elu等激活函数 ReLu: 让激活函数的导数为1; LeakyRelu: 包含了ReLu的几乎所有有点，同时解决了ReLu中0区间带来的影响; ELU: 和LeakyRelu一样，都是为了解决0区间问题，相对于来，elu计算更耗时一些	
	2. BatchNormalization: 本质上是解决传播过程中的梯度问题	
	3. ResNet残差结构	
	4. LSTM结构: 可以改善RNN中的梯度消失问题，主要原因在于LSTM内部复杂的“门”	
	5. 预训练加微调 基本思想是每次训练一层隐藏层节点，将上一层隐藏层的输出作为输入，而本层的输出作为下一层的输入，这就是逐层预训练。 训练完成后，再对整个网络进行“微调(fine-tunning)”。 此方法相当于是找全局最优，然后整合起来寻找全局最优，但是现在基本都是直接拿imagenet的预训练模型直接进行finetunning。	
	6. 梯度剪切:思想是设值一个剪切阈值，如果更新梯度时，梯度超过了这个阈值，那么就将其强制限制在这个范围之内	
	7. 权重正则化: 是通过对网络权重做正则来限制过拟合。	

1.4 如何确定是否出现梯度爆炸？

- 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。

1.5 神经网络中有哪些正则化技术？

L2 正则化（Ridge）； L1 正则化（Lasso）；

权重衰减; 丢弃法;
批量归一化; 数据增强
早停法

1.6 批量归一化(BN) 如何实现？作用？

实现过程： 计算训练阶段 mini_batch 数量激活函数前结果的均值和方差，然后对其进行归一化，最后对其进行缩放和平移。

作用： 1.可以使用更高的学习率进行优化；

2.移除或使用较低的 dropout；

3.降低 L2 权重衰减系数；

4.调整了数据的分布，不考虑激活函数，它让每一层的输出归一化到了均值为 0 方差为 1 的分布，这保证了梯度的有效性，可以解决反向传播过程中的梯度问题。

1.7 神经网络中权值共享的理解？

权值共享这个词是由 LeNet5 模型提出来的。以 CNN 为例，在对一张图偏进行卷积的过程中，使用的是同一个卷积核的参数。

比如一个 $3 \times 3 \times 1$ 的卷积核，这个卷积核内 9 个的参数被整张图共享，而不会因为图像内位置的不同而改变卷积核内的权系数。

通俗说：就是用用一个卷积核不改变其内权系数的情况下卷积处理整张图片。

1.8 对 fine-tuning(微调模型)的理解？为什么要修改最后几层神经网络权值？

使用预训练模型的好处：在于利用训练好的 SOTA 模型权重去做特征提取，可以节省我们训练模型和调参的时间。

理由：

1.CNN 中更靠近底部的层（定义模型时先添加到模型中的层）编码的是更加通用的可复用特征，而更靠近顶部的层（最后添加到模型中的层）编码的是更专业化的特征。微调这些更专业化的特征更加有用，它更代表了新数据集上的有用特征。

2.训练的参数越多，过拟合的风险越大。很多 SOTA 模型拥有超过千万的参数，在一个不大的数据集上训练这么多参数是有过拟合风险的，除非你的数据集像 Imagenet 那样大。

1.9 什么是 Dropout？为什么有用？它是如何工作的？

背景：如果模型的参数太多，数据量又太小，则容易产生过拟合。为了解决过拟合，就同时训练多个网络。然后多个网络取均值。**费时！**

介绍：Dropout 可以防止过拟合，在前向传播的时候，让某个神经元的激活值以一定的概率 p 停止工作，这样可以使模型的泛化性更强。

Dropout 效果跟 bagging 效果类似（bagging 是减少方差 variance，而 boosting 是减少偏差 bias）。

加入 dropout 会使神经网络训练时间长，模型预测时不需要 dropout，记得关掉。

具体流程：

i.随机删除（临时）网络中一定的隐藏神经元，输入输出保持不变，

ii.让输入通过修改后的网络。然后把得到的损失同时修改后的网络进行反向传播。在未删除的神经元上面进行参数更新

iii.重复该过程（恢复之前删除掉的神经元，以一定概率删除其他神经元。前向传播、反向传播更新参数）

1.10 如何选择 dropout 的概率？

input 的 dropout 概率推荐是 0.8， hidden layer 推荐是 0.5

为什么 dropout 可以解决过拟合？

1.取平均的作用：

Dropout 产生了许多子结构之后的操作，父神经网络有 N 个节点，加入 Dropout 之后可以看做在权值不变的情况下（参数共享）将模型数量扩增到指数级别

2.减少神经元之间复杂的共适应关系，迫使网络去学习更加鲁棒；

Dropout 在训练和测试的区别？

训练时随机删除一些神经元，在测试模型时将所有的神经元加入。

1.11 什么是 Adam？Adam 和 SGD 之间的主要区别是什么？

1.12 一阶优化和二阶优化的方法有哪些？为什么不使用二阶优化？

一阶优化方法有：

SGD -> SGDM -> NAG -> AdaGrad -> AdaDelta / RMSProp(加速梯度下降) -> Adam -> Nadam

二阶优化

定义：对目标函数进行二阶泰勒展开，也就是二阶梯度优化方法

牛顿法 + BFGS 法

利用二阶泰勒展开,即牛顿法需要计算 **Hessian** 矩阵的逆，计算量大，在深度学习中并不常用。因此一般使用的是一阶梯度优化。

算法	介绍	公式
SGD	每一次迭代计算数据集的mini-batch的梯度, 然后对参数进行更新	
SGDM	在SGD基础上引入了一阶动量:	$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$
NAG	牛顿加速梯度, 是Momentum动量算法的变种	$\begin{cases} v_t = \alpha v_{t-1} + \eta_t \Delta J(W_t - \alpha v_{t-1}) \\ W_{t+1} = W_t - v_t \end{cases}$
AdaGrad	单调递减的学习率变化过于激进; 该维度上, 迄今为止所有梯度值的平方和。	$V_t = \sum_{\tau=1}^t g_{\tau}^2$
AdaDelta	不累积全部历史梯度, 而只关注过去一段时间窗口的下降梯度; 避免了二阶动量持续累积、导致训练过程提前结束的问题	$V_t = \beta_2 * V_{t-1} + (1 - \beta_2) g_t^2$
RMSProp	对梯度计算了微分平方加权平均数; 有利于消除了摆动幅度大的方向, 用来修正摆动幅度, 使得各个维度的摆动幅度都较小; 另一方面也使得网络函数收敛更快。	$\begin{aligned} s_{dw} &= \beta s_{dw} + (1 - \beta) dW^2 \\ s_{db} &= \beta s_{db} + (1 - \beta) db^2 \\ W &= W - \alpha \frac{dW}{\sqrt{s_{dw} + \epsilon}} \\ b &= b - \alpha \frac{db}{\sqrt{s_{db} + \epsilon}} \end{aligned}$
Adam	用梯度的一阶矩估计和二阶矩估计动态调整每个参数的学习率, 在经过偏置的校正后, 每一次迭代后的学习率都有个确定的范围, 使得参数较为平稳	$\begin{cases} m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \\ \hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \\ W_{t+1} = W_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \end{cases}$
	Adam = Adaptive + Momentum, Adam算法是将SGDM的二阶动量和RMSProp的二阶动量结合起来。	
Nadam	Nadam在Adam的基础上加入了一阶动量的累积	

1.12 为什么 Momentum 可以加速训练？

动量其实累加了历史梯度更新方向，所以在每次更新时，要是当前时刻的梯度与历史时刻梯度方向相似，这种趋势在当前时刻则会加强；要是不同，则当前时刻的梯度方向减弱。动量方法限制了梯度更新方向的随机性，使其沿正确方向进行。

1.13 什么时候使用 Adam 和 SGD?

Adam 等自适应学习率算法对于**稀疏数据**具有优势，且收敛速度很快；

但精调参数的 SGD (+Momentum) 往往能够取得更好的最终结果

Adam+SGD 组合策略：先用 Adam 快速下降，再用 SGD 调优。

如果模型是非常稀疏的，那么优先考虑自适应学习率的算法；

在模型设计实验过程中，要快速验证新模型的效果，用 **Adam** 进行快速实验优化；

在模型上线或者结果发布前，可以用精调的 **SGD** 进行模型的极致优化。

1.14 batch size 和 epoch 的平衡

batchsize	批大小。在深度学习中，一般采用 SGD 训练，即每次训练在训练集中取 batchsize 个样本训练	
	Batch size 大，收敛速度会比较慢，因为参数每次更新所需要的样本量增加了，但是会沿着比较准确的方向进行；	增大 Batch_Size，相对处理速度加快，所需内存容量增加。
	Batch_Size 过小，训练数据就会非常难收敛，从而导致欠拟合。	
iteration	1 个 iteration 等于使用 batchsize 个样本训练一次	
epoch	1 个 epoch 等于使用训练集中的全部样本训练一次	
	当一个完整的数据集通过了神经网络一次并且返回了一次的过程	
举例	例：训练集有 1000 个样本，batchsize=10，那么：训练完整个样本集需要：100 次 iteration，1 次 epoch	

1.15 SGD 每步做什么，为什么能 online learning?

online learning 强调的是学习是实时的，流式的，每次训练不用使用全部样本，而是以之前训练好的模型为基础，每来一个样本就更新一次模型，这种方法叫做 OGD (online gradient descent)，目的是快速地进行模型的更新，提升模型时效性。而 SGD 的思想正是在线学习的思想。

1.16 学习率太大(太小)时会发生什么？如何设置学习率？

1. 基于经验的手动调整

通过尝试不同的固定学习率，如 3、1、0.5、0.1、0.05、0.01、0.005、0.0001 等，观察迭代次数和 loss 的变化关系，找到 loss 下降最快关系对应的学习率。

太小，收敛过程将变得十分缓慢；

太大，梯度可能会在最小值附近来回震荡(**loss 爆炸**)，甚至可能无法收敛。

2. 基于策略的调整

① fixed、exponential、polynomial

② 自适应动态调整。adadelta、adagrad、ftrl、momentum、rmsprop、sgd

```
lr = tf.Variable(0.001, dtype=tf.float32) # 学习率
# 定义损失函数和优化器，损失函数用sigmoid交叉熵，优化器选择AdamOptimizer
# tf.reduce_mean: 计算张量tensor沿着指定的数轴上的的平均值，
# 主要用作降维或者计算tensor（图像）的平均值。
loss = tf.reduce_mean(tf.nn.sigmoid_cross_entropy_with_logits
                      (logits=pred, labels=one_hot_labels))
# 是一个寻找全局最优点的优化算法，引入了二次方梯度校正
optimizer = tf.train.AdamOptimizer(learning_rate=lr).minimize(loss)
```

