法律声明

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

关注 小象学院

第6讲　深度学习基础

计算机视觉

主讲人：叶梓

上海交通大学博士
主要研究方向：机器学习、深度学习、人工智能

本章内容

- 深度学习基础
  - 与传统神经网络的区别
  - 目标函数
  - 改进的梯度下降
  - 避免过适应
- CNN初步介绍
  - CNN的基本组件
  - 卷积层的误差反向传播
  - 池化层的误差反向传播

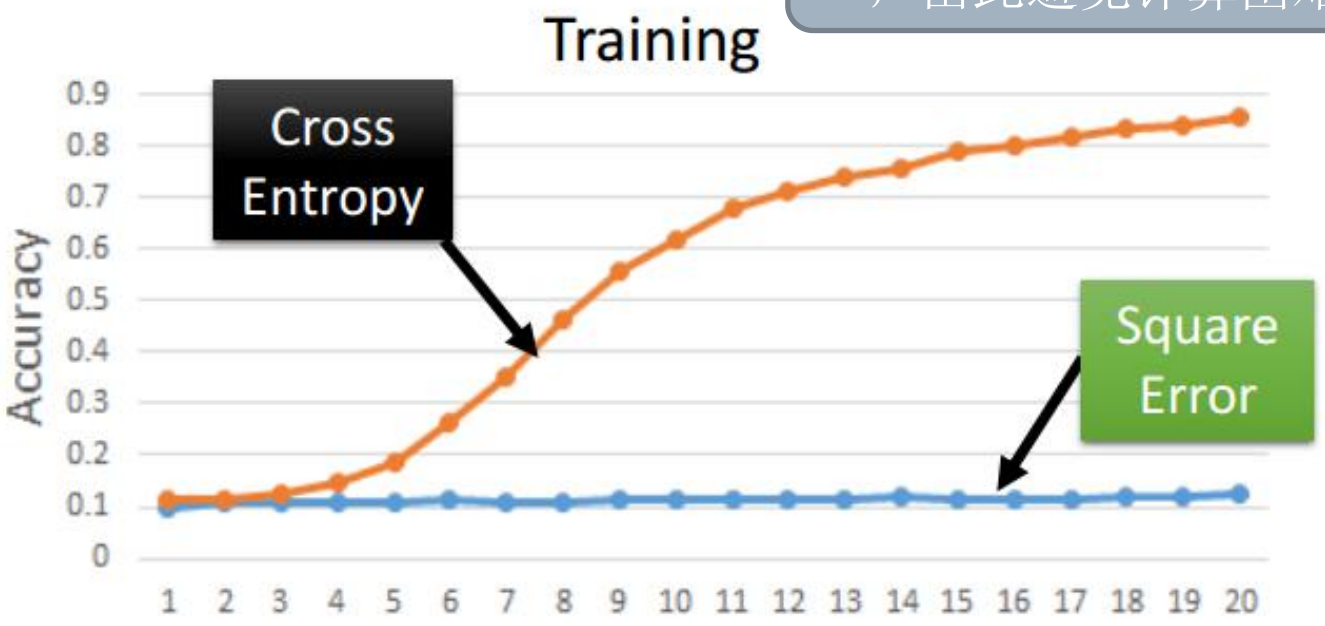
深度学习与神经网络的区别

- 网络架构
  - 3层以内→可达上千层
- 层间连接
  - 通常是全连接→形式多样：共享权值、跨层的反馈等
- 目标函数
  - MSE→CE等
- 激活函数
  - Sigmoid等→ReLU等
- 梯度下降的方法
  - GD等→Adam等
- 避免过适应
  - 凭经验→Dropout等

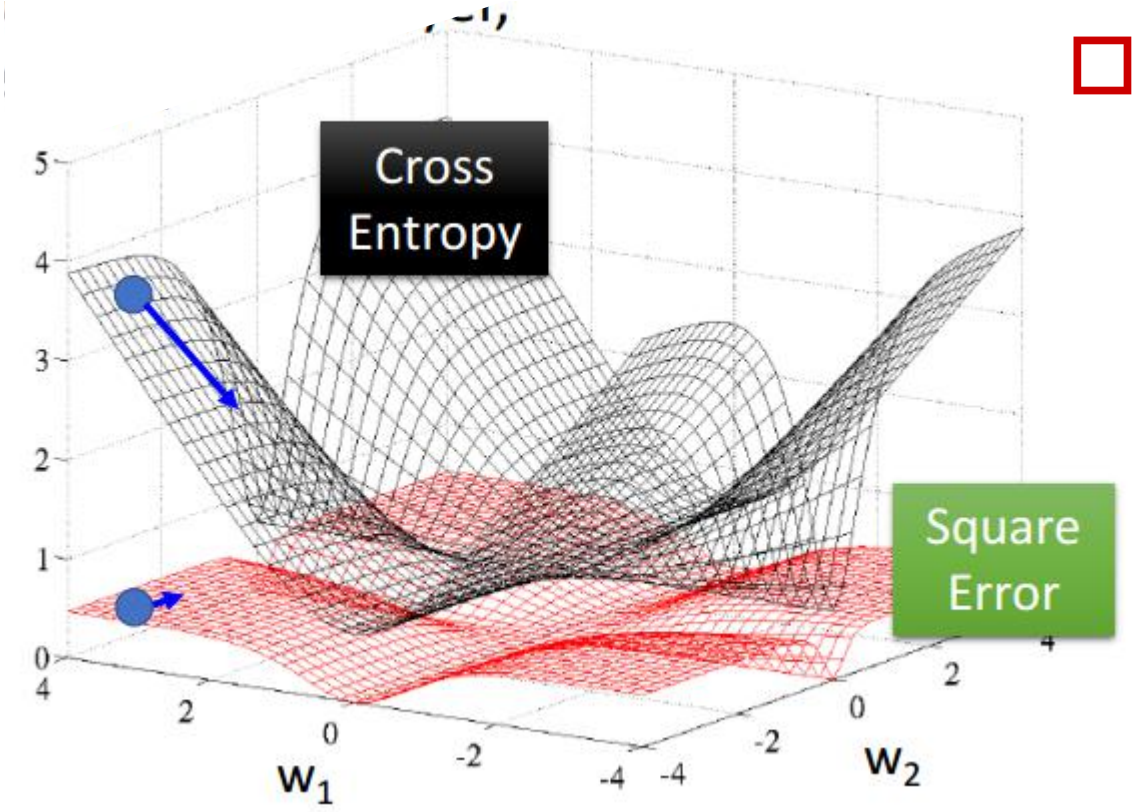
选择合适的目标函数

- 均方误差：$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = 0$
- 交叉熵：$-\sum_{i=1}^{n}\hat{y}_i \ln y_i = 0$

