

CNN调参技巧

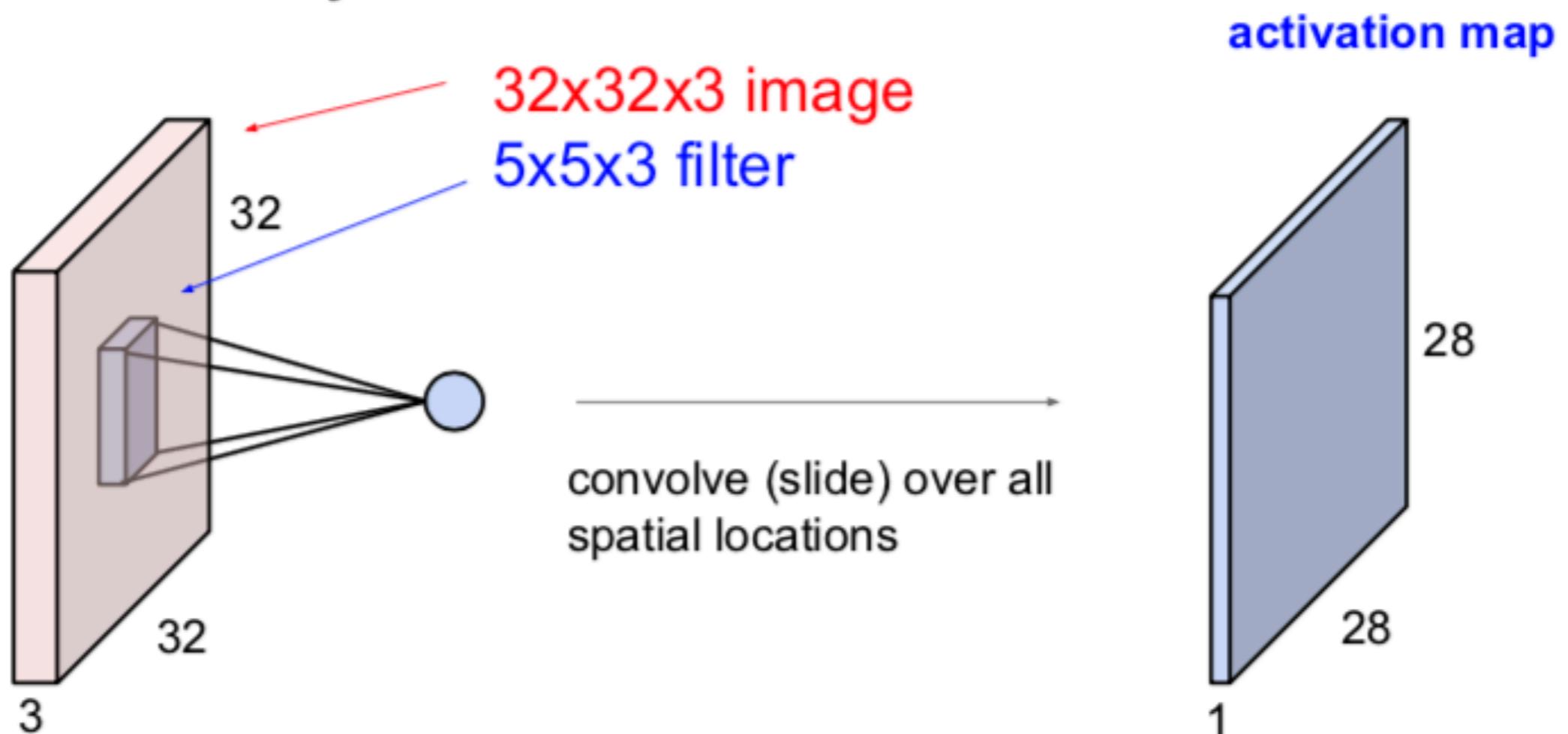
-- By Fanli 28.12.2017

Outline

- CNN回顾
- 为什么要用CNN进行图像处理？
- CNN调参技巧汇总

卷积层

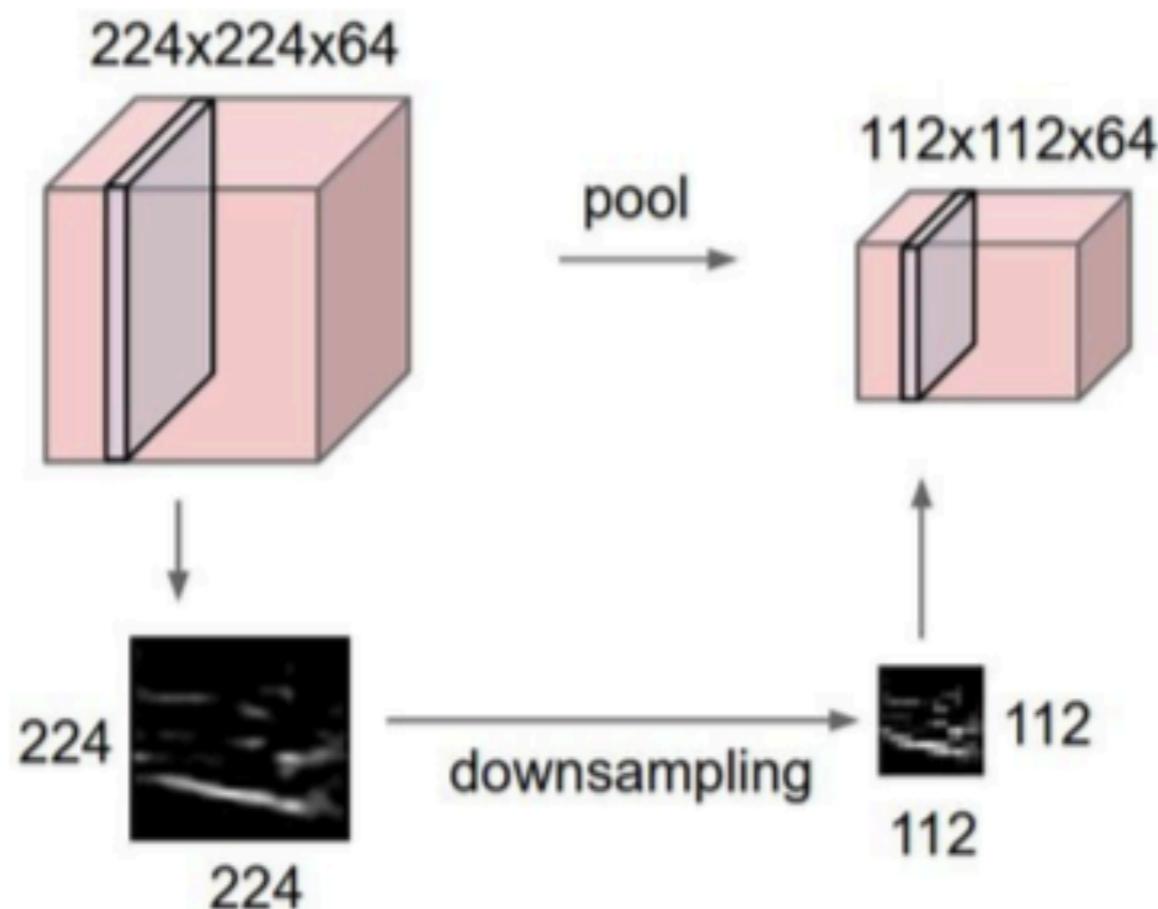
Convolution Layer



池化层

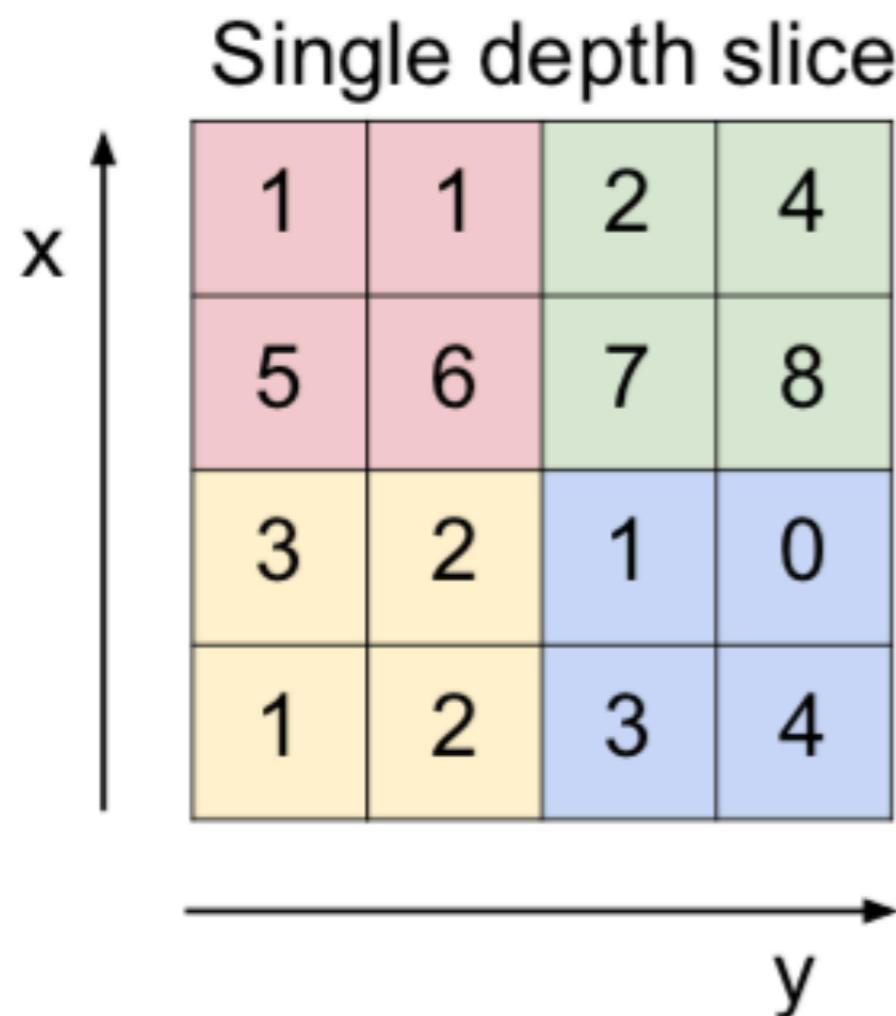
Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



