机器学习
Machine Learning

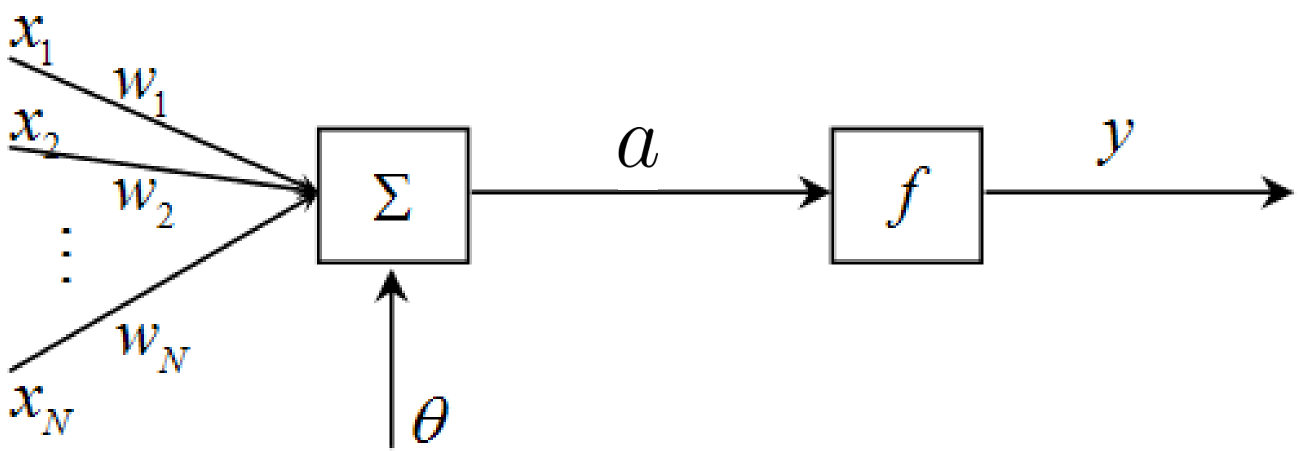
北京航空航天大学计算机学院智能识别与图像处理实验室
IRIP Lab, School of Computer Science and Engineering, Beihang University
黄 迪　刘庆杰

2018年秋季学期
Fall 2018

人工神经元的MP模型

MP模型

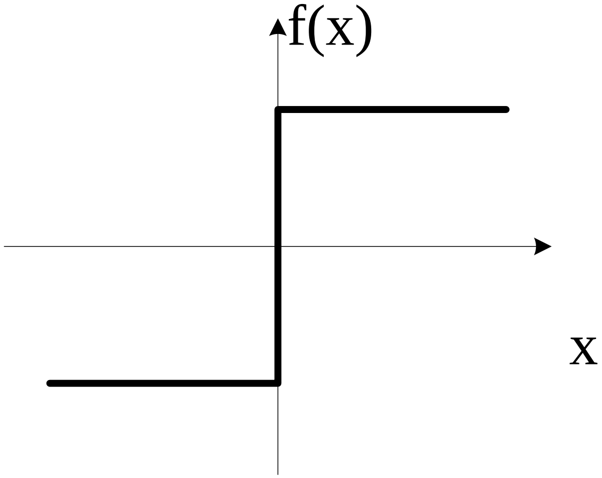
- 是一种人工神经元的数学模型，它最早是由美国的McCulloch和Pitts提出的神经元模型，是大多数神经网络模型的基础。



人工神经元的MP模型

- MP模型采用**阶跃函数**作为激活函数

$$f(x)=\begin{cases}1, & x\ge 0\\ -1, & x<0\end{cases}$$

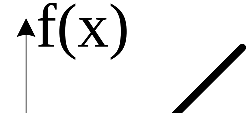


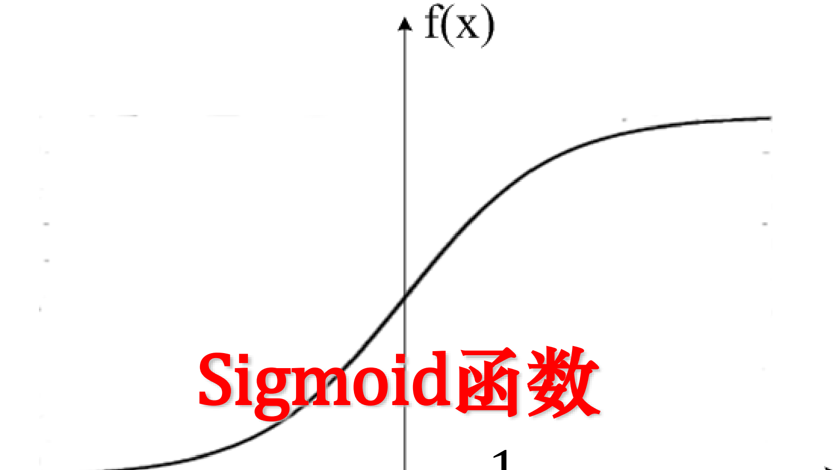
$$y=\operatorname{sgn}\left(\sum_{i=0}^{N} w_i x_i\right)$$

- 当激活函数 f 为阈值(阶跃)函数时，神经元就可以看作是一个线性分类器。

人工神经元的MP模型

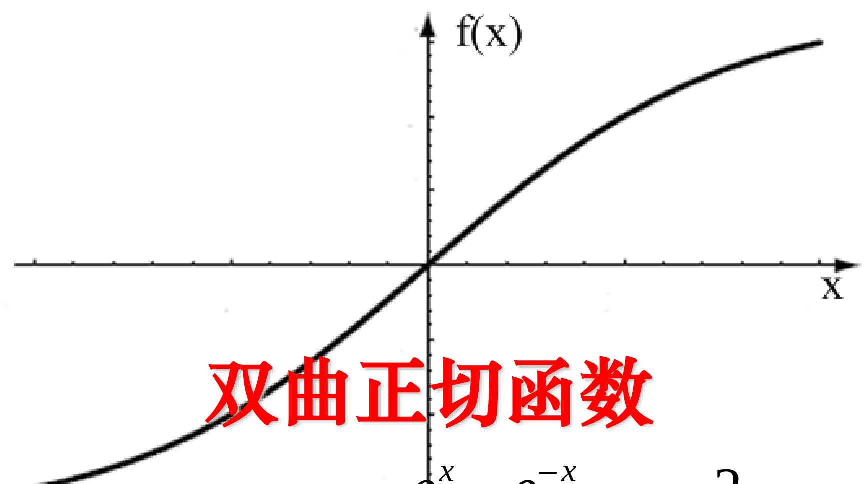
- 激活函数:
  - 线性函数
  - 非线性斜面函数(Ramp Function)
  - Sigmoid输出函数(S型函数)





Sigmoid函数

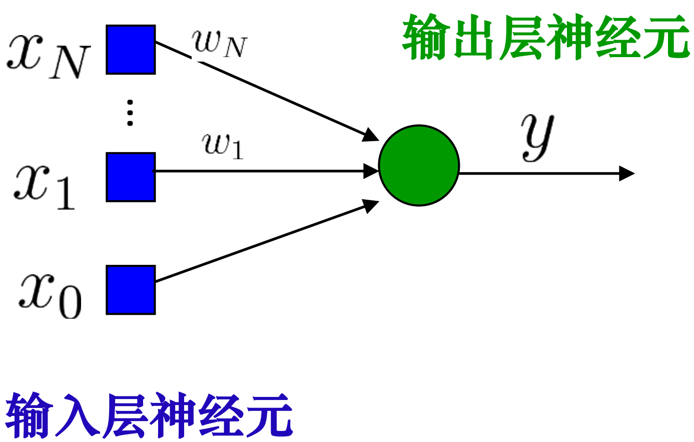
$$f(x)=\frac{1}{1+e^{-x}}$$

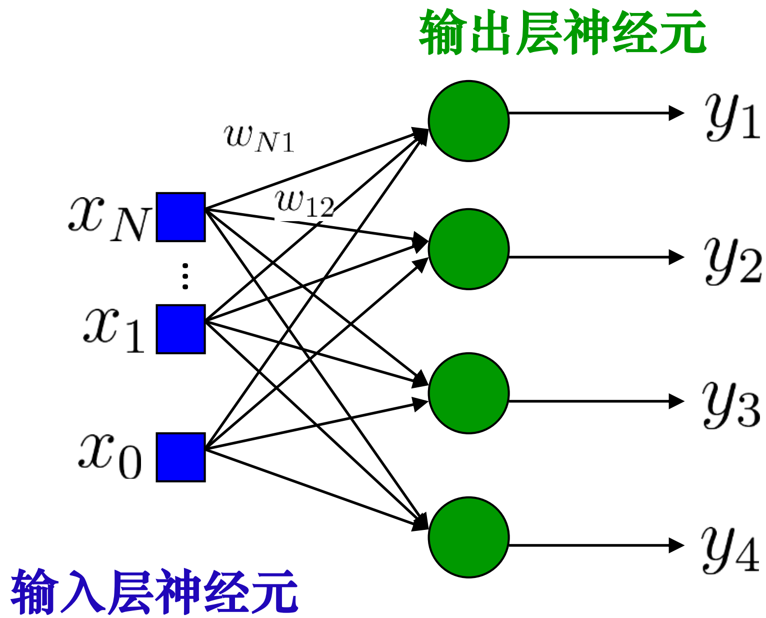


双曲正切函数

$$f(x)=\tanh(x)=\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}=\frac{2}{1+e^{-2x}}-1$$

感知器



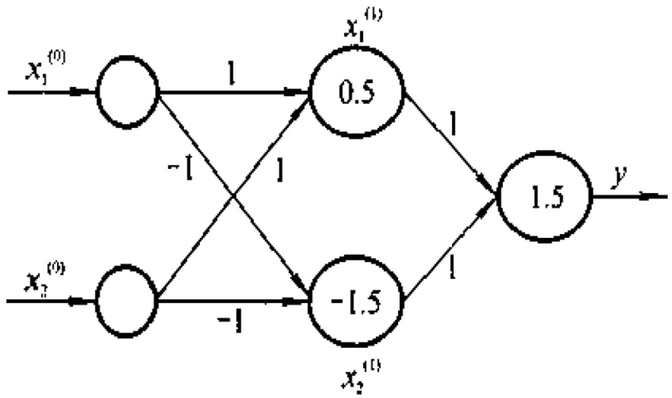


$$y = f(a) = f\left(\sum_{i=1}^{N} w_i x_i\right)$$

$$y_j = f(a_j) = f\left(\sum_{i=1}^{N} w_{ij} x_i\right)$$

多层感知器

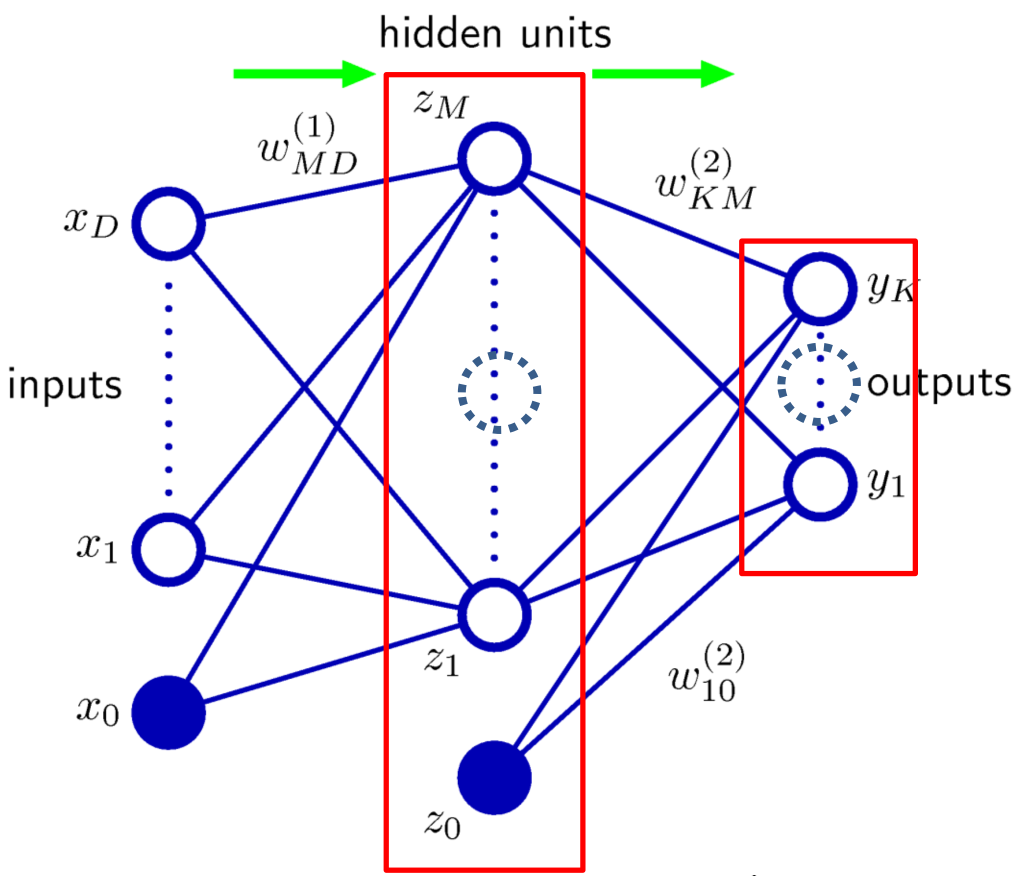
- 对于异或问题，用一个简单的三层感知器就可得到解决。



x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0

理论证明，三层感知器可以实现任意的逻辑运算，在激活函数为Sigmoid函数的情况下，可以逼近任何非线性多元函数。

前馈神经网络



$$a_j = \sum_{i=0}^{D} w_{ji}^{(1)} x_i$$

$$z_j = h(a_j)$$

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j$$

$$y_k = \sigma(a_k)$$

$$\sigma(a) = \frac{1}{1+\exp(-a)}$$

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{i=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right)$$

如何训练神经网络?

- 定义准则函数

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2$$

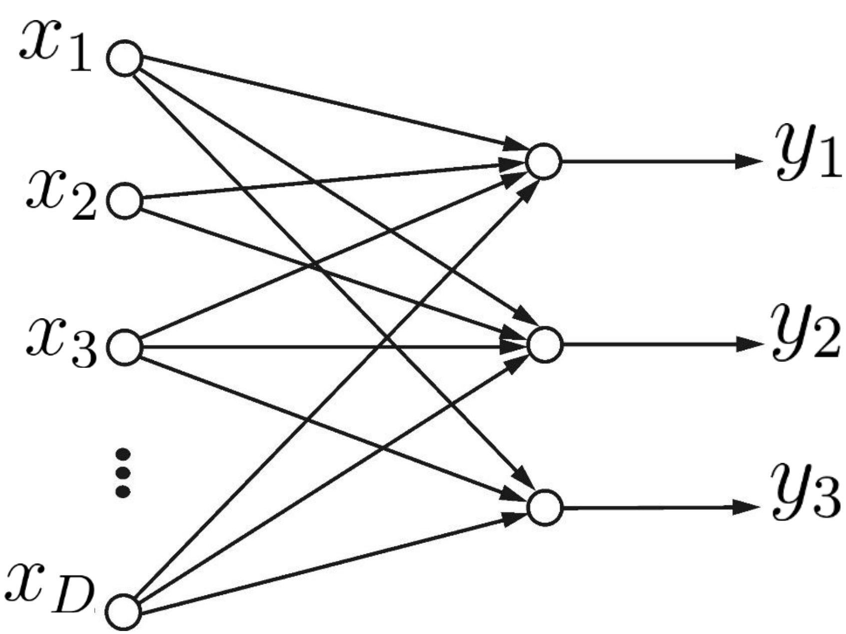
寻找一个 $\mathbf{w}$，使得 $E(\mathbf{w})$最小。

$$\nabla E(\mathbf{w}) = 0$$

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

BP反向传播算法

- 样本集 $\{\mathrm{x}_n, \mathbf{t}_n\}_{n=1}^{N}, \mathrm{x} \in \mathbb{R}^D, \mathbf{t} \in \mathbb{R}^3$



$$y_k = \sum_{i=1}^{D} w_{ki} x_i$$

$$E_n(\mathrm{w}) = \frac{1}{2} \sum_{k=1}^{3} (y_{nk} - t_{nk})^2$$

$$\Rightarrow \frac{\partial E_n}{\partial w_{ki}} = (y_{nk} - t_{nk}) x_{ni}$$

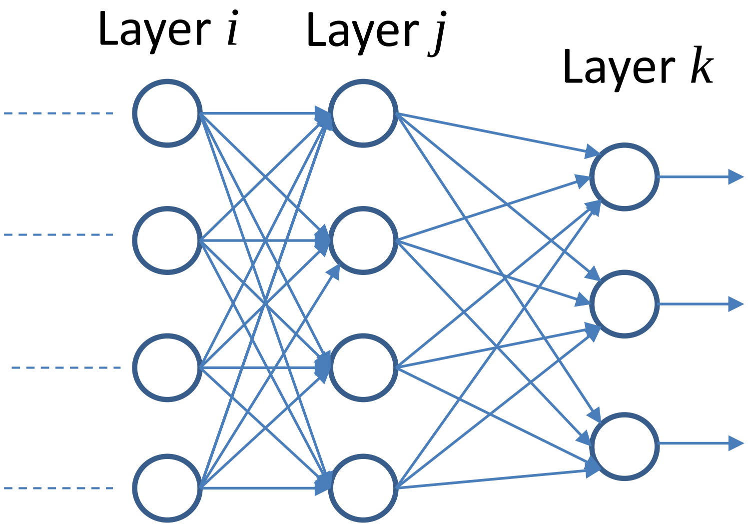
BP反向传播算法

- **练习：** 对非线性 $y_k = h(a_k) = h(\sum_i w_{ki} x_i)$

计算 $\dfrac{\partial E_n}{\partial w_{ki}}$ $\quad E_n = \frac{1}{2} \sum_k (y_k - t_k)^2$

$$
\begin{aligned}
\frac{\partial E_n}{\partial w_{ki}} &= \frac{\partial E_n}{\partial y_k} \frac{\partial y_k}{\partial a_k} \frac{\partial a_k}{\partial w_{ki}} \\
&= (y_k - t_k) \frac{\partial y_k}{\partial a_k} \frac{\partial a_k}{\partial w_{ki}} \\
&= (y_k - t_k) h'(a_k) \frac{\partial a_k}{\partial w_{ki}} \\
&= (y_k - t_k) h'(a_k) x_i
\end{aligned}
$$