计算出的y值不可能为0，由此避免计算困难



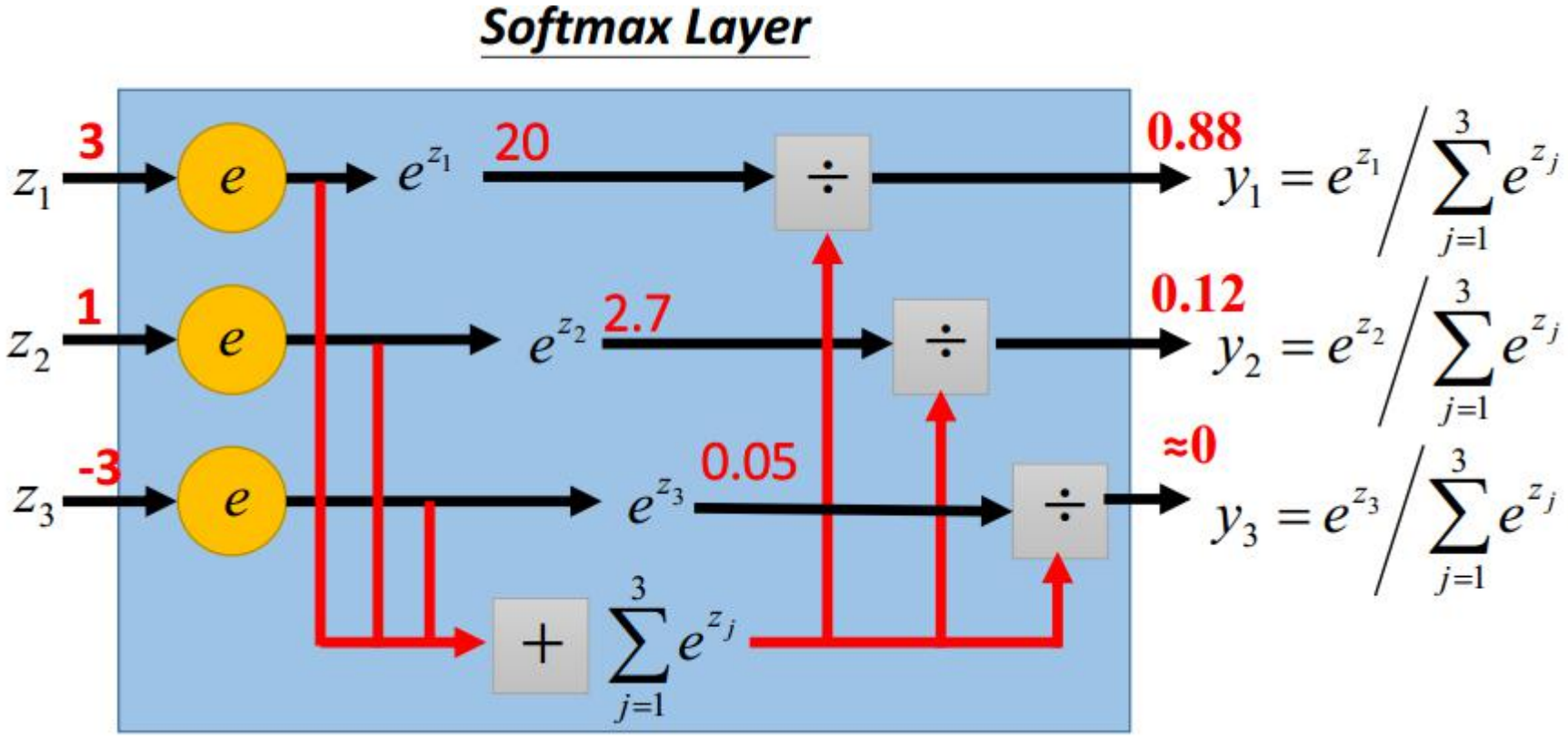
交叉熵与均方误差



- 可以想象为交叉熵目标函数的最优值搜索空间的“地形”更“陡”，更有利于快速的找到最优值。

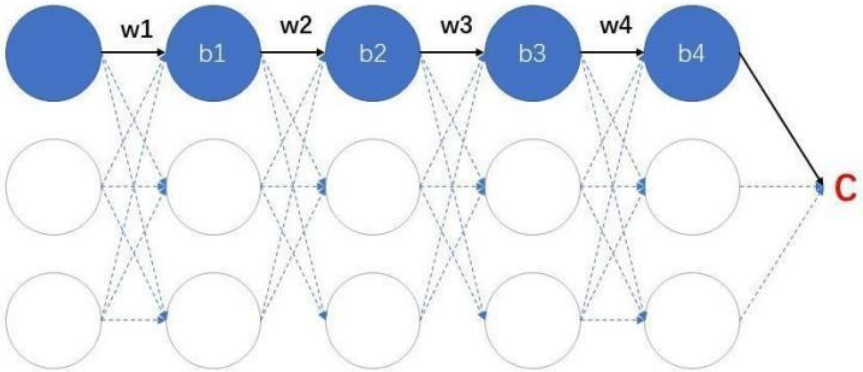
Softmax层

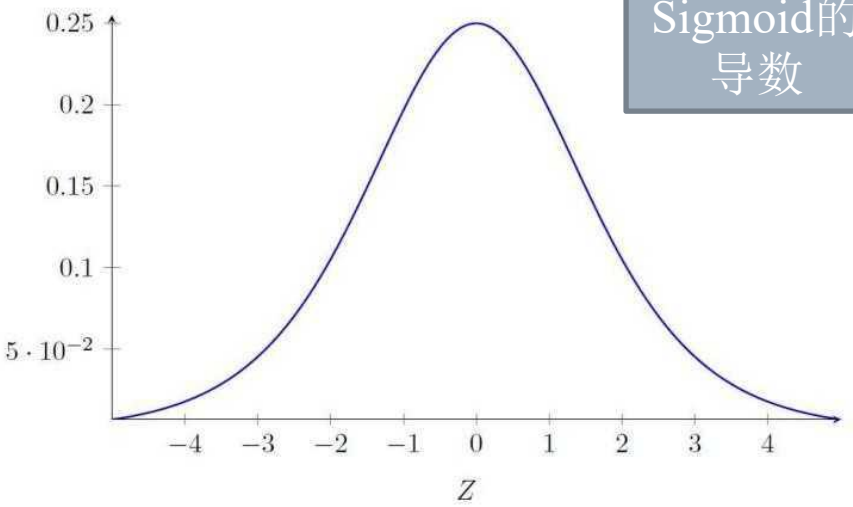
- Softmax层的作用是突出"最大值"并转换成概率的形式。



梯度消失的直观解释

- 神经元的激活函数采用Sigmoid函数，
- 则大部分情况下$|w|<1$，而Sigmoid的导数$\sigma' < 1/4$。





Sigmoid的导数

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial a_4} \underbrace{\sigma(z_4)'}_{<\frac{1}{4}} w_4 \underbrace{\sigma(z_3)'}_{<\frac{1}{4}} w_3 \underbrace{\sigma(z_2)'}_{<\frac{1}{4}} w_2 \sigma(z_1)'$$

$$\frac{\partial C}{\partial b_2} = \frac{\partial C}{\partial a_4} \underbrace{\sigma(z_4)'}_{<\frac{1}{4}} w_4 \underbrace{\sigma(z_3)'}_{<\frac{1}{4}} w_3 \sigma(z_2)'$$

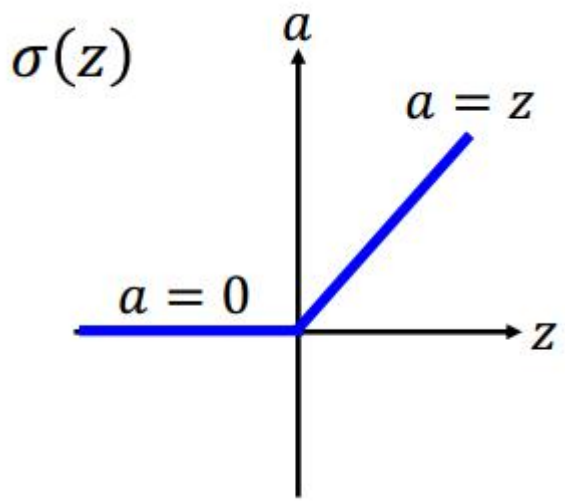
$$\frac{\partial C}{\partial b_3} = \frac{\partial C}{\partial a_4} \underbrace{\sigma(z_4)'}_{<\frac{1}{4}} w_4 \sigma(z_3)'$$

$$\frac{\partial C}{\partial b_4} = \frac{\partial C}{\partial a_4} \sigma(z_4)'$$

小象学院 ChinaHadoop.cn

激活函数

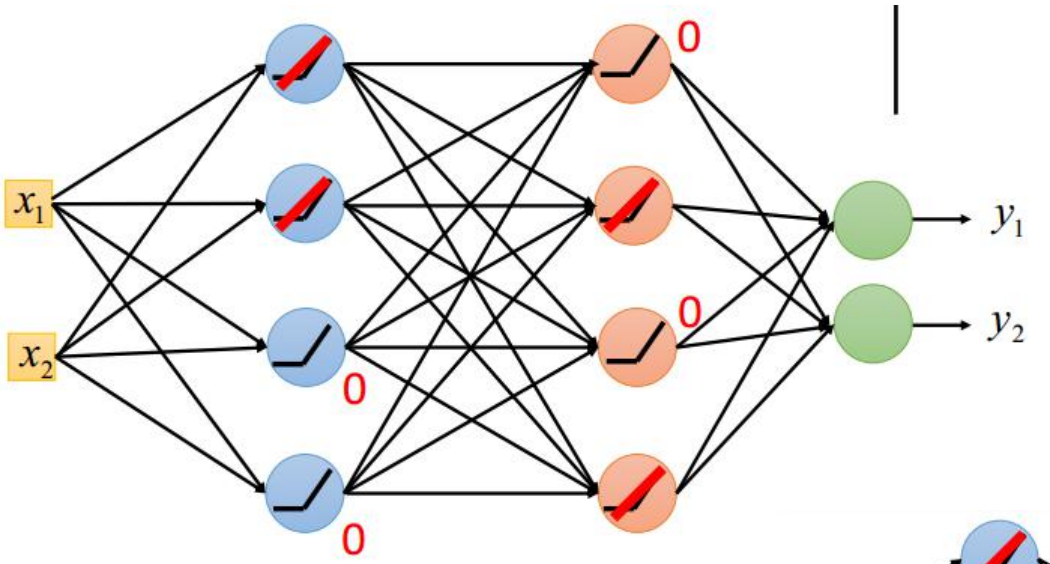
- 2006年，人们(Hinton等)用RBM预训练的方式解决梯度消失的问题；
- 到了2015年，人们用ReLU激活函数。



- 计算速度快
- 有生物学依据
- 更重要的是：

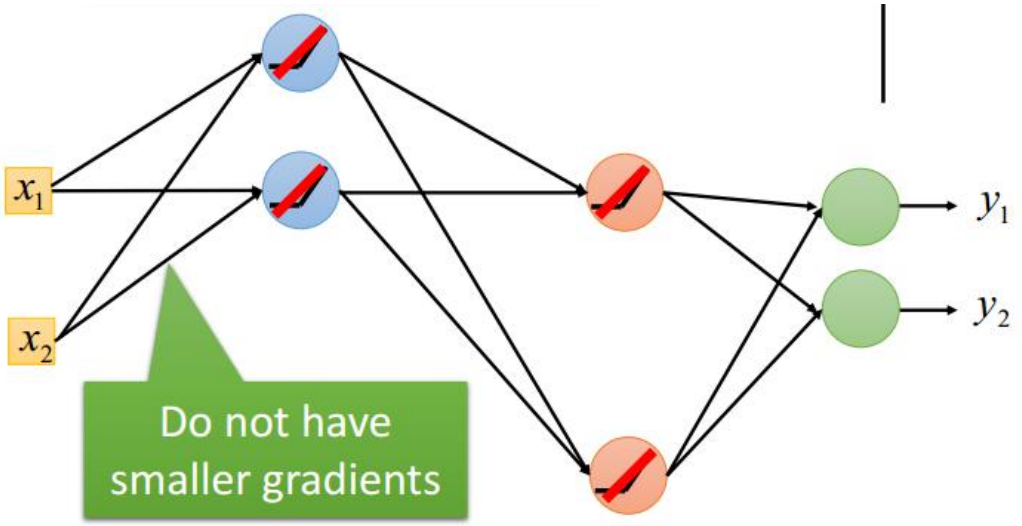
  能解决梯度消失问题

采用ReLU激活函数后



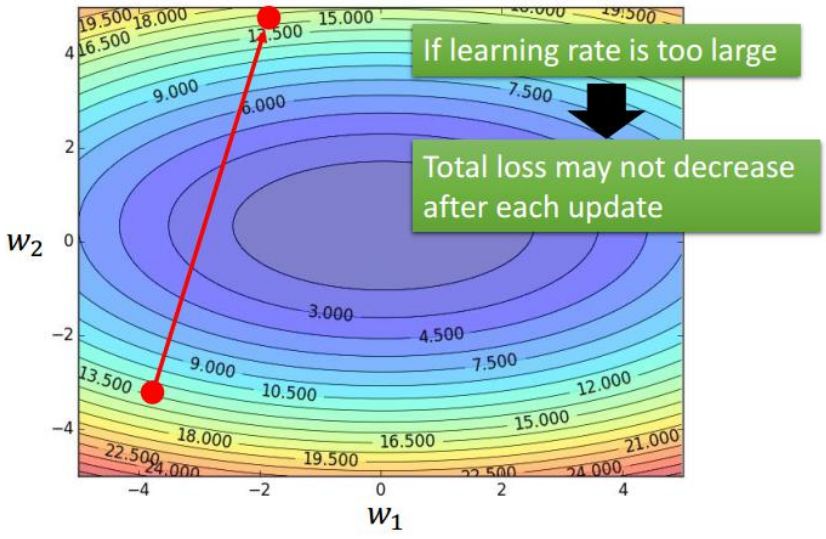
- 一些单元的输出成了0，而另一些则成了完全线性的单元。

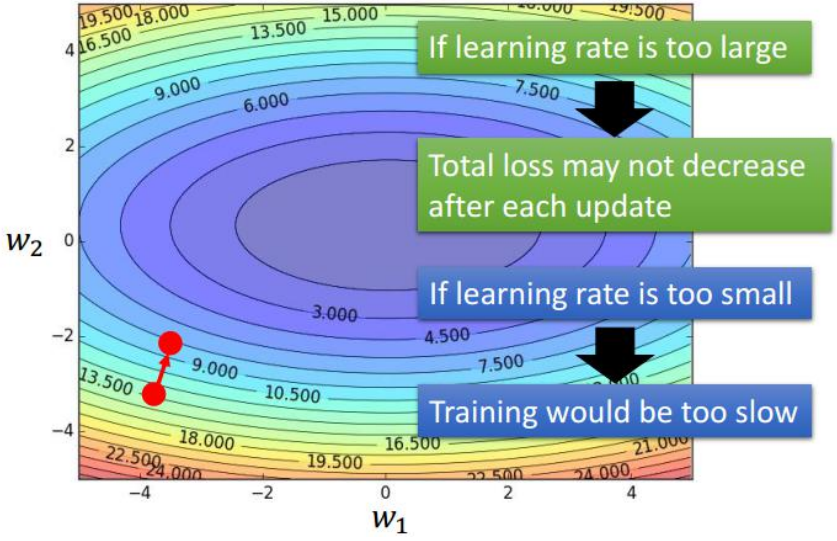
- 这也避免了调整各层权重时的"梯度消失"的问题。



学习步长

- 学习步长的设置是个难题：若学习步长过大，则目标函数可能不降低；但若学习步长过小，则训练过程可能非常缓慢。
- 解决之道：训练几轮后就按一些因素调整学习步长。
- 例如：$\eta^{(t)} = \eta / \sqrt{t+1}$