Logit: 对它取对

数。 $Logit(Odds) = \log(\frac{P}{1-P})$

1.17 神经网络为什么不用拟牛顿法而是用梯度下降？

	梯度下降法 (SGD 为例)	牛顿法	拟牛顿法
时间复杂度	只需计算一阶导，时间复杂度低， $O(n)$	二阶导，每一步都需要求解目标函数的Hessian 矩阵及其逆，时间复杂度高， $O(n^3)$	不用二阶偏导数而构造出可以近似海塞矩阵的正定对称阵， $O(n^2)$
收敛速度	收敛慢，迭代次数大	收敛快，迭代次数小	收敛快，迭代次数小
几何上	用一个二次曲面去拟合你当前所处位置的局部曲面；二次曲面的拟合会比平面更好，所以牛顿法选择的下降路径会更符合真实的最优下降路径	用一个平面去拟合当前的局部曲面	
初始值要求	无太强要求，容易逃离鞍点	对初始值有一定要求，非凸问题容易陷入鞍点（牛顿法步长会越来越小）	
应用场景	特征维度较大的场景，如特征数 > 10k	特征维度较小的场景	需满足拟牛顿条件，更适合凸问题
总结：在神经网络（非凸问题）的训练中，大多数都采用梯度下降方法；而在训练逻辑回归（凸问题）等模型时，可采用梯度下降和拟牛顿方法。			

牛顿法原理：使用函数 $f(x)$ 的泰勒级数的前面几项来寻找方程 $f(x)=0$ 的根。

1.18 BN 和 Dropout 在训练和测试时的差别？

BN:

训练时：是对每一个 batch 的训练数据进行归一化，也即是用每一批数据的均值和方差。

测试时：都是对单个样本进行测试。这个时候的均值和方差是全部训练数据的均值和方差，这两个数值是通过移动平均法求得。

当一个模型训练完成之后，它的所有参数都确定了，包括均值和方差，gamma 和 bata。

Dropout: 只有在训练的时候才采用，是为了减少神经元对部分上层神经元的依赖，类似将多个不同网络结构的模型集成起来，减少过拟合的风险。

1.19 若网络初始化为 0 的话有什么问题？

w 初始化全为 0，很可能直接导致模型失效，无法收敛。

1.20 sigmoid 和 softmax 的区别？softmax 的公式？

Softmax回归模型输出层的激活函数如下所示：	
$z^{[L]} = W^{[L]}a^{[L-1]} + b^{[L]}$	$a_i^{[L]} = \frac{e^{z_i^{[L]}}}{\sum_{i=1}^C e^{z_i^{[L]}}}$
输出层每个神经元的输出 $a_i^{[L]}$ 对应属于该类的概率，满足：	$\sum_{i=1}^C a_i^{[L]} = 1$

	定义	使用	公式
sigmoid	将一个real value映射到 (0,1) 的区间	用来做二分类	
softmax	把一个 k 维的 real value 向量 (a1,a2,a3,a4 ...) 映射成一个 (b1,b2,b3,b4...)，其中 bi 是一个 0~1 的常数，输出神经元之和为 1.0	根据 bi 的概率大小来进行多分类	$S_i = \frac{e^i}{\sum_j e^j}$
二分类问题时 sigmoid 和 softmax 是一样的，求的都是 cross entropy loss(交叉熵损失)			

1.21 改进的 softmax 损失函数有哪些？

1.Large Margin Softmax Loss (**L-Softmax**)

加了一个 margin，要保证大于某一个阈值。这样可以尽量实现让类间间距更大化，类内距离更小。

2.Center Loss

通过将特征和特征中心的距离和 softmax loss 一同作为损失函数，使得类内距离更小，有点 L1，L2 正则化。

1.22 深度学习调参有哪些技巧？

1.准备数据

- 1.1 保证有大量、高质量并且带有干净标签的数据；
- 1.2 样本要随机化，防止大数据淹没小数据；
- 1.3 样本要做归一化。

2.预处理：0 均值和 1 方差化

3.minibatch：建议值 128，不要用过大的数值，否则很容易过拟合

4.梯度归一化：其实就是计算出来梯度之后，要除以 minibatch 的数量。

5.学习率

- 5.1 用一个一般的 lr 开始，然后逐渐的减小它；
- 5.2 建议值是 **0.01**，适用于很多 NN 的问题，一般倾向于小一点；
- 5.3 对于调度学习率的建议：如果在验证集上性能不再增加就让学习率除以 2 或者 5，然后继续，学习率会一直变得很小，到最后就可以停止训练了。
- 5.4 很多人用的一个设计学习率的原则就是监测一个比率（范数和权值范数之间的比率），如果这个比率在 10^{-3} 附近，如果小于这个值，学习会很慢，如果大于这个值，那么学习很不稳定，由此会带来失败。

6.使用验证集，可以知道什么时候开始降低学习率，和什么时候停止训练。

7.权重初始化的选择：

- 7.1 用高斯分布乘上一个很小的数
- 7.2 在深度网络中，随机初始化权重，使用 SGD 的话一般处理的都不好，这是因为初始化的权重太小了。这种情况下对于浅层网络有效，但是当足够深的时候就不行了，因为 weight 更新的时候，是靠很多 weight 相乘的，越乘越小，有点类似梯度消失。

8.如果训练 RNN 或者 LSTM，务必保证 gradient 的 norm 被约束在 15 或者 5(前提还是要先归一化 gradient)，这一点在 RNN 和 LSTM 中很重要。

9.Adam 收敛速度的确要快一些，可结果往往没有 **sgd + momentum** 的解好（如果模型比较复杂的话，sgd 是比较难训练的，这时候用 adam 较好）

10.如果使用 LSTM 来解决长时依赖的问题，记得初始化 bias 的时候要太一点

11.尽可能想办法多的**扩增训练数据**，如果使用的是图像数据，不妨对图像做一点扭转之类的操作，来扩充数据训练集合。

12.使用 **dropout**。

1. **early stop**，发现 `val_loss` 没更新，就尽早停止。
2. 评价最终结果的时候，多做几次，然后平均一下他们的结果。

1.23 神经网络调参，要往哪些方向想？

损失函数的选择是否是合适的
学习率的选取是否合适
batch size 选择是否合适
训练样本是否正常，是否需要增强
是否有设置 batch normalization(数据归一化)
激活函数的类型需要选取恰当
选取合适的优化算法（例如梯度下降，Adam 等）
是否存在过拟合

1.24 深度学习训练中是否有必要使用 L1 获得稀疏解？

没必要。对于加了 L1 正则的神经网络，大部分深度学习框架自带的优化器训练获得不了稀疏解；如果稀疏解的模式是无规律的，这种稀疏意义不大。

1.25 神经网络数据预处理方法有哪些？中心化/零均值，归一化

1.26 如何初始化神经网络的权重？神经网络怎样进行参数初始化？

- a: 将所有的参数初始化为 0
- b: 对 w 随机初始化
- c: Xavier 初始化：尽可能的让输入和输出服从相同的分布，这样就能够避免后面层的激活函数的输出值趋向于 0
- d: 何氏初试法

如果权重初始化太小，会导致神经元的输入过小，随着层数的不断增加，会出现信号消失的问题；也会导致 sigmoid 激活函数丢失非线性的能力，因为在 0 附近 sigmoid 函数近似是线性的。如果参数初始化太大，会导致输入状态太大。

对 sigmoid 激活函数来说，激活函数的值会变得饱和，从而出现梯度消失的问题。

1.27 为什么构建深度学习模型需要使用 GPU？

VGG16（在 DL 应用中经常使用 16 个隐藏层的卷积神经网络）大约具有 1.4 亿个参数，又称权重和偏见。如果用传统的方法，训练这种系统需要几年的时间。

神经网络的计算密集部分由多个矩阵乘法组成，可以简单地通过同时执行所有操作，而不是一个接一个地执行，要使用 GPU（图形处理单元）而不是 CPU 来训练神经网络。

1.28 前馈神经网络(FNN),递归神经网络(RNN)和 CNN 区别？

FNN 是指在一个 vector 内部的不同维度之间进行一次信息交互。

CNN 是以逐次扫描的方式在一个 vector 的局部进行多次信息交互。

RNN 就是在 hidden state 上增加了 loop，看似复杂，其实就是从时间维度上进行了展开。是在不同时刻的多个 vectors 之间进行信息交互。

1.29 神经网络可以解决哪些问题？

CNN	用于图像、计算机视觉领域	图像识别，目标检测，视频检测
RNN	用于序列数据的处理	语音领域：语音识别、语音分离；自然语言处理领域(文本检测，机器翻译)；图像视频领域
GAN	生成式对抗网络	语音、图像、视频、文字等
解决	1. 计算机视觉(包括图像与视频领域)	2. 语音领域（语音识别等）
	3. 自然语言处理(主要面向文本)	4. 智能驾驶、智慧城市、医疗

1.30 如何提高小型网络的精度？

- 1.模型蒸馏技术；
- 2.利用 AutoML 进行网络结构的优化。

1.31 列举你所知道的神经网络中使用的损失函数

欧氏距离，交叉熵，对比损失，合页损失

1.32 什么是鞍点问题？梯度为 0，海森矩阵不定的点，不是极值点。

1.33 网络设计中，为什么卷积核设计尺寸都是奇数？

- ① 保证像素点中心位置，避免位置信息偏移
- ② 填充边缘时能保证两边都能填充，原矩阵依然对称

2.CNN

对图像（不同的数据窗口数据）和滤波矩阵做内积（逐个元素相乘再求和）的操作就是所谓的『卷积』操作。

卷积神经网络由输入层、卷积层、激励层、池化层、全连接层组成。

①最左边:

数据输入层，对数据做一些处理：

去均值（把输入数据各个维度都中心化为 0，避免数据过多偏差，影响训练效果）

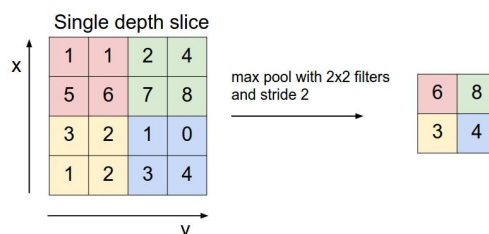
归一化（把所有的数据都归一到同样的范围）、PCA/白化等。CNN 只对训练集做“去均值”这一步。