池化层

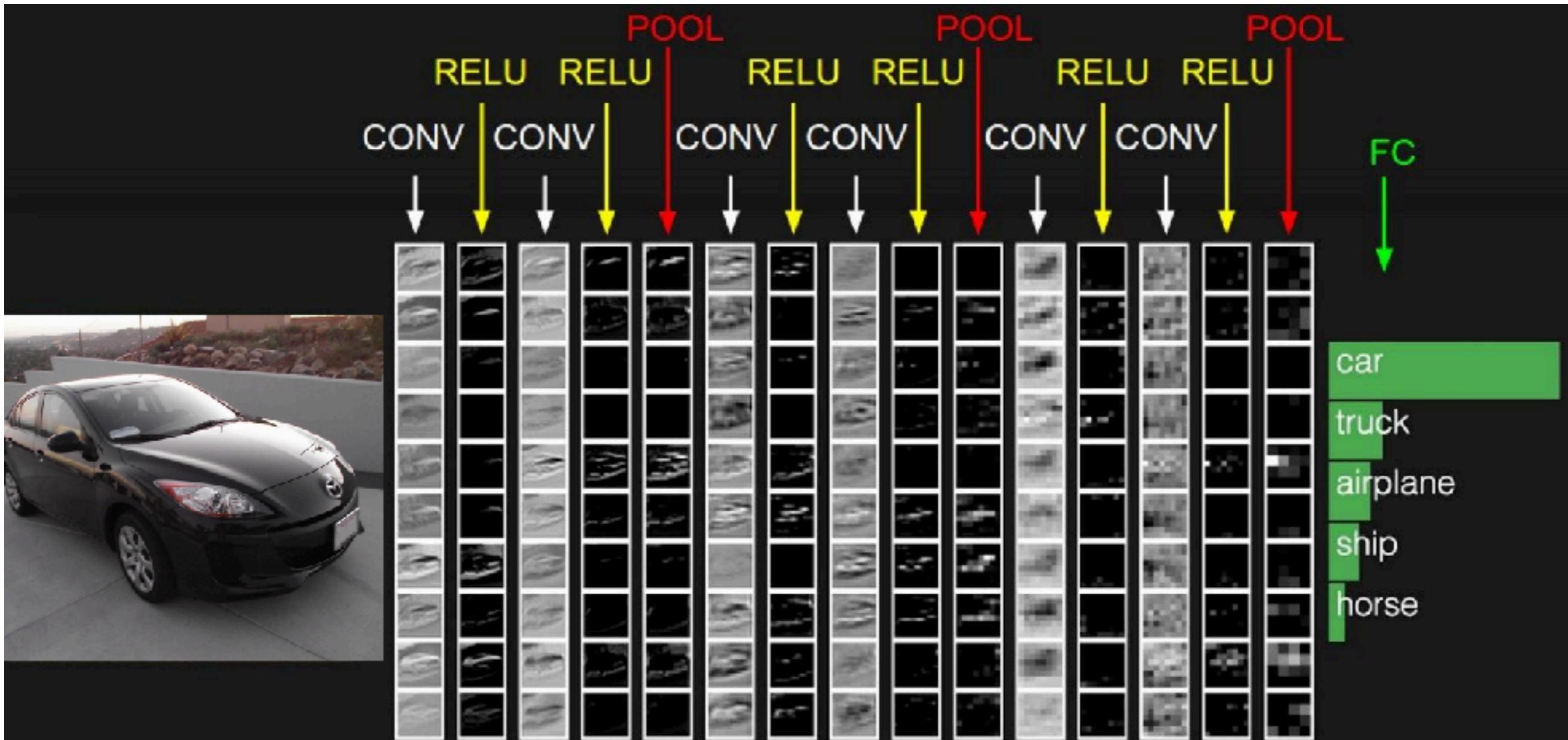
MAX POOLING



max pool with 2x2 filters
and stride 2

6	8
3	4

CNN层级结构



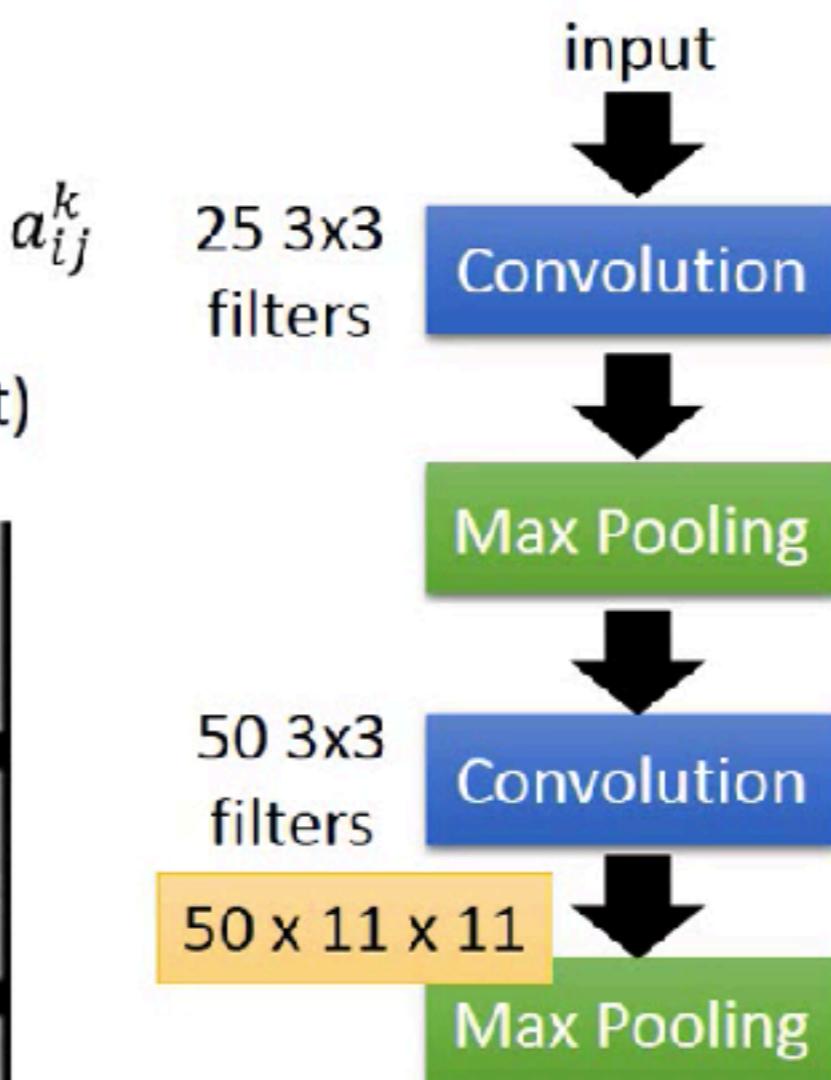
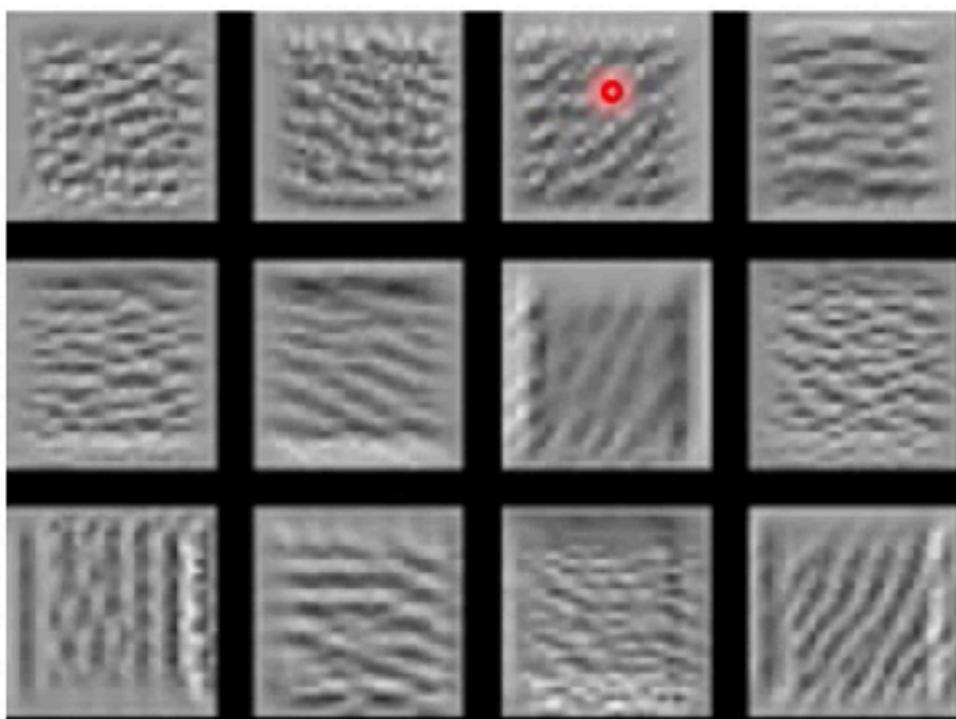
CNN到底在学什么？

What does CNN learn?

The output of the k-th filter is a 11×11 matrix.

Degree of the activation of the k-th filter: $a^k = \sum_{i=1}^{11} \sum_{j=1}^{11} a_{ij}^k$

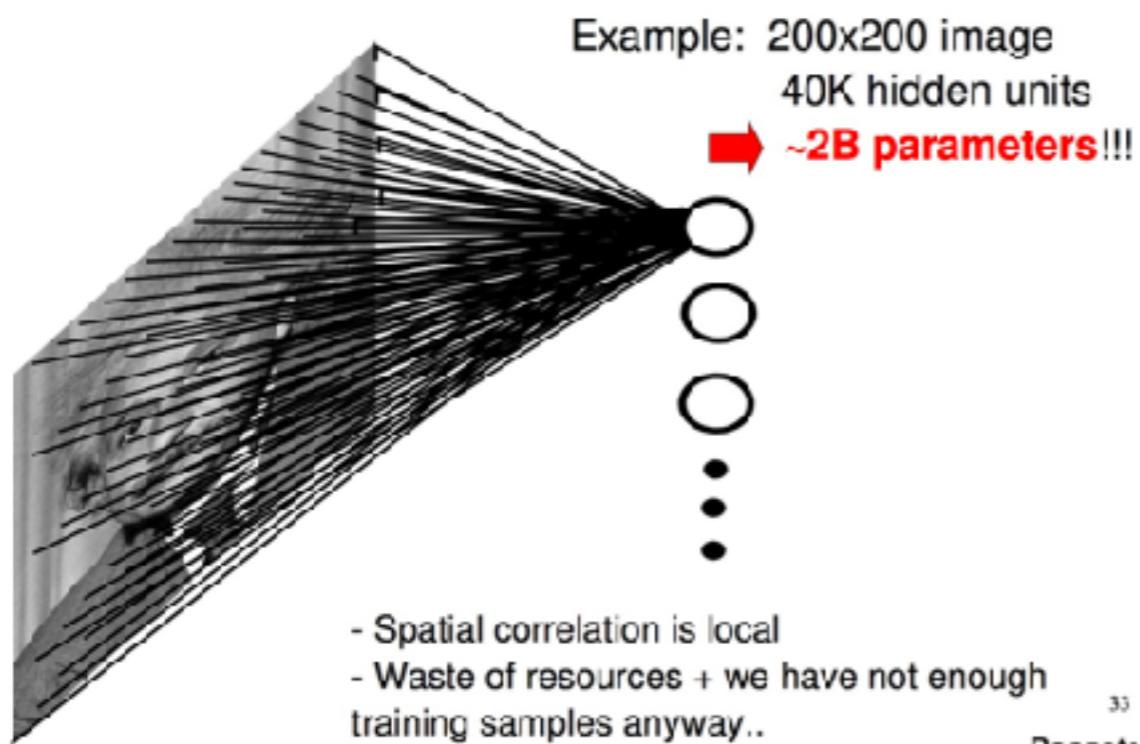
$$x^* = \arg \max_x a^k \text{ (gradient ascent)}$$



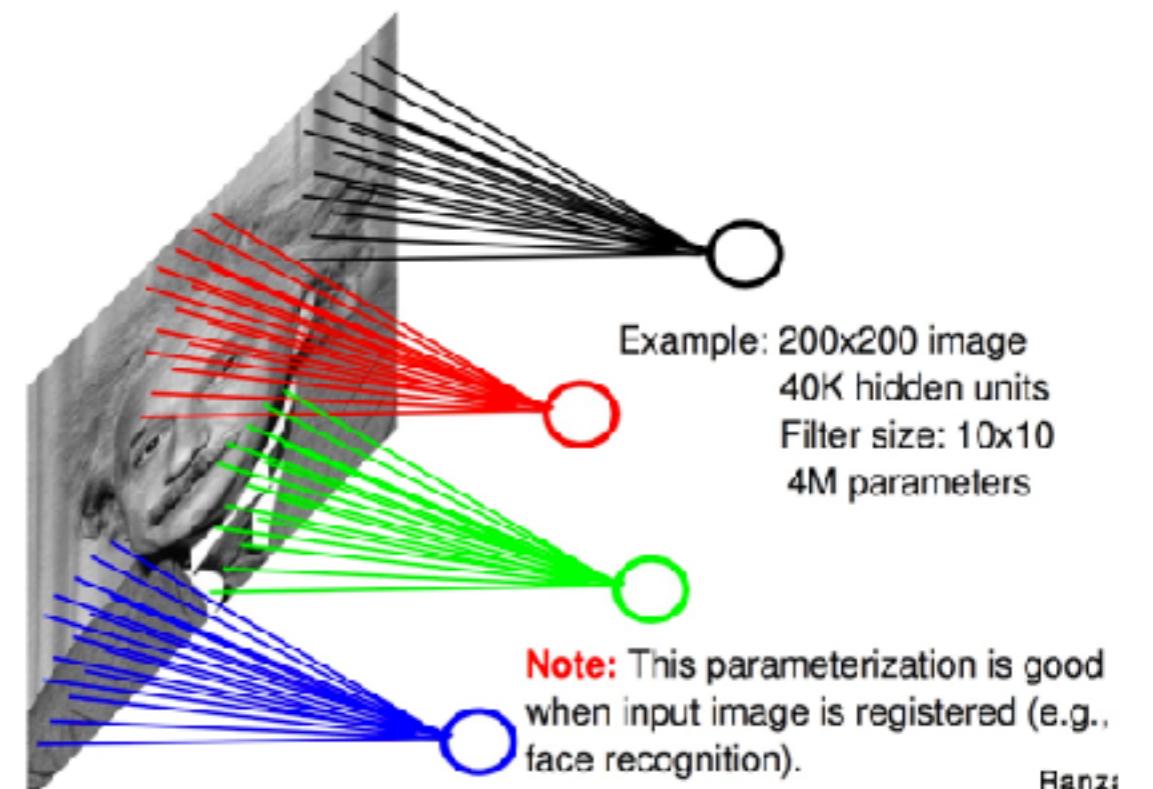
为什么要用CNN进行图像处理?

理由一：

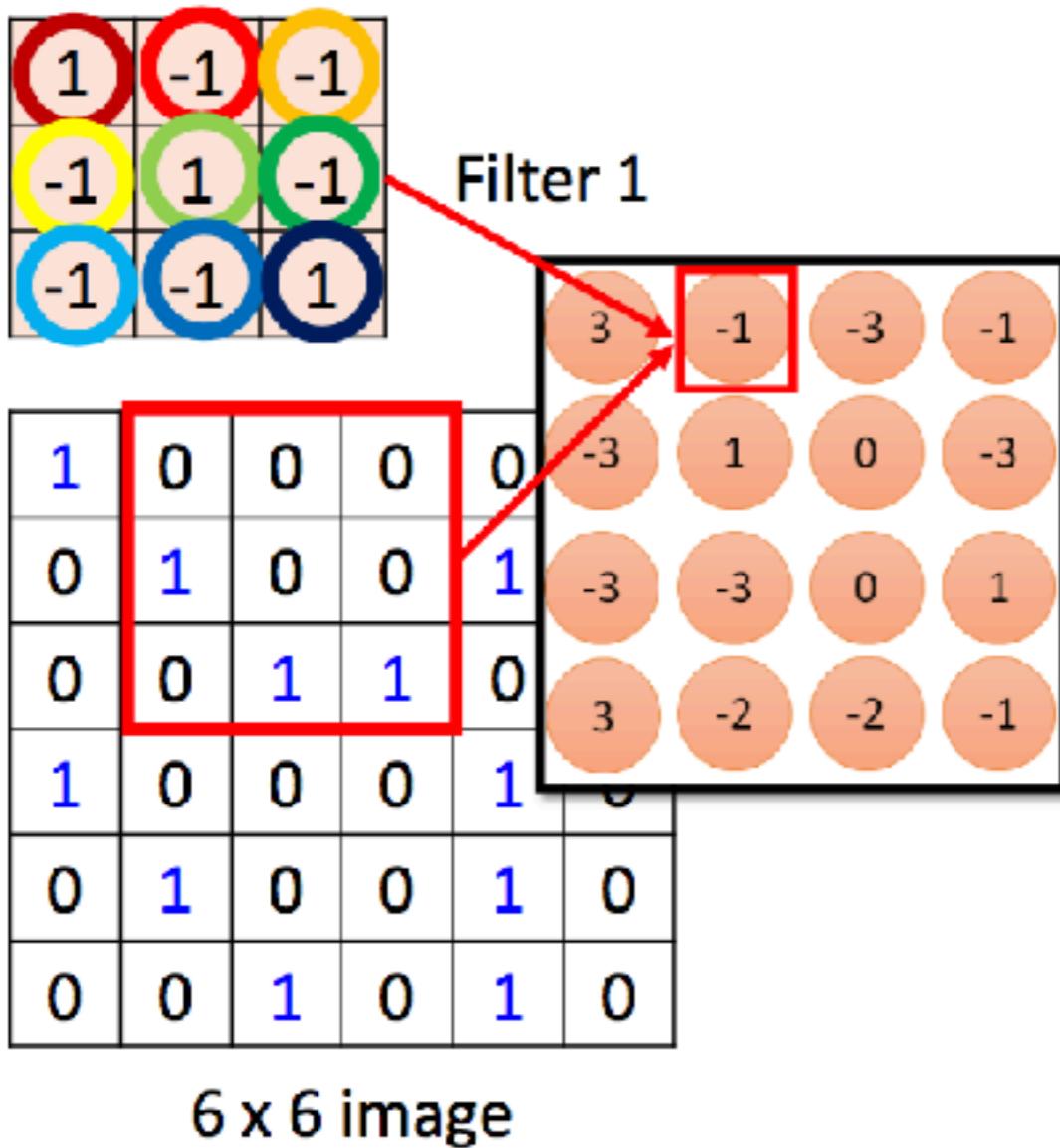
用MLP处理图片的参数量太大了！



33
.....