SGD的问题

- learning rate不易确定，如果选择的太小，收敛速度会很慢；如果太大，loss function 就会在极小值处不停地震荡甚至偏离。
- 每个参数的 learning rate 都是相同的，如果数据是稀疏的，则希望对出现频率低的特征进行大一点的更新。
- 深层神经网络之所以比较难训练，并不是因为容易进入局部最小，而是因为学习过程容易陷入到马鞍面中，在这种区域中，所有方向的梯度值都几乎是 0。

Momentum（动量）

- Momentum借用了物理中的动量概念，即，前几次的梯度也会参与运算。
- 为了表示动量，引入了一个新的变量v。v是之前的梯度的累加，但每回合都有一定的衰减。
- 前后梯度方向一致时，能够加速学习；前后梯度方向不一致时,能够抑制震荡。

$$\hat{g} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta), y_i)$$

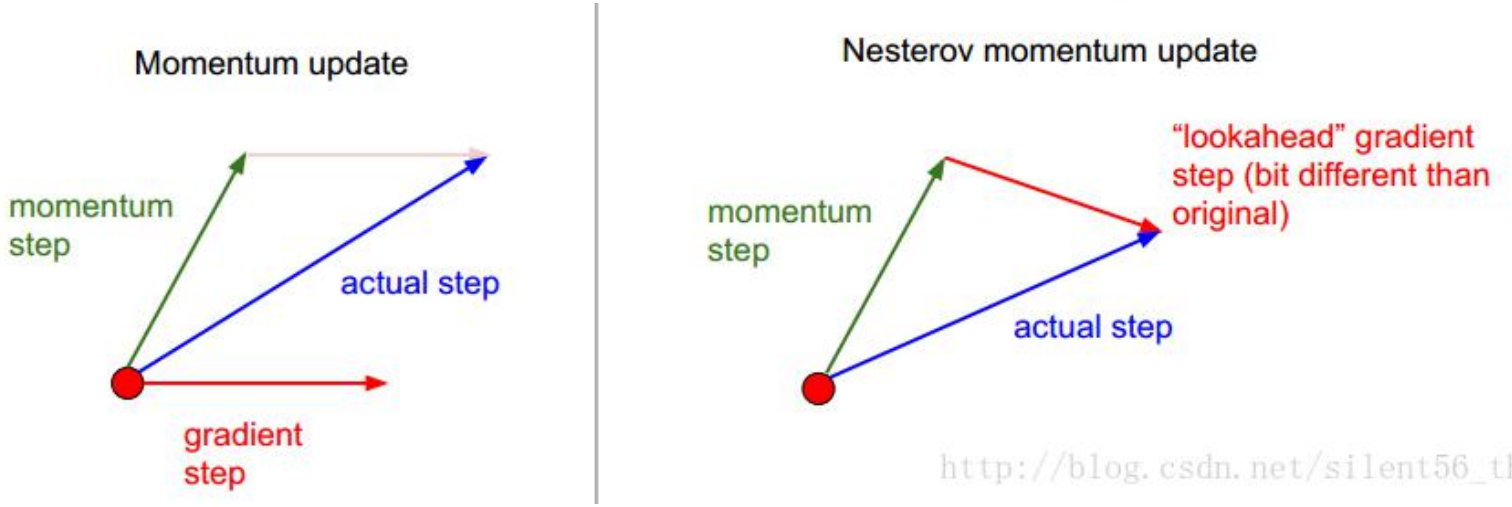
$$v \leftarrow \alpha v - \epsilon \hat{g}$$

$$\theta \leftarrow \theta + v$$

Nesterov Momentum

- 对Momentum的一种改进：先对参数进行估计，然后使用估计后的参数来计算误差。

$$\hat{g} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta+\alpha v), y_i)$$
$$v \leftarrow \alpha v - \epsilon \hat{g}$$
$$\theta \leftarrow \theta + v$$



http://blog.csdn.net/silent56_th

Adagrad

- 应该为不同的参数设置不同的学习步长。

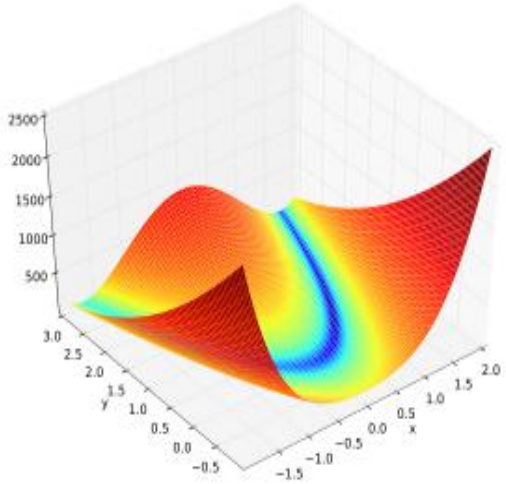
$$\hat{g} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta), y_i)$$

$$r \leftarrow r + \hat{g} \odot \hat{g}$$

累加梯度的平方，
注意：永远是递增。

$$\triangle\theta = -\frac{\epsilon}{\delta + \sqrt{r}} \odot \hat{g}$$

$$\theta \leftarrow \theta + \triangle\theta$$



- 梯度越小，则学习步长越大，反之亦然。
- 想象一下：在缓坡上，可以大步地往下跑；而且陡坡上，只能小步地往下挪。

RMSprop

- RMSprop是一种改进的Adagrad，通过引入一个衰减系数，让r每回合都衰减一定比例。
- 这种方法很好的解决了Adagrad过早结束的问题，适合处理非平稳目标，对于RNN效果很好。

$$\hat{g} \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta), y_i)$$

$$r \leftarrow \rho r + (1-\rho)\hat{g} \odot \hat{g}$$

$$\triangle\theta = -\frac{\epsilon}{\delta + \sqrt{r}} \odot \hat{g}$$

$$\theta \leftarrow \theta + \triangle\theta$$

这里未必是递增，通过参数来协调当前和过往。

Adam

- Adam 这个名字来源于 adaptive moment estimation，自适应矩估计。
- Adam本质上是带有动量项的RMSprop，它利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。

$$g \leftarrow +\frac{1}{m}\nabla_\theta \sum_i L(f(x_i;\theta), y_i)$$

$$s \leftarrow \rho_1 s + (1-\rho_1) g$$

一阶矩估计

$$r \leftarrow \rho_2 r + (1-\rho_2) g \odot g$$

二阶矩估计

$$\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$$

0.9

$$\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$$

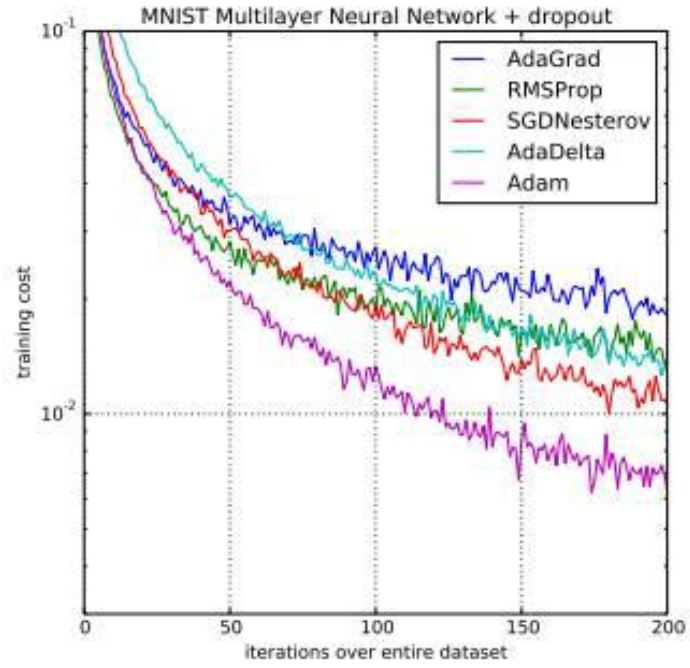
0.999

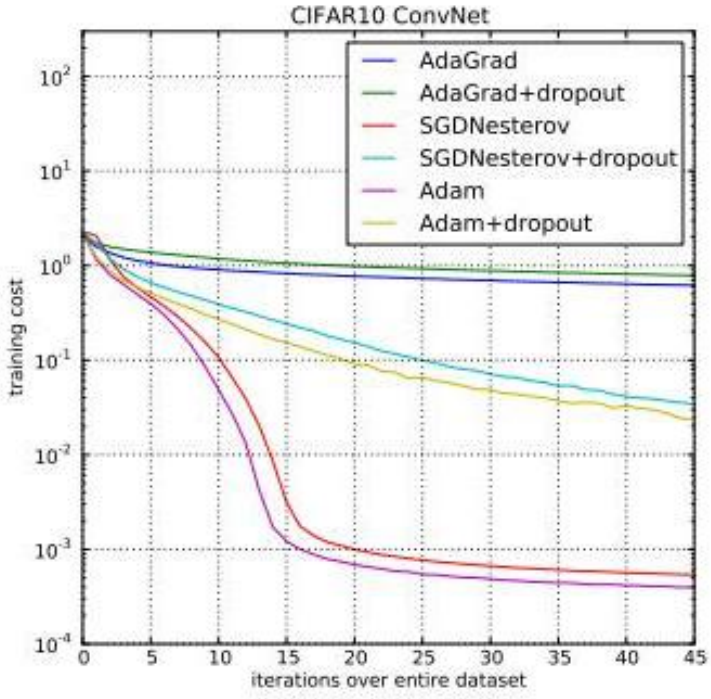
$$\Delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$$

$$\theta \leftarrow \theta + \Delta\theta$$

各种梯度下降算法的比较

- 左：在 MNIST 上训练多层神经网络
- 右：在CIFAR10上训练卷积神经网络





各种梯度下降算法的比较

- 如果数据是稀疏的，就用自适用方法，即Adagrad, Adadelta, RMSprop, Adam。
- RMSprop, Adadelta, Adam 在很多情况下的效果是相似的。通常，Adam 是最好的选择。
- 很多论文里都会用 SGD，没有 momentum 等。SGD 虽然也能达到极小值，但是比其它算法用的时间长，而且可能会被困在鞍点。
- SGD相对稳定——据说老司机都是开手动挡的。