②中间是:

CONV: 卷积层，线性乘积 求和。

RELU: 激励层，ReLU 是激活函数的一种。

POOL: 池化层，即取区域平均或最大。



③最右边是:

FC: 全连接层

2.1 卷积神经网络的结构

CNN层次结构	输出尺寸	作用
输入层	$W_1 \times H_1 \times 3$	卷积网络的原始输入，可以是原始或预处理后的像素矩阵
卷积层	$W_1 \times H_1 \times K$	参数共享、局部连接，利用平移不变性从全局特征图提取局部特征
激活层	$W_1 \times H_1 \times K$	将卷积层的输出结果进行非线性映射
池化层	$W_2 \times H_2 \times K$	进一步筛选特征，可以有效减少后续网络层次所需的参数量
全连接层	$(W_2 \cdot H_2 \cdot K) \times C$	将多维特征展平为2维特征，通常低维度特征对应任务的学习目标（类别或回归值）

池化层的作用：

减小图像尺寸即数据降维，缓解过拟合，保持一定程度的旋转和平移不变性。

2.2 Keras 搭建 CNN

```

from keras import layers
from keras.models import Sequential
from keras.optimizers import RMSprop
from keras.datasets import imdb
from keras.preprocessing import sequence

max_features = 10000
max_len = 500
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
x_train = sequence.pad_sequences(x_train, maxlen=max_len)
x_test = sequence.pad_sequences(x_test, maxlen=max_len)

model = Sequential()
# 输出空间的维度: 32 | 1维卷积窗口的大小: 7*7
model.add(layers.Embedding(max_features, 128, input_length=max_len))
model.add(layers.Conv1D(32, 7, activation='relu')) # 500-7+1 = 494
model.add(layers.MaxPooling1D(5)) # 494/5 = 98
model.add(layers.Conv1D(32, 7, activation='relu')) # 98-7+1 = 92
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))
model.summary()
model.compile(optimizer=RMSprop(lr=1e-4),
              loss='binary_crossentropy',
              metrics=['acc'])
# validation_split: 按一定比例从训练集中取出一部分作为验证集
history = model.fit(x_train, y_train, epochs=10,
                    batch_size=128, validation_split=0.2)

```

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 500, 128)	1280000
conv1d_1 (Conv1D)	(None, 494, 32)	28704
max_pooling1d_1 (MaxPooling1D)	(None, 98, 32)	0
conv1d_2 (Conv1D)	(None, 92, 32)	7200
global_max_pooling1d_1 (GlobalMaxPooling1D)	(None, 32)	0
dense_1 (Dense)	(None, 1)	33

总结：通常情况下，一维 CNN 的架构与 CV 的二维 CNN 很相似，它将 **Conv1D** 层和 **MaxPooling1D** 层

堆叠在一起，最后是一个全局池化运算或展平操作。

RNN 在处理非常长的序列时计算代价很大，但一维 CNN 的计算代价很小，所以在 RNN 之前使用一维 CNN 作为预处理步骤是一个好主意，这样可以使序列变短，并提取出有用的表示交给 RNN 来处理。

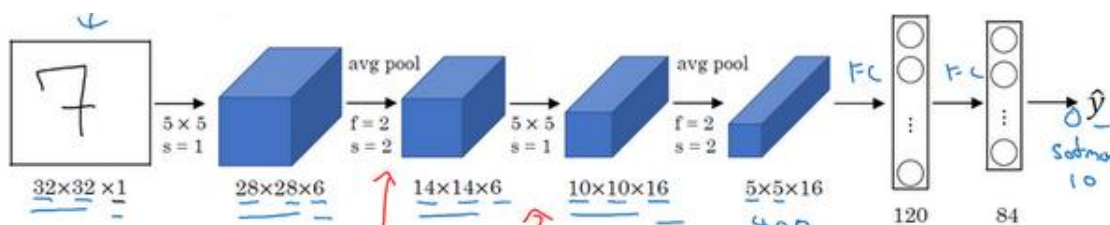
2.2 经典网络分类

2.2.1 LeNet

最早用于数字识别；针对灰度图进行训练的，输入图像大小为 $32 \times 32 \times 1$ ， 5×5 卷积核，不包含输入层的情况下共有 7层，每层都包含可训练参数。

输入的二维图像，先经过两次卷积层到池化层，再经过全连接层，最后使用 softmax 分类作为输出层。
(conv1->pool->conv2->pool2 再接全连接层)

- (1)每个卷积层包含三个部分：卷积、池化和非线性激活函数
- (2)使用卷积提取空间特征
- (3)降采样 (Subsample) 的平均池化层 (Average Pooling)
- (4)双曲正切 (Tanh) 或S型 (Sigmoid) 的激活函数
- MLP作为最后的分类器
- (5)层与层之间的稀疏连接减少计算复杂度



2.2.2 AlexNet

用多层小卷积叠起来替换单个的大卷积。

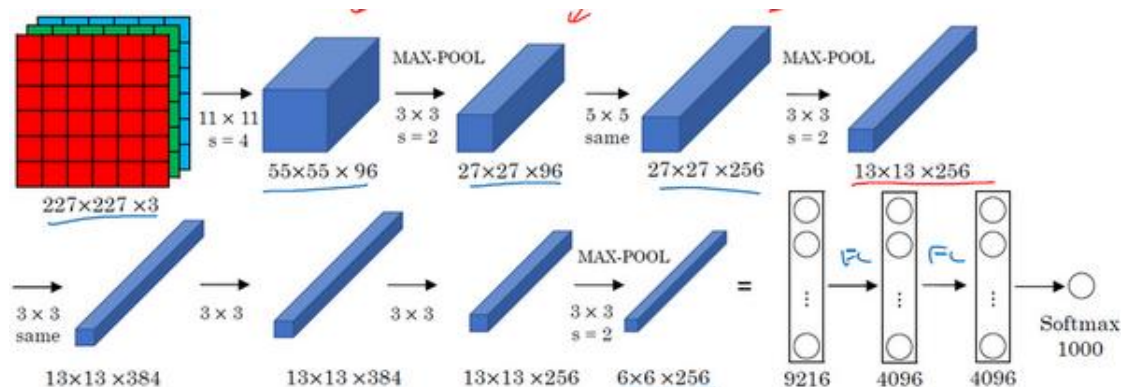
输入尺寸: $227 \times 227 \times 3$

卷积层: 5 个

降采样层(池化层): 3 个

全连接层: 2 个 (不包含输出层)

输出层: 1 个。1000 个类别



AlexNet 比 LeNet 表现更为出色的另一个原因是它使用了 ReLU 激活函数。

2.2.2.1 AlexNet 对比 LeNet 的优势?

1. AlexNet 比 LeNet 更深;
2. 用多层的小卷积来替换单个的大卷积;
3. 非线性激活函数: ReLU

- 4.防止过拟合的方法: Dropout, 数据增强
- 5.大数据训练: 百万级 ImageNet 图像数据
- 6.其他: GPU 实现, LRN 归一化层的使用

2.2.3 VGG

构筑了 16~19 层深的卷积神经网络, **VGG-16** 中的 16: 含有参数的有 16 个层

VGGNet 论文中全部使用了 3*3 的小型卷积核和 2*2 的最大池化层, 通过不断加深网络结构来提升性能。

卷积层: CONV=3*3 filters, s = 1, padding = same convolution。

池化层: MAX_POOL = 2*2, s = 2。

优点: 简化了卷积神经网络的结构; 缺点: 训练的特征数量非常大。

随着网络加深, 图像的宽度和高度都在以一定的规律不断减小, 每次池化后刚好缩小一半, 信道数目不断增加一倍。

2.2.3.1 VGG 使用 2 个 3*3 卷积的优势在哪里?

①减少网络层参数:

用两个 3*3 卷积比用 1 个 5*5 卷积拥有更少的参数量, 只有后者的 $2*3*3/(5*5)=0.72$ 。但是起到的效果是一样的, 两个 33 的卷积层串联相当于一个 55 的卷积层, 感受野的大小都是 5*5, 即 1 个像素会跟周围 5*5 的像素产生关联。

②更多的非线性变换:

2 个 33 卷积层拥有比 1 个 55 卷积层更多的非线性变换 (前者可以使用两次 ReLU激活函数, 而后者只有一次), 使得卷积神经网络对特征的学习能力更强。

2.2.3.2 每层卷积是否只能用一种尺寸的卷积核?

可以, 经典的神经网络一般都属于层叠式网络, 每层仅用一个尺寸的卷积核, 如 **VGG** 结构中使用了大量的 3*3 卷积层。

同一层特征图也可以分别使用多个不同尺寸的卷积核, 以获得不同尺度的特征, 再把这些特征结合起来, 得到的特征往往比使用单一卷积核的要好。比如 **GoogLeNet**、**Inception** 系列的网络。

2.2.4 Inception(GoogLeNet)

增加了卷积神经网络的宽度, 在多个不同尺寸的卷积核上进行卷积后再聚合, 并使用 **1*1** 卷积降维减少参数量。

2.2.4.1 inception 结构能不能缓解梯度消失?