BP反向传播算法



$$a_j = \sum_i w_{ji} z_i \qquad z_j = h(a_j)$$

$$\frac{\partial E_n}{\partial w_{ji}} = \boxed{\frac{\partial E_n}{\partial a_j}} \frac{\partial a_j}{\partial w_{ji}}$$

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} \qquad \frac{\partial a_j}{\partial w_{ji}} = z_i$$

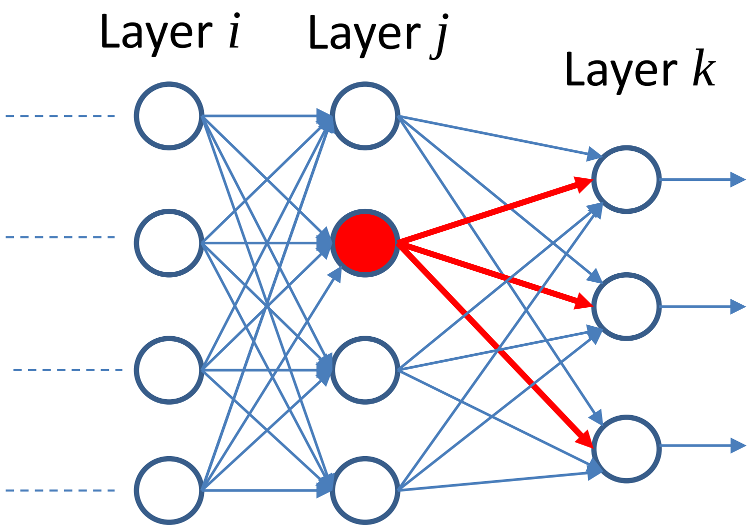
$$\Rightarrow \quad \frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$$

BP反向传播算法

对于输出层而言： $\delta_k = y_k - t_k$

对于隐藏层而言： $\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j}$

$$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k$$



BP反向传播算法

- **练习：** $a_k = \sum_j w_{kj} z_j \qquad z_j = h(a_j)$

推导 $\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$

$$\delta_j = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j} = \sum_k \delta_k \frac{\partial a_k}{\partial a_j}$$

$$= \sum_k \delta_k \frac{\partial a_k}{\partial z_j} \frac{\partial z_j}{\partial a_j} = \sum_k \delta_k w_{kj} \frac{\partial h(a_j)}{\partial a_j}$$

$$= h'(a_j) \sum_k \delta_k w_{kj}$$

BP反向传播算法

- 初始化权重w_{ij}
- 对于输入的训练样本，求取每个节点输出和最终输出层的输出值
- 对输出层求取 $\delta_k = y_k - t_k$
- 对于隐藏层求取 $\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$
- 求取输出误差对于每个权重的梯度 $\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i$
- 更新权重 $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} - \eta \Delta E(\mathbf{w}^{(\tau)})$

BP反向传播算法

- 多层前馈网络的学习算法比较复杂，其主要困难是中间的隐层不直接与外界连接，无法直接计算其误差。

- 反向逐层传播输出层的误差，以间接算出隐层误差。算法分为两个阶段：
  - 第一阶段(正向传播过程)输入信息从输入层经隐层逐层计算各单元的输出值；
  - 第二阶段(反向传播过程)由输出误差逐层向前算出隐层各单元的误差，并用此误差修正前层权值。

第六讲：人工神经网络（续）

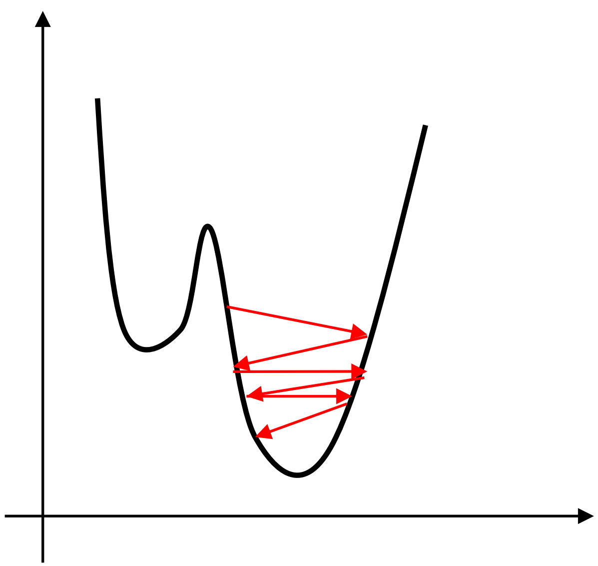
Chapter 6: Artificial Neural Networks

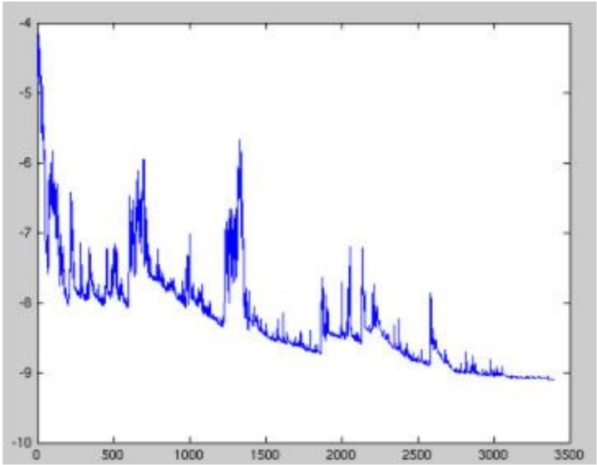
随机梯度下降

- 每个训练轮次使用单个样本的梯度进行参数更新，叫**随机梯度下降**（Stochastic gradient descent，SGD）
  - ➢ 引起梯度震荡
  - ➢ 相似样本，冗余计算

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} - \eta \Delta E(\boxed{\mathrm{x}_i}; \mathbf{w}^{(\tau)})$$

一个样本





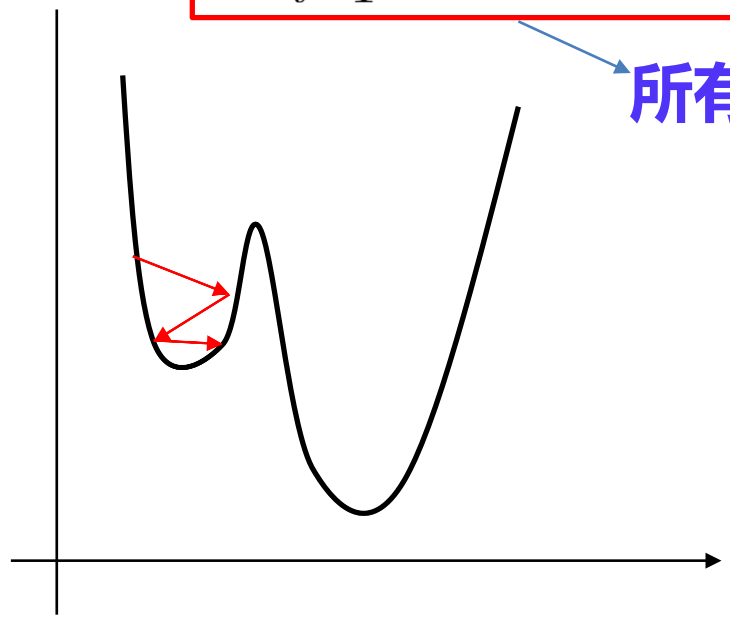
批梯度下降

- 使用所有样本的梯度和进行参数更新，叫**批梯度下降**（Batch gradient descent，BGD）

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} - \eta \boxed{\frac{1}{N}\sum_{i=1}^{N} \Delta E(\mathrm{x}_i; \mathbf{w}^{(\tau)})}$$

所有样本

  - 计算量大
  - 易陷入局部极小值



小批量梯度下降

- 使用一部分样本梯度和进行参数更新，叫**小批量梯度下降**（Mini-Batch gradient descent，MBGD）

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} - \eta \boxed{\frac{1}{m}\sum_{i=1}^{m} \Delta E(\mathrm{x}_i; \mathbf{w}^{(\tau)})}$$

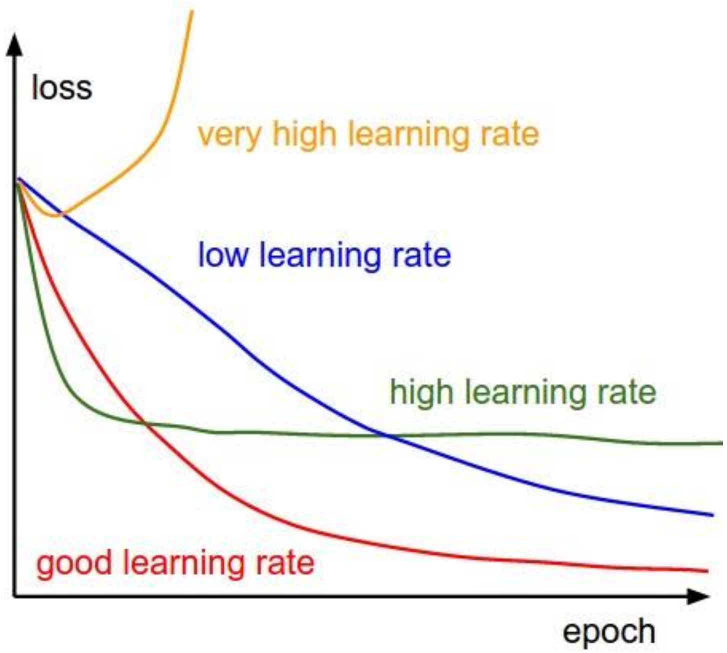
部分样本

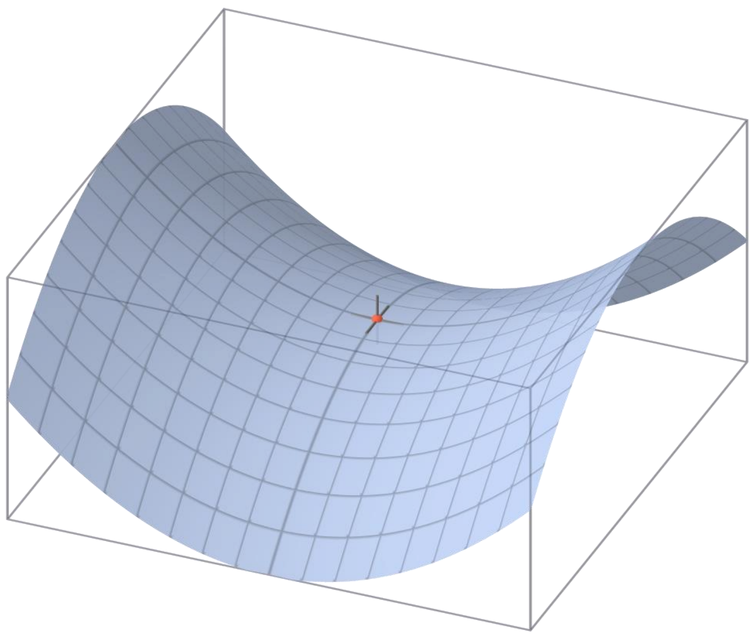
- 计算小批量数据的梯度更加高效
- 收敛更稳定
- 样本使用更灵活

目前最常用的策略。在使用中，**这种方法通常被称为SGD**。

SGD的缺点

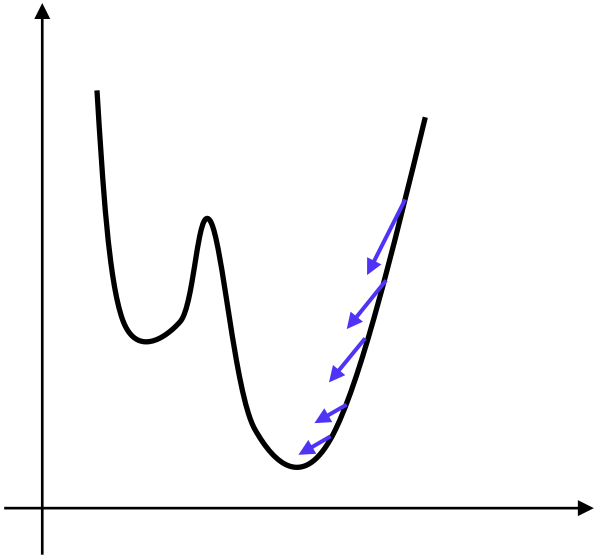
- 学习率难以选择
- 对于非凸函数，容易陷入到局部极小值和鞍点





可变学习率

- 固定学习率不是一个好的选择
- 学习的不同阶段梯度不同



- 自适应学习率
  - 线性衰减

    $$\eta^k = \alpha\eta^{k-1} \quad \alpha = 0.5,\ 0.2,\ 0.1,\ \cdots$$

  - 指数衰减

    $$\eta^k = \eta^0 e^{-kt}$$

  - …

Adagrad

- 对稀疏参数进行大幅更新和对频繁参数进行小幅更新
- 适合处理稀疏数据

[redacted]

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} - \eta^{\tau} \Delta E(\mathbf{w}^{(\tau)})$$

RMSprop

- Adagrad引起学习率衰减

[redacted]

学习率衰减

- 减弱梯度累积

$$\boxed{\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} - \frac{\eta^{0}}{\sqrt{g^{(\tau)}} + \epsilon} \Delta E(\mathbf{w}^{(\tau)})}$$

Adadelta

- 使用前一次的梯度开方 $\sqrt{\mathbf{w}^{(\tau-1)}}+\epsilon$ 代替 η^0

[redacted]

$$\mathbf{w}^{(\tau+1)}=\mathbf{w}^{\tau}-\frac{\sqrt{\mathbf{w}^{(\tau-1)}}+\epsilon}{\sqrt{g^{(\tau)}}+\epsilon}\Delta E(\mathbf{w}^{(\tau)})$$