关于算法选择的建议

- **1.首先，各大算法孰优孰劣并无定论。**如果是刚入门，优先考虑 SGD+Nesterov Momentum或者Adam.
- **2. 选择熟悉的算法**——这样可以更加熟练地利用经验进行调参。
- **3.充分了解数据**——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。
- **4. 根据需求来选择**——在模型设计实验过程中，要快速验证新模型的效果，可以先用Adam；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。
- **5. 先用小数据集进行实验。**有论文研究指出，随机梯度下降算法的收敛速度和数据集的大小的关系不大。因此可以先用一个具有代表性的小数据集进行实验。

关于算法选择的建议

- **6. 考虑不同算法的组合。**先用Adam进行快速下降，而后再换到SGD进行充分的调优。
- **7. 数据集一定要充分的打散（shuffle）。**这样在使用自适应学习率算法的时候，可以避免某些特征集中出现，而导致的有时学习过度、有时学习不足，使得下降方向出现偏差的问题。
- **8. 训练过程中持续监控训练数据和验证数据上的目标函数值以及精度或者AUC等指标的变化情况。**对训练数据的监控是要保证模型进行了充分的训练；对验证数据的监控是为了避免出现过拟合。
- **9. 制定一个合适的学习率衰减策略。**可以使用定期衰减策略，比如每过多少个epoch就衰减一次；或者利用精度或者AUC等性能指标来监控。

Batch Normalization的由来

- 在CNN训练时，绝大多数都采用基于mini-batch的随机梯度下降算法为基础的算法进行训练；
- 随着输入数据的不断变化，及网络中参数不断调整，网络的各层输入数据的分布则会不断变化，
- 则各层在训练的过程中就需要不断的改变以适应这种新的数据分布，从而造成网络训练困难，难以拟合的问题。

Batch Normalization的步骤

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

- 在每次SGD时，通过mini-batch来对相应的activation做规范化操作，使得结果（输出信号各个维度）的均值为0，方差为1.
- 最后的“scale and shift”操作则是为了让因训练所需而“刻意”加入的BN能够有可能还原最初的输入。

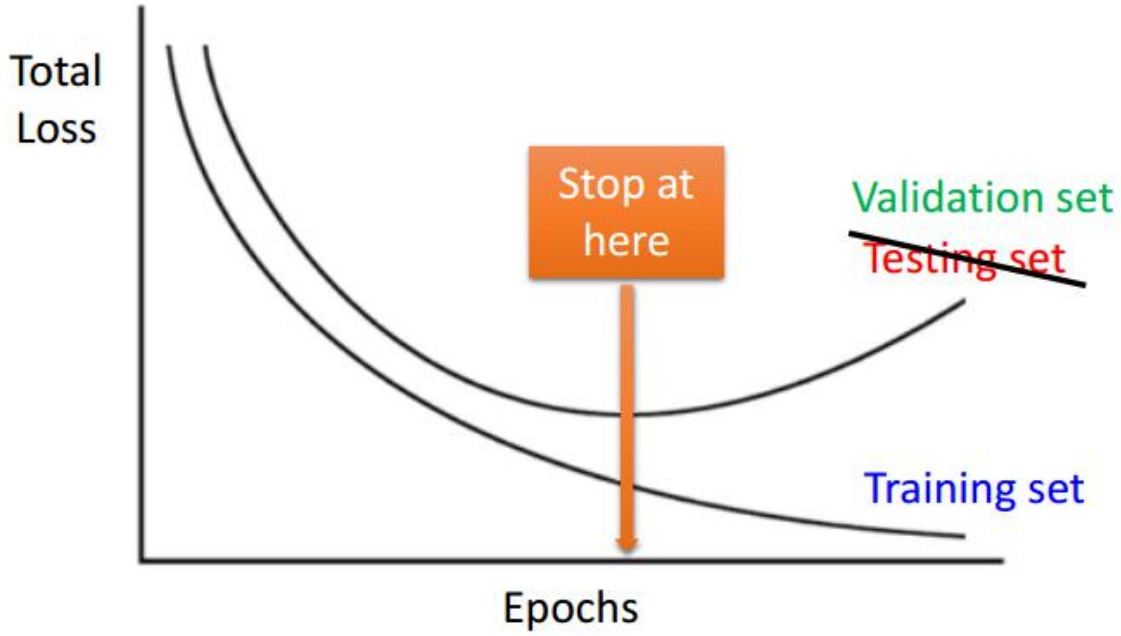
避免过适应

- 只要在训练集图片中稍加一些噪声，学习机就不能做出正确的判断。
- 过适应的根本原因：权重参数太多，而样本量不足。
- 如何避免过适应？
  - 早期停止训练
  - 权重衰减
  - Dropout



早期停止训练

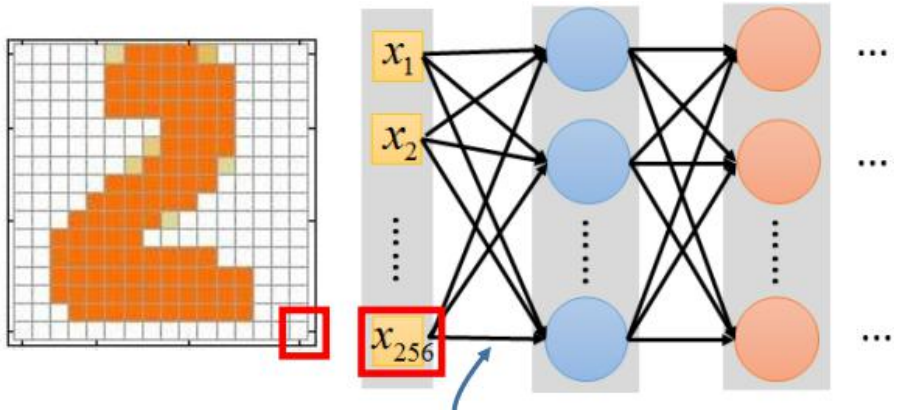
- 当目标函数在“验证集”上不再减小时，训练就应该停止了。
- 不能一味追求“训练集”的误差减小。



权重衰减

- 有些权重是“无用的”。
- 原梯度下降的公式：

$$w \leftarrow w - \eta \frac{\partial L}{\partial w}$$

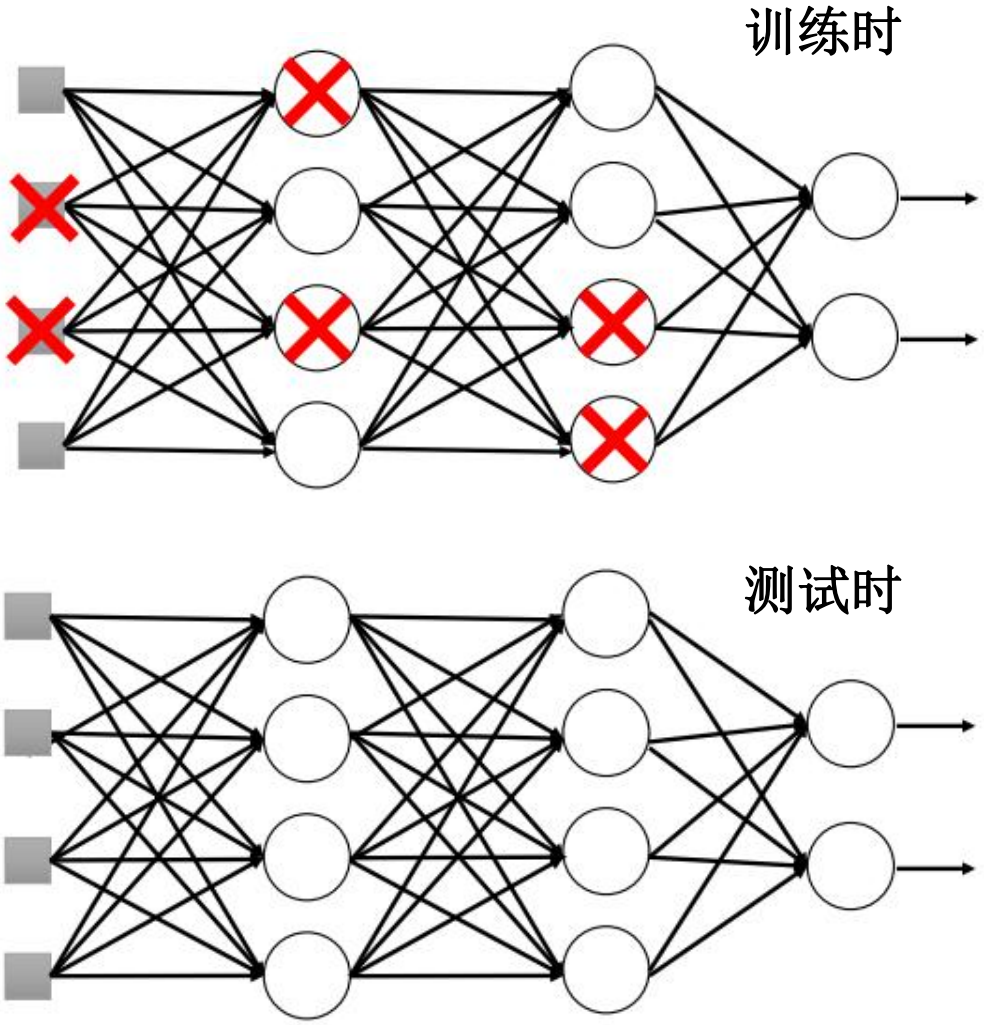


- 经过“权重衰减”后的公式：

$$w \leftarrow \underbrace{(1-\lambda)}_{\text{越来越小}} w - \eta \frac{\partial L}{\partial w}$$