可以, 因为 inception 结构额外计算了两个中间 loss, 防止了较深网络传播过程中的梯度消失问题。

2.2.5 ResNet

残差网络解决了网络退化的问题 (随着网络的深度增加, 准确度反而下降了)

2.2.5.1 ResNet 为什么不用 Dropout?

BN 在训练过程对每个单个样本的 forward 均引入多个样本 (Batch 个) 的统计信息, 相当于自带一定噪音, 起到正则效果, 所以也就基本消除了 Dropout 的必要。 (ResNet 训练 152 层深的神经网络)

2.2.5.2 ResNet 网络越来越深, 准确率会不会提升?

训练精度和测试精度迅速下降。

神经网络在反向传播过程中要不断地传播梯度, 而当网络层数加深时, 梯度在传播过程中会逐渐消失, 导致无法对前面网络层的权重进行有效的调整。

2.2.5.3 ResNet v1 与 ResNet v2 的区别?

通过 ResNet 残差学习单元的传播公式, 发现前馈和反馈信号可以直接传输,

因此 捷径连接 的非线性激活函数（如 ReLU）替换为 Identity Mappings。

同时，ResNet V2 在每一层中都使用了 Batch Normalization。这样处理之后，新的残差学习单元将比以前更容易训练且泛化性更强。

2.2.6 DenseNet

含义：前面所有层与后面层的密集连接，每一层的输入都是前面所有层输出的并集，而该层所学习的特征图也会被直接传给其后面所有层作为输入

优点：缓解梯度消失问题，特征复用，加强特征传播，减少参数量

缺点：内存占用高

梯度消失原因：每一层都直接连接 input 和 loss。

参数量少原因：每一层已经能够包含前面所有层的输出，只需要很少的特征图就可以了。

2.2.6.1 DenseNet 比 ResNet 好？

1.ResNet 连接方式可能会阻碍信息的流动，但是 DenseNet 每层的输出都和最终的输出直接相连，梯度可以直接从末端流到之前的所有的层。

2.DenseNet 连接有正则化的作用，可以减少过拟合。

3.DenseNet 直接连接不同层的特征图，而不是像 ResNet 一样 element-wise sum。**2.2.6.2 为什么 DenseNet 比 ResNet 更耗显存？**

DenseNet 的特征图像比 ResNet 大很多，导致卷积过程的计算量比 resnet 大很多。

2.3 卷积层有哪些基本参数？

①卷积核大小 (Kernel Size):

定义了卷积的感受野 在过去常设为 5，如 LeNet-5；现在多设为 3，通过堆叠 3×3×3 的卷积核来达到更大的感受域。

②卷积核步长 (Stride):

常见设置为 1，可以覆盖所有相邻位置特征的组合；当设置为更大值时相当于对特征组合降采样。

③填充方式 (Padding)

④输入通道数：指定卷积操作时卷积核的深度

⑤输出通道数：指定卷积核的个数

感受野：CNN 每一层输出的特征图上的像素点在原始图像上映射的区域大小。

2.4 如何计算卷积层的输出的大小？

$$O = \frac{(W - K + 2P)}{S} + 1$$

K 是过滤器尺寸，P 是填充，S 是步幅

2.5 如何计算卷积层参数数量？

卷积层参数量 = (filter size * 前一层特征图的通道数) * 当前层 filter 数量 + 当前层 filter 数量。
(卷积核长度*卷积核宽度*通道数+1) * 卷积核个数

假设输入层矩阵维度是 96×96×3，第一层卷积层使用尺寸为 5×5、深度为 16 的过滤器（卷积核尺寸为 5×5、卷积核数量为 16），那么这层卷积层的参数个数为 5×5×3×16 + 16=1216 个。

2.6 有哪些池化方法？

池化操作也叫做子采样(Subsampling)或降采样(Downsampling)，往往会用在卷积层之后，通过池化来降低卷积层输出的特征维度，有效减少网络参数的同时还可以防止过拟合现象。

①最大池化 和 ②平均池化

以最大池化为例，池化范围(2×2)(2×2)和滑窗步长(stride=2)(stride=2) 相同，仅提取一次相同区域的范化

特征。

2.7 1*1 卷积的作用？

- ①加入非线性函数。卷积层之后经过激励层，提升网络的表达能力;
- ②对卷积核通道数进行降维和升维，减小参数量。

2.8 卷积层和池化层有什么区别？

	卷积层	池化层
结构	零填充时输出维度不变，而通道数改变	通常特征维度会降低，通道数不变
稳定性	输入特征发生细微改变时，输出结果会改变	感受域内的细微变化不影响输出结果
作用	感受域内提取局部关联特征	感受域内提取泛化特征，降低维度
参数量	与卷积核尺寸、卷积核个数相关	不引入额外参数

- ①卷积层有参数，池化层没有参数;
- ②经过卷积层节点矩阵深度会改变。池化层不会改变节点矩阵的深度，但是它可以缩小节点矩阵的大小。

2.9 卷积核是否一定越大越好？

不一定，**缺点**：会导致计算量大幅增加，不利于训练更深层的模型，相应的计算性能也会降低。
卷积神经网络（VGG、GoogLeNet 等），发现通过堆叠 2 个 3×3 卷积核可以获得与 5×5 卷积核相同的感受视野，同时参数量会更少（ $3 \times 3 \times 2 + 1 < 5 \times 5 \times 1 + 1$ ）

优点：

文本特征有时需要有较广的感受域让模型能够组合更多的特征（如词组和字符）
卷积核的大小并没有绝对的优劣，需要视具体的应用场景而定，但是极大和极小的卷积核都是不合适的，单独的 1×1 极小卷积核只能用作分离卷积而不能对输入的原始特征进行有效的组合，极大的卷积核通常会组合过多的无意义特征从而浪费了大量的计算资源。

2.10 卷积在图像中有什么直观作用？

用来提取**图像的特征**，但不同层次的卷积操作提取到的特征类型是不相同的：

- 浅层卷积： 边缘特征
- 中层卷积： 局部特征
- 深层卷积： 全局特征

2.11 CNN 中空洞卷积的作用是什么？

空洞卷积也叫扩张卷积，在保持参数个数不变的情况下增大了卷积核的感受野，同时它可以保证输出的特征映射的大小保持不变。一个扩张率为 2 的 3×3 卷积核，感受野与 5×5 的卷积核相同，但参数量仅为 9 个。

2.12 怎样才能减少卷积层参数量？

- ①使用**堆叠小卷积核**代替大卷积核：
VGG 网络中 2 个 3×3 的卷积核可以代替 1 个 5×5 的卷积核
- ②使用**分离卷积**操作：
将原本 K×K×C 的卷积操作分离为 K×K×1 和 1×1×C 的两部分操作
- ③添加 **1×1** 的卷积操作：与分离卷积类似，但是通道数可变，在 K×K×C1 卷积前添加 1×1×C2 的卷积核（满足 $C2 < C1$ ）
- ④在卷积层前使用**池化**操作：池化可以降低卷积层的输入特征维度

2.13 在进行卷积操作时，必须同时考虑通道和区域吗？

①**标准卷积**同时考虑通道和区域

②**通道分离（深度分离）卷积网络(Xception 网络):**

首先对每一个通道进行各自的卷积操作，有多少个通道就有多少个过滤器。得到新的通道特征矩阵之后，再对这批新通道特征进行标准的 1×1 跨通道卷积操作。

2.14 采用宽卷积,窄卷积的好处有什么？

宽卷积、窄卷积其实是一种**填充方式**。

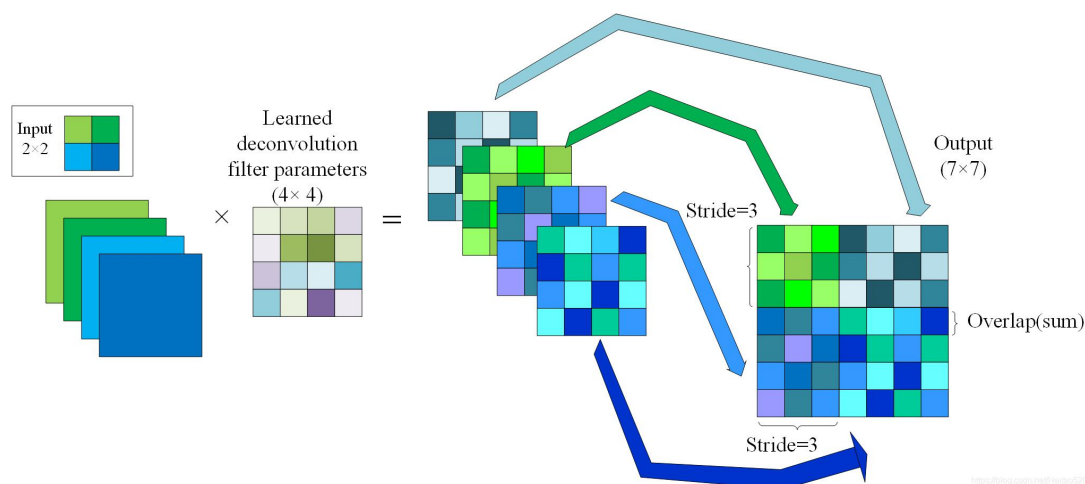
①**宽卷积('SAME'填充):**

对卷积核不满足整除条件的输入特征进行补全，以使卷积层的输出维度保持与输入特征维度一致。

②**窄卷积('VALID'填充):**

不进行任何填充，在输入特征边缘位置若不足以进行卷积操作，则对边缘信息进行舍弃,因此在步长为 1 的情况下该填充方式的卷积层输出特征维度可能会略小于输入特征的维度。

2.15 介绍反卷积(转置卷积)



正向传播时乘以卷积核的转置矩阵，反向传播时乘以卷积核矩阵，由卷积输出结果近似重构输入数据，上采样。

输入： 2×2 ，卷积核： 4×4 ，滑动步长：3，输出： 7×7

过程如下：

①输入图片每个像素进行一次 **full 卷积**，根据 full 卷积大小计算可以知道每个像素的卷积后大小为 $1+4-1=4$ ，即 4×4 大小的特征图，输入有 4 个像素所以 4 个 4×4 的特征图。

②将 4 个特征图进行步长为 3 的相加；输出的位置和输入的位置相同。步长为 3 是指每隔 3 个像素进行相加，重叠部分进行相加，即输出的第 1 行第 4 列是由红色特阵图的第一行第四列与绿色特征图的第一行第一列相加得到，其他类推。

可以看出反卷积的大小是由卷积核大小与滑动步长决定，in 是输入大小，k 是卷积核大小，s 是滑动步长，out 是输出大小 得到 $out=(in-1)*s+k$ 上图过程就是 $(2-1)*3+4=7$ 。

2.16 如何提高卷积神经网络的泛化能力？

1. 增加数据	
2. 使用更大批次	在相同迭代次数和学习率的条件下，每批次采用更多的数据将有助于模型更好的学习到正确的模式
3. 调整数据分布	模型过多地学习某类数据容易导致其输出结果偏向于该类型的数据
4. 调整目标函数	
5. 调整网络结构	<p>浅层卷积神经网络中，参数量较少往往使模型的泛化能力不足而导致欠拟合，此时通过叠加卷积层可以有效地增加网络参数，提高模型表达能力；</p> <p>深层卷积网络中，若没有充足的训练数据则容易导致模型过拟合，此时通过简化网络结构减少卷积层数可以起到提高模型泛化能力的作用。</p>
6. 数据增强	通过平移、旋转、加噪声等一些列变换来增加训练数据，同类数据的表现形式也变得更多样，有助于模型提高泛化能力，需要注意的是数据变化应尽可能不破坏元数数据的主体特征
7. 权值正则化	在损失函数中添加一项权重矩阵的正则项作为惩罚项，用来惩罚损失值较小时网络权重过大的情况，此时往往是网络权值过拟合了数据样本。
8. dropout	即网络结构上的正则化，通过随机性地屏蔽某些神经元的输出让剩余激活的神经元作用，可以使模型的容错性更强。