动量SGD

- SGD在遇到局部极值和鞍点时容易震荡
- 引入动量momentum，抑制梯度的震蒎

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} - \eta \boxed{\frac{1}{m}\sum_{i=1}^{m} \Delta E(\mathrm{x}_i; \mathbf{w}^{(\tau)})}$$

$$\Big\Downarrow$$

$$\nabla_{\mathbf{w}} E^{(\tau)}$$

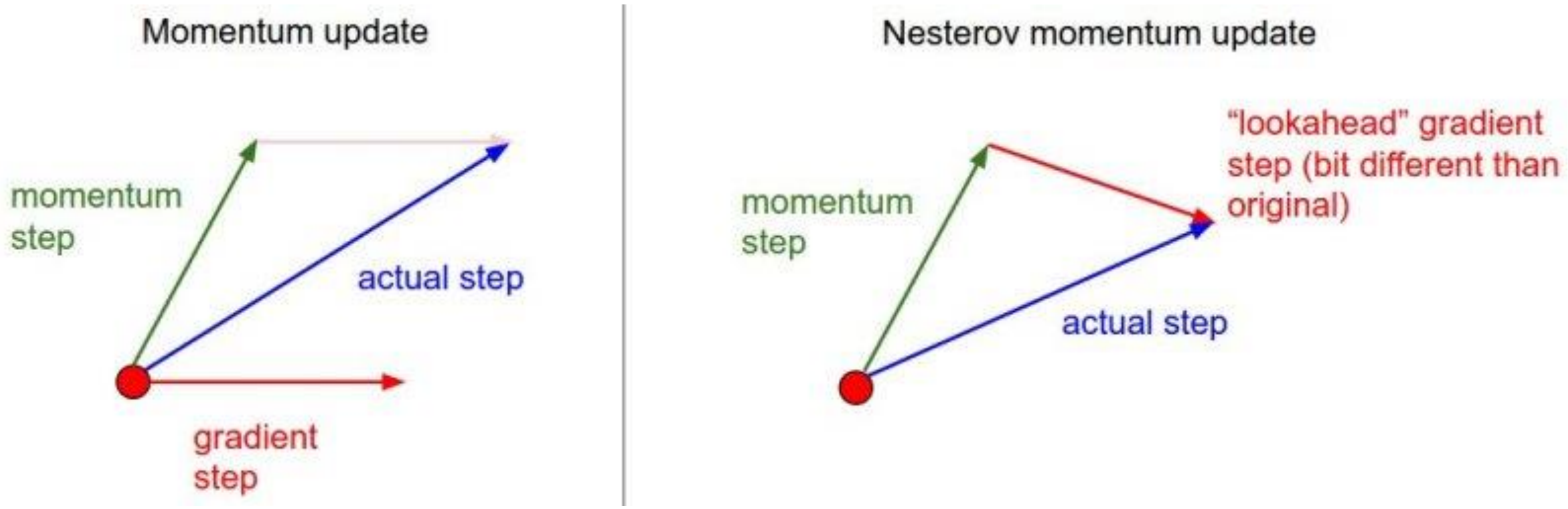
$g^{(\tau)}$ **表示** τ **时刻的优化方向，且** $g^{(0)} = \nabla_{\mathbf{w}} E^{(0)}$

τ **时刻的优化方向为：** $g^{(\tau)} = \alpha g^{(\tau-1)} + \eta \nabla_{\mathbf{w}} E^{(\tau)}$

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} - g^{\tau}$$

Nesterov梯度（NAG）

- 具有一定的预测性



Adam

- **最常用的方法**
- **Adaptive + momentum**

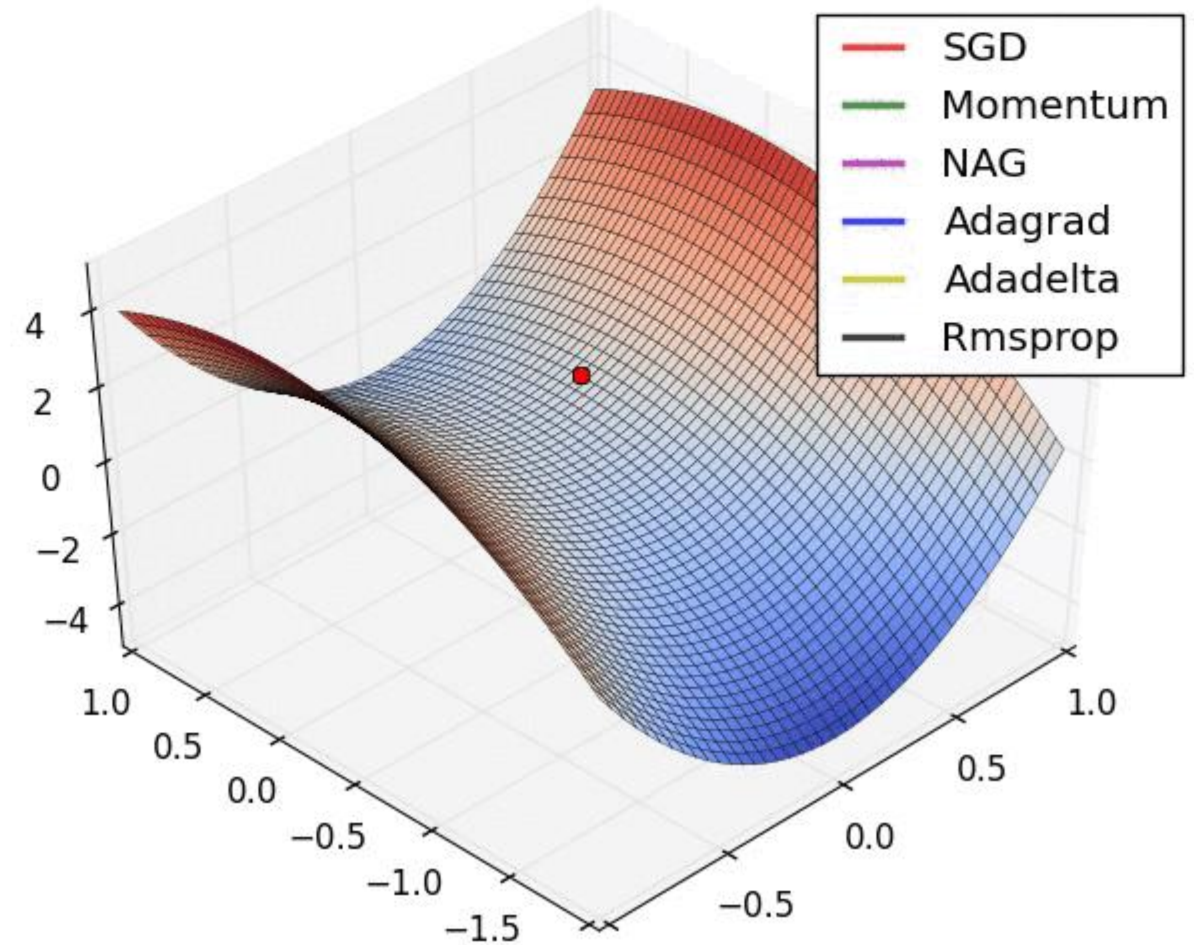
$$m^{(\tau)} = \alpha \cdot m^{(\tau-1)} + (1-\alpha) \cdot \nabla_{\mathbf{w}} E^{(\tau)}$$

$$n^{(\tau)} = \beta \cdot n^{(\tau-1)} + (1-\beta) \cdot [\nabla_{\mathbf{w}} E^{(\tau)}]^2$$

$$\hat{m}^{(\tau)} = \frac{m^{\tau}}{1-u^{\tau}} \qquad \hat{n}^{(\tau)} = \frac{m^{\tau}}{1-v^{\tau}}$$

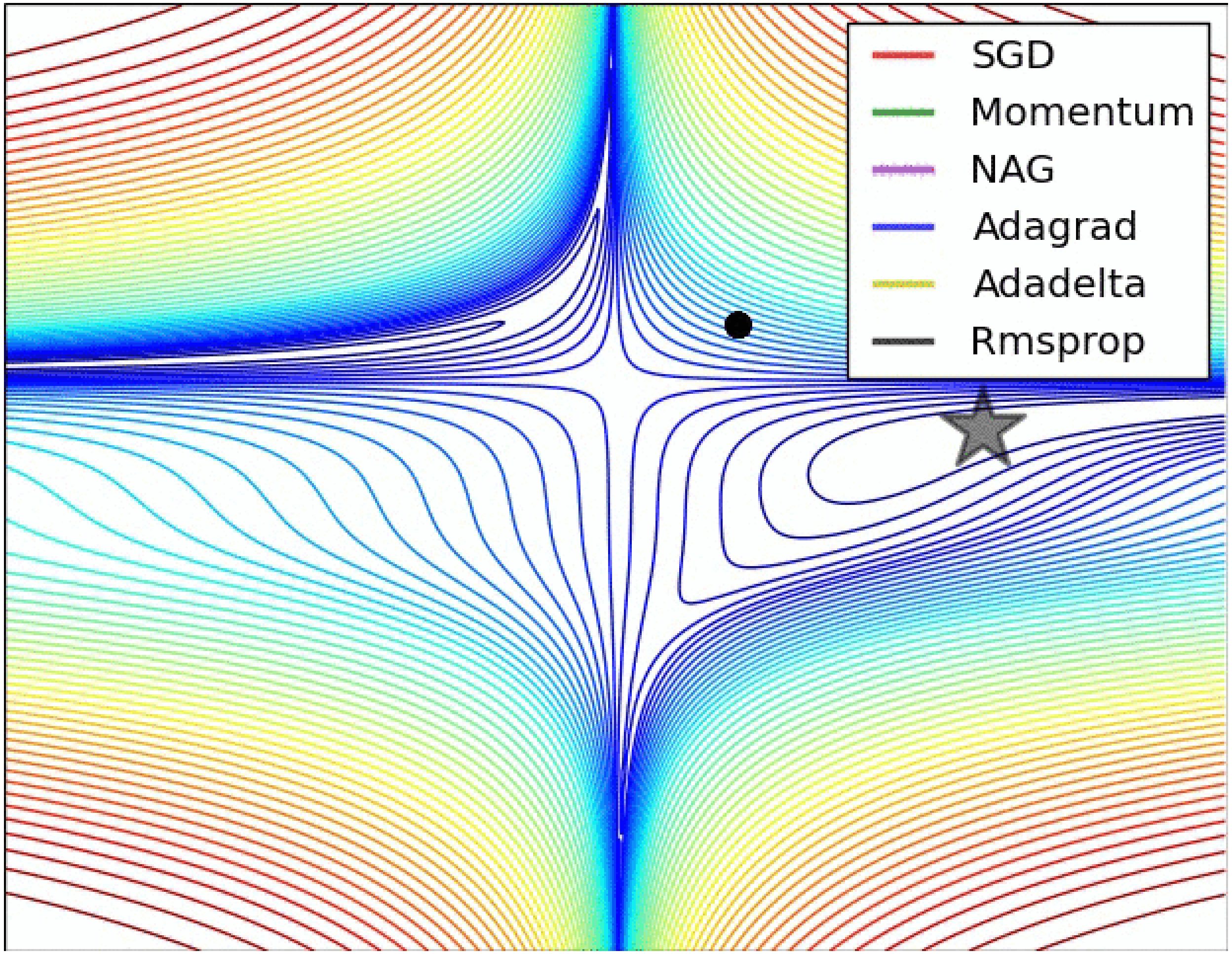
$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{\tau} - \frac{1}{\sqrt{n^{\tau}} + \epsilon} \hat{m}^{\tau}$$

各种优化的对比



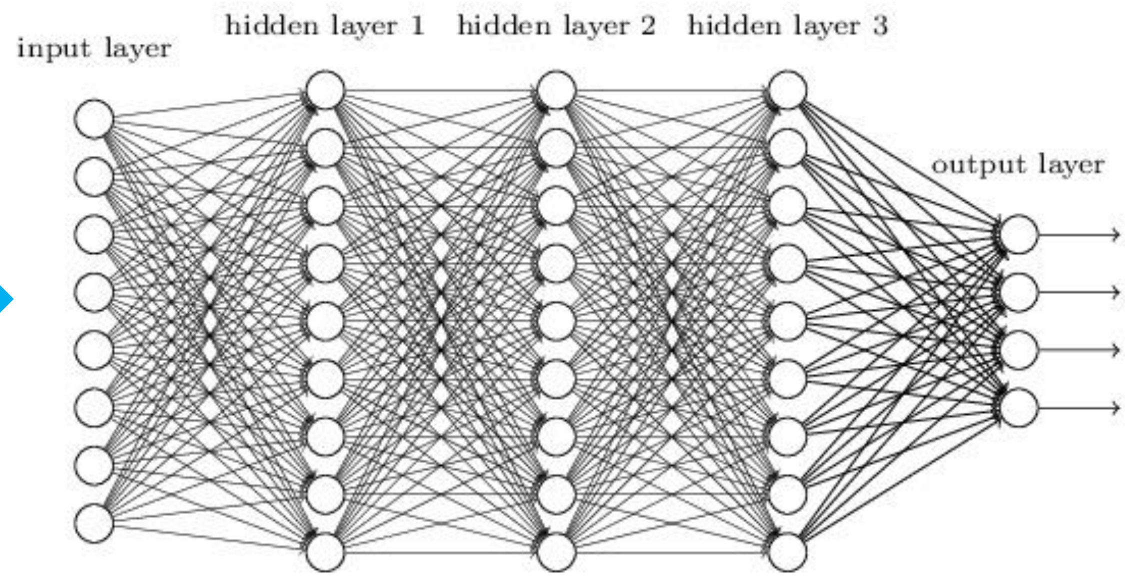
SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
4
2
0
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.5
1.0
0.5
0.0
−0.5