其中，λ为衰减率，例如可为0.01

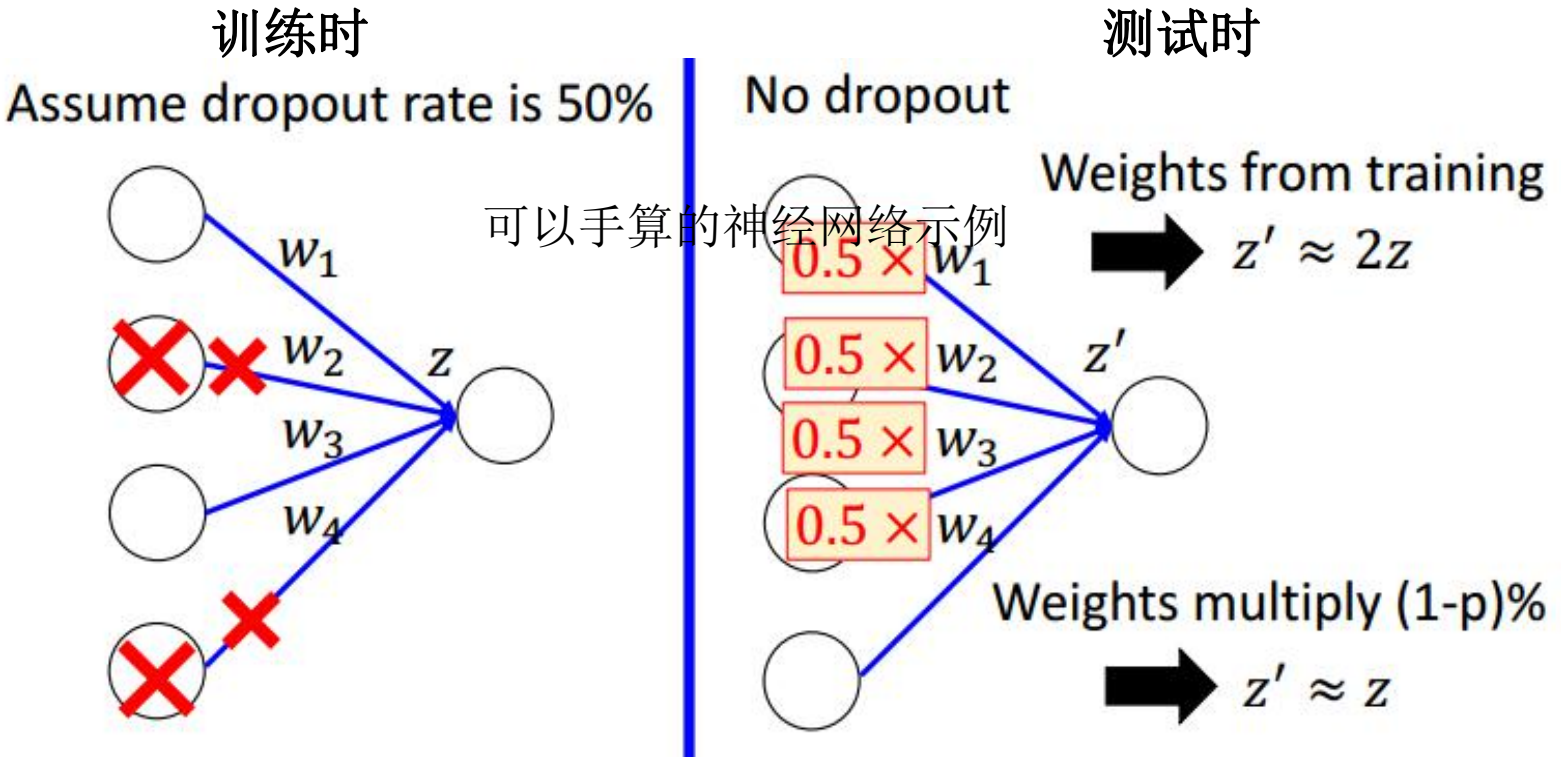
Dropout



- 每次更新参数前，按一定的比例删减部分神经元；
- 删减后整个网络变得“更瘦”；
- 本质上，Dropout就是用一小块数据来训练一系列“子网络”；
- Dropout是“集成学习”的一种。

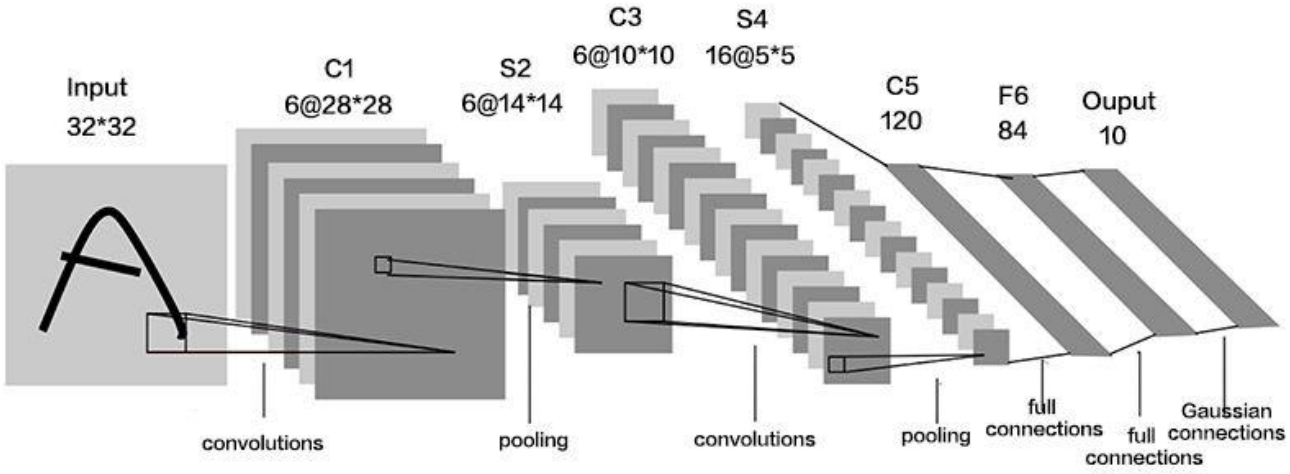
测试时权重应减小

- 测试时，使用所有的神经元，且权重应根据删减比例缩小。



可以手算的神经网络示例

卷积神经网络（CNN）



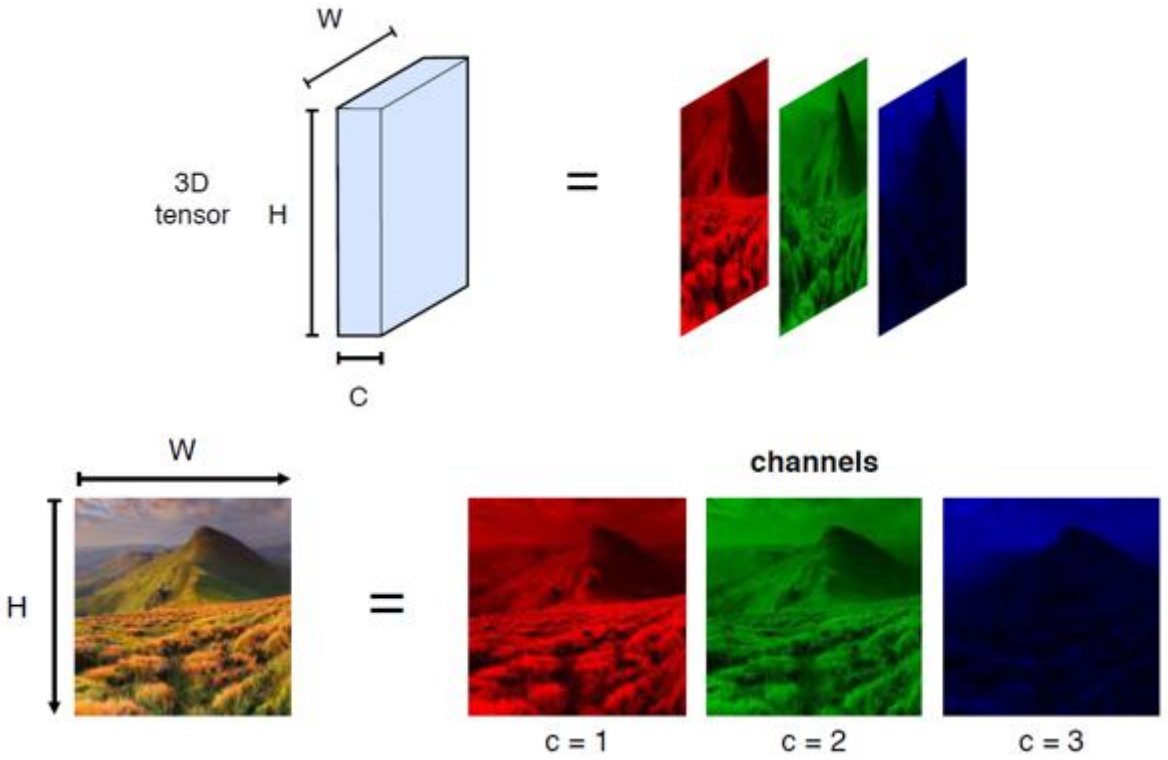
CNN的基本组件

- 卷积层（Convolutional layer）
  - 激活函数（Sigmoid, ReLU, ...）
- 池化层（Pooling layer）
  - 平均池化（Average pooling）
  - 最大化池化（Max pooling）
- 全连接层（Fully-Connected layer）