2.17 卷积神经网络在 NLP 与 CV 领域应用的区别？

自然语言处理对一维信号（词序列）做操作，输入数据通常是离散取值（例如表示一个单词或字母通常表示为词典中的 one hot 向量）

计算机视觉则是对二维（图像）或三维（视频流）信号做操作。输入数据是连续取值（比如归一化到 0，1 之间的- 灰度值）。

2.18 全连接、局部连接、全卷积与局部卷积的区别？

连接方式	示意图	说明
全连接		层间神经元完全连接 ，每个输出神经元可以获取到所有输入神经元的信息，有利于信息汇总，常置于网络末层；连接与连接之间独立参数，大量的连接大大增加模型的参数规模。
局部连接		层间神经元只有局部范围内的连接 ，在这个范围内采用全连接的方式，超过这个范围的神经元则没有连接；连接与连接之间独立参数，相比于全连接减少了感受域外的连接，有效 减少参数规模
全卷积		层间神经元只有局部范围内的连接 ，在这个范围内采用全连接的方式，连接所采用的 参数在不同感受域之间共享 ，有利于提取特定模式的特征；相比于局部连接，共用感受域之间的参数可以进一步减少参数量。
局部卷积		层间神经元只有局部范围内的连接 ， 感受域内采用全连接的方式 ，而感受域之间间隔采用 局部连接与全卷积的连接方式 ；相比与全卷积成倍引入额外参数，但有更强的灵活性和表达能力；相比于局部连接，可以 有效控制参数量

2.19 卷积层和全连接层的区别？

- 1.卷积层是局部连接，所以提取的是局部信息；全连接层是全局连接，所以提取的是全局信息；
- 2.当卷积层的局部连接是全局连接时，全连接层是卷积层的特例；

2.20 Max pooling 如何工作？还有其他池化技术吗？

- 1.Max pooling:选取滑动窗口的最大值
- 2.Average pooling: 平均滑动窗口中的所有值
- 3.Global average pooling: 平均每页特征图的所有值

2.21 卷积神经网络的优点？为什么用小卷积核？

多个小的卷积核叠加使用要远比一个大的卷积核单独使用效果要好的多。

1.局部连接

这个是最容易想到的，每个神经元不再和上一层的所有神经元相连，而只和一小部分神经元相连。这样就减少了很多参数。

2.权值共享

一组连接可以共享同一个权重，而不是每个连接有一个不同的权重，这样又减少了很多参数。

3.下采样

Pooling 层利用图像局部相关性的原理，对图像进行子抽样，可以减少数据处理量同时保留有用信息。通过去掉 Feature Map 中不重要的样本，进一步减少参数数量。

2.22 CNN 拆成 3x1 1x3 的优点？

为了压缩模型参数量（这里参数由 $3 \times 3 = 9$ 降低到 $1 \times 3 + 3 \times 1 = 6$ ），但是计算量基本没变（乘数目没变）。

2.23 BN、LN、IN、GN 和 SN 的区别？

将输入的 feature map shape 记为 $[N, C, H, W]$ ，其中 N 表示 batch size，即 N 个样本； C 表示通道数； H 、 W 分别表示特征图的高度、宽度。

	名称	区别
BN	批归一化 15年	在batch上，对N、H、W做归一化，而保留通道 C 的维度。BN对较小的batch size效果不好。BN适用于固定深度的前向神经网络，如CNN，不适用于RNN；
LN	层归一化 16年	LN在通道方向上，对C、H、W归一化，主要对RNN效果明显
IN	实例规范化 17年	在图像像素上，对H、W做归一化，用在风格化迁移
GN	组归一化 18年	将channel分组，然后再做归一化
SN	自适应归一化	将 BN、LN、IN 结合，赋予权重，让网络自己去学习归一化层应该使用什么方法，自动为神经网络的每个归一化层确定一个合适的归一化操作。

为什么需要卷积？不能使用全连接层吗？

为什么降采用使用 max pooling，而分类使用 average pooling？

CNN 是否抗旋转？如果旋转图像，CNN 的预测会怎样？

什么是数据增强？为什么需要它们？你知道哪种增强？

如何选择要使用的增强？

什么是迁移学习？它是如何工作的？
什么是目标检测？你知道有哪些框架吗？
什么是对象分割？你知道有哪些框架吗？

3.RNN

核心思想：像人一样拥有记忆能力。用以往的记忆和当前的输入，生成输出。

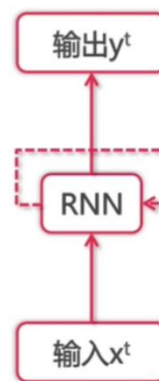
RNN 和 传统神经网络 最大的区别：

在于每次都会将前一次的输出结果，带到下一次的隐藏层中，一起训练。

- ◆ 维护一个状态作为下一步的额外输入
- ◆ 每一步使用同样的激活函数和参数

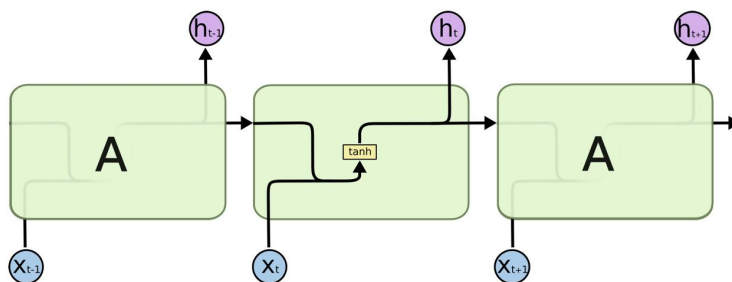
$$s_t = f_w(s_{t-1}, x_t)$$

新状态 旧状态 输入



RNN 应用场景：

1.文本生成 2.语音识别 3.机器翻译 4.生成图像描述 5.视频标记



缺点：

RNN 有短期记忆问题，无法处理很长的输入序列

训练 RNN 需要投入极大的成本

RNN 是一种死板的逻辑，越晚的输入影响越大，越早的输入影响越小，且无法改变这个逻辑。

3.1 RNNs 训练和传统 ANN 训练异同点？

相同点：都使用 BP 误差反向传播算法。

不同点：

RNNs 网络参数 W, U, V 是共享的，而传统神经网络各层参数间没有直接联系。

对于 RNNs，在使用梯度下降算法中，每一步的输出不仅依赖当前步的网络，还依赖于之前若干步的网络状态。

3.2 为什么 RNN 训练的时候 Loss 波动很大？

由于 RNN 特有的 memory 会影响后期其他的 RNN 的特点，梯度时大时小，lr 没法个性化的调整，导致 RNN 在 train 的过程中，Loss 会震荡起伏，为了解决 RNN 的这个问题，在训练的时候，可以设置临界值，当梯度大于某个临界值，直接截断，用这个临界值作为梯度的大小，防止大幅震荡。

3.3 RNN 中为什么会出现梯度消失？

梯度消失现象：累乘会导致激活函数导数的累乘，如果取 tanh 或 sigmoid 函数作为激活函数的话，

那么必然是一堆小数在做乘法，结果就是越乘越小。随着时间序列的不断深入，小数的累乘就会导致梯度越来越小直到接近于 0，这就是“梯度消失”现象。

实际使用中，会优先选择 **tanh** 函数，原因是 tanh 函数相对于 sigmoid 函数来说梯度较大，收敛速度更快且引起梯度消失更慢。

3.4 如何解决 RNN 中的梯度消失问题？

1. 选取更好的激活函数，如 **Relu** 激活函数。ReLU 函数的左侧导数为 0，右侧导数恒为 1，这就避免了“梯度消失”的发生。但恒为 1 的导数容易导致“梯度爆炸”，但设定合适的阈值可以解决这个问题。

2. 加入 **BN** 层，其优点：加速收敛. 控制过拟合，可以少用或不用 Dropout 和正则。降低网络对初始化权重不敏感，且能允许使用较大的学习率等。

3. 改变传播结构，**LSTM** 结构可以有效解决这个问题。

3.5 CNN VS RNN

- ◆ CNN不能完美的解决序列式问题
- ◆ CNN卷积相当于N-gram，LSTM提取更长的依赖
- ◆ Pre-train的embedding
- ◆ 双向RNN会增强效果
- ◆ CNN模型并行程度高，更快
- ◆ embedding模型压缩

不同点

1. CNN 空间扩展，神经元与特征卷积；RNN 时间扩展，神经元与多个时间输出计算

2. RNN 可以用于描述时间上连续状态的输出，有记忆功能，CNN 用于静态输出。

3.6 Keras 搭建 RNN

```
from keras.datasets import imdb
from keras.preprocessing import sequence
from keras.layers import Dense
from keras.layers import SimpleRNN
# from keras.layers import LSTM

max_features = 10000 # 作为特征的单词个数
maxlen = 500
# 在这么多单词之后截断文本（这些单词都属于前 max_features 个最常见的单词）
batch_size = 32
(input_train, y_train), (input_test, y_test) = imdb.load_data(num_words=max_features)

input_train = sequence.pad_sequences(input_train, maxlen=maxlen)
input_test = sequence.pad_sequences(input_test, maxlen=maxlen)

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(SimpleRNN(32)) #LSTM
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(input_train, y_train,
                    epochs=10,
                    batch_size=512,
                    validation_split=0.2)
```

```

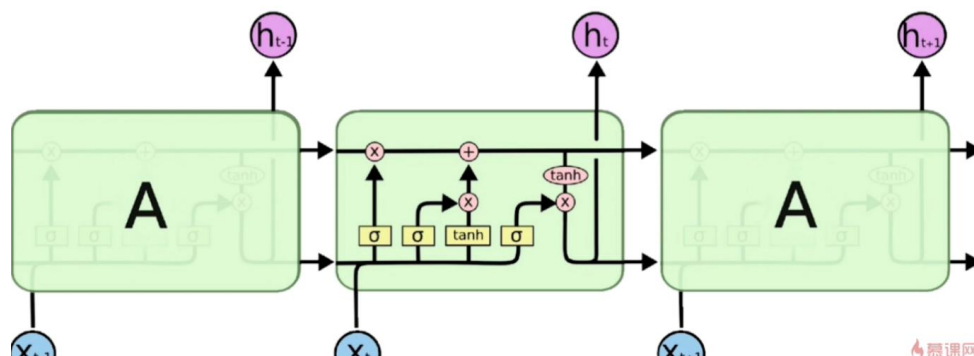
Train on 20000 samples, validate on 5000 samples
Epoch 1/2
20000/20000 [=====] - 49s 2ms/step - loss: 0.6408 - acc: 0.6818 - val_loss: 0.5006 - val_acc: 0.8162
Epoch 2/2
20000/20000 [=====] - 45s 2ms/step - loss: 0.4442 - acc: 0.8265 - val_loss: 0.4372 - val_acc: 0.7996