各种优化的对比



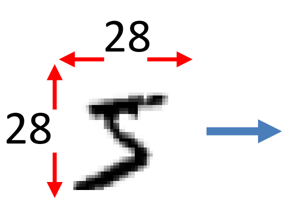
梯度消失问题

MNIST



0
1
2
3
4
5
6
7
8
9

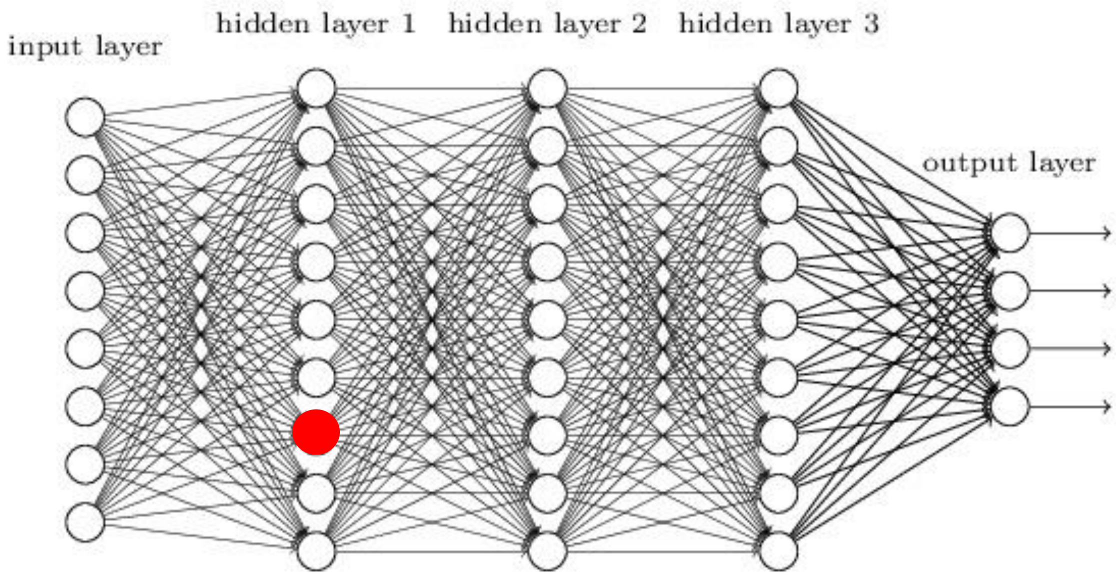
784 30 30 30 10 输出



28×28=784维

部分内容来源于：http://neuralnetworksanddeeplearning.com

梯度消失问题



隐藏层神经元： $neuro_i^l = \sigma(\mathrm{w}_i^l \mathrm{x}_i^l + \mathrm{b}_i^l)$

梯度为：$\delta_i^l = \frac{\partial E}{\partial b_i^l}$ ➡ $||\delta^l||$ 表示l层的学习速度

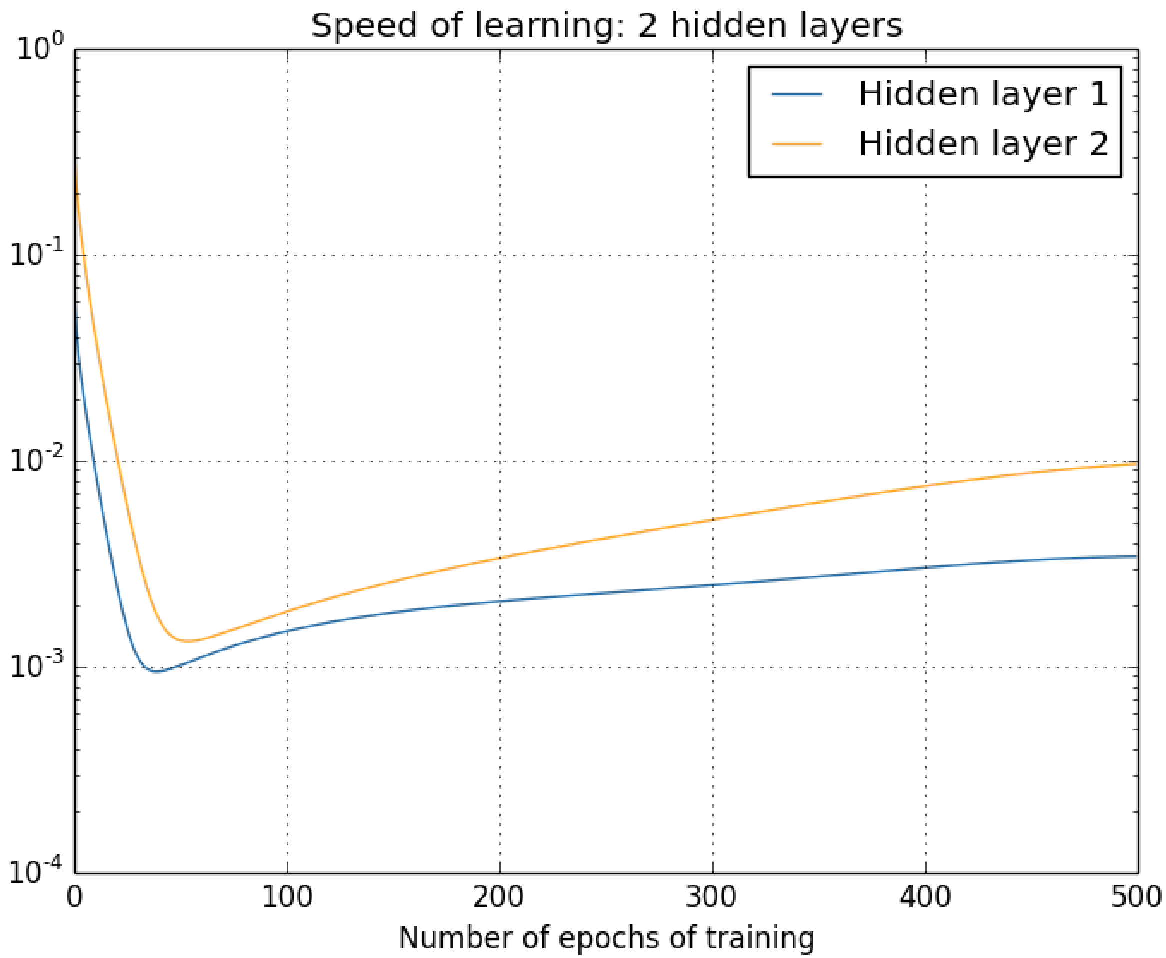
梯度消失问题

- 在MNIST数据集上对[728 30 30 10]的网络进行训练

初始化：

$$W \sim \mathcal{N}(0, 0.2)$$

$$\eta = 0.1$$



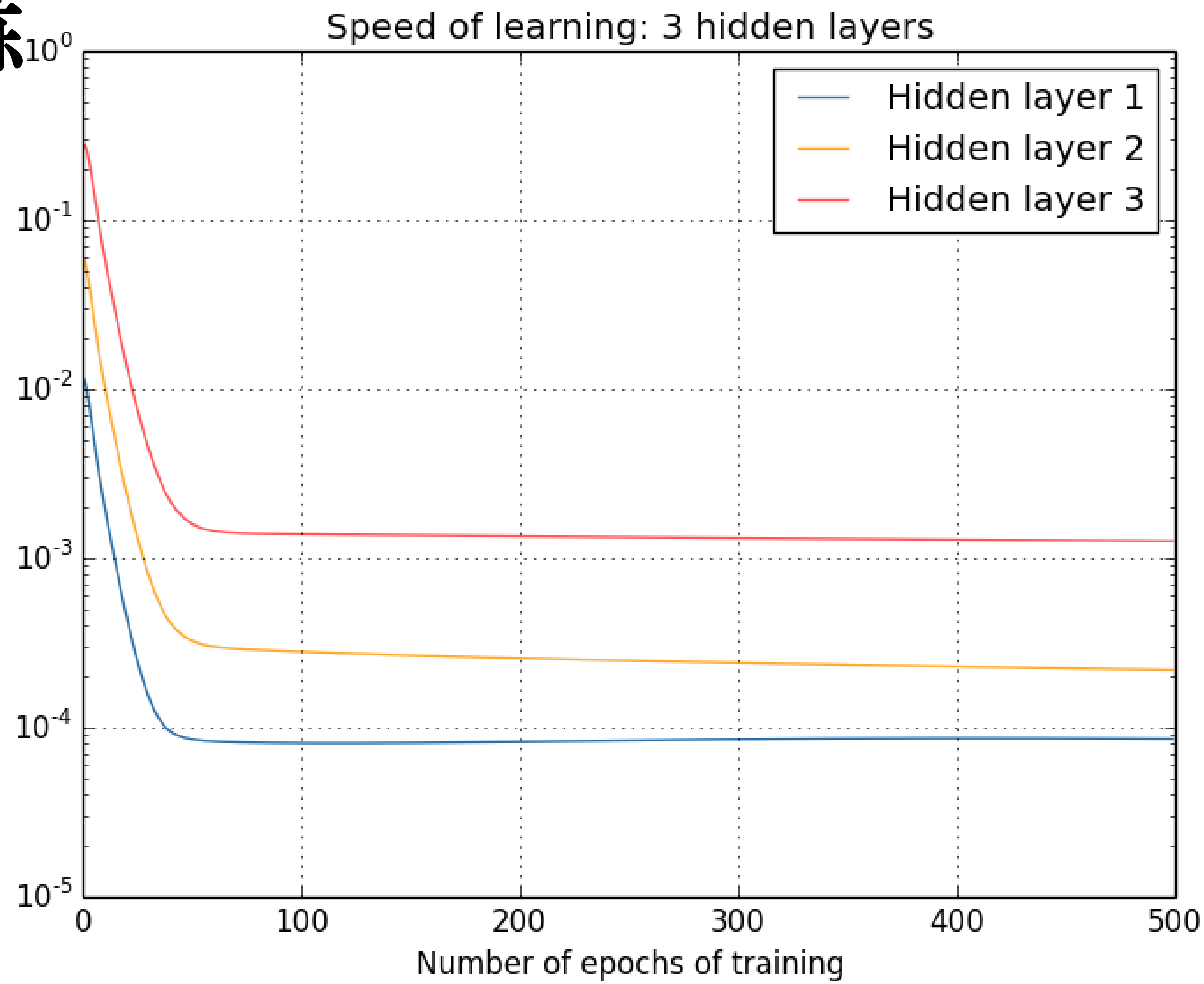
梯度消失问题

- 在MNIST数据集上对[728 30 30 30 10]的网络进行训练

初始化：

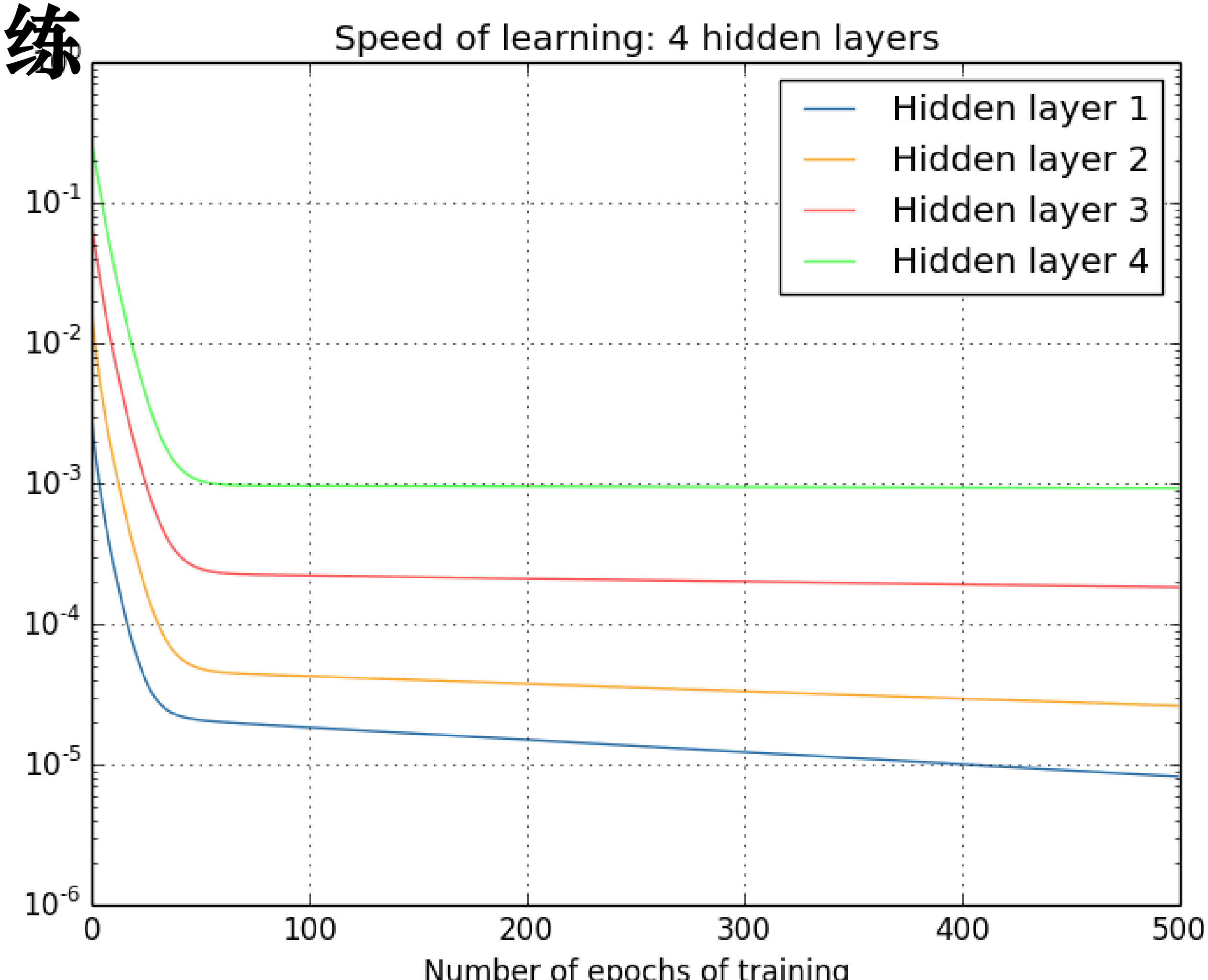
$$W \sim \mathcal{N}(0, 0.2)$$

$$\eta = 0.1$$



梯度消失问题

- 在MNIST数据集上对[728 30 30 30 30 10]的网络进行训练



初始化：

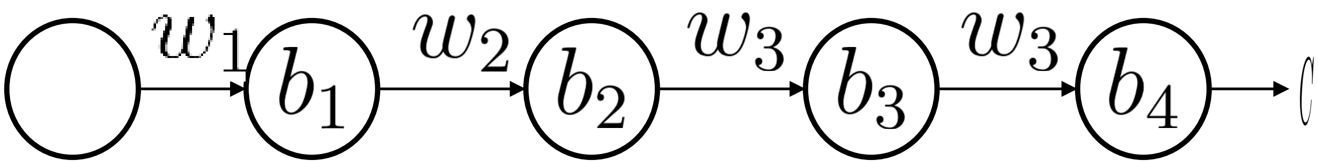
$$W \sim \mathcal{N}(0, 0.2)$$

$$\eta = 0.1$$

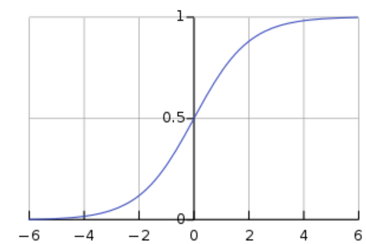
梯度消失问题

- 梯度消失的起因

考虑一个每层只有一个神经元的多层神经网络



$$y_i = \sigma(z_i) = \sigma(w_i x_i + b_i) \quad \sigma - \text{sigmoid}$$



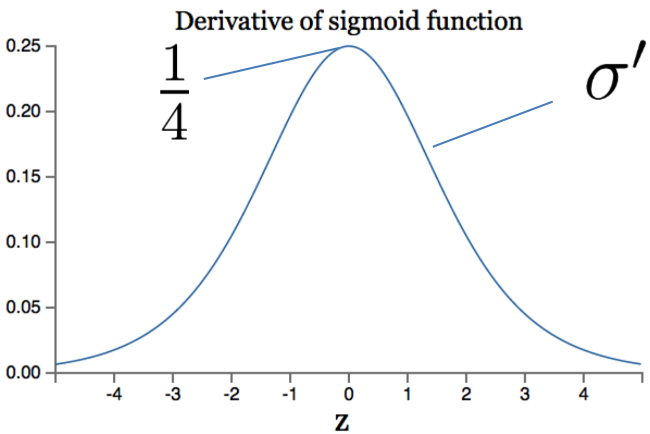
可以推出：

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial y_4}\frac{\partial y_4}{\partial z_4}\frac{\partial z_4}{\partial x_4}\frac{\partial x_4}{\partial z_3}\frac{\partial z_3}{\partial x_3}\frac{\partial x_3}{\partial z_2}\frac{\partial z_2}{\partial x_2}\frac{\partial x_2}{\partial z_1}\frac{\partial z_1}{\partial b_1}$$
$$= \frac{\partial C}{\partial y_4}\sigma'(z_4)\, w_4 \sigma'(z_3)\, w_3 \sigma'(z_2)\, w_2 \sigma'(z_1)$$

梯度消失问题

- **梯度消失的起因**

$$
\begin{aligned}
\frac{\partial C}{\partial b_1} &= \frac{\partial C}{\partial y_4}\frac{\partial y_4}{\partial z_4}\frac{\partial z_4}{\partial x_4}\frac{\partial x_4}{\partial z_3}\frac{\partial z_3}{\partial x_3}\frac{\partial x_3}{\partial z_2}\frac{\partial z_2}{\partial x_2}\frac{\partial x_2}{\partial z_1}\frac{\partial z_1}{\partial b_1} \\
&= \frac{\partial C}{\partial y_4}\sigma'\left(z_4\right)w_4\sigma'\left(z_3\right)w_3\sigma'\left(z_2\right)w_2\sigma'\left(z_1\right)
\end{aligned}
$$



$$
\left.\begin{aligned} \sigma'(z) &\le \frac{1}{4} \\ |w| &< 1 \end{aligned}\right\} \Rightarrow |\sigma'(z)w| \le \frac{1}{4}
$$

梯度消失问题

- 梯度消失的起因

[redacted]

$$|w| < 1$$

$\Rightarrow$ [redacted]

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

common terms

$$\frac{\partial C}{\partial b_3} = \sigma'(z_3)\overbrace{w_4\sigma'(z_4)\frac{\partial C}{\partial a_4}}$$

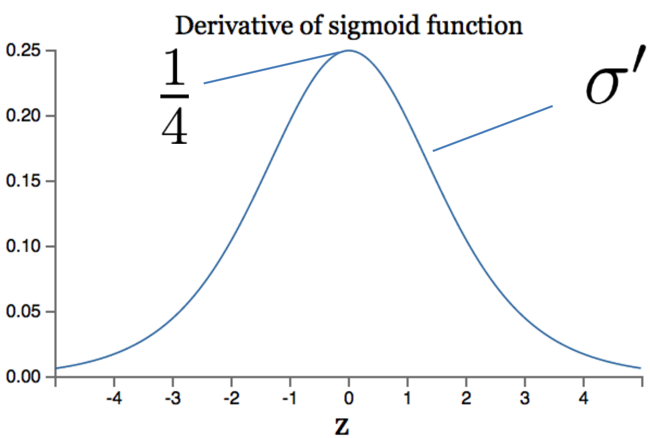
$$\frac{\partial C}{\partial b_1} < \frac{1}{16}\frac{\partial C}{\partial b_3} \ll \frac{\partial C}{\partial b_n}, \quad n > 3$$