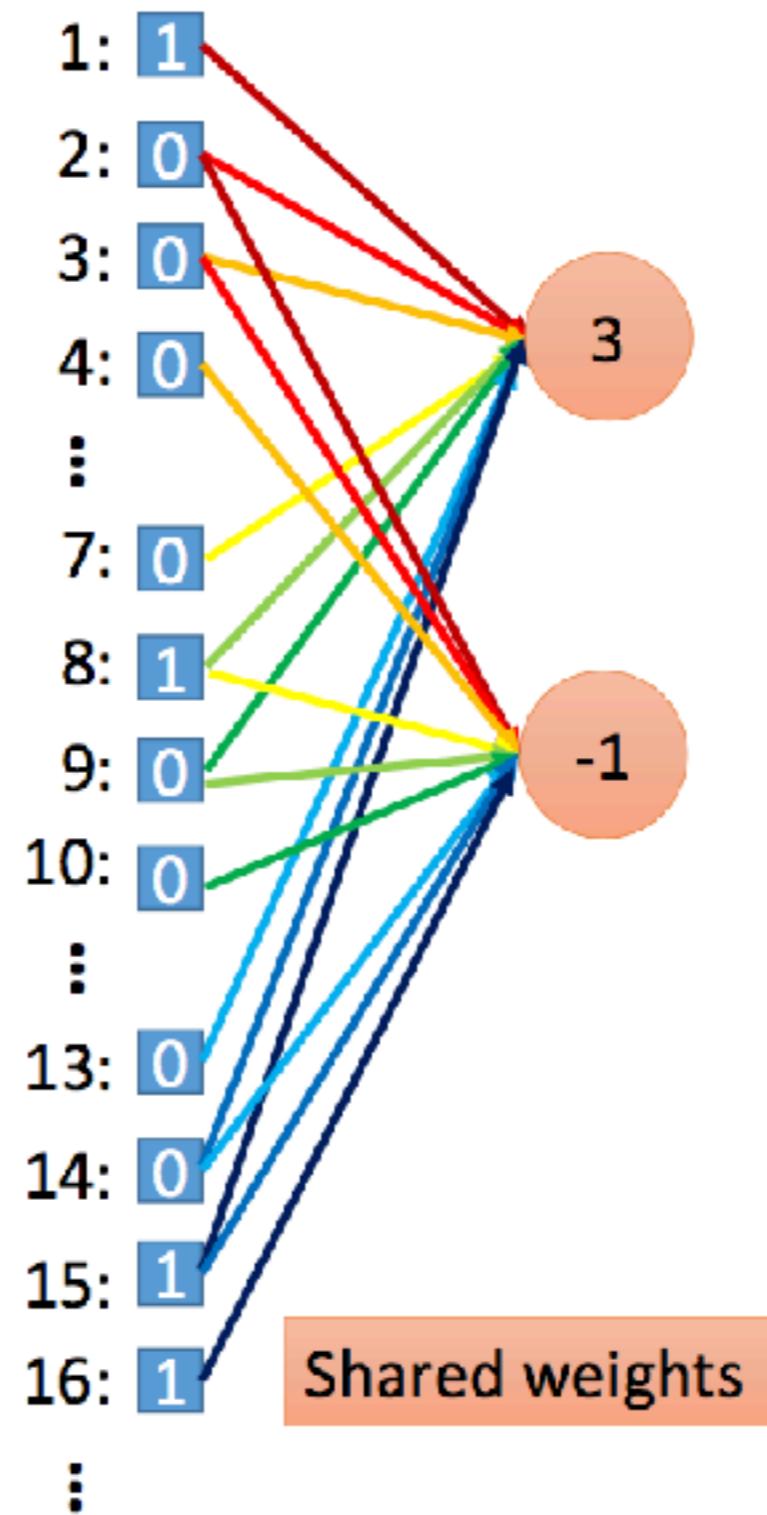


CNN是一个权重共享和稀疏连接的DNN。



Less parameters!

Even less parameters!



为什么要用CNN进行图像处理?

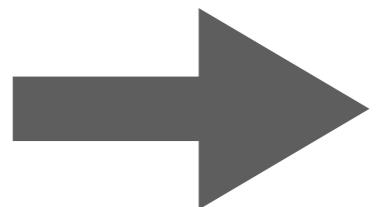
理由二：

由图像本身的特性决定。

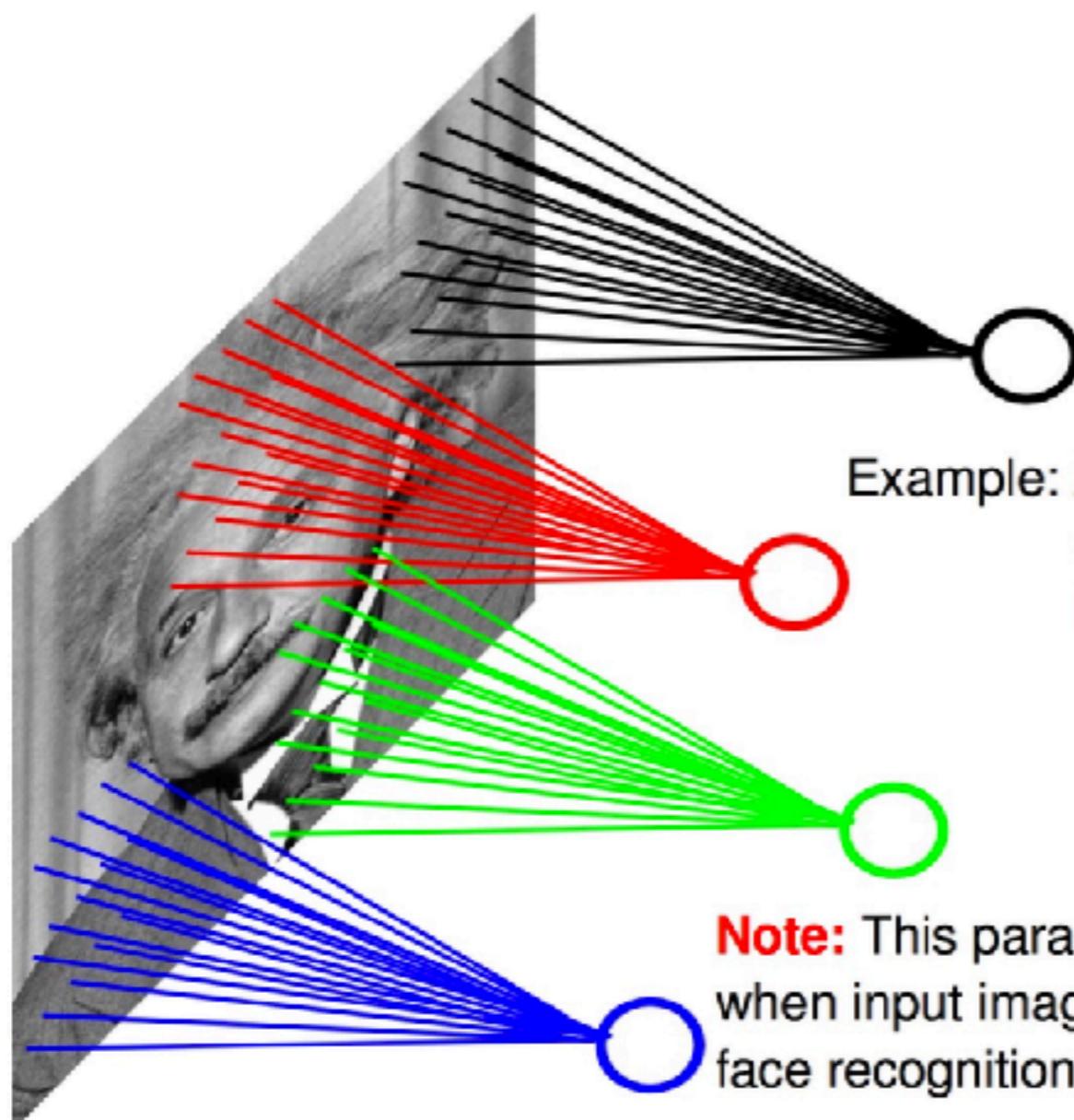
图像特性一：

侦测特定模式 (pattern) 无需看整张图片，只需看图片中的部分区域





体现在卷积层中的局部连接



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

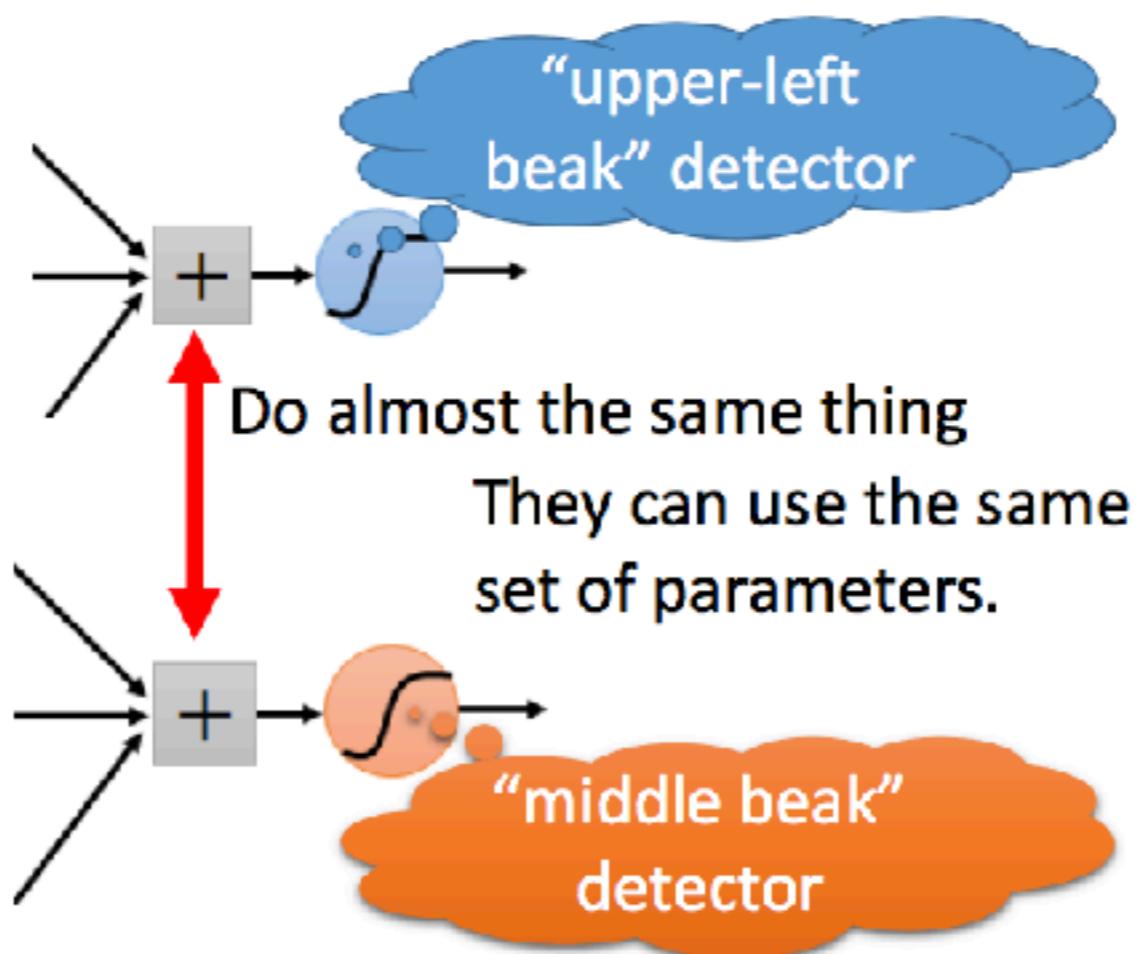
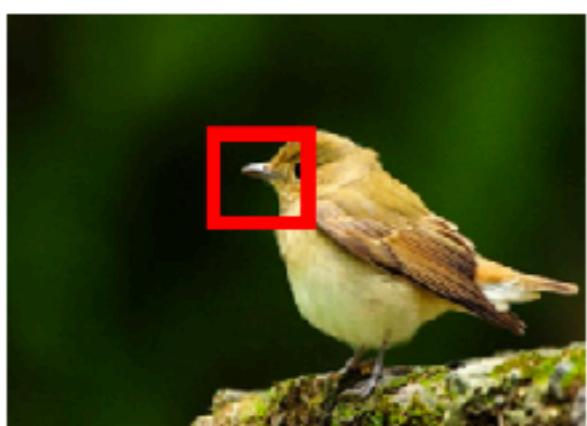
Note: This parameterization is good
when input image is registered (e.g.,
face recognition).