```

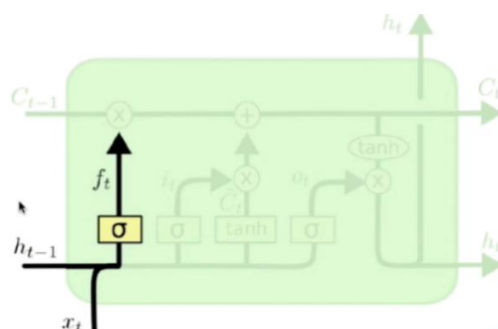
4.LSTM

长短期记忆网络（Long Short-Term Memory）是一种时间循环神经网络，是为了解决一般的 RNN 存在的长期依赖问题而专门设计出来的，所有的 RNN 都具有一种重复神经网络模块的链式形式。

三个门（遗忘门，输入门，输出门），两个状态（ C_t, h_t ）



遗忘门

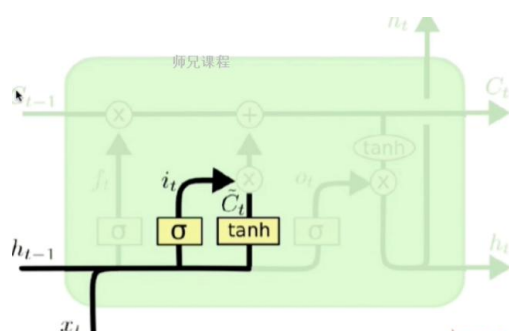


作用对象：细胞状态。

作用：将细胞状态中的信息选择性的遗忘。

f_t 和 C_{t-1} 做点积操作， f_t 确保 C_{t-1} 有哪些东西需要被遗忘调

输入层门



作用对象：细胞状态

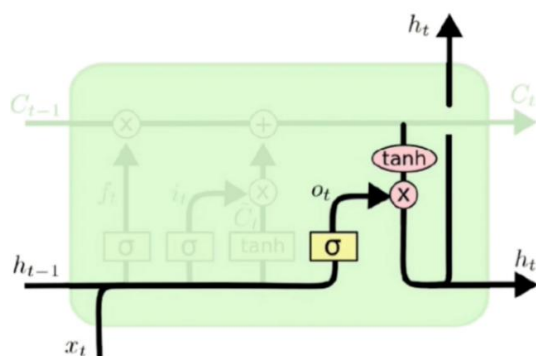
作用：将新的信息选择性的记录到细胞状态中。

操作步骤：

步骤一: sigmoid 层称 “输入门层” 决定什么值我们将要更新

步骤二, tanh 层创建一个新的候选值向量加入到状态中

输出层门



作用对象：隐层 h_t 作用：确定输出什么值。

操作步骤：

步骤一:通过 sigmoid 层来确定细胞状态的哪个部分将输出。

步骤二:把细胞状态通过 \tanh 进行处理，并将它和 sigmoid 门的输出相乘，最终我们仅仅会输出我们确定输出的那部分。

4.1 LSTM 结构推导，为什么比 RNN 好？

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。

4.2 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？

sigmoid 用在了各种 gate 上，产生 0~1 之间的值，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。

4.3 LSTM 中为什么经常是两层双向 LSTM？

有些时候预测需要由前面若干输入和后面若干输入共同决定，这样会更加准确。

4.4 RNN 扩展改进

4.4.1 Bidirectional RNNs

将两层 RNNs 叠加在一起，当前时刻输出(第 t 步的输出)不仅仅与之前序列有关，还与之后序列有关。例如：为了预测一个语句中的缺失词语，就需要该词汇的上下文信息。Bidirectional RNNs 是一个相对较简单的 RNNs，是由两个 RNNs 上下叠加在一起组成的。输出由前向 RNNs 和后向 RNNs 共同决定。

4.4.2 CNN-LSTMs

该模型中，CNN 用于提取对象特征，LSTMs 用于预测。CNN 由于卷积特性，其能够快速而且准确地捕捉对象特征。LSTMs 的优点：能够捕捉数据间的长时依赖性。

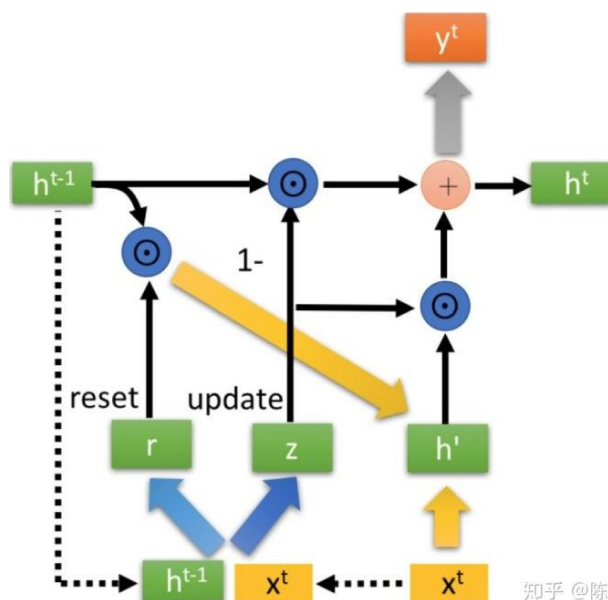
4.4.3 Bidirectional LSTMs

有两层 LSTMs。一层处理过去的训练信息，另一层处理将来的训练信息。通过前向 LSTMs 获得前向隐藏状态，后向 LSTMs 获得后向隐藏状态，当前隐藏状态是前向隐藏状态与后向隐藏状态的组合。

4.4.4 GRU

（14 年提出）是一般的 RNNs 的变型版本，其主要是从以下两个方面进行改进。

- 1.以语句为例，序列中不同单词处的数据对当前隐藏层状态的影响不同，越前面的影响越小，即每个之前状态对当前的影响进行了距离加权，距离越远，权值越小。
- 2.在产生误差 error 时，其可能是由之前某一个或者几个单词共同造成，所以应当对对应的单词 weight 进行更新。GRUs 的结构如下图所示。GRUs 首先根据当前输入单词向量 word vector 以及前一个隐藏层状态 hidden state 计算出 update gate 和 reset gate。再根据 reset gate、当前 word vector 以及前一个 hidden state 计算新的记忆单元内容(new memory content)。当 reset gate 为 1 的时候，new memory content 忽略之前所有 memory content，最终的 memory 是由之前的 hidden state 与 new memory content 一起决定。



4.5 LSTM、RNN、GRU 区别？

	LSTM	GRU	RNN
复杂度比较	LSTM > GRU > RNN，当数据集不大时，GRU和LSTM难分伯仲、但是数据集变大时LSTM更优		
	不仅传递隐状态还传递细胞状态	都只有隐状态	
结构比较	有三个门 (forget, input, output)	有两个门 (update和reset)	
	用memory cell 把hidden state 包装起来	直接将hidden state 传给下一个单元	

与 LSTM 相比，GRU 内部少了一个”门控“，参数比 LSTM 少，但是却也能够达到与 LSTM 相当的功能。考虑到硬件的计算能力和时间成本，因而很多时候我们也会选择更加实用的 GRU。

4.6 LSTM 是如何实现长短期记忆功能的？

4.7 LSTM 的原理、写 LSTM 的公式、手推 LSTM 的梯度反向传播

5.反向传播

5.1 什么是反向传播？

通俗解释：

类比几个人站成一排，第一个人看一幅画（输入数据），描述给第二个人（隐层）……依此类推，到最后一个人（输出）的时候，画出来的画肯定不能看了（误差较大）。

反向传播就是：把画拿给最后一个人看（**求取误差**），然后最后一个人就会告诉前面的人下次描述时需要注意哪里（**权值修正**）

一种与最优化方法（如梯度下降法）结合使用的，用来训练人工神经网络的常见方法。

目的是更新神经元参数,而神经元参数正是 $z=wx+b$ 中的 (w,b) .对参数的更新,利用损失值 loss 对参数的导数,并沿着负梯度方向进行更新。

“正向传播”求损失，“反向传播”回传误差

5.2 反向传播是如何工作的？

1.输入层接收 x

2.使用权重 w 对输入进行建模

3.每个隐藏层计算输出，数据在输出层准备就绪

4.实际输出和期望输出之间的差异称为误差

5.返回隐藏层并**调整权重**，以便在以后的运行中减少此错误

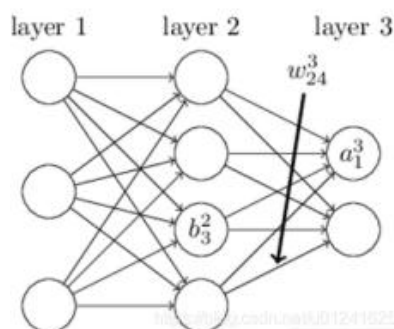
这个过程一直重复，直到我们得到所需的输出。训练阶段在监督下完成。一旦模型稳定下来，就可以用于生产。

5.3 为什么需要反向传播？

- 反向传播快速、简单且易于实现
- 没有要调整的参数
- 不需要网络的先验知识
- 模型不需要学习函数的特性

5.4 手推 BP

推导：**链式求导法**则反复用



w^l_{jk} 表示第 l 层的第 j 个神经元连接到第 $(l-1)$ 层的第 k 个神经元的权重；

b^l_j 表示第 l 层的第 j 个神经元的偏置；

z^l_j 表示第 l 层的第 j 个神经元的输入，即： $z^l_j = \sum_k w^l_{jk} a^{l-1}_k + b^l_j$

a^l_j 表示第 l 层的第 j 个神经元的输出，即： $a^l_j = \sigma \left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j \right)$

σ 表示激活函数

C 表示损失函数，用来计算最终输出与实际值的误差。

L 表示神经网络的最大层数。

第 l 层第 j 个神经元产生的误差定义为： $\delta^l_j = \frac{\partial C}{\partial z^l_j}$

公式1（计算最后一层神经网络产生的误差）:

$$\delta^L = \nabla_a C \odot \sigma' (z^L)$$

其中 \odot 表示Hadamard乘积，用于矩阵或向量之间点对点的乘法运算。

推导过程：

$$\begin{aligned} \therefore \delta_j^L &= \frac{\partial C}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \cdot \sigma' (z_j^L) \\ \therefore \delta^L &= \frac{\partial C}{\partial a^L} \odot \frac{\partial a^L}{\partial z^L} = \nabla_a C \odot \sigma' (z^L) \end{aligned}$$