卷积神经网络（CNN）

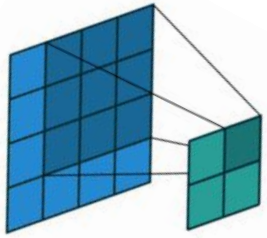
CNN卷积层

- 3通道（RGB）输入图片→3D tensor

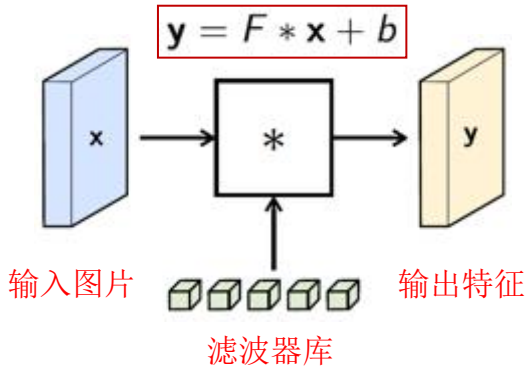


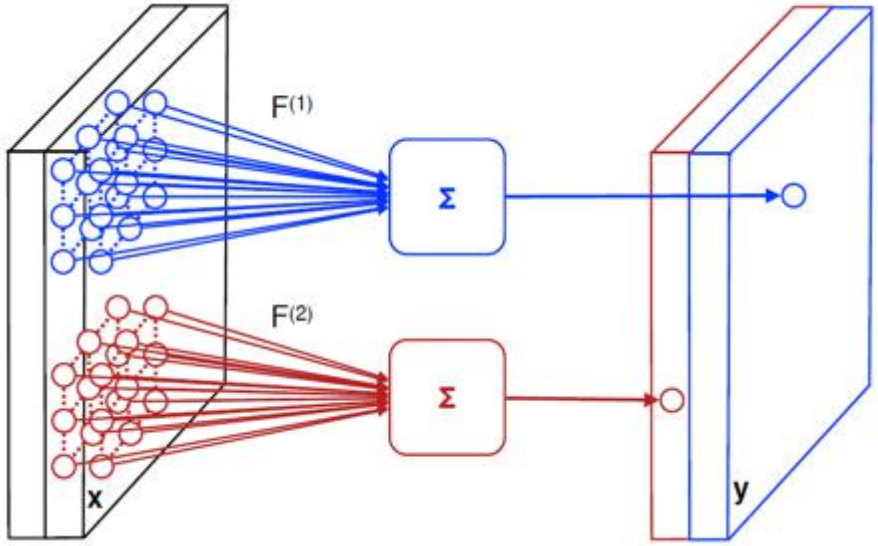
卷积神经网络（CNN）

CNN卷积层



- 3D滤波器/卷积核
  - 以扫描窗的方式，对图像做卷积
  - 每层含有多个核，每个核对应一个输出通道
  - 提取局部特征
  - 权重参数需要学习

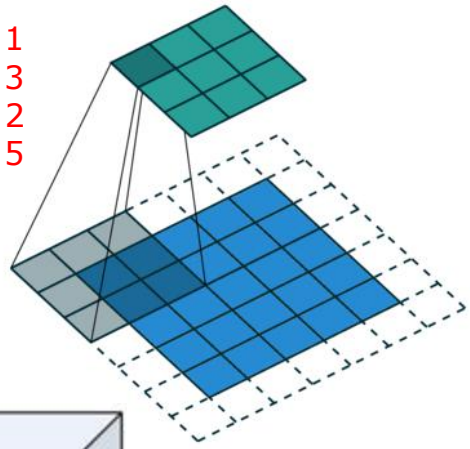




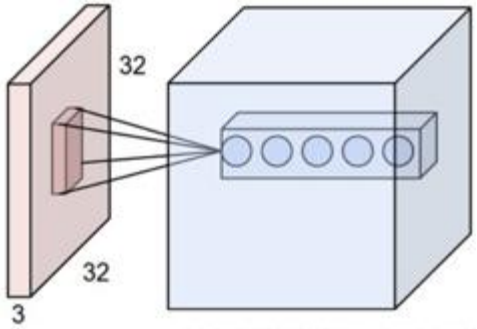
卷积神经网络（CNN）

CNN卷积层

零填充：1
核尺寸：3
步长：2
核数量：5



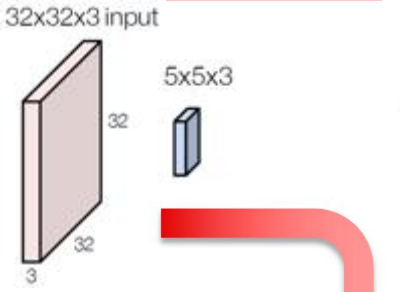
- 3D滤波器/卷积核的超参数
  - 滤波器/卷积核数量（output number）
  - 核尺寸（kernel size）
  - 步长（stride）
  - 零填充（zero padding）
- 尺寸计算（W, H, D）
  - W = （W – size + 2 * padding）/ stride + 1
  - H = （H – size + 2 * padding）/ stride + 1
  - D = output number