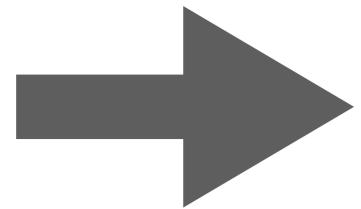
为什么要用CNN进行图像处理?

理由二：

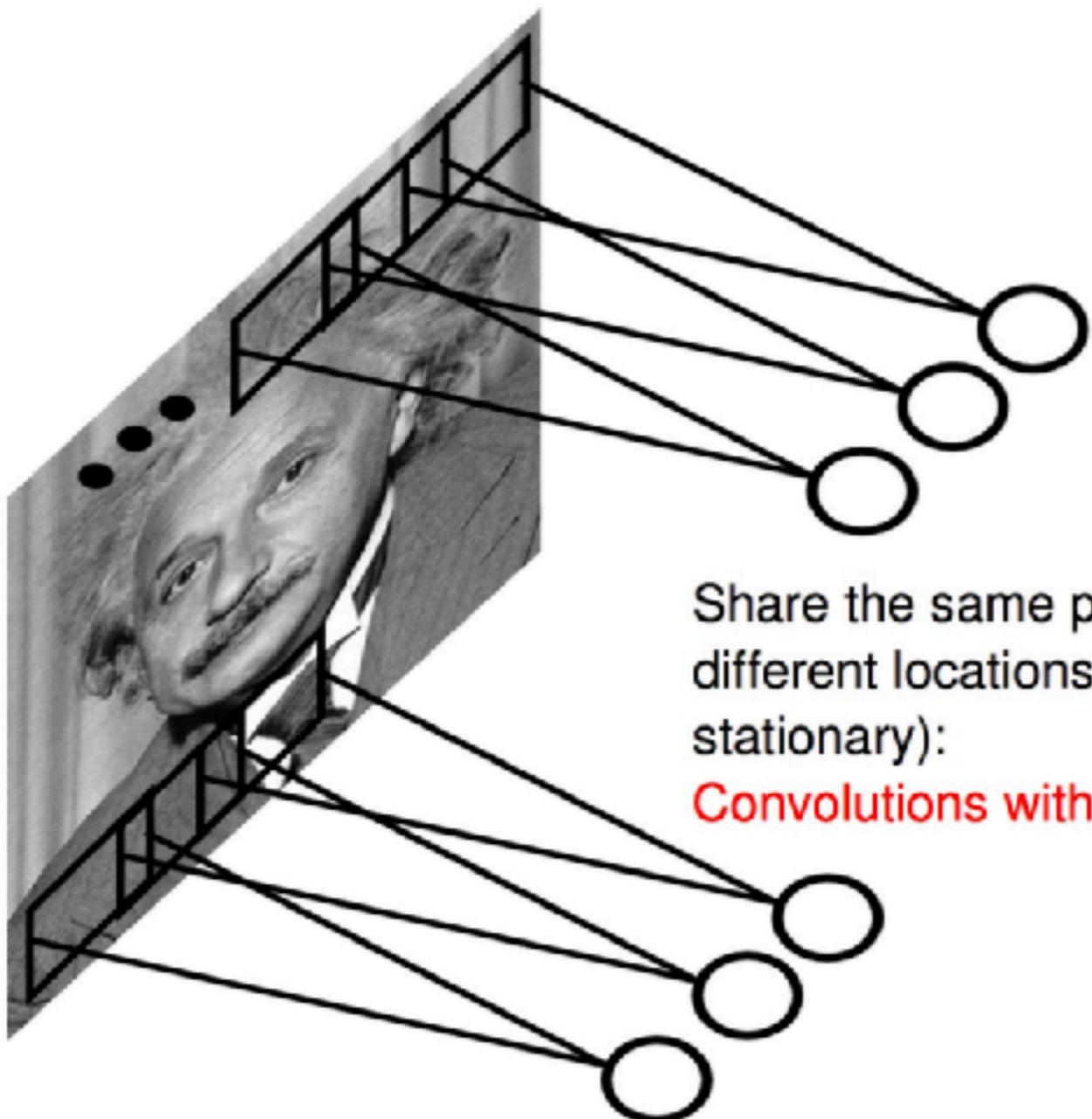
由图像本身的特性决定。

图像特性二：同样的模式（pattern）可能会存在于图像中的不同区域





体现在卷积层中的权重共享



Ranz

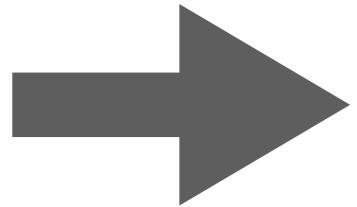
为什么要用CNN进行图像处理?

理由二：

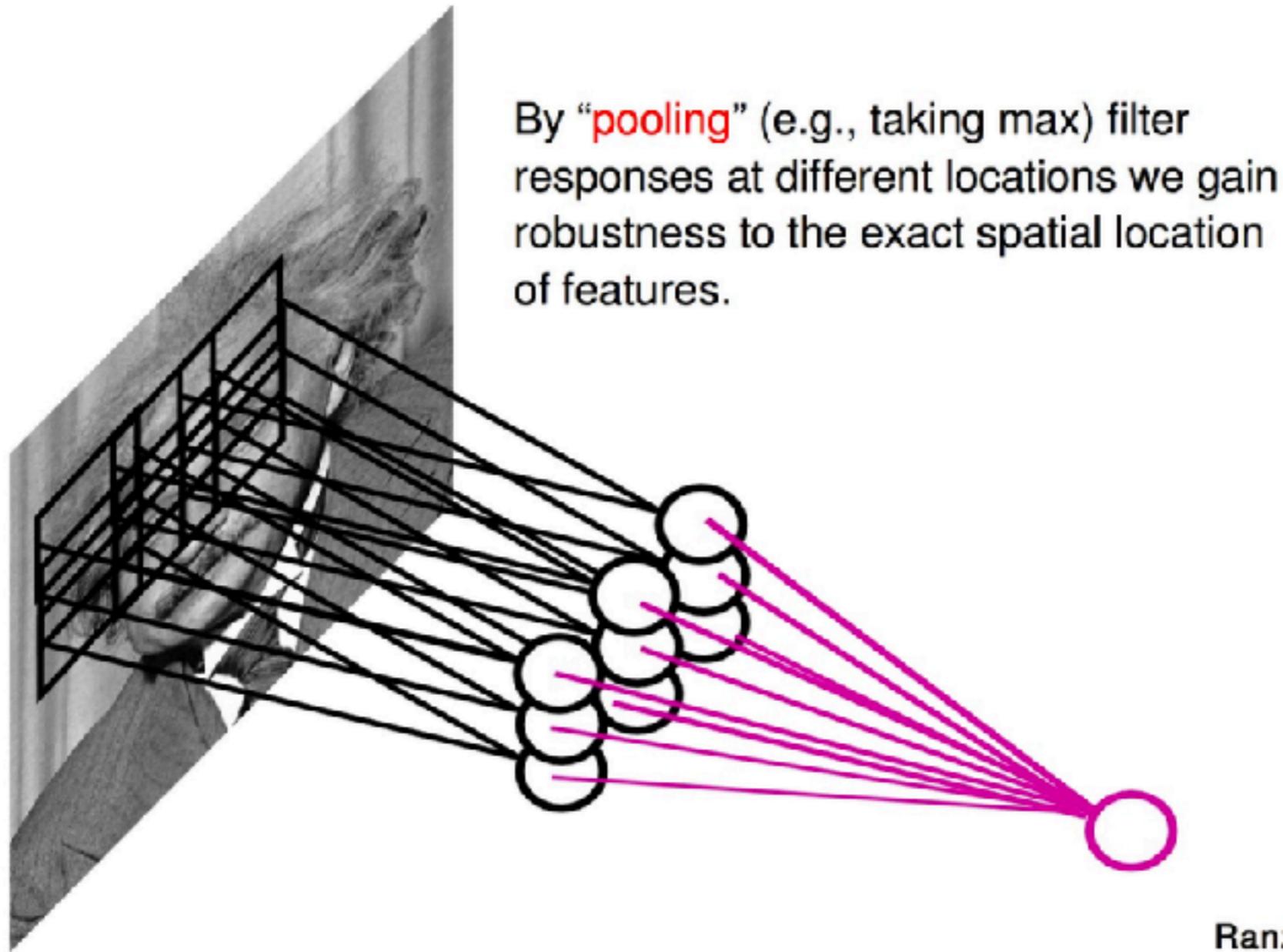
由图像本身的特性决定。

图像特性三：缩小图像不会改变图像识别的结果





体现在池化层，实现数据降维，从而
减少参数个数



CNN调参技巧汇总

A. 网络设计架构

- 激活函数
- 网络层大小
- 预训练模型调优

B. 参数相关

- 权重初始化
- 从图像中学习
- 适应性的学习率

C. Overfitting

- 正则化
- Dropout
- 数据扩增
- Early Stopping

D. 其他

- 集成
- 数据预处理

权重初始化



1. 将权重全部初始化为0?

不可以！因为这样每个神经元的输出都相同，反向传播时每个神经元的权重更新也相同，也就是说，本来我们希望不同的神经元学习到不同的权值，但是现在每个神经元学习到的特征都相同，这就失去了网络学习的意义。

2. 随机初始化为较小的值？比如： $\text{weights} \sim 0.001 \times N(0, 1)$

不好，因为初始化值太低和标准高斯分布都会使得梯度更新值过小，导致学习速率太低，也会有梯度消失的问题。

参考：<http://blog.csdn.net/lanchunhui/article/details/50099521>

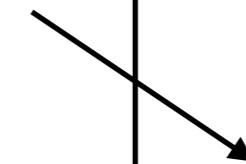
初始化的正确姿势

找到一种初始权重的方法使得神经元输入和输出值的方差相等。

- Initialize all the intercepts (b) = 0
- Initialize model weights uniformly from $[-U, U]$

$$U = \frac{\sqrt{6}}{\sqrt{\text{fan_in} + \text{fan_out}}}$$

- fan_in = # neurons in the last layer
- fan_out = # neurons in the current layer



Xavier Initialization

原始论文：<http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf>

参考链接：<https://prateekvjoshi.com/2016/03/29/understanding-xavier-initialization-in-deep-neural-networks/>

预训练模型调优 (Fine-Tune)

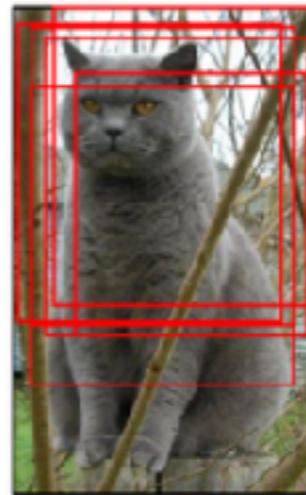
	数据集与预训练模型采用的数据集非常相似	数据集与预训练模型采用的数据集不同
数据集较小	在预训练模型前几层输出特征上再训练一个线性分类器	这种情况比较麻烦，在网络较靠前的特征层就训练线性分类器可能比较好
数据集较大	用较小的学习率对预训练模型的前几层进行调优	用较小的学习率对预训练模型的很多层进行调优

- 在自己的数据集上调优 (fine-tune) 预训练模型。不同情况下采用不同的调优策略。对于数据集，_Caltech-101_与_ImageNet_比较相似，都是关于物体的数据集；而_Place Database_就与_ImageNet_不同，它是关于场景的数据集
- 可以从Caffe Model Zoo可以下载不同的预训练模型