公式2（由后往前，计算前后两层神经网络直接的误差关系，递推公式）：

$$\delta^l = \left((w^{l+1})^T \delta^{l+1} \right) \odot \sigma' (z^l)$$

推导过程：

$$\begin{aligned} \therefore \delta_j^l &= \frac{\partial C}{\partial z_j^l} = \sum_k \frac{\partial C}{\partial z_k^{l+1}} \cdot \frac{\partial z_k^{l+1}}{\partial a_j^l} \cdot \frac{\partial a_j^l}{\partial z_j^l} \\ &= \sum_k \delta_k^{l+1} \cdot \frac{\partial (w_{kj}^{l+1} a_j^l + b_k^{l+1})}{\partial a_j^l} \cdot \sigma' (z_j^l) \\ &= \sum_k \delta_k^{l+1} \cdot w_{kj}^{l+1} \cdot \sigma' (z_j^l) \end{aligned}$$

公式3（计算权重的梯度）：

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

推导过程：

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{jk}^l} = \delta_j^l \cdot \frac{\partial (w_{jk}^l a_k^{l-1} + b_j^l)}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$$

公式4（计算偏置的梯度）：

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l$$

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l \cdot \frac{\partial (w_{jk}^l a_k^{l-1} + b_j^l)}{\partial b_j^l} = \delta_j^l$$

伪代码：

对于训练集中的每个样本 x ，设输入层（Input layer）对应的激活值 a^1 ：

- 前向传播： $z^l = w^l a^{l-1} + b^l, a^l = \sigma(z^l)$
- 计算输出层产生的错误： $\delta^L = \nabla_a C \odot \sigma'(z^L)$
- 反向传播错误： $\delta^l = \left((w^{l+1})^T \delta^{l+1} \right) \odot \sigma'(z^l)$
- 使用梯度下降，训练参数：

$$w^l \rightarrow w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T$$
$$b^l \rightarrow b^l - \frac{\eta}{m} \sum_x \delta^{x,l}$$

6.GAN

生成式对抗网络，由一个生成器网络和一个判别器网络组成。判别器的训练目的是能够区分生成器的输出与来自训练集的真实图像，生成器的训练目的是欺骗判别器。值得注意的是，生成器从未直接见过训练集中的图像，它所知道的关于数据的信息都来自于判别器。

GAN 相关的技巧：

- 我们使用 tanh 作为生成器最后一层的激活，而不用 sigmoid，后者在其他类型的模型中更加常见。
- 我们使用正态分布（高斯分布）对潜在空间中的点进行采样，而不用均匀分布。
- 随机性能够提高稳健性。训练 GAN 得到的是一个动态平衡，所以 GAN 可能以各种方式“卡住”。在训练过程中引入随机性有助于防止出现这种情况。我们通过两种方式引入随机性：一种是在判别器中使用 dropout，另一种是向判别器的标签添加随机噪声。
- 稀疏的梯度会妨碍 GAN 的训练。在深度学习中，稀疏性通常是我们需要的属性，但在 GAN 中并非如此。有两件事情可能导致梯度稀疏：最大池化运算和 ReLU 激活。我们推荐使用步进卷积代替最大池化来进行下采样，还推荐使用 LeakyReLU 层来代替 ReLU 激活。LeakyReLU 和 ReLU 类似，但它允许较小的负数激活值，从而放宽了稀疏性限制。
- 在生成的图像中，经常会见到棋盘状伪影，这是由生成器中像素空间的不均匀覆盖导致的。为了解决这个问题，每当在生成器和判别器中都使用步进的 Conv2DTranpose 或 Conv2D 时，使用的内核大小要能够被步幅大小整除。

6.1 生成器

它将一个向量（来自潜在空间，训练过程中对其随机采样）转换为一张候选图像。GAN 常见的诸多问题之一，就是生成器“卡在”看似噪声的生成图像上。

一种可行的解决方案是在判别器和生成器中都使用 dropout。

```

latent_dim = 32,height = 32,width = 32,channels = 3
generator_input = keras.Input(shape=(latent_dim,))

# 将输入转换为大小为 16×16 的128 个通道的特征图
x = layers.Dense(128 * 16 * 16)(generator_input)
x = layers.LeakyReLU()(x)
x = layers.Reshape((16, 16, 128))(x)

# 添加一个卷积层
x = layers.Conv2D(256, 5, padding='same')(x)
x = layers.LeakyReLU()(x)

# 上采样为 32×32 Transpose调换x, y, z的位置
x = layers.Conv2DTranspose(256, 4, strides=2, padding='same')(x)
x = layers.LeakyReLU()(x)

x = layers.Conv2D(256, 5, padding='same')(x)
x = layers.LeakyReLU()(x)
x = layers.Conv2D(256, 5, padding='same')(x)
x = layers.LeakyReLU()(x)

# 生成一个大小为 32×32 的单通道特征图 (即 CIFAR10 图像的形状)
x = layers.Conv2D(channels, 7, activation='tanh', padding='same')(x)
#将生成器模型实例化, 它将形状为 (latent_dim,)的输入映射到
#形状为 (32, 32, 3) 的图像
generator = keras.models.Model(generator_input, x)

```

6.2 判别器

接下来开发 discriminator 模型, 它接收一张候选图像 (真实的或合成的) 作为输入, 并将其划分到这两个类别之一: “生成图像” 或 “来自训练集的真实图像”。

```

discriminator_input = layers.Input(shape=(height, width, channels))
x = layers.Conv2D(128, 3)(discriminator_input)
x = layers.LeakyReLU()(x)
x = layers.Conv2D(128, 4, strides=2)(x)
x = layers.LeakyReLU()(x)
x = layers.Conv2D(128, 4, strides=2)(x)
x = layers.LeakyReLU()(x)
x = layers.Conv2D(128, 4, strides=2)(x)
x = layers.LeakyReLU()(x)
x = layers.Flatten()(x)

x = layers.Dropout(0.4)(x) # 一个 dropout 层, 这是很重要的技巧
x = layers.Dense(1, activation='sigmoid')(x) # 判别层

discriminator = keras.models.Model(discriminator_input, x)

# 将判别器模型实例化, 它将形状为 (32, 32, 3)的输入转换为一个二进制分类决策 (真 / 假)
discriminator_optimizer = keras.optimizers.RMSprop(lr=0.0008, clipvalue=1.0, decay=1e-8)
# clipvalue=1.0: 在优化器中使用梯度裁剪 (限制梯度值的范围)
# decay=1e-8: 为了稳定训练过程, 使用学习率衰减
discriminator.compile(optimizer=discriminator_optimizer, loss='binary_crossentropy')

```

6.3 训练技巧

- ① 输入规范化到 (-1, 1) 之间, 最后一层的激活函数使用 tanh (BEGAN 除外)

- ② 使用 wasserstein GAN 的损失函数
- ③ 如果有标签数据的话，尽量使用标签，也有人提出使用反转标签效果很好，另外使用标签平滑，单边标签平滑或者双边标签平滑
- ④ 使用 mini-batch norm， 如果不用 batch norm 可以使用 instance norm 或者 weight norm
- ⑤ 避免使用 RELU 和 pooling 层，减少稀疏梯度的可能性，可以使用 leakrelu 激活函数
- ⑥ 优化器尽量选择 ADAM，学习率不要设置太大，初始 $1e-4$ 可以参考，另外可以随着训练进行不断缩小学习率
- ⑦ 给 D 的网络层增加高斯噪声，相当于是一种正则

7. 超参数调整

7.1 神经网络中包含哪些超参数？

通常可以将超参数分为三类：网络参数、优化参数、正则化参数。

网络参数：可指网络层与层之间的交互方式（相加、相乘或者串接等）、卷积核数量和卷积核尺寸、网络层数（也称深度）和激活函数等。

优化参数：一般指学习率、批样本数量（batch size）、不同优化器的参数以及部分损失函数的可调参数。

正则化：权重衰减系数，丢弃比率（dropout）

7.2 为什么要进行超参数调优？

本质上，这是模型优化寻找最优解和正则项之间的关系。网络模型优化调整的目的是为了寻找到全局最优解（或者相比更好的局部最优解），而正则项又希望模型尽量拟合到最优。两者通常情况下，存在一定的对立，但两者的目标是一致的，即最小化期望风险。模型优化希望最小化经验风险，而容易陷入过拟合，正则项用来约束模型复杂度。所以如何平衡两者之间的关系，得到最优或者较优的解就是超参数调整优化的目的。

7.3 超参数的重要性顺序

首先，**学习率，损失函数上的可调参数**。在网络参数、优化参数、正则化参数中最重要的超参数可能就是学习率了。学习率直接控制着训练中网络梯度更新的量级，直接影响着模型的有效容限能力；损失函数上的可调参数，这些参数通常情况下需要结合实际损失函数来调整，大部分情况下这些参数也能很直接的影响到模型的有效容限能力。这些损失一般可分成三类，第一类辅助损失结合常见的损失函数，起到辅助优化特征表达的作用。例如度量学习中的Center loss，通常结合交叉熵损失伴随一个权重完成一些特定的任务。这种情况一般建议辅助损失值不高于或者不低于交叉熵损失值的两个数量级；第二类，多任务模型的多个损失函数，每个损失函数之间或独立或相关，用于各自任务，这种情况取决于任务之间本身的相关性，目前笔者并没有一个普适的经验由于提供参考；第三类，独立损失函数，这类损失通常会在特定的任务有显著性的效果。例如RetinaNet中的focal loss，其中的参数 ν, α ，对最终的效果会产生较大的影响。这类损失通常论文中会给出特定的建议值。

其次，**批样本数量，动量优化器 (Gradient Descent with Momentum) 的动量参数 β** 。批样本决定了数量梯度下降的方向。过小的批数量，极端情况下，例如batch size为1，即每个样本都去修正一次梯度方向，样本之间的差异越大越难以收敛。若网络中存在批归一化 (batchnorm)，batch size过小则更难以收敛，甚至垮掉。这是因为数据样本越少，统计量越不具有代表性，噪声也相应的增加。而过大的batch size，会使得梯度方向基本稳定，容易陷入局部最优解，降低精度。一般参考范围会取在[1:1024]之间，当然这个不是绝对的，需要结合具体场景和样本情况；动量衰减参数 β 是计算梯度的指数加权平均数，并利用该值来更新参数，设置为 0.9 是一个常见且效果不错的选择；