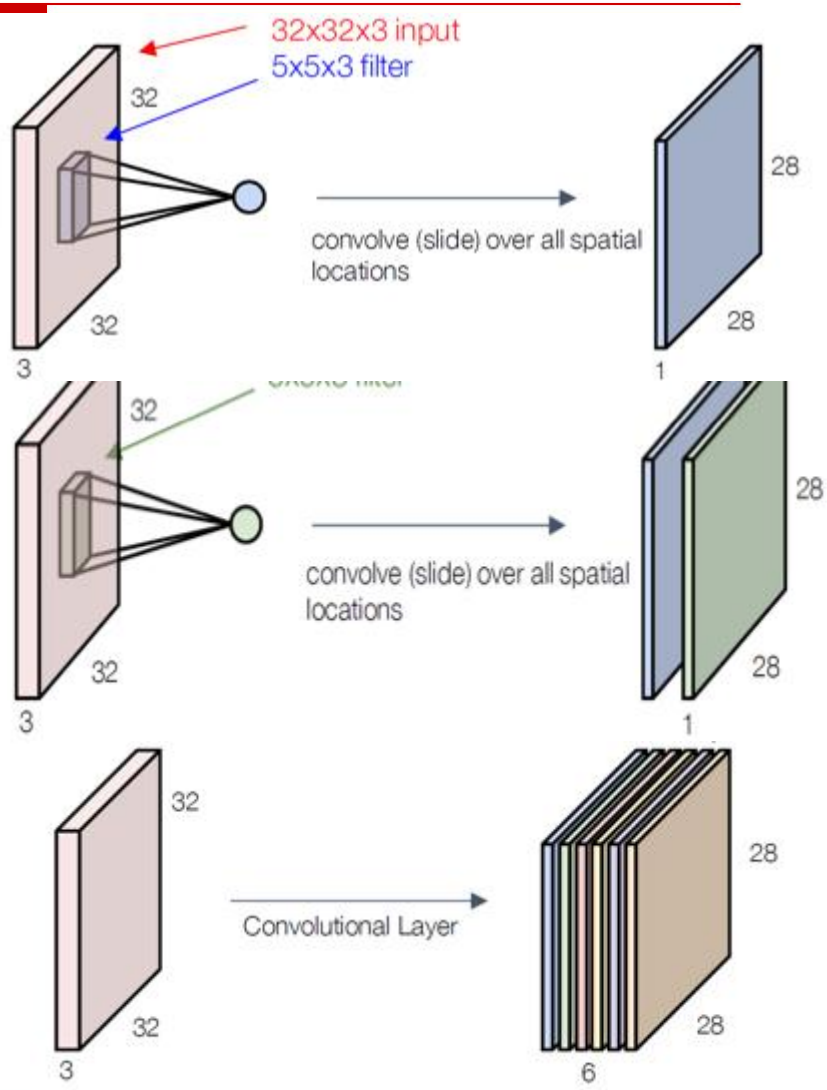


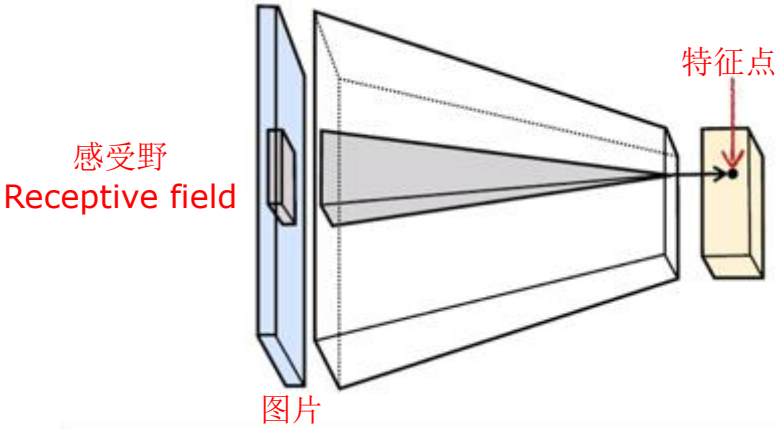
卷积神经网络（CNN）

CNN卷积层

- 原理演示
  - 输入图片32x32x3
  - 卷积核5x5x3
  - 卷积核数量6
  - 输出特征28x28x6





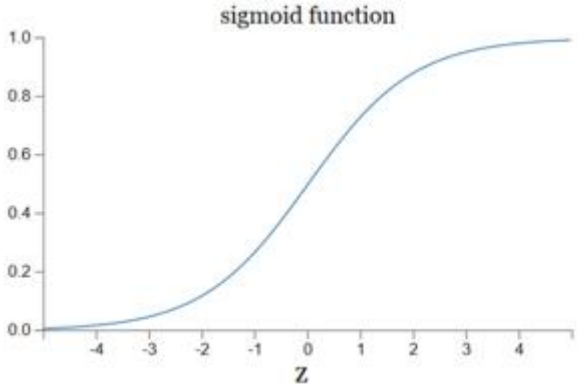


卷积神经网络（CNN）

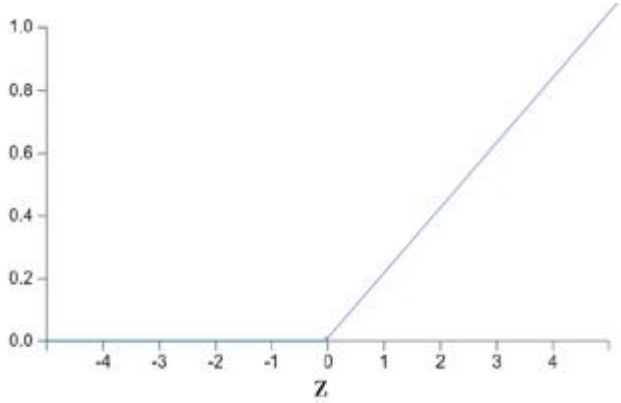
CNN卷积层

- 非线性激活函数
  - Sigmoid
  - ReLU（Rectified Linear Unit）

$$\sigma(z) \equiv \frac{1}{1+e^{-z}}$$



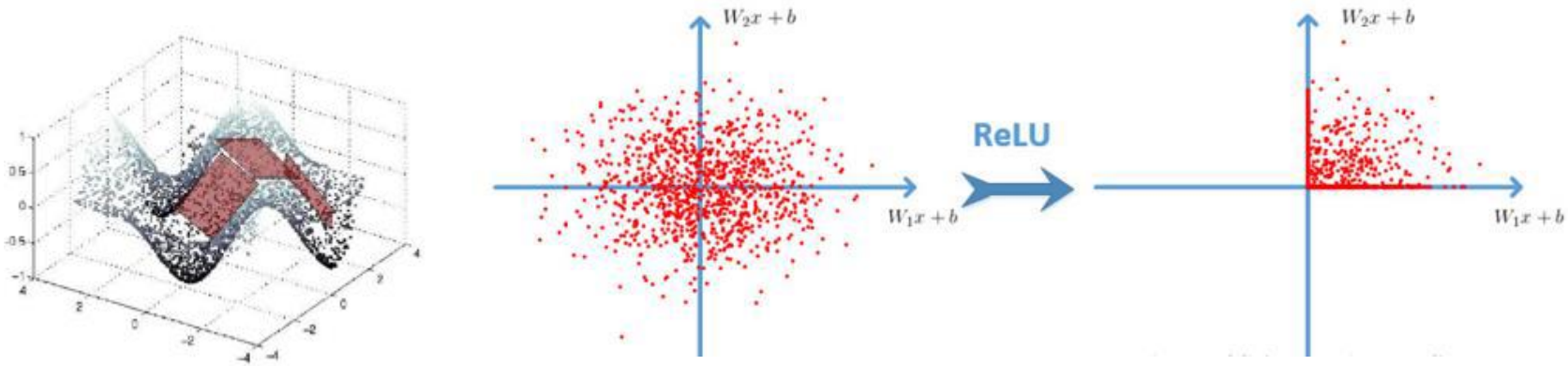
$$f(x) = \max(0, x)$$



卷积神经网络（CNN）

CNN卷积层

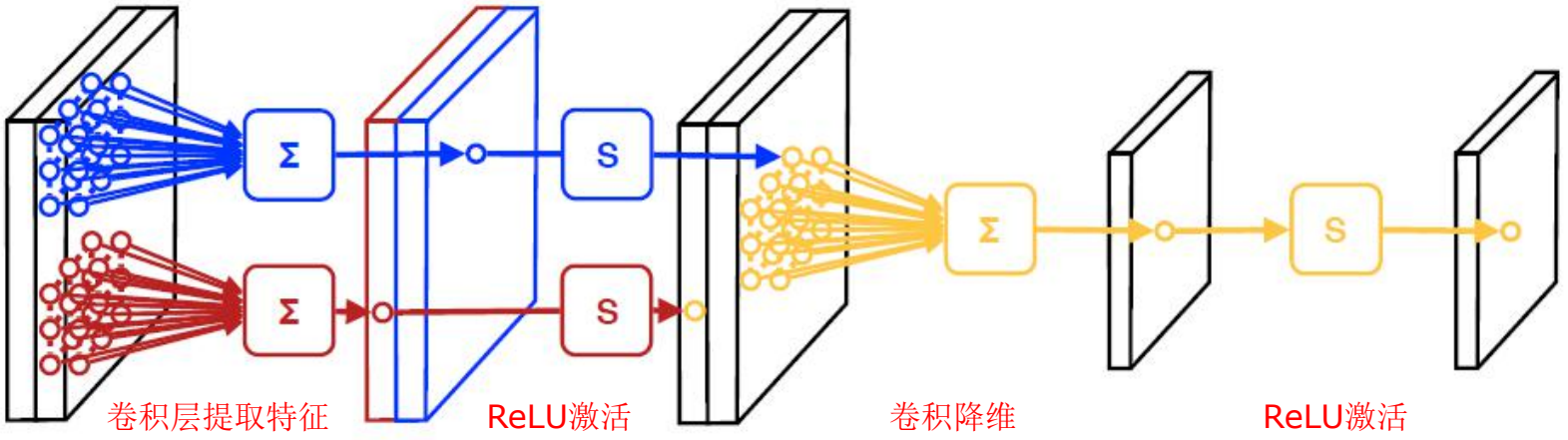
- ReLU激活函数
  - 分段线性函数
  - 无饱和问题，明显减轻梯度消失问题
  - 深度网络能够进行优化的关键



卷积神经网络（CNN）

CNN卷积层

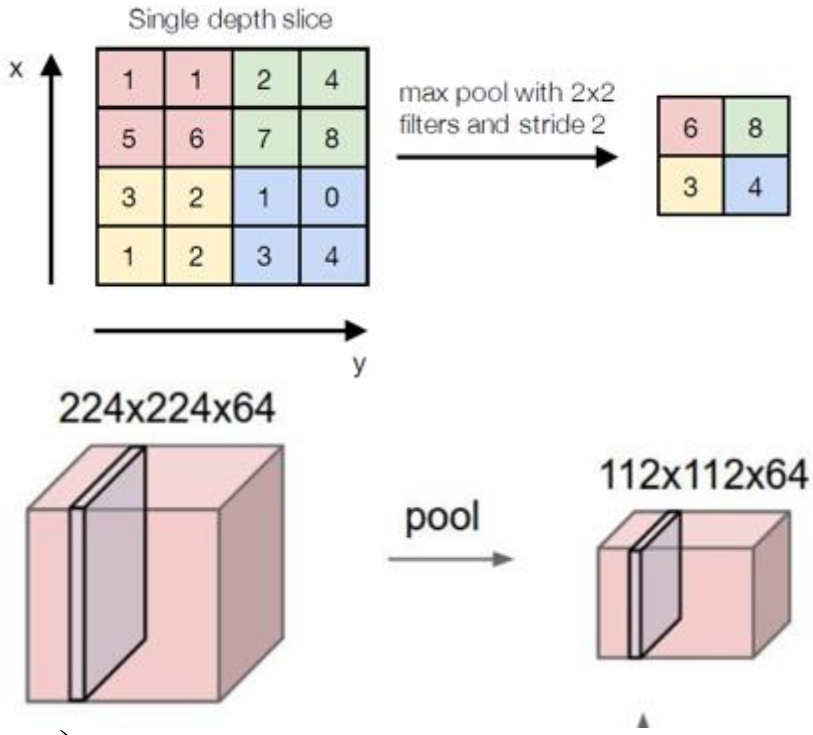
- 组合简例
  - 卷积步长大于1，有降维作用



卷积神经网络（CNN）

CNN池化层

- 作用：特征融合，降维
- 无参数需要学习
- 超参数
  - 尺寸（size）
  - 步长（step）
  - 计算类别
    - 最大化池化（Max pooling）
    - 平均池化（Average pooling）



卷积神经网络（CNN）

CNN-Softmax层

- 指数归一化函数
  - 将一个实数值向量压缩到（0, 1）
  - 所有元素和为1

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{for } j = 1, \ldots, K.$$

- 最后一个全连接层对接1000-way的softmax层
- 得出1000类标签的概率值
- 用于构建loss

复习一下上节课

- a^l表示第l层的输出（a^0是网络最开始的输入），$a^l=\sigma(z^l)$；网络的激活函数假设都是 σ()
- w^l和 b^l表示第l层的参数，z^l是第l层神经元的输入，即 $z^l=w^l a^{l-1}+b^l$ 。
- C表示 loss function，δ^l表示第l层的残差，
- 我们就得到第l层的残差:

$$\delta^{l-1} = W^{lT}\delta^l \odot \sigma'(z^{l-1})$$

$$\Updownarrow$$

$$\delta_j = f'\left(net_j\right)\sum_{k=1}^{c}\delta_k w_{kj}$$

池化层的误差反向传播

- 当接在卷积层的下一层为池化层，已知池化层残差，则卷积层的残差计算方法为：
- 假设第l-1层为卷积层，第l层为池化层，池化层的残差为δ_j^l，卷积层的残差为δ_j^{l-1}，有
  - $\delta_j^{l-1}=\text{upsample}(\delta_j^l)\odot\sigma'(z_j^{l-1})$
- 其中，$\sigma'(z_j^{l-1})$表示的是第l-1层第j个节点处激励函数的导数。
- 符号⊙代表对应位置的元素相乘。
- 假设卷积层的矩形大小为4×4，池化后得到的矩形大小为2×2。
- 假定此时池化后的残差如图：

1	3
2	4

池化层的的误差反向传播

- 先考虑mean-pooling：得到的卷积层应该是4×4大小，其值分布为(等值复制)左图：
- 由于需要满足反向传播时各层间残差总和不变，所以卷积层对应每个值需要平摊：
  - 除以pooling区域大小即可，这里pooling层大小为2×2=4，
- 最后的卷积层值分布为右图：

1	1	3	3
1	1	3	3
2	2	4	4
2	2	4	4



0.25	0.25	0.75	0.75
0.25	0.25	0.75	0.75
0.5	0.5	1	1
0.5	0.5	1	1

池化层的误差反向传播

- 再考虑max-pooling，则需要记录前向传播过程中pooling区域中最大值的位置，
- 这里假设pooling层值1,3,2,4对应的pooling区域位置分别为右下、右上、左上、左下。
- 则此时对应卷积层误差敏感值分布为：

0	0	0	3
0	1	0	0
2	0	0	0
0	0	4	0

- 注意：无论哪种i，都还要乘上$\sigma'(z_j^{l-1})$

卷积层运算

- 假设我们要处理如下卷积操作：

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} * \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{pmatrix} = \begin{pmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \end{pmatrix}$$

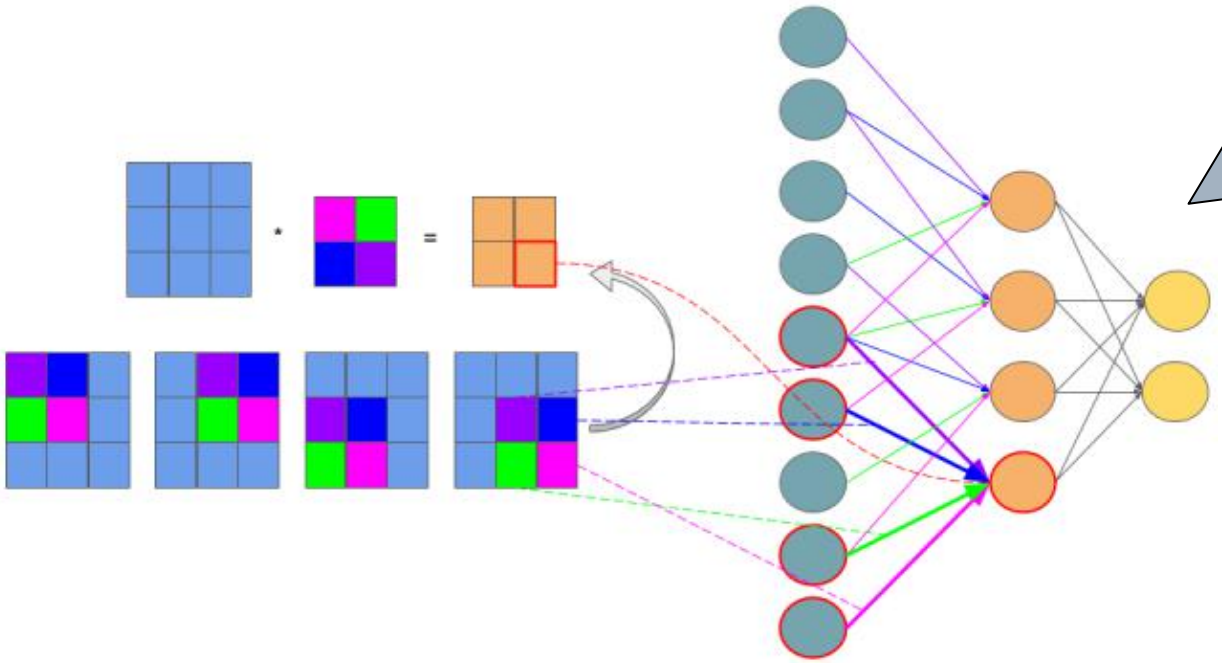
- 这个操作乍看完全不同于全联接层的操作，可以把卷积操作表示成下面的等式：

$$\begin{aligned} z_{11} &= a_{11}w_{11} + a_{12}w_{12} + a_{21}w_{21} + a_{22}w_{22} \\ z_{12} &= a_{12}w_{11} + a_{13}w_{12} + a_{22}w_{21} + a_{23}w_{22} \\ z_{21} &= a_{21}w_{11} + a_{22}w_{12} + a_{31}w_{21} + a_{32}w_{22} \\ z_{22} &= a_{22}w_{11} + a_{23}w_{12} + a_{32}w_{21} + a_{33}w_{22} \end{aligned}$$