梯度消失问题

- 梯度消失的起因

$$\frac{\partial C}{\partial b_1}=\frac{\partial C}{\partial y_4}\frac{\partial y_4}{\partial z_4}\frac{\partial z_4}{\partial x_4}\frac{\partial x_4}{\partial z_3}\frac{\partial z_3}{\partial x_3}\frac{\partial x_3}{\partial z_2}\frac{\partial z_2}{\partial x_2}\frac{\partial x_2}{\partial z_1}\frac{\partial z_1}{\partial b_1}$$

$$=\frac{\partial C}{\partial y_4}\sigma'\left(z_4\right)w_4\sigma'\left(z_3\right)w_3\sigma'\left(z_2\right)w_2\sigma'\left(z_1\right)$$



$$\left.\begin{array}{c}\sigma'(z)\le\frac{1}{4}\\ |w|<1\end{array}\right\}\Rightarrow|\sigma'(z)w|\le\frac{1}{4}$$

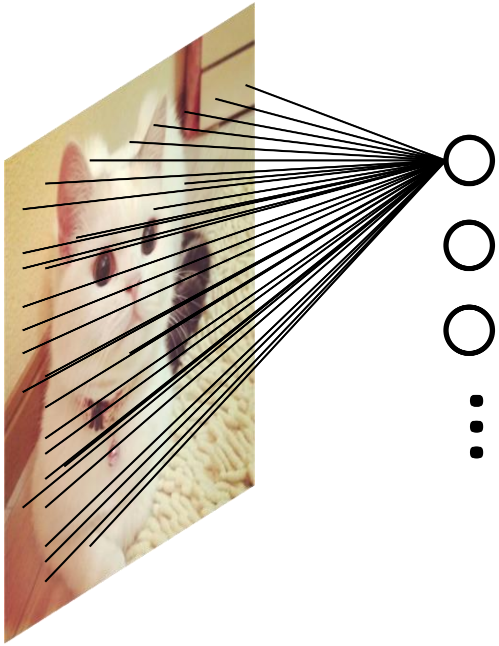
$$\Rightarrow\frac{\partial C}{\partial b_1}\to 0\ \text{梯度消失}$$

$$|\sigma'(z)w|>1\to\ \text{梯度爆炸}$$

深度神经网络

卷积神经网络（CNN）

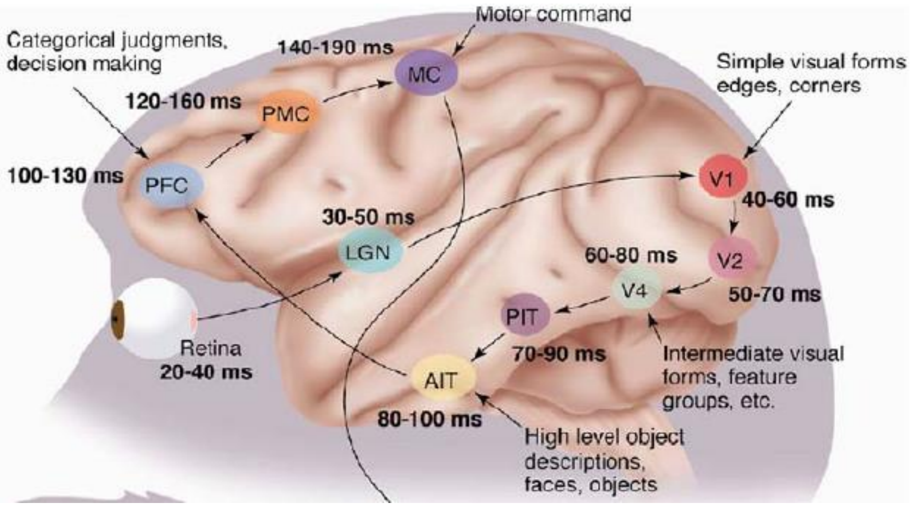
- **Fully Connected Neural Net**

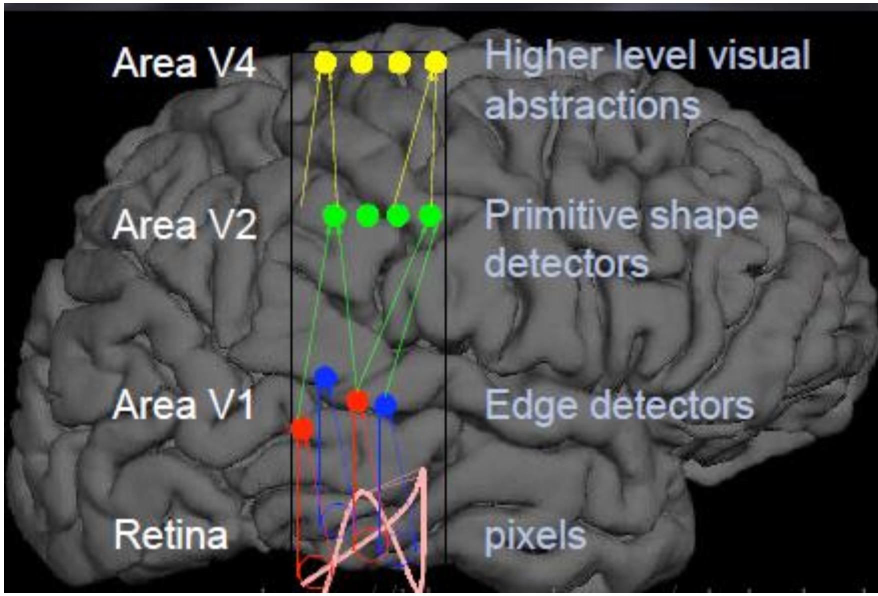


Example: 1000×1000 image
1M hidden units

➡ 10^{12} Parameters!!!

卷积神经网络（CNN）

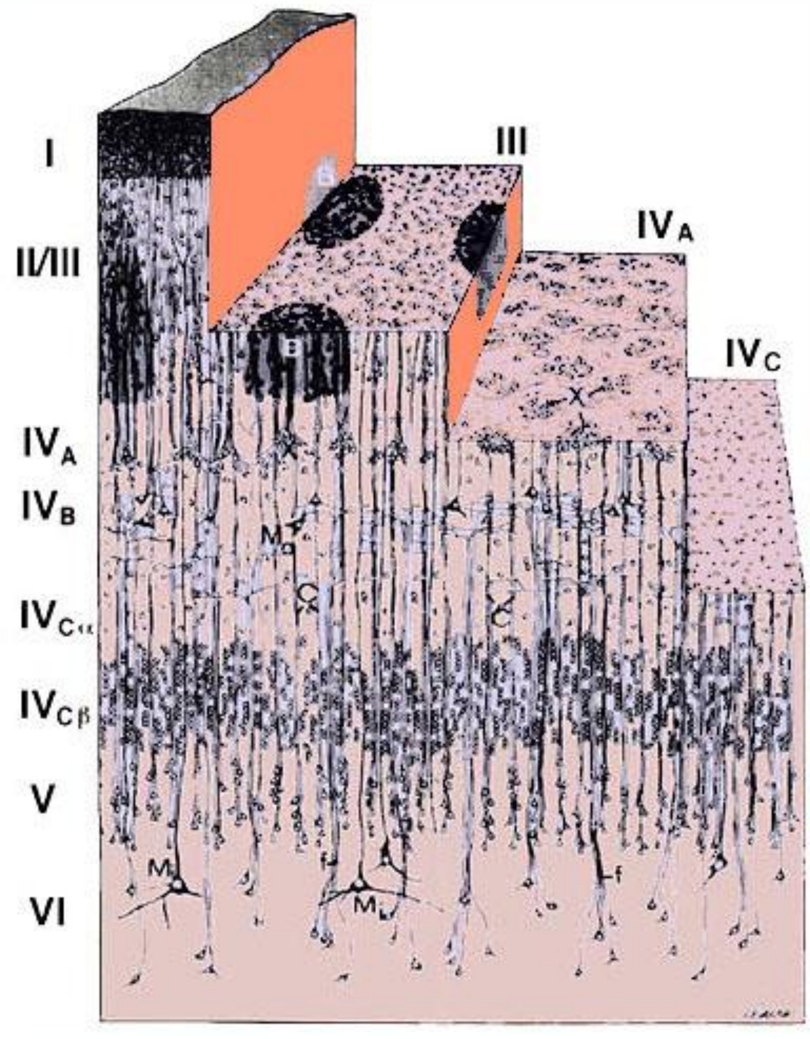




Animal visual cortex contains two basic cell types:

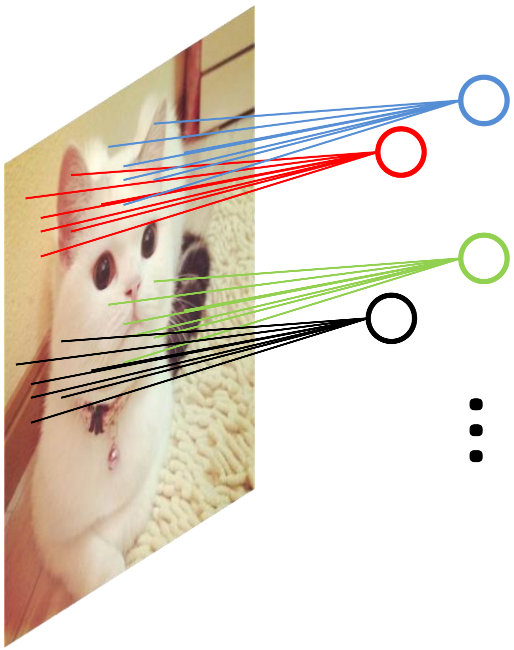
1. simple cells: respond maximally to specific edge-like patterns within their receptive field
2. complex cells: have larger receptive fields and are locally invariant to the exact position of the pattern