数据扩增



水平翻转



随机剪裁

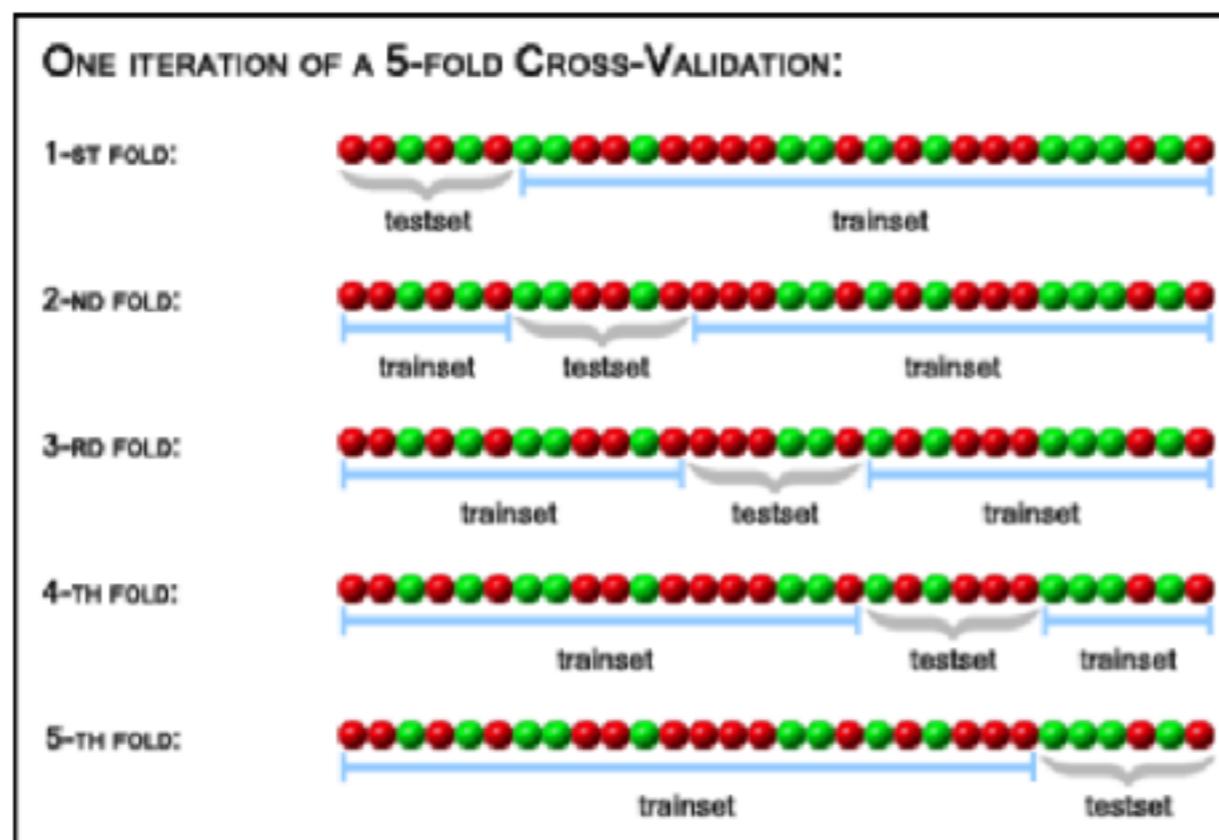


颜色抖动

也可以尝试多种组合，比如同时对图像进行旋转和缩放，调节图像中的饱和度，明度等等

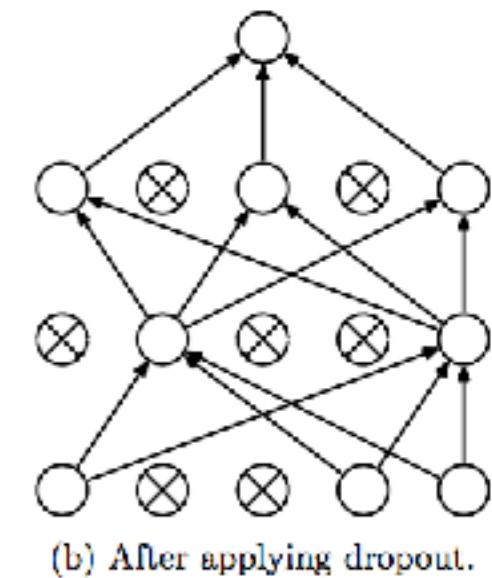
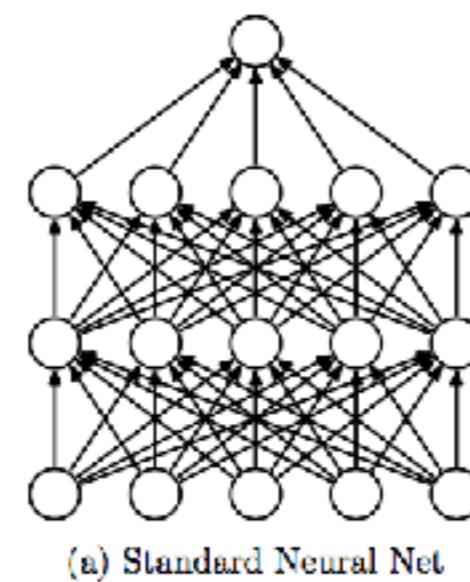
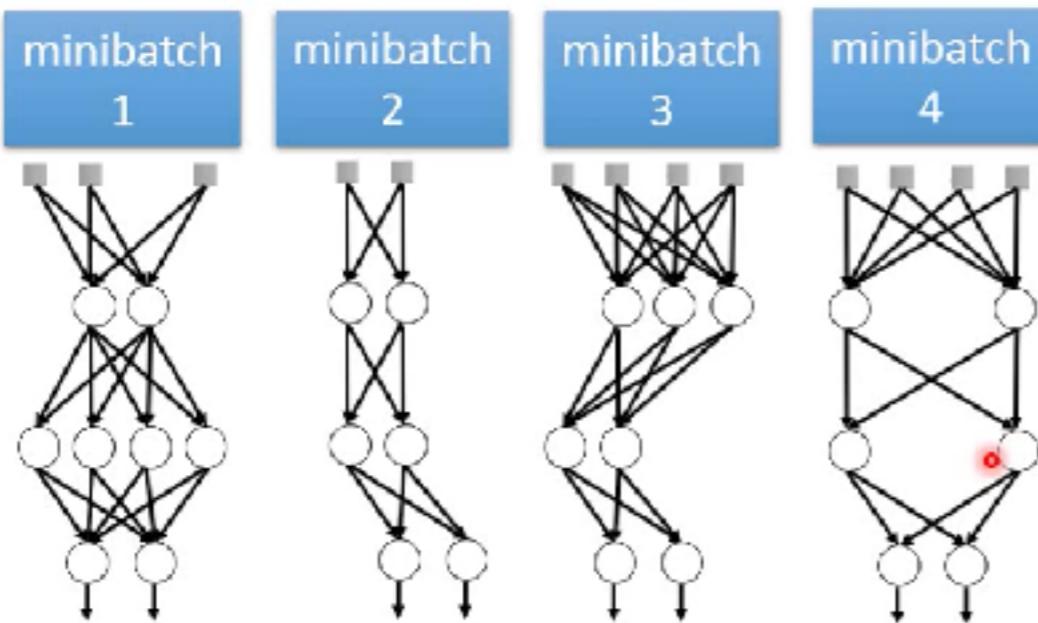
集成 (Ensemble)

- 用不同的初始化训练不同的模型
- 用不同的数据集训练不同的模型
- 用交叉验证训练不同模型
- 用不同的测试集训练结果，然后将结果平均

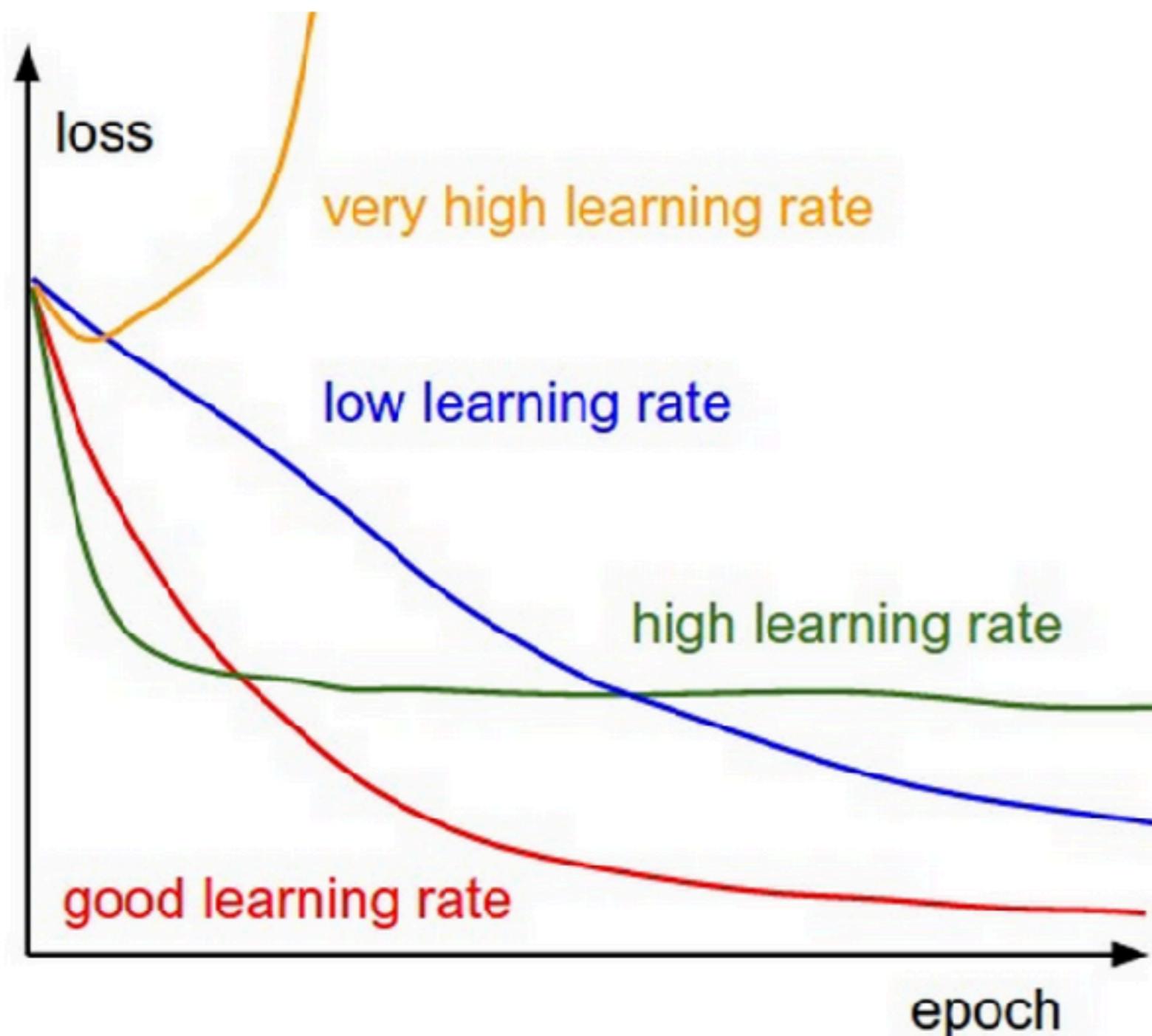


Dropout

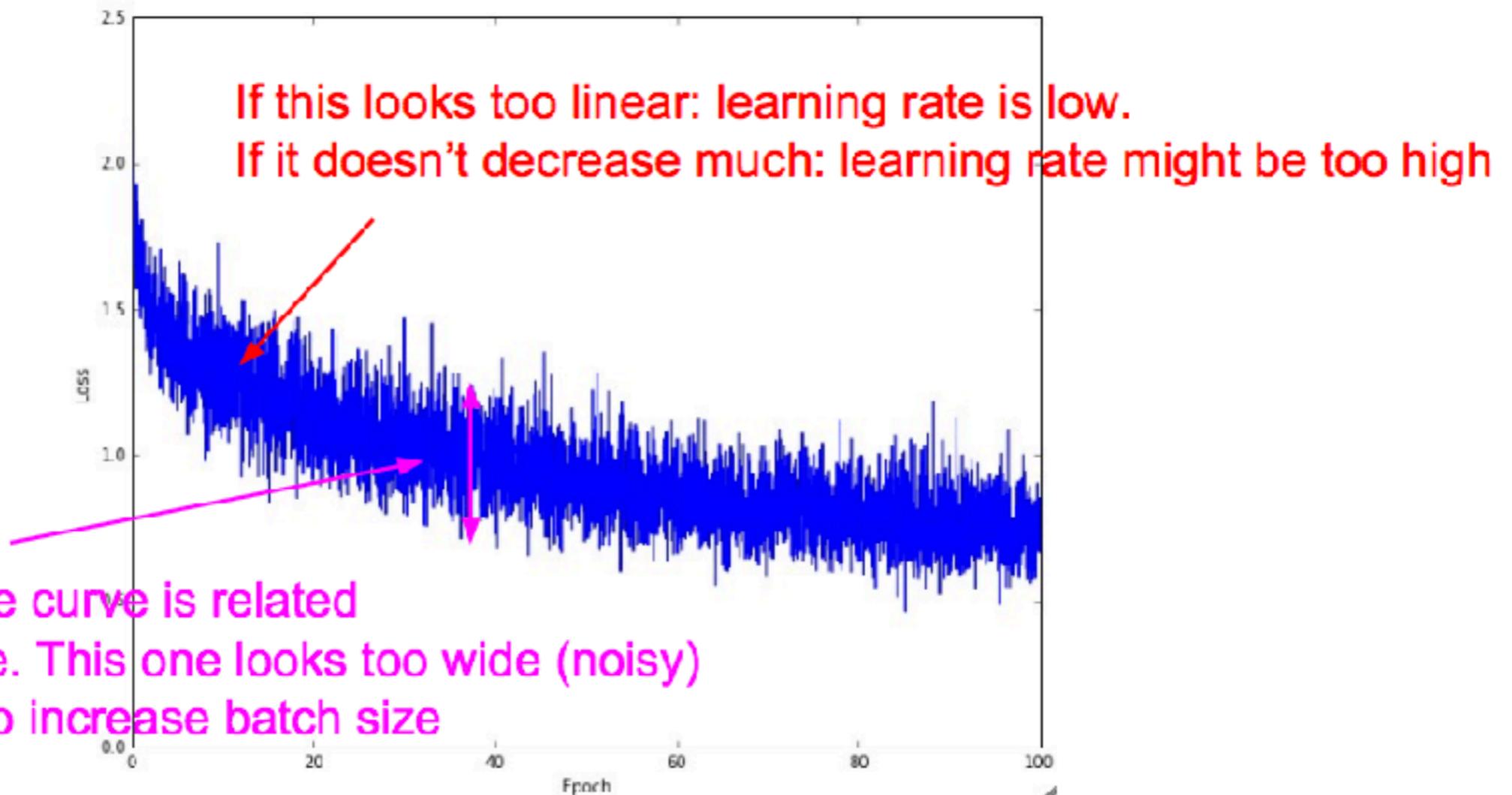
- 每一次更新参数的时候，丢掉 $p\%$ 的神经元
- 用新的网络进行训练
- Dropout只在训练的时候使用，测试的时候不用
- 但是权重都要乘以 $(1-p)\%$
- Dropout ratio一般设为0.5
- Dropout是一种集成方法