- 卷积操作一般是要把卷积核旋转 180 度再相乘的，但由于 CNN 中的卷积参数本来就是训练过程"学"出来的，所以旋不旋转，关系其实不大。

卷积层运算的展开表示

- 左边青色的神经元表示 a_{11} 到 a_{33}，中间橙色的表示 z_{11}到 z_{22}。
- 注意：青色和橙色神经元之间的权值连接用了不同的颜色标出，紫色线表示 w_{11}，蓝色线表示 w_{12}，依此类推。



卷积层其实就是一种特殊的连接层，它是部分连接的，而且参数也是共享的。

卷积层的误差反向传播

- 先计算 δ^{l-1}。
- 假设 a^{l-1} 是前一层经过某些操作后得到的输出，即 $a^{l-1}=\sigma(z^{l-1})$，同时也是本层的输入。
  - 可能是激活函数，也可能是 pooling 层等，无论哪种操作，都可以用 σ() 来表示
- 根据链式法则：

$$\delta^{l-1}=\frac{\partial C}{\partial z^{l-1}}=\frac{\partial C}{\partial z^{l}}\frac{\partial z^{l}}{\partial a^{l-1}}\frac{\partial a^{l-1}}{\partial z^{l-1}}=\delta^{l}\frac{\partial z^{l}}{\partial a^{l-1}}\odot\sigma'(z^{l-1})$$

$$\frac{\partial C}{\partial a_{ij}}$$

$$\delta^{l-1}=W^{lT}\delta^{l}\odot\sigma'(z^{l-1})$$

$z^l=w^la^{l-1}+b^l$

卷积层的误差反向传播

$$
\begin{aligned}
z_{11} &= a_{11}w_{11} + a_{12}w_{12} + a_{21}w_{21} + a_{22}w_{22} \\
z_{12} &= a_{12}w_{11} + a_{13}w_{12} + a_{22}w_{21} + a_{23}w_{22} \\
z_{21} &= a_{21}w_{11} + a_{22}w_{12} + a_{31}w_{21} + a_{32}w_{22} \\
z_{22} &= a_{22}w_{11} + a_{23}w_{12} + a_{32}w_{21} + a_{33}w_{22}
\end{aligned}
$$

- 这里的重点是要计算$\frac{\partial z^l}{\partial a^{l-1}}$，这也是卷积层区别于全联接层之处。
- 根据前面展开的卷积操作的等式，这个导数其实比全联接层更容易求。
- 以a_{11}^{l-1}和a_{12}^{l-1}为例，简洁起见，下面去掉右上角的层数符号l：

∇a_{ij} 表示 $\frac{\partial C}{\partial a_{ij}}$

$$
\begin{aligned}
\nabla a_{11} &= \frac{\partial C}{\partial z_{11}}\frac{\partial z_{11}}{\partial a_{11}} + \frac{\partial C}{\partial z_{12}}\frac{\partial z_{12}}{\partial a_{11}} + \frac{\partial C}{\partial z_{21}}\frac{\partial z_{21}}{\partial a_{11}} + \frac{\partial C}{\partial z_{22}}\frac{\partial z_{22}}{\partial a_{11}} \\
&= \delta_{11}w_{11} \\
\nabla a_{12} &= \frac{\partial C}{\partial z_{11}}\frac{\partial z_{11}}{\partial a_{12}} + \frac{\partial C}{\partial z_{12}}\frac{\partial z_{12}}{\partial a_{12}} + \frac{\partial C}{\partial z_{21}}\frac{\partial z_{21}}{\partial a_{12}} + \frac{\partial C}{\partial z_{22}}\frac{\partial z_{22}}{\partial a_{12}} \\
&= \delta_{11}w_{12} + \delta_{12}w_{11}
\end{aligned}
$$

卷积层的误差反向传播

- 如果把所有式子都写出来，就会发现可以用一个卷积运算来计算所有 ∇a^{l-1}：

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & \delta_{11} & \delta_{12} & 0 \\ 0 & \delta_{21} & \delta_{22} & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} w_{22} & w_{21} \\ w_{12} & w_{11} \end{pmatrix} = \begin{pmatrix} \nabla a_{11} & \nabla a_{12} & \nabla a_{13} \\ \nabla a_{21} & \nabla a_{22} & \nabla a_{23} \\ \nabla a_{31} & \nabla a_{32} & \nabla a_{33} \end{pmatrix}$$

- 这样，δ^{l-1}可以改写为

$$\delta^{l-1} = \frac{\partial C}{\partial z^{l-1}} = \delta^l * rot180(W^l) \odot \sigma'(z^{l-1})$$

- 跟原始的 BP 不同的是，这里需要将后一层的误差 δ^l写成矩阵的形式，并用 0 填充到合适的维度。
- 注意：这里不再是跟矩阵 W^{lT} 相乘，而是先将 W^l **旋转 180 度**后，再跟其做卷积运算。

卷积层的误差反向传播

- 假设第l-1层(池化层)有N个通道，即有N张特征图。第l层(卷积层)有M个特征，l-1层中每个通道图都对应有自己的残差。
- 第l层中第j个核对第l-1层第i个通道的残差计算方法：
  - $\delta_i^{l-1} = \sum_{j=1}^{M} \delta_j^l \star w_{ij}$
- 符号★表示离散卷积，即卷积核旋转180°后再卷积。
- 另外因为这里默认池化层是线性激励函数，所以后面没有乘相应节点的导数。

卷积层的误差反向传播

- 第l-1层某个通道图，大小为3×3，第l层有2个特征核，核大小为2×2，则在前向传播卷积时第l层会有2个大小为2×2的卷积图。
- 假设2个特征核分别为：

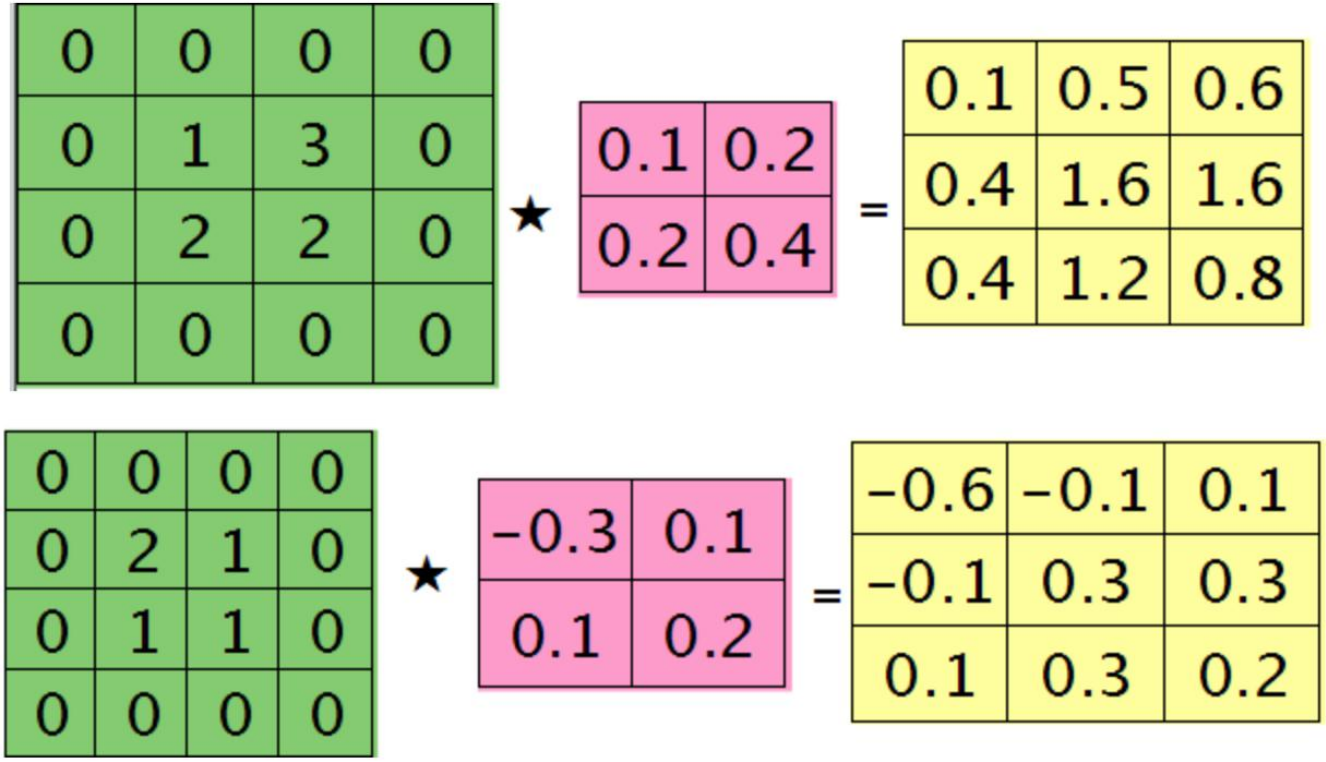
0.1	0.2	-0.3	0.1
0.2	0.4	0.1	0.2

- 反向传播求误差敏感项时，假设已经知道第l层2个卷积图的残差：

1	3	2	1
2	2	1	1

卷积层的误差反向传播

- 经过padding后的矩阵和对应的核进行卷积的结果如下：



卷积层的误差反向传播

- 第*l*-1层中每个通道图都对应有自己的残差，其计算依据为第*l*层所有特征核的贡献之和。
- 即，求第i个通道的残差时需要将第*l*层的所有核都计算一遍，然后求和。
- 本质上还是BP算法，只不过这里是有重叠的，第*l*-1层中某个点会对第*l*层中的多个点有影响。

0.1	0.5	0.6
0.4	1.6	1.6
0.4	1.2	0.8

+

-0.6	-0.1	0.1
-0.1	0.3	0.3
0.1	0.3	0.2

=

-0.5	0.4	0.7
0.3	1.9	1.9
0.5	1.5	1.0

卷积层的误差反向传播

- 假设现在需要求第l-1层的第i个通道，与第l层的第j个通道之间的权值和偏置的导数
- 则计算公式如下：
  - $\frac{\partial C}{\partial w_{ij}} = a_i^{l-1} * \delta_j^l$
  - $\frac{\partial C}{\partial b_j} = \delta_j^l$
- 其中符号*表示卷积操作。

参考资料

- 李宏毅：Deep Learning Tutorial
- https://blog.csdn.net/u014595019/article/details/52989301
- https://blog.csdn.net/u010089444/article/details/76725843
- https://www.cnblogs.com/tornadomeet/p/3468450.html
- http://jermmy.xyz/2017/12/16/2017-12-16-cnn-back-propagation/

问答互动

在所报课的课程页面，

1、点击“全部问题”显示本课程所有学员提问的问题。

2、点击“提问”即可向该课程的老师和助教提问问题。



联系我们

小象学院：互联网新技术在线教育领航者

– 微信公众号：**小象学院**