卷积神经网络（CNN）



卷积神经网络（CNN）

■ 局部感知



Example: 1000×1000 image

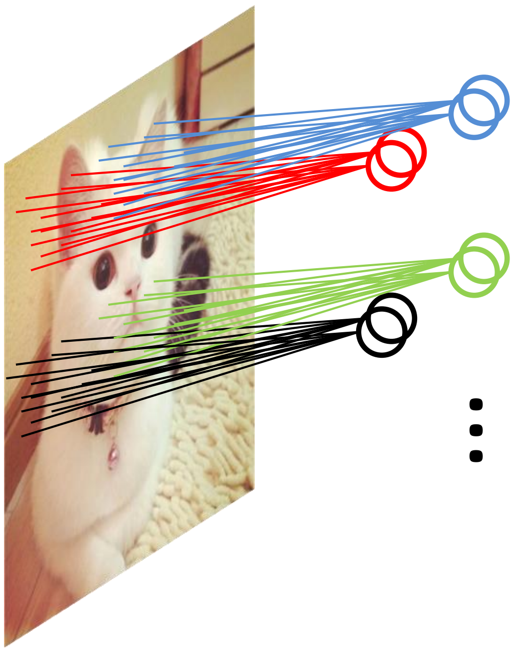
1M hidden units

Filter size: 10×10

100M parameters

卷积神经网络（CNN）

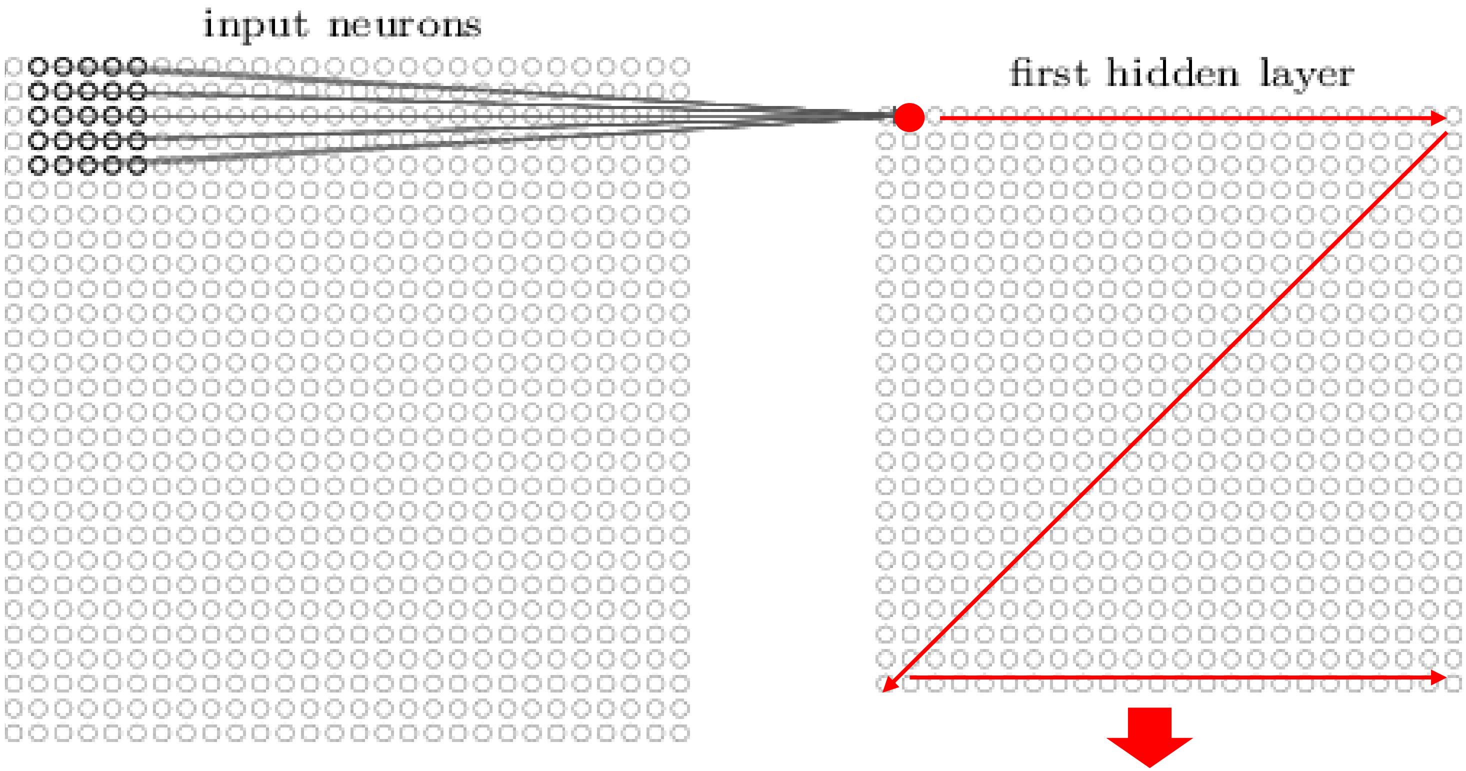
- 权值共享



Example: 1000×1000 image
100 filters
Filter size: 10×10
10K parameters

卷积神经网络（CNN）

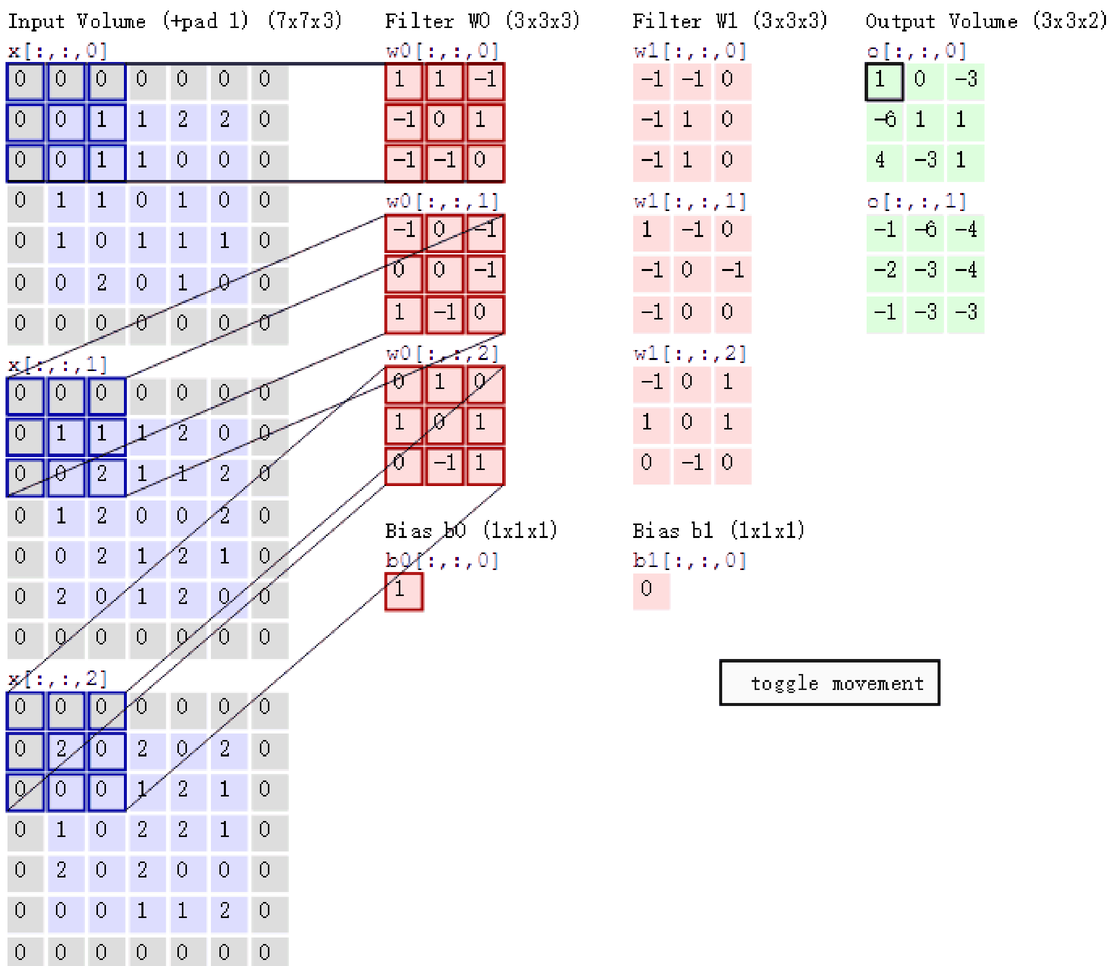
- 局部感受野+权值共享



所有的神经元具有相同的权重w和偏置b

卷积神经网络（CNN）

- 卷积操作

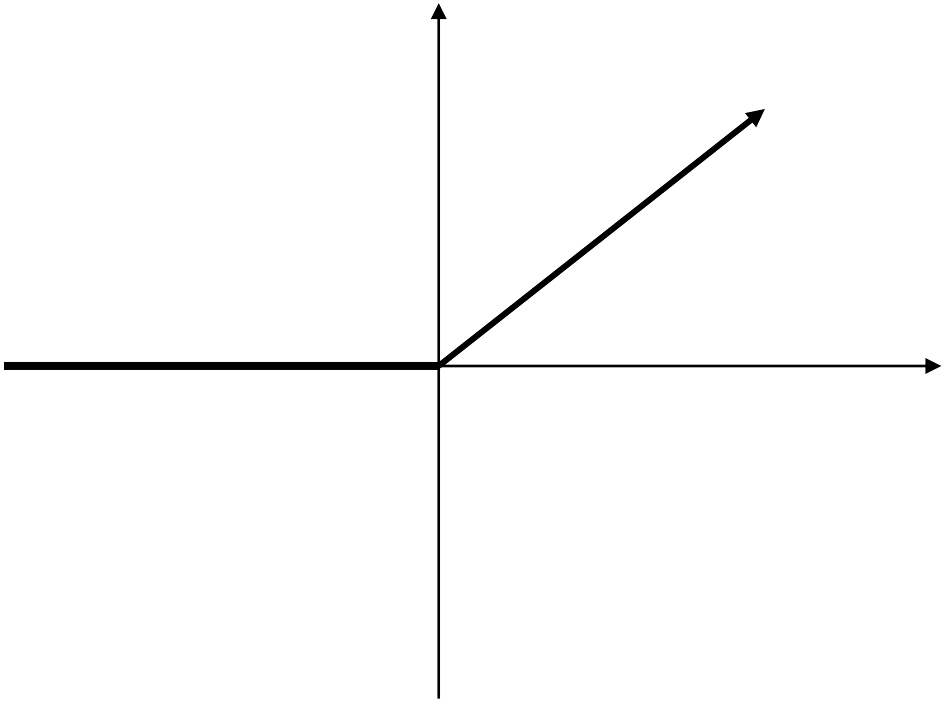


CS231n

卷积神经网络（CNN）

- 线性整流激活函数（**Rectified Linear Units, ReLU**）

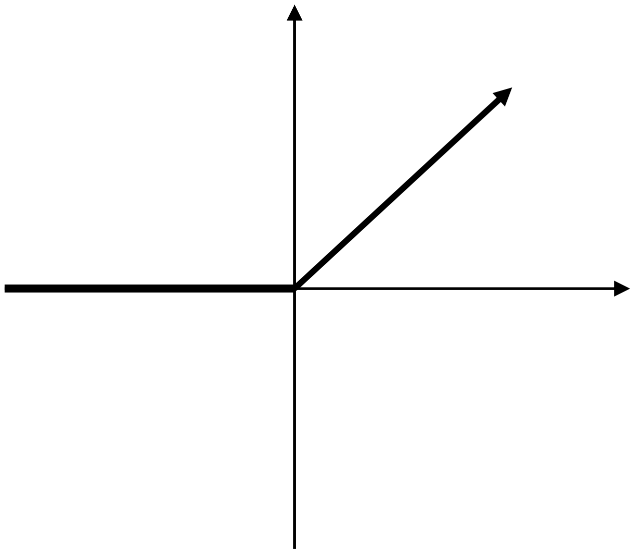
$$f(x) = \max(0, x)$$



卷积神经网络（CNN）

- 线性整流激活函数（**Rectified Linear Units, ReLU**）

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

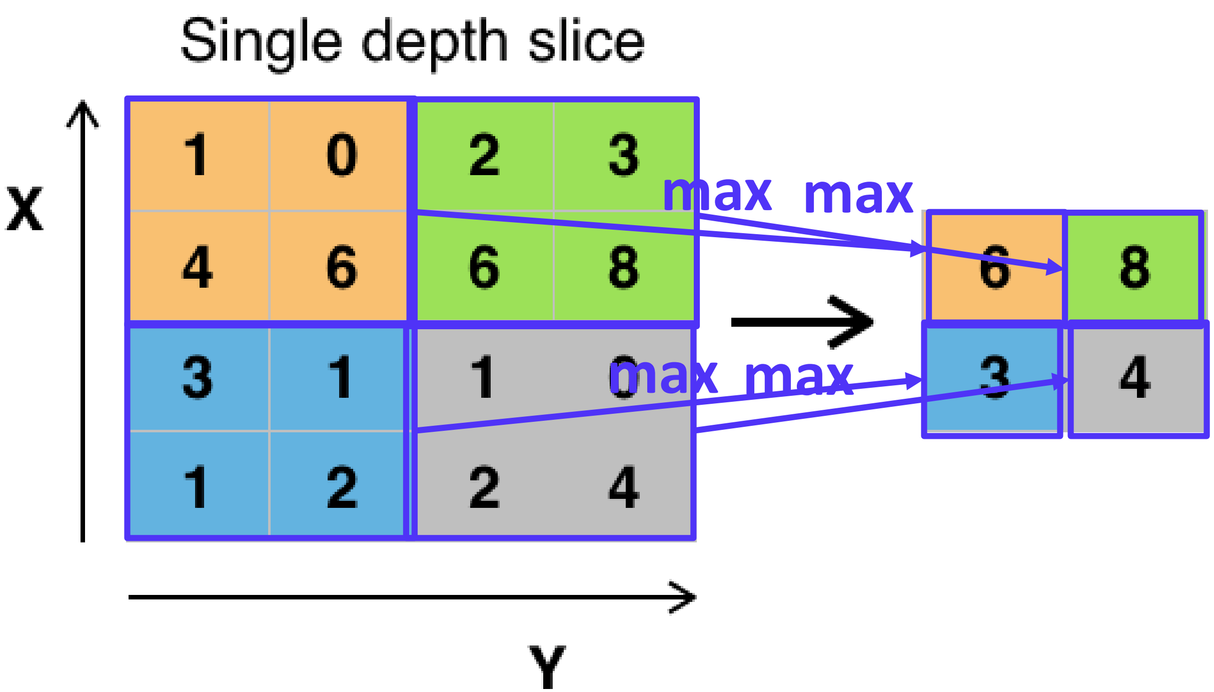


优势：

1. 避免了梯度爆炸和梯度消失问题
2. 简化计算过程
3. 训练稀疏网络

卷积神经网络（CNN）

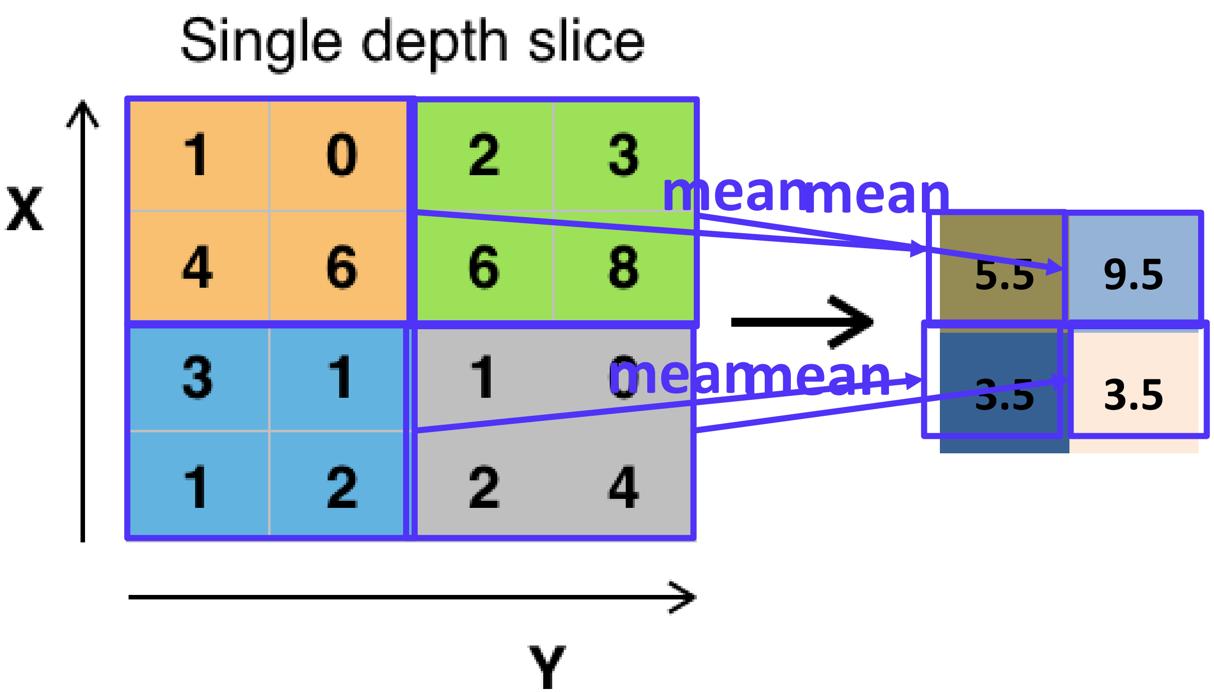
- 池化操作（**pooling**）



Max Pooling

卷积神经网络（CNN）

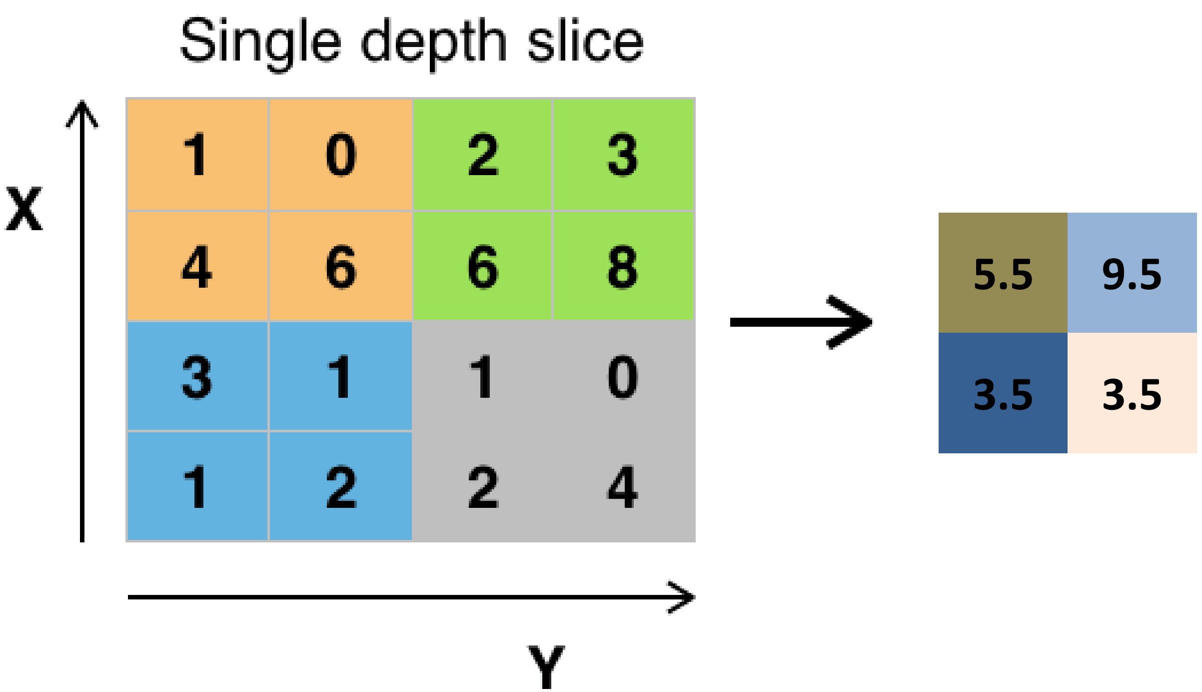
- 池化操作（**pooling**）



Mean Pooling

卷积神经网络（CNN）

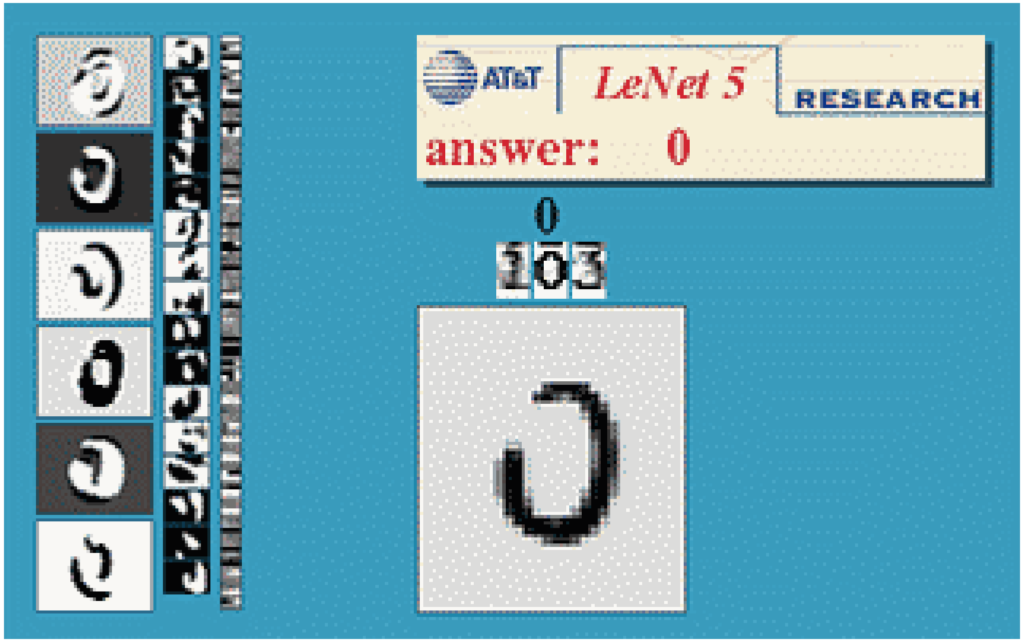