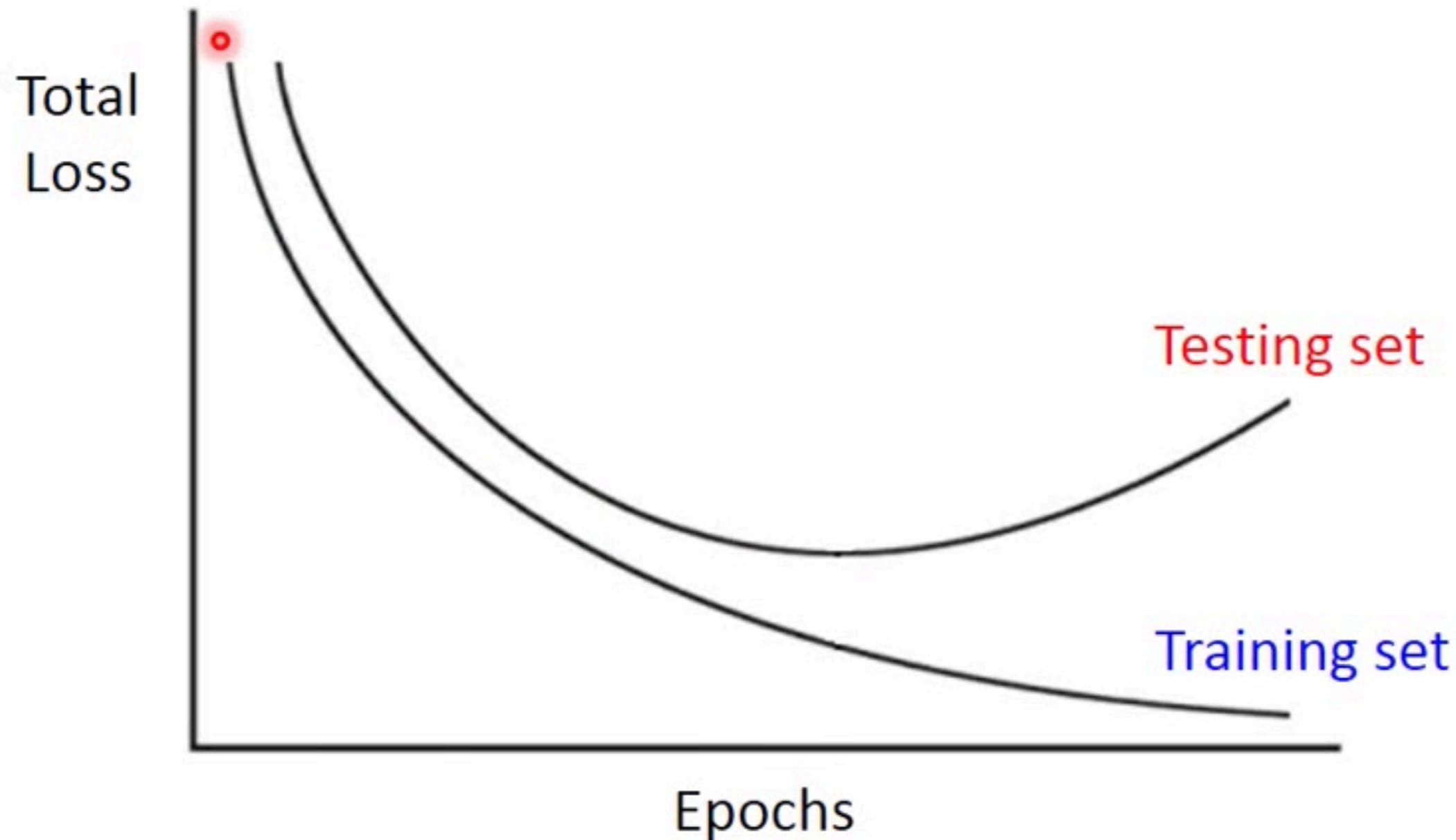
从图像中学习——学习率



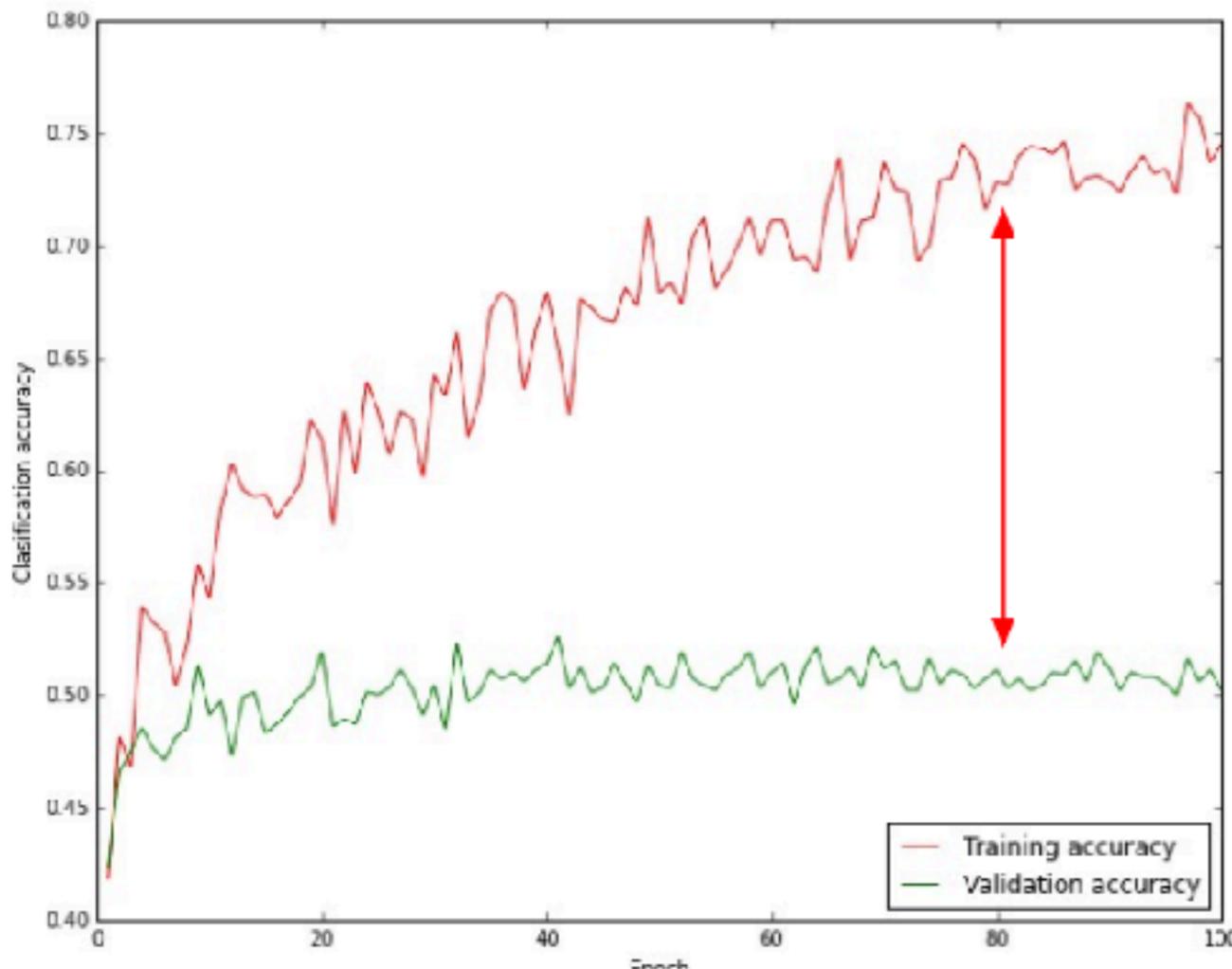
从图像中学习——Batch-size



从图像中学习——Early Stopping



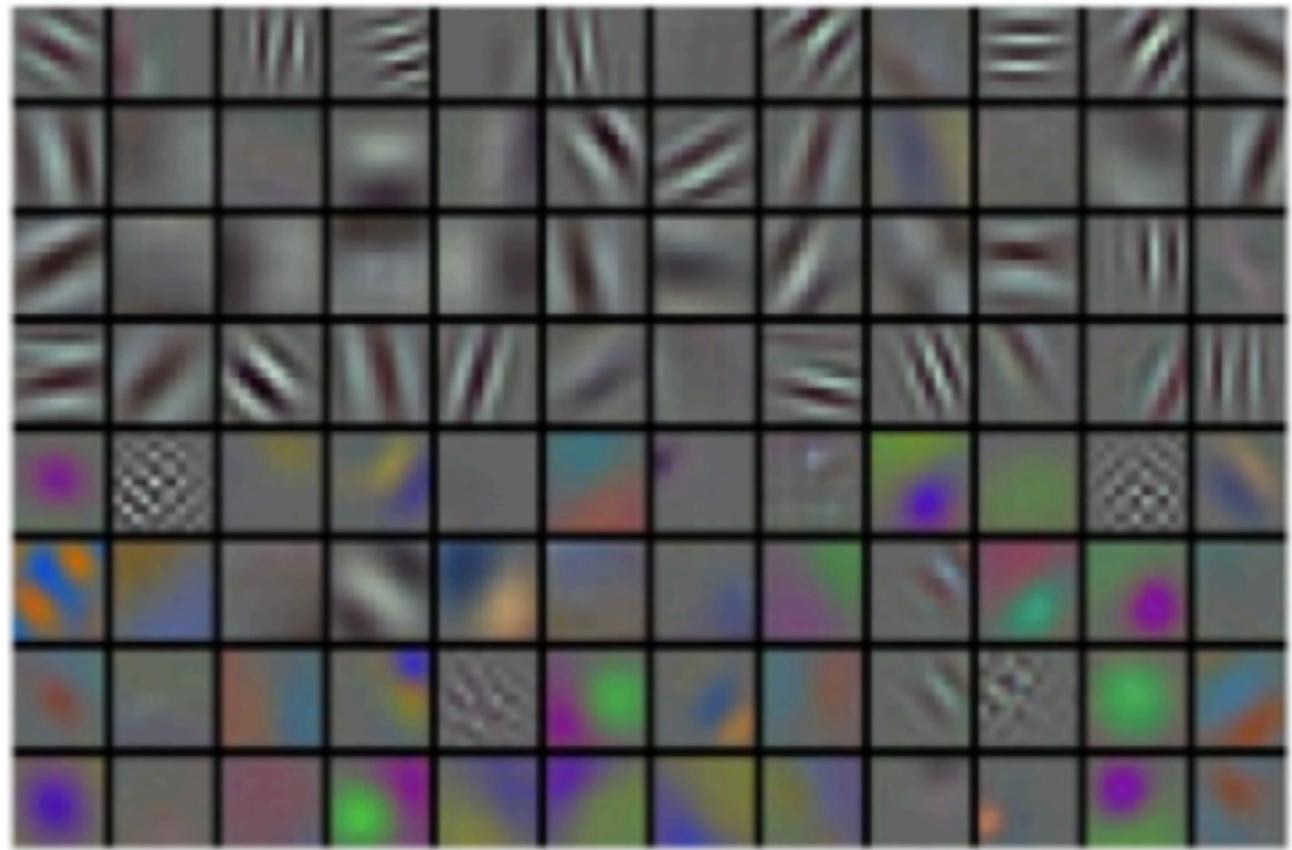
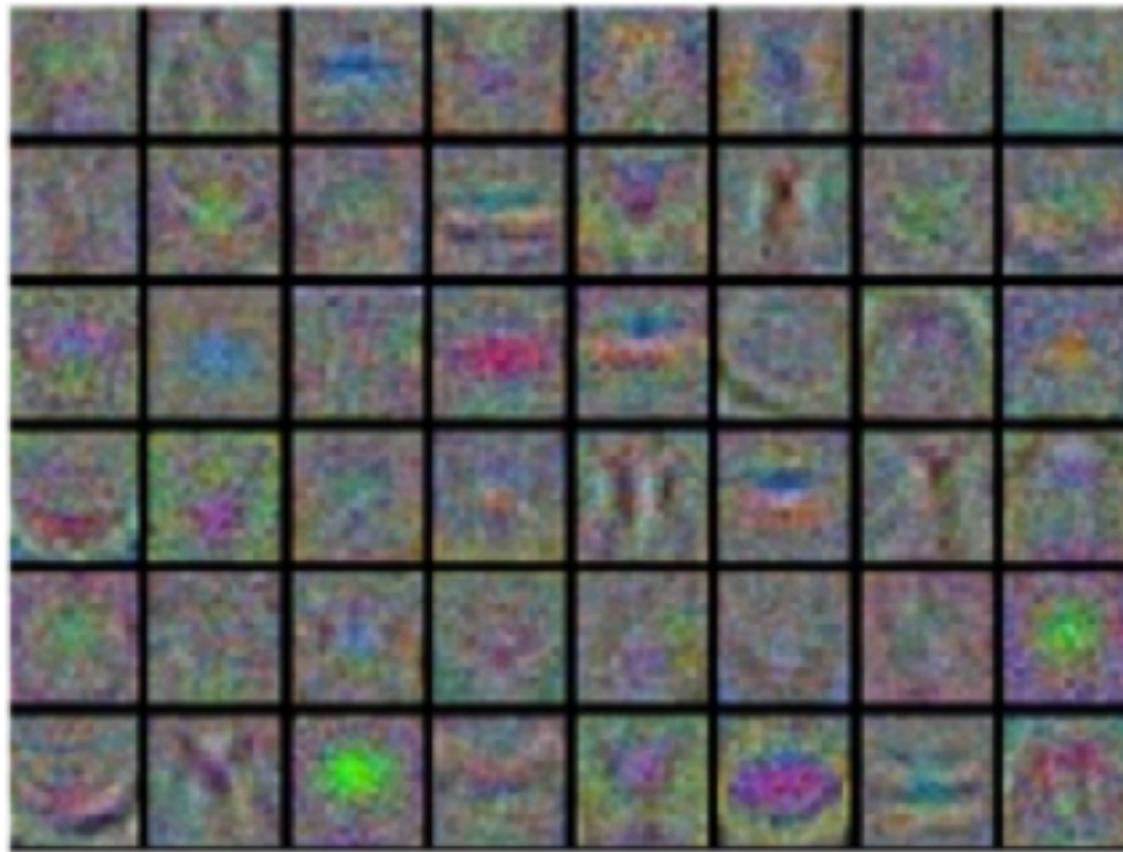
从图像中学习——过拟合vs.欠拟合