最后，**Adam优化器的超参数、权重衰减系数、丢弃法比率 (dropout) 和网络参数**。在这里说明下，这些参数重要性放在最后并不等价于这些参数不重要。而是表示这些参数在大部分实践中不建议过多尝试，例如Adam优化器中的 $\beta_1, \beta_2, \epsilon$ ，常设为 0.9、0.999、 10^{-8} 就会有不错的表现。权重衰减系数通常会有个建议值，例如0.0005，使用建议值即可，不必过多尝试。dropout通常会在全连接层之间使用防止过拟合，建议比率控制在[0.2, 0.5]之间。使用dropout时需要特别注意两点：一、在RNN中，如果直接放在memory cell中，循环会放大噪声，扰乱学习。一般会建议放在输入和输出层；二、不建议dropout后直接跟上batchnorm，dropout很可能影响batchnorm计算统计量，导致方差偏移，这种情况下会使得推理阶段出现模型完全垮掉的极端情况；网络参数通常也属于超参数的范围内，通常情况下增加网络层数能增加模型的容限能力，但模型真正有效的容限能力还和样本数量和质量、层之间的关系等有关，所以一般情况下会选择先固定网络层数，调优到一定阶段或者有大量的硬件资源支持可以在网络深度上进行进一步调整。

7.4 极端批样本数量下，如何训练网络？

极端批样本情况一般是指 batch size 为 1 或者 batch size 在 6000 以上的情况。这两种情况，在使用不合理的情况下都会导致模型最终性能无法达到最优甚至是崩溃的情况。

在目标检测、分割或者 3D 图像等输入图像尺寸较大的场景，通常 batch size 会非常小。batch size 过小通常对 batchnorm 的影响是最大的，若网络模型中存在 batchnorm，batch size 若只为 1 或者 2 时会对训练结果产生非常大的影响。这时通常有两种策略：

一、若模型使用了预训练网络，可冻结预训练网络中 batchnorm 的模型参数，有效降低 batch size 引起的统计量变化的影响。

二、在网络不是过深或者过于复杂时可直接移除 batchnorm 或者使用 groupnorm 代替 batchnorm，前者不多阐释，后者是有 FAIR 提出的一种用于减少 batch 对 batchnorm 影响，其主要策略是先将特征在通道上进行分组，然后在组内进行归一化。即归一化操作上完全与 batch size 无关。这种 groupnorm 的策略被证实能在极小批量网络训练上能达到较优秀的性能。当然这里也引入里 group 这个超参数，一般情况下建议不宜取 group 为 1 或者各通道单独为组的 group 数量，可结合实际网络稍加调试。

为了降低训练时间的成本，多机多卡的分布式系统通常会使用超大的 batch size 进行网络训练。可采用层自适应速率缩放（LARS）算法。从理论认知上将，batch size 增大会减少反向传播的梯度更新次数，但为了达到相同的模型效果，需要增大学习率。但学习率一旦增大，又会引起模型的不收敛。为了解决这一矛盾，LARS 算法就在各层上自适应的计算一个本地学习率用于更新本层的参数，这样能有效的提升训练的稳定性。

7.5 合理使用预训练网络

7.5.1 什么是微调（fine-tune）

指稍微调整参数即可得到优秀的性能，是迁移学习的一种实现方式。微调和从头训练的本质区别在于模型参数的初始化，train from scratch 通常指对网络各类参数进行随机初始化（当然随机初始化也存在一定技巧），随机初始化模型通常不具有任何预测能力，通常需要大量的数据或者特定域的数据进行从零开始的训练，这样需要训练到优秀的模型通常是稍困难的。而微调的网络，网络各类参数已经在其他数据集（例如 ImageNet 数据集）完成较好调整的，具备了较优秀的表达能力。因此，我们只需要以较小的学习速率在自己所需的数据集领域进行学习即可得到较为优秀的模型。微调通常情况下，无须再重新设计网络结构，预训练模型提供了优秀的结构，只需稍微修改部分层即可。在小数据集上，通常微调的效果比从头训练要好很多，原因在于数据量较小的前提下，训练更多参数容易导致过度拟合。

7.5.2 微调有哪些不同方法？

以图像分类为例，通常情况下由于不同数据集需要的类别数不同，我们需要修改网络的输出顶层。这种情况下有两种微调方式：

不冻结网络模型的任何层，对最后的改动层使用较大的学习率，对未改动层以较小的学习率进行训练全模型训练，进行多轮训练即可。即一步完成训练。

冻结除了顶部改动层以外的所有层参数，即不对冻结部分的层进行参数训练更新，进行若干轮的微调训练后，放开顶部层以下的若干层或者全部放开所有层的参数，再次进行若干轮训练即可。即分多步训练。以上两种都属于微调。目前由于存在大量优秀的预训练模型，如何确定哪个模型适合自己的任务并能得到最佳性能需要花大量的时间探索。此时，上述的前者是种不错训练方式，你无须进行过多分步的操作。而当探索到一个比较适合模型时，你不妨可以再次重新尝试下以第二种方式进行训练，或许能得到相比于前者稍高些的性能，因为小数据集上调整过多的参数过拟合的机率也会增大，当然这并不是绝对的。

7.5.3 微调先冻结底层，训练顶层的原因？

首先冻结除了顶部改动层以外的所有层参数，对顶层进行训练，这个过程可以理解为顶层的域适应训练，主要用来训练适应模型的现有特征空间，防止顶层糟糕的初始化，对已经具备一定表达能力的层的干扰和破坏，影响最终的性能。之后，在很多深度学习框架教程中会使用放开顶层往下一半的层数，继续进行微调。这样的好处在于越底层的特征通常是越通用的特征，越往上其整体的高层次语义越完备，这通过感受野很容易理解。所以，若预训练模型的数据和微调训练的数据语义差异越大（例如 ImageNet 的

预模型用于医学图像的训练），那越往顶层的特征语义差异就越大，因此通常也需要进行相应的调整。

7.5.4 不同的数据集特性下如何微调？

数据集数据量少，数据和原数据集类似。这是通常做法只需修改最后的输出层，训练即可，训练过多参数容易过拟合。

数据集数据量少，数据和原数据集差异较大。由于数据差异较大，可以在完成输出顶层的微调后，微调顶层往下一半的层数，进行微调。

数据集数据量大，数据与原数据集差异较大。这种情况下，通常已经不需要用预训练模型进行微调，通常直接重新训练即可。

数据集数据量大，数据与原数据类似。这时预训练模型的参数是个很好的初始化，可利用预训练模型放开所有层以较小的学习率微调即可。

7.5.5 目标检测中使用预训练模型的优劣？

目标检测中无论是一阶段的 YOLO、SSD 或者 RetinaNet 还是二阶段的 Faster R-CNN、R-FCN 和 FPN 都是基于 ImageNet 上预训练好的分类模型。

优势在于：

- 1、正如大部分微调的情况一样，使用预训练网络已拥有优秀的语义特征，能有效的加快训练速度；
- 2、其次，对于大部分二阶段的模型来说，并未实现严格意义上的完全端对端的训练，所以使用预训练模型能直接提取到语义特征，能使两个阶段的网络更容易实现模型的优化。

劣势在于，分类模型和检测模型之间仍然存在一定任务上的差异：

- 1、分类模型大部分训练于单目标数据，对同时进行多目标的捕捉能力稍弱，且不关注目标的位置，在一定程度上让模型损失部分空间信息，这对检测模型通常是不利的；
- 2、域适应问题，若预训练模型（ImageNet）和实际检测器的使用场景（医学图像，卫星图像）差异较大时，性能会受到影响；
- 3、使用预训练模型就意味着难以自由改变网络结构和参数限制了应用场合。

7.5.6 目标检测中如何从零开始训练(train from scratch)？

- 1、数据集不大时，同样需要进行数据集增强。
- 2、预训练模型拥有更好的初始化，train from scratch 需要更多的迭代次数以及时间训练和优化检测器。而二阶段模型由于并不是严格的端对端训练，此时可能需要更多的迭代次数以及时间，而一阶段检测模型训练会相对更容易些。
- 3、目标检测中 train from scratch 最大的问题还是 batch size 过小。所以可采取的策略是增加 GPU 使用异步 batchnorm 增大 batch size，若条件限制无法使用更多 GPU 时，可使用 groupnorm 代替 batchnorm
- 4、由于分类模型存在对多目标的捕捉能力弱以及对物体空间位置信息不敏感等问题，可借鉴 DetNet 训练一个专属于目标检测的模型网络，增强对多目标、尺度和位置拥有更强的适应性。

7.6 自动化超参数搜索方法有哪些？

主要包含网格搜索，随机搜索，基于模型的超参优化

7.6.1 网格搜索：

通常当超参数量较少的时候，可以使用网格搜索法。即列出每个超参数的大致候选集合。利用这些集合 进行逐项组合优化。在条件允许的情况下，重复进行网格搜索会当优秀，当然每次重复需要根据上一步得到的最优参数组合，进行进一步的细粒度的调整。网格搜索最大的问题就在于计算时间会随着超参数的数量指数级的增长。

7.6.2 随机搜索：

随机搜索，是一种用来替代网格搜索的搜索方式。随机搜索有别于网格搜索的一点在于，不需要设定一个离散的超参数集合，而是对每个超参数定义一个分布函数来生成随机超参数。随机搜索相比于网

格搜索在一些不敏感超参上拥有明显优势。例如网格搜索对于批样本数量（batch size），在[16,32,64]这些范围内进行逐项调试，这样的调试显然收益更低下。当然随机搜索也可以进行细粒度范围内的重复的搜索优化。

7.6.3 基于模型的超参优化：

调优问题转化为了优化问题。直觉上会考虑是否进行一个可导建模，然后利用梯度下降进行优化。但不幸的是我们的超参数通常情况下是离散的，而且其计算代价依旧很高。

基于模型的搜索算法，最常见的就是贝叶斯超参优化。有别于的网格搜索和随机搜索独立于前几次搜索结果的搜索，贝叶斯则是利用历史的搜索结果进行优化搜索。其主要有四部分组成，

1. 目标函数，大部分情况下就是模型验证集上的损失。
- 2、搜索空间，即各类待搜索的超参数。
- 3、优化策略，建立的概率模型和选择超参数的方式。
- 4、历史的搜索结果。首先对搜索空间进行一个先验性的假设猜想，即假设一种选择超参的方式，然后不断的优化更新概率模型，最终的目标是找到验证集上误差最小的一组超参数。