- 池化操作（pooling）



1. 减少参数和计算量，防止过拟合
2. 使模型对尺度、平移、旋转变化具有一定的不变性

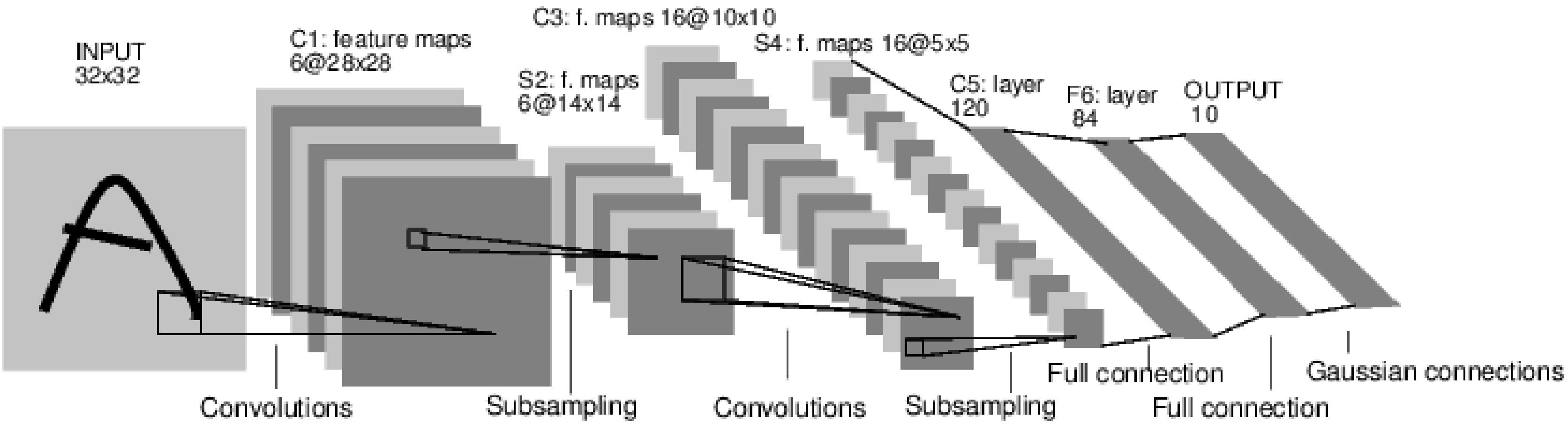
卷积神经网络（CNN）

- LeNet-5: 1989



卷积神经网络（CNN）

- **LeNet-5: 1989**



■ C1层为卷积层

✓ 6个特征图（feature maps）大小分别为28×28，特征图中的每个神经元与输入中5×5的邻域相连。

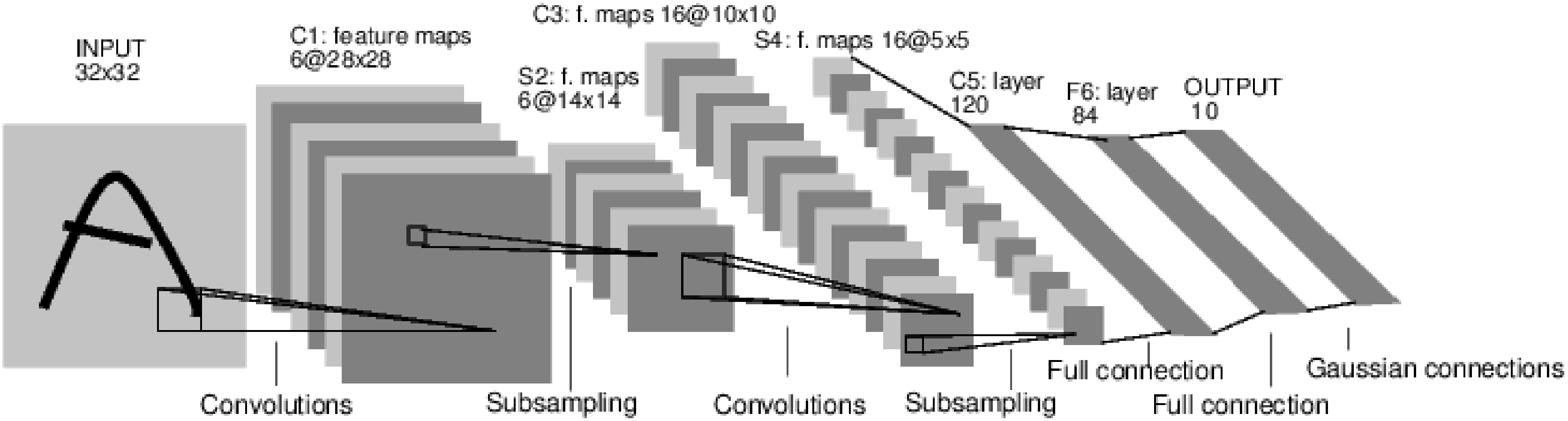
✓ 每个神经元的参数数目： 5×5+1=26

✓ C1层参数总数：(5×5+1)×6=156

✓ C1层与输入连接数：(5×5+1)×6×(28×28)=122,404

卷积神经网络（CNN）

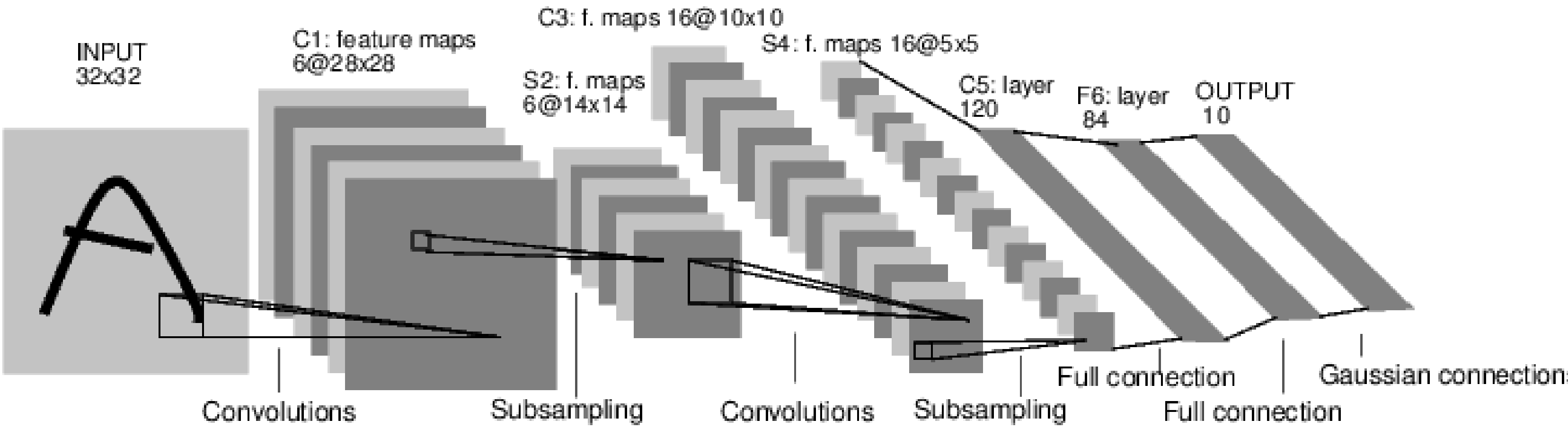
- **LeNet-5: 1989**



- S2层为下采样层
  - ✓ 12个特征图,大小分别为14×14，特征图中的每个单元与C1中2×2的邻域相连，不重叠（stride = 2）
  - ✓ S2中每个单元的4个输入相加，乘1个可以训练的参数w，加一个偏置b，结果通过sigmoid激活输出
  - ✓ S2层参数总数：2×6=12

卷积神经网络（CNN）

- **LeNet-5: 1989**



■ C3为卷积层

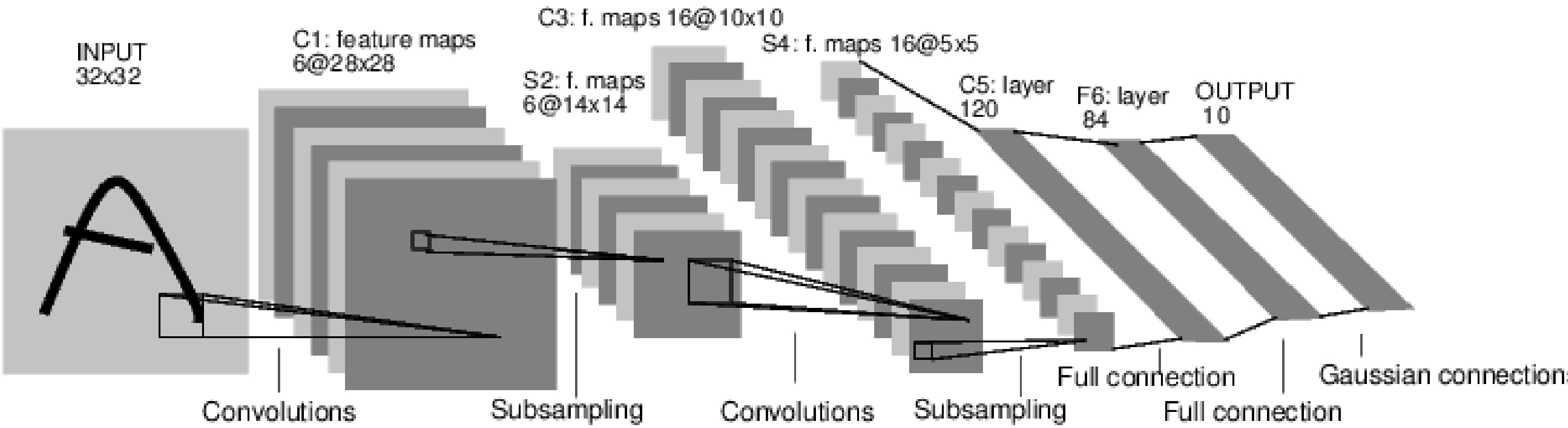
✓ 16个卷积核，得到16个特征图，大小分别为10×10

✓ 每个特征图的神经元与S2层的某几层的多个5×5领域相连

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X

卷积神经网络（CNN）

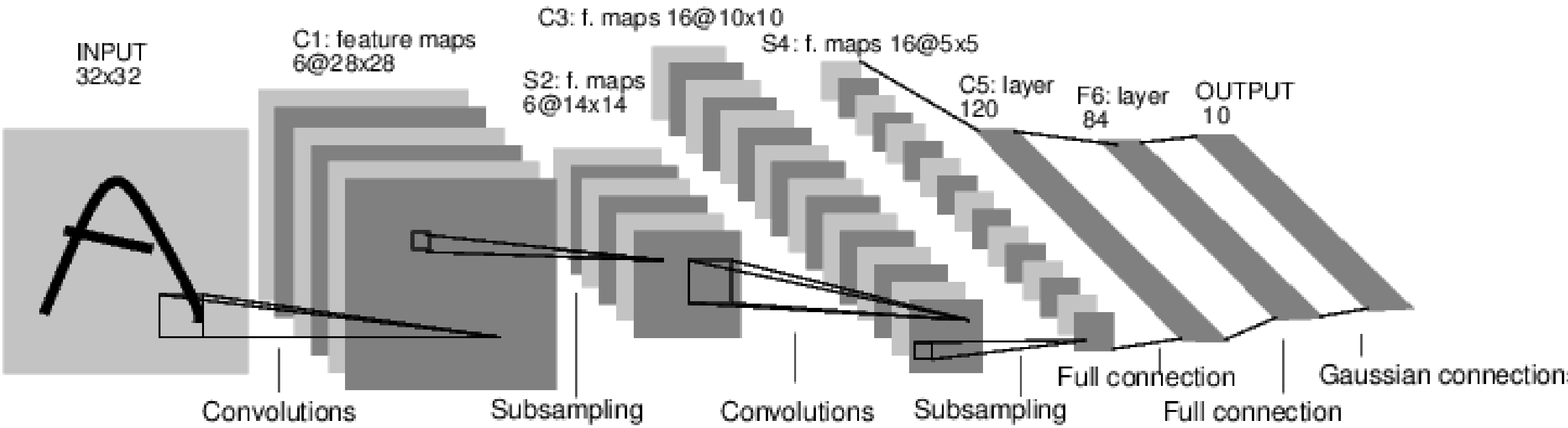
- **LeNet-5: 1989**



■ S4层为下采样层
- ✓ 16个5×5特征图,大小分别为14×14，特征图中的每个单元与C2中2×2的邻域相连，不重叠（stride = 2）
- ✓ S4中每个需要训练1个参数w，1个偏置b
- ✓ S2层参数总数：2×16=32