big gap = overfitting
=> increase regularization strength

no gap
=> increase model capacity

从图像中学习——可视化卷积层的权重



良好的卷积层的权重可视化出来会具有smooth的特性，
上图中左边图片有很多噪点，右边则比较平滑。

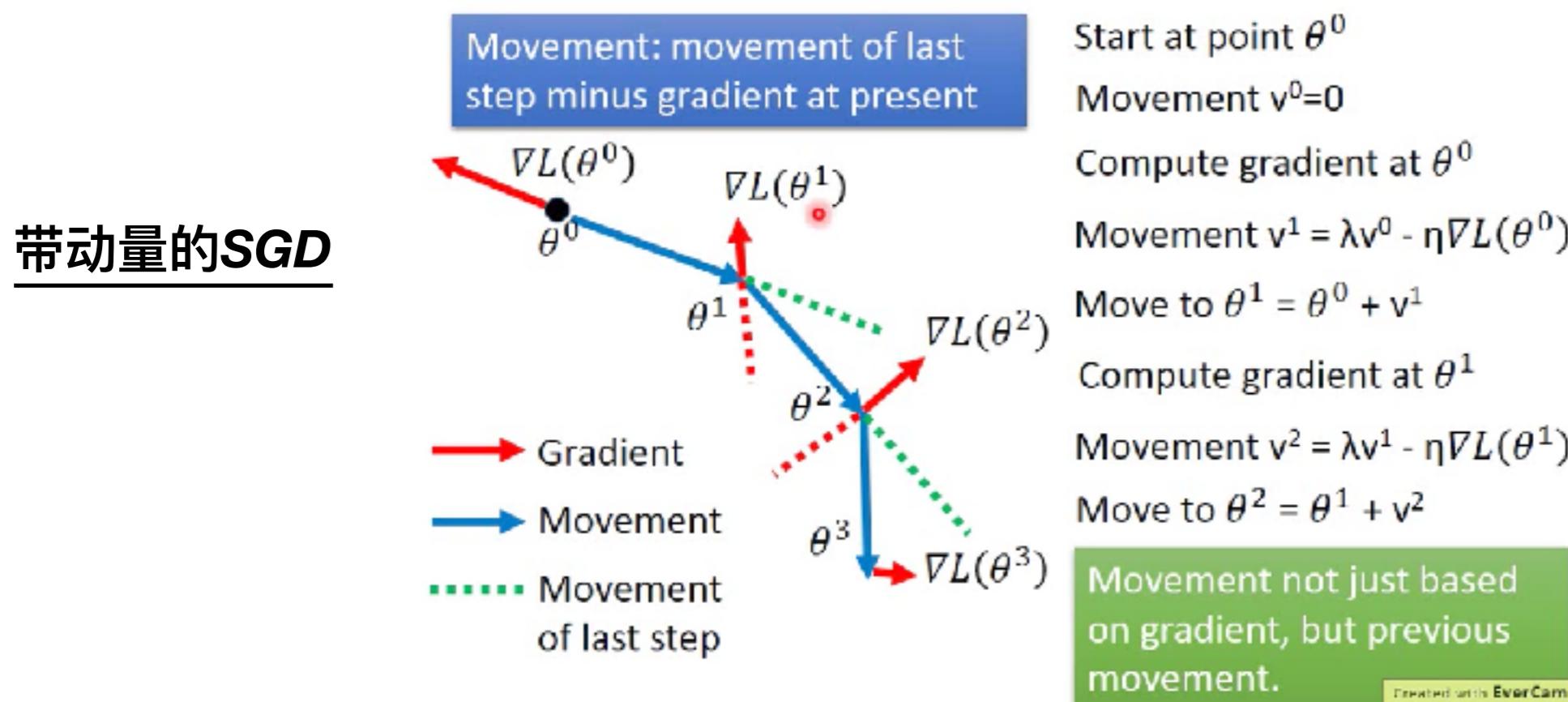
适应性的学习率

Adagrad

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sqrt{\sum_{i=0}^t (g^i)^2}} g^t$$

RMSPro

$$w^{t+1} \leftarrow w^t - \frac{\eta}{\sigma^t} g^t \quad \sigma^t = \sqrt{\alpha(\sigma^{t-1})^2 + (1-\alpha)(g^t)^2}$$



适应性的学习率——Adam算法

Adam

RMSProp + Momentum

Algorithm 1: Adam, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t .

Require: α : Stepsize

Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates

Require: $f(\theta)$: Stochastic objective function with parameters θ

Require: θ_0 : Initial parameter vector

$m_0 \leftarrow 0$ (Initialize 1st moment vector) → for momentum

$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
 $t \leftarrow 0$ (Initialize time step)

$t \leftarrow 0$ (Initialize timestep) → for RMSprop

while θ_t not converged **do**

$$t \leftarrow t + 1$$

$g_t \leftarrow \nabla_{\theta} f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
 $\hat{\sigma}_{\epsilon,t}^2 \leftarrow \hat{\sigma}_{\epsilon,t}^2 / (1 - \beta_2^t)$ (Compute bias-corrected second moment estimate)

$\hat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \bar{m}_t / (\sqrt{\bar{v}_t} + \epsilon)$ (Update parameters)

end while

return θ_t (Resulting parameters)

正则化

正则项加在数据损失函数之后，使损失函数更平滑，可以防治过拟合。

以下是几种正则化形式：

1. L2 正则化

- 最常用的正则化形式
- 分散权重向量，惩罚非常不一样的权重向量

$$\frac{1}{2} \lambda \omega^2$$

2. L1 正则化

- 使得权重向量在优化期间变得稀疏
- L2一般比L1效果要好

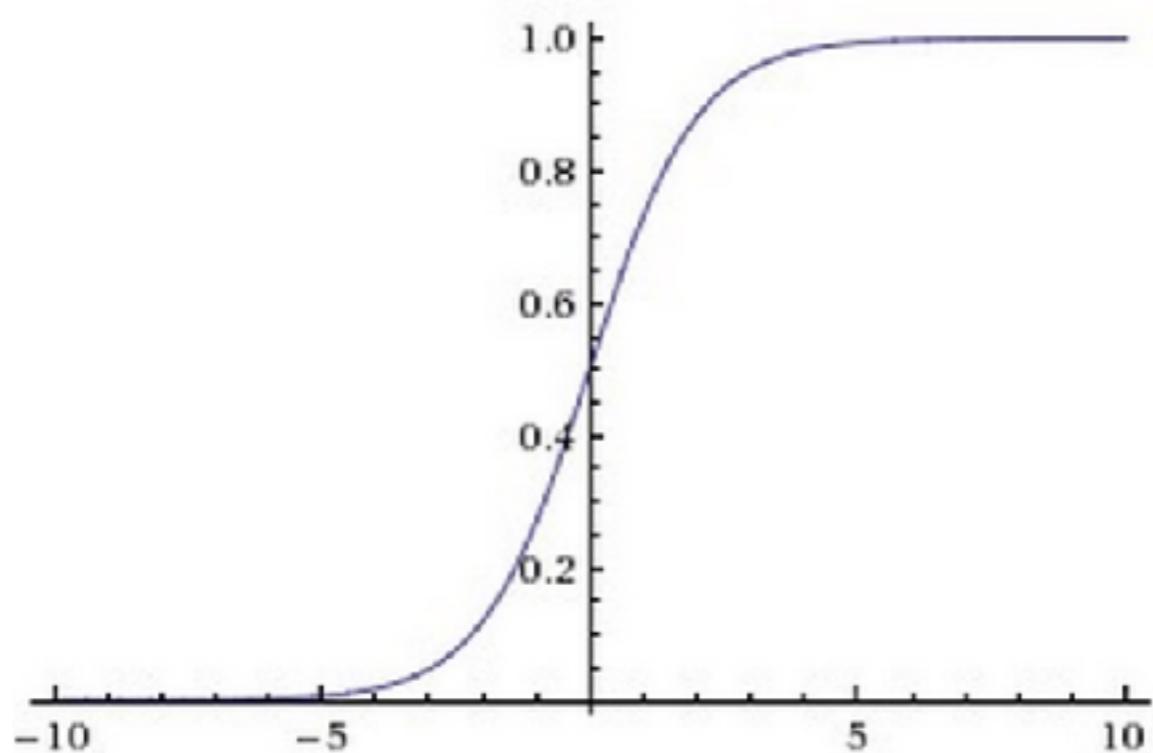
$$\lambda |\omega|$$

3. 最大范数约束

- 正常更新权重，但是限制更新后的权重大大小
- 典型的c值为3或者4
- 权重的更新更稳定

$$\|\omega\|_2 < c$$

激活函数



Sigmoid

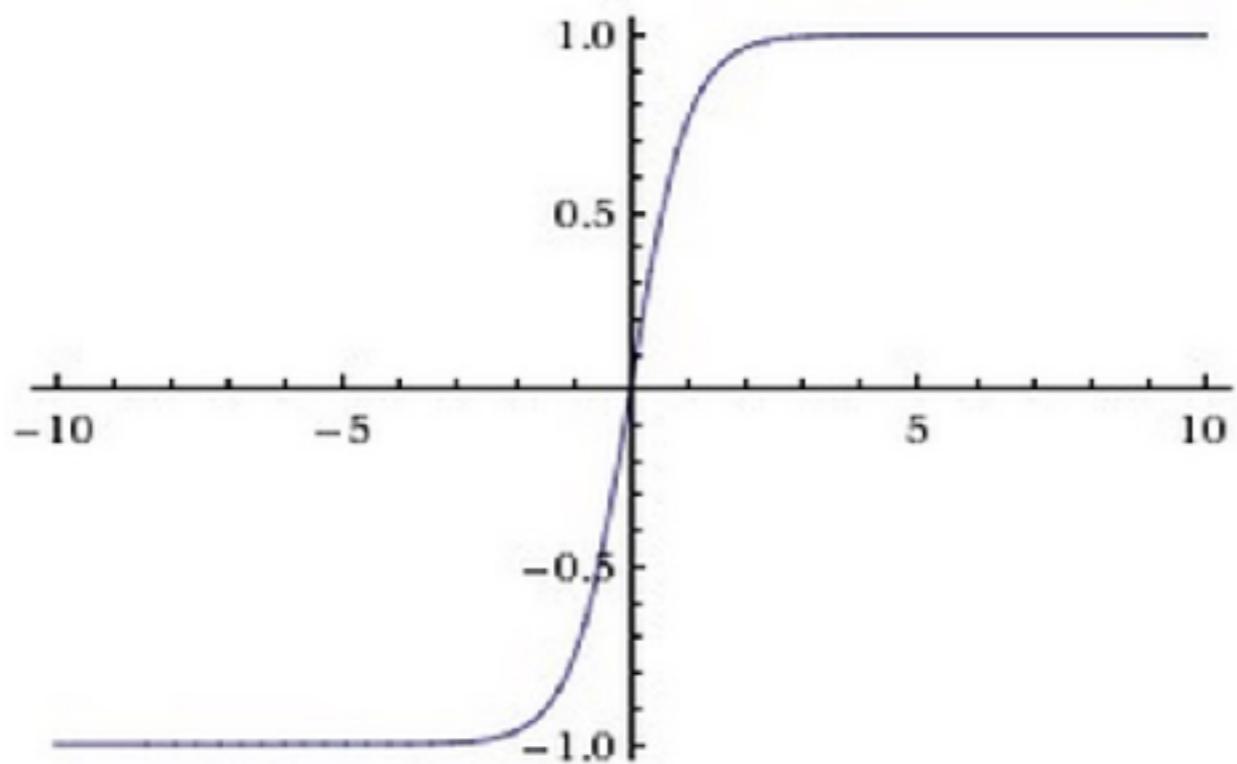
优点:

1. 将实数压扁到区间[0, 1]
2. 很好地模拟了神经元的激活状态

缺点:

1. 容易饱和，并终止梯度传递
2. sigmoid函数的输出不是零中心化的，影响梯度下降过程。

激活函数



优点：

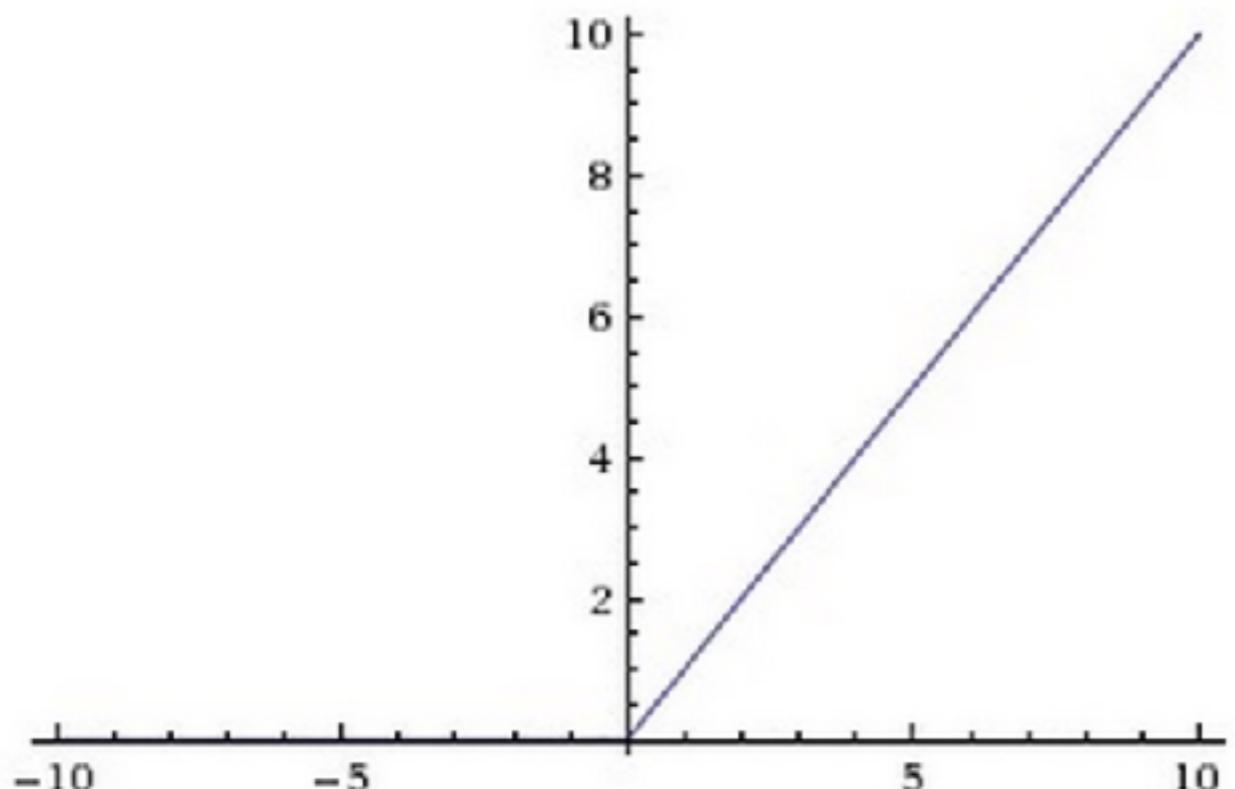
1. 将实数压扁到区间[0, 1]
2. 输出是零中心化的

缺点：

1. 容易饱和，并终止梯度传递

tanh

激活函数



ReLU

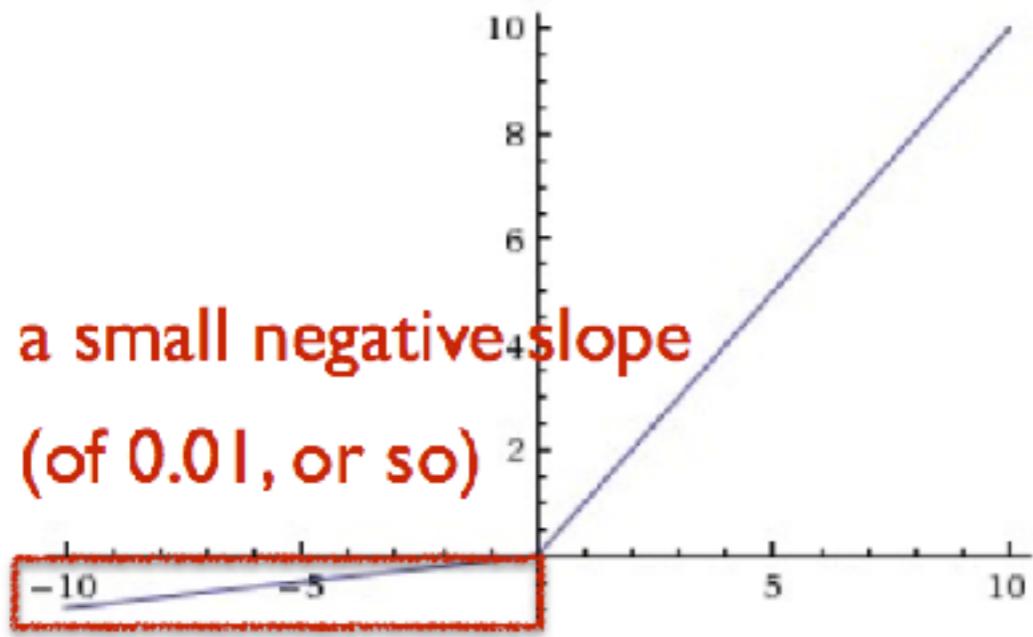
优点：

1. 计算速度快
2. 没有饱和梯度的问题
3. 能加速梯度的收敛过程

缺点：

1. ReLU单元可能会“死掉”

激活函数

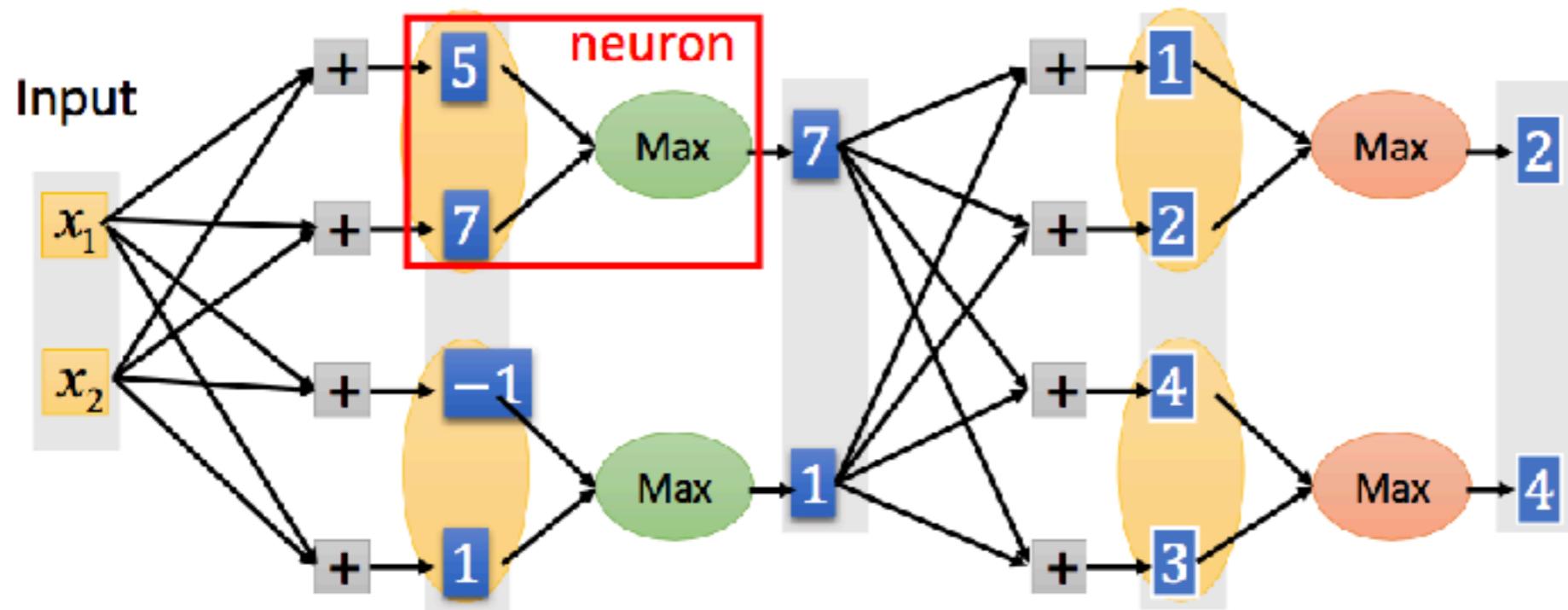


Leaky ReLU

优点：

1. 非饱和函数
2. 计算速度快
3. 比sigmoid和tanh函数收敛都快
4. 不会“杀死”传来的梯度

激活函数



Maxout “Neuron”: $\max(w_1^T x + b_1, w_2^T x + b_2)$

优点：

1. 作为ReLU和Leaky ReLU的泛化，拥有ReLU的所有优点，并没有它的缺点

缺点：

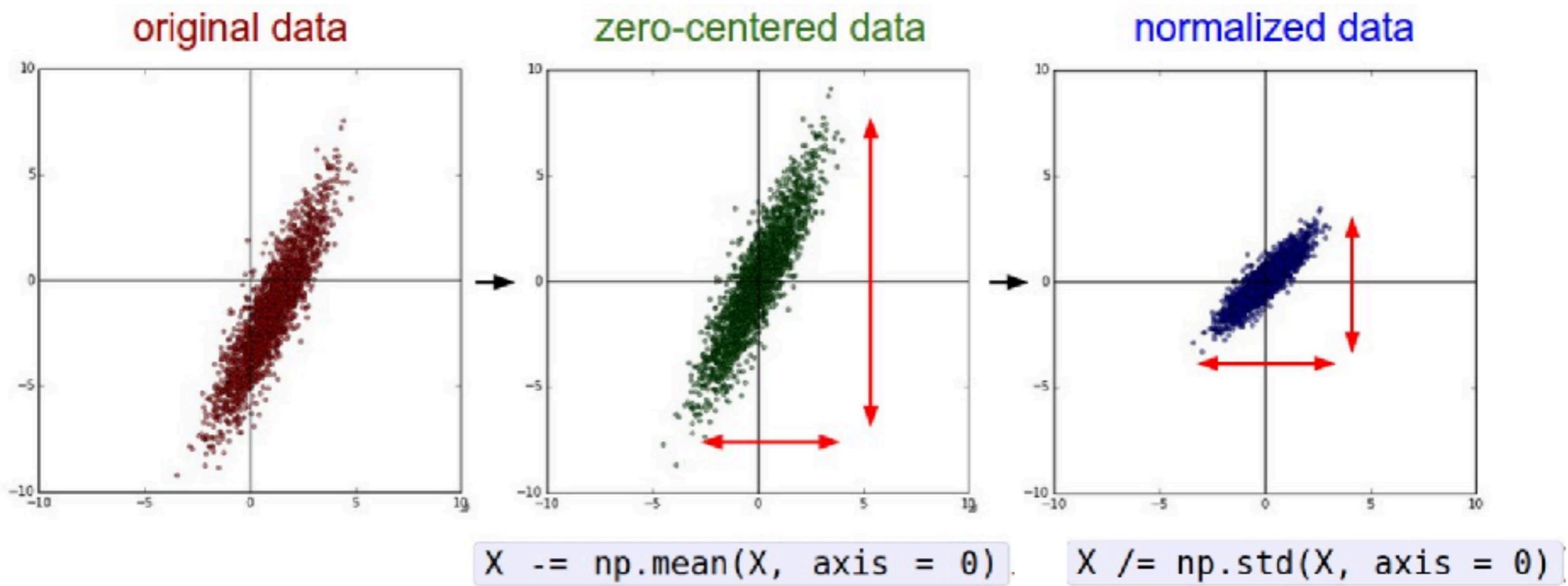
1. 参数增多

选择激活函数的正确姿势

1. 使用ReLU，但注意学习率
2. 尝试Leaky ReLU或者Maxout
3. 可以尝试用tanh，但不要期待太多
4. 不要使用sigmoid

— Advised by F.-F. Li and A. Karpathy

数据预处理



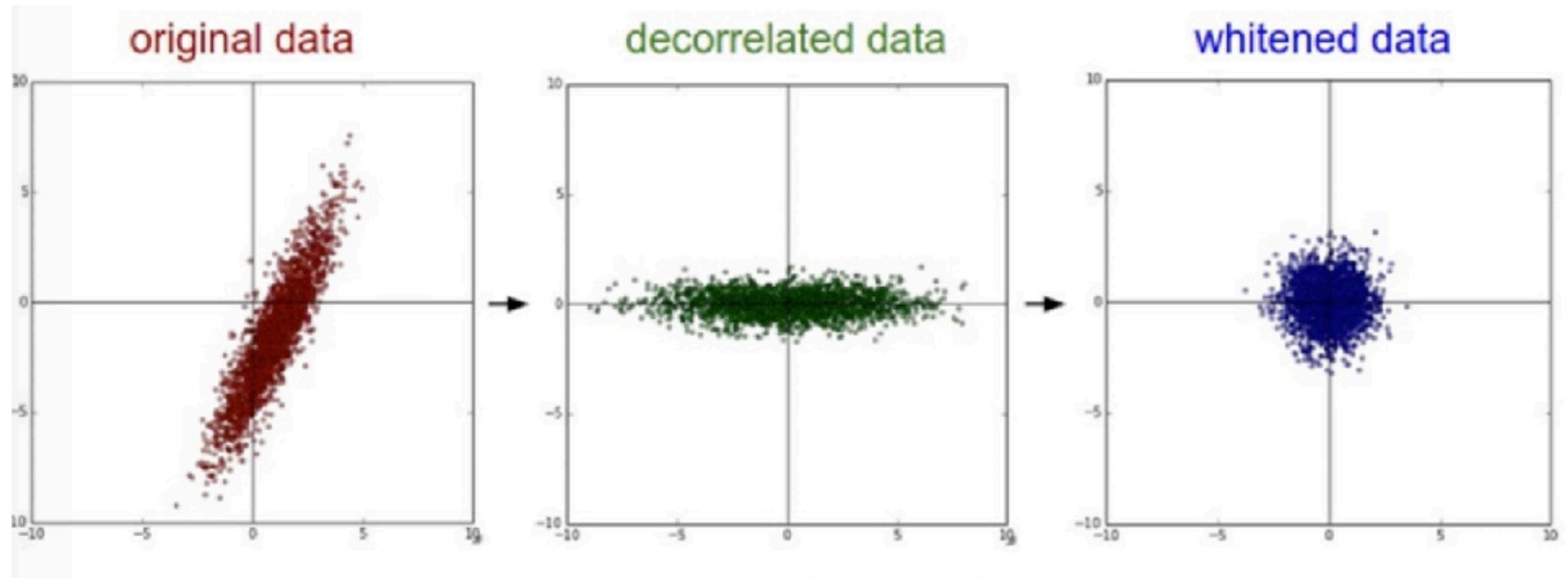
原始数据

去均值/零中心化

归一化

归一化处理使得最小值和最大值分别为-1和1，但它只对于输入特征具有不同尺度的时候才有意义。图像像素的尺度是差不多的（0~255之间）。

数据预处理



原始数据

去相关

白化

网络层大小

输入层大小（图像大小）

- 最好为2的幂，比如32, 64, 244等

Filter个数

- 最好为2的幂，比如32, 64, 244等

Minibatch大小

- 建议128，但不要太大，容易过拟合

卷积核大小

- 较小的卷积核 (3x3) 和较小的滑动步长 (1) 并进行zero-padding操作

池化层大小

- 使用大小为2x2的池化层
- max pooling比average pooling效果要好一些

参考资料

- Hung-yi Lee, “Machine Learning (2017, Spring)”,
<http://speech.ee.ntu.edu.tw/~tlkagk/courses.html>
- Xiu-Shen Wei, "Must Know Tips/Tricks in Deep Neural Networks", <http://lamda.nju.edu.cn/weixs/project/CNNTricks/CNNTricks.html>
- Deep Learning Tutorial at Stanford University,
<http://ufldl.stanford.edu/tutorial/>