卷积神经网络（CNN）

- **LeNet-5: 1989**



■ C5为卷积层

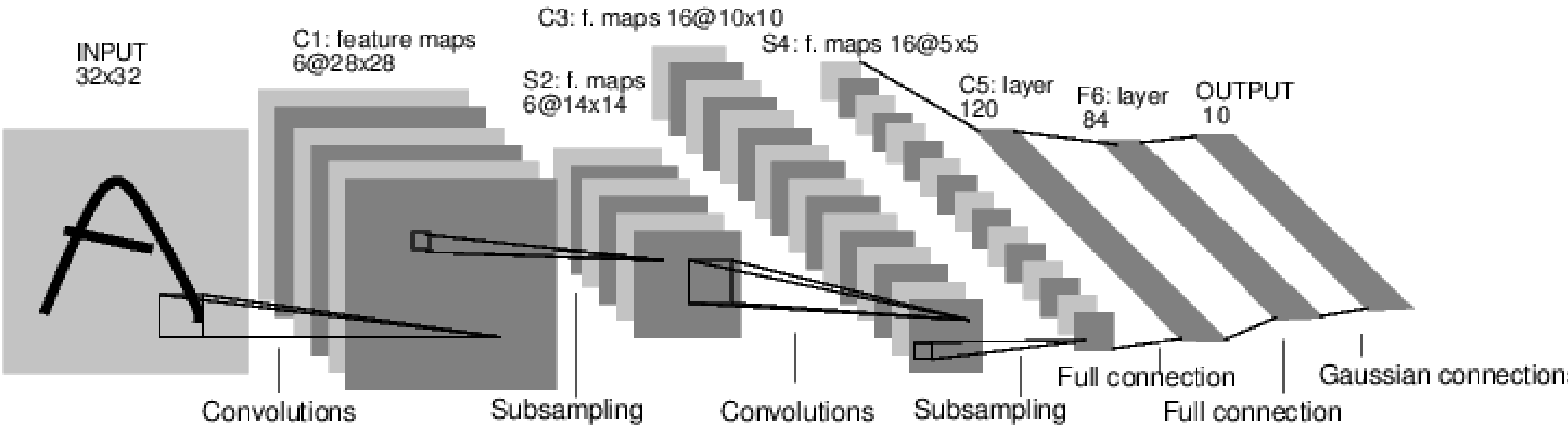
✓ 120个神经元，可以看作120个特征图，大小为1×1

✓ 每个神经元与S4层全部16个特征图的5×5领域相连

✓ 连接数=参数数：(5×5×16+1)×120=48,120

卷积神经网络（CNN）

- **LeNet-5: 1989**

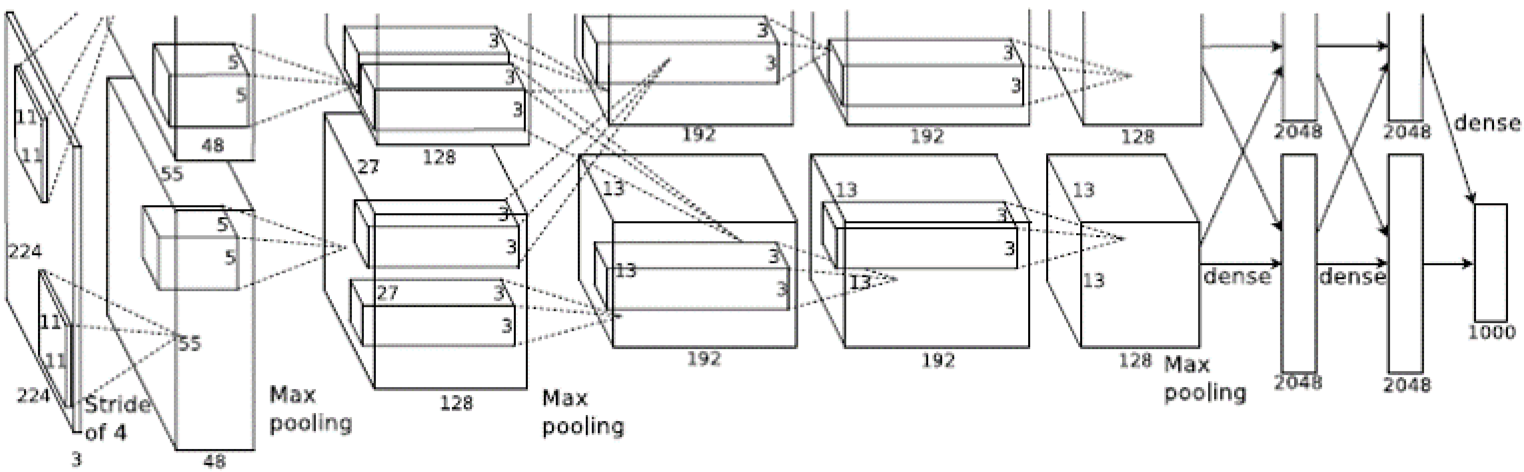


■ 在MNIST 60,000训练集上的错误率为0.95%

■ 60,000+540,000数据增强 错误率0.8%

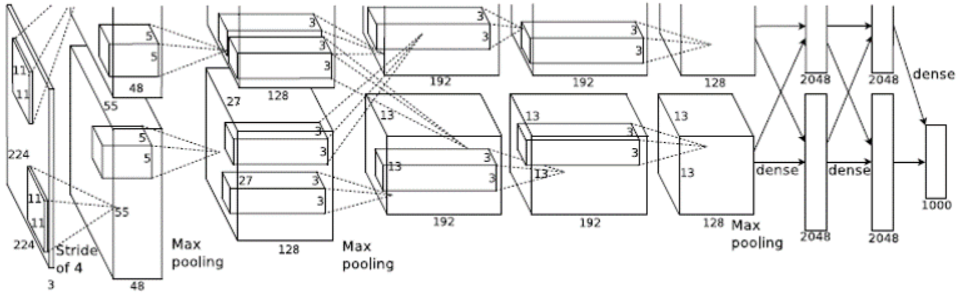
卷积神经网络（CNN）

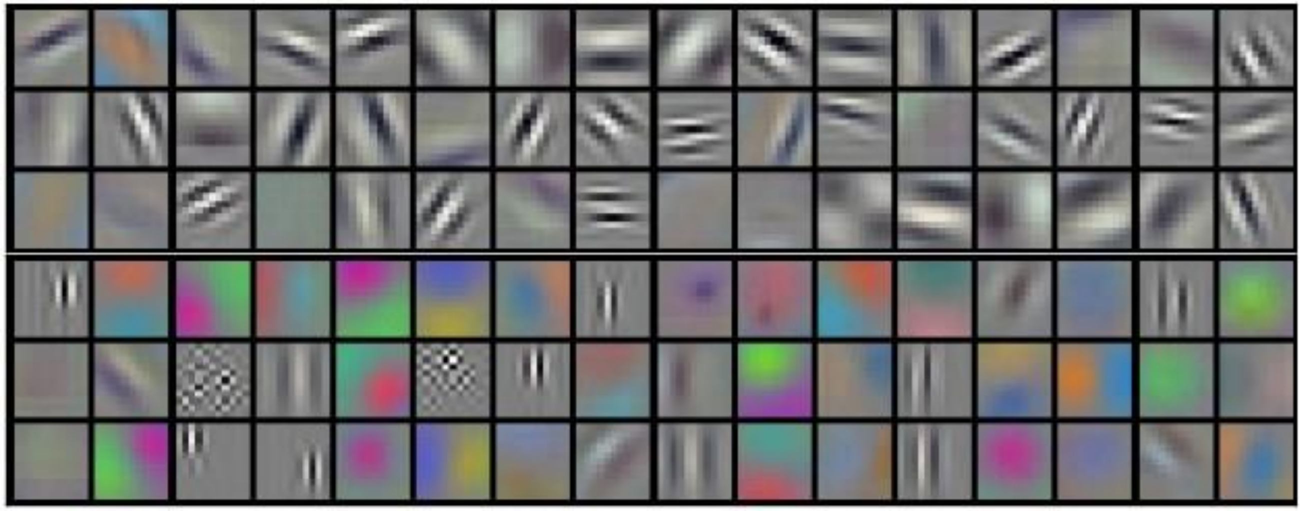
- **AlexNet: 2012**



卷积神经网络（CNN）

- **AlexNet: 2012**





AlexNet 学到的第一层卷积核

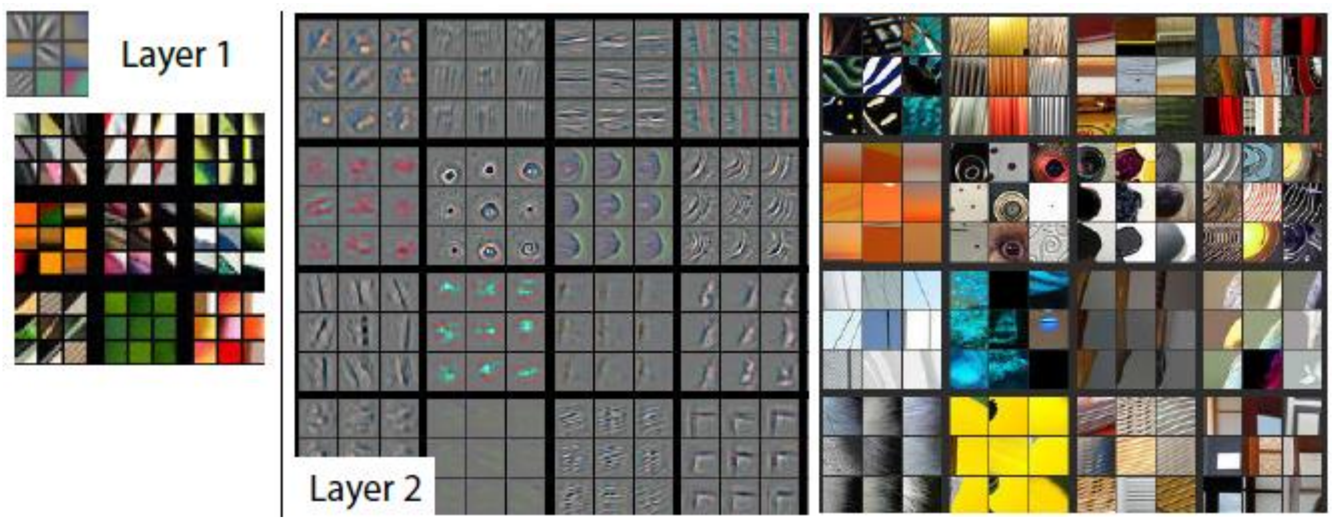
卷积神经网络（CNN）

● Other Nets

- ■ ZF Net: 2013
- ■ GoogLeNet: 2014
- ■ VGGNet: 2014
- ■ ResNet: 2015

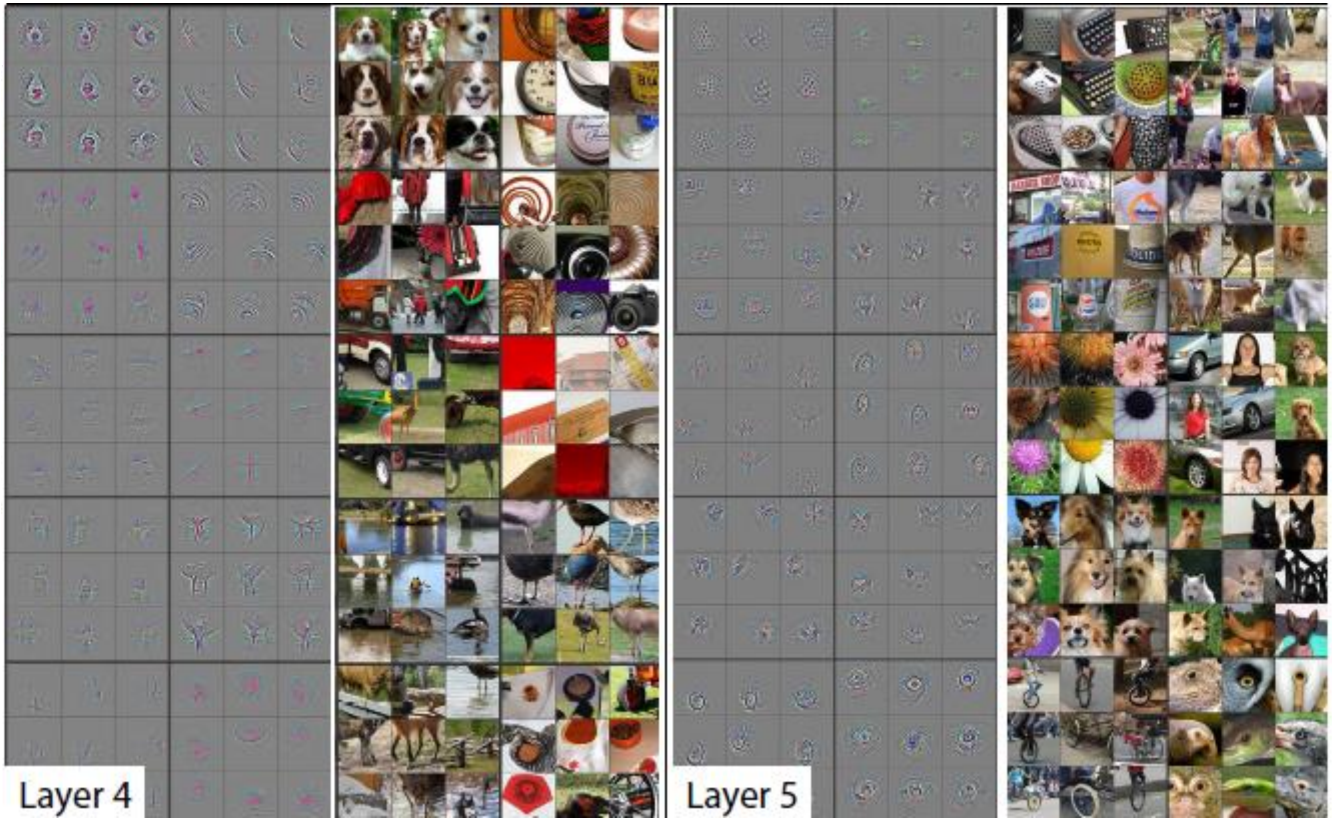
卷积神经网络（CNN）

- CNN可视化



卷积神经网络（CNN）

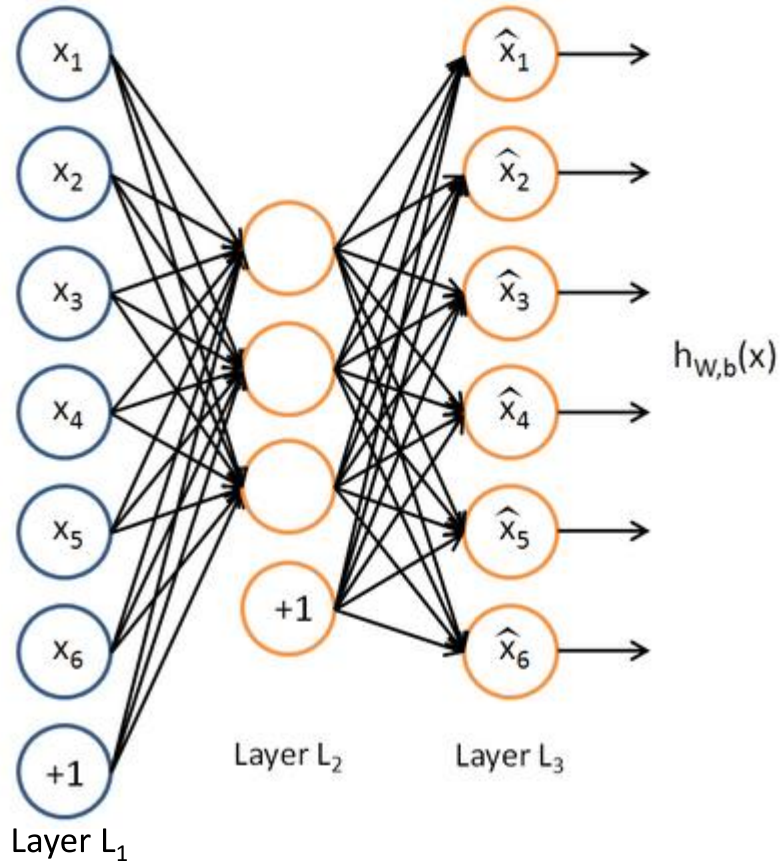
- CNN可视化



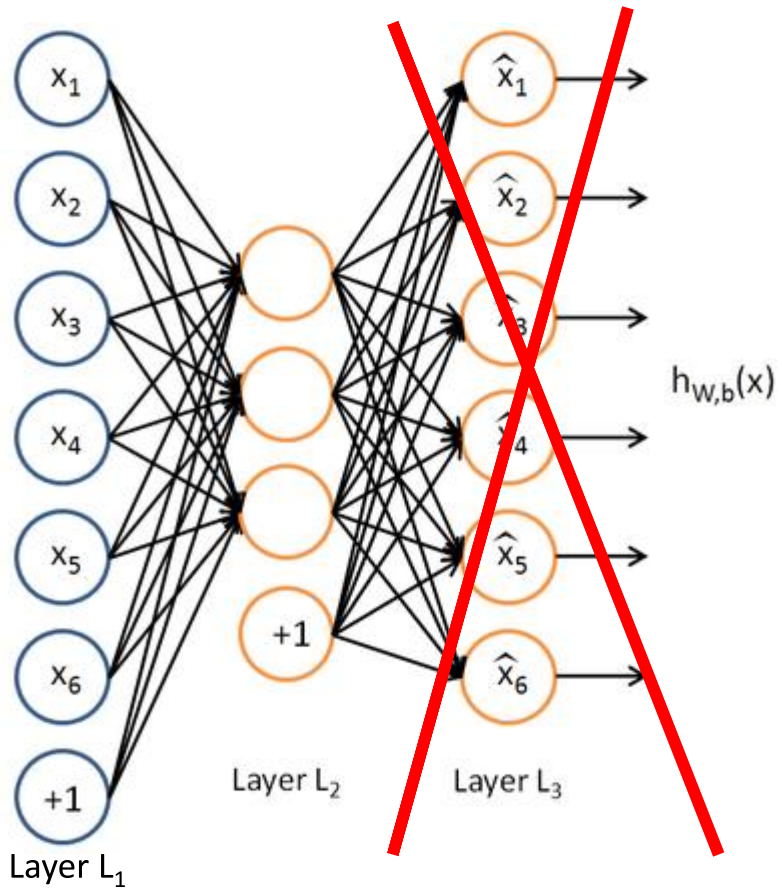
自编码机（AutoEncoder）

- AutoEncoder是一种无监督的学习算法，他利用反向传播算法，让目标值等于输入值：

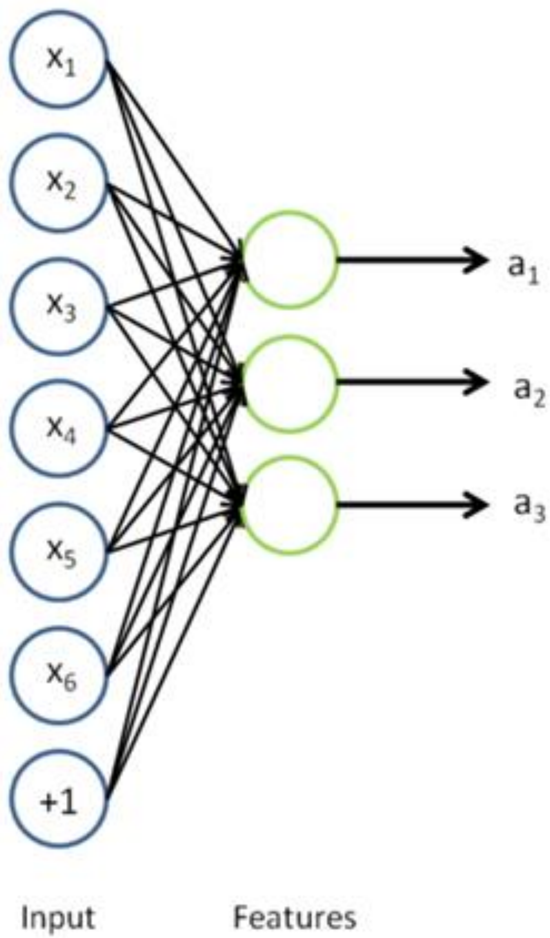
$$x \approx h_{w,b}(x)$$



自编码机（AutoEncoder）



自编码机（AutoEncoder）



输入的特征表示

其它深度学习模型

- DBN（Deep Belief Networks, 深度信念网络）
- RNN（Recursive neural networks，递归神经网络）
- LSTM（Long short term memory，长短时记忆网络）
- DBM（Deep Boltzmann machines，深度玻尔兹曼机）
- Stacked (de-noising) auto-encoders，栈式自编码机
- Deep Q-networks，深度Q网络
- …