深度学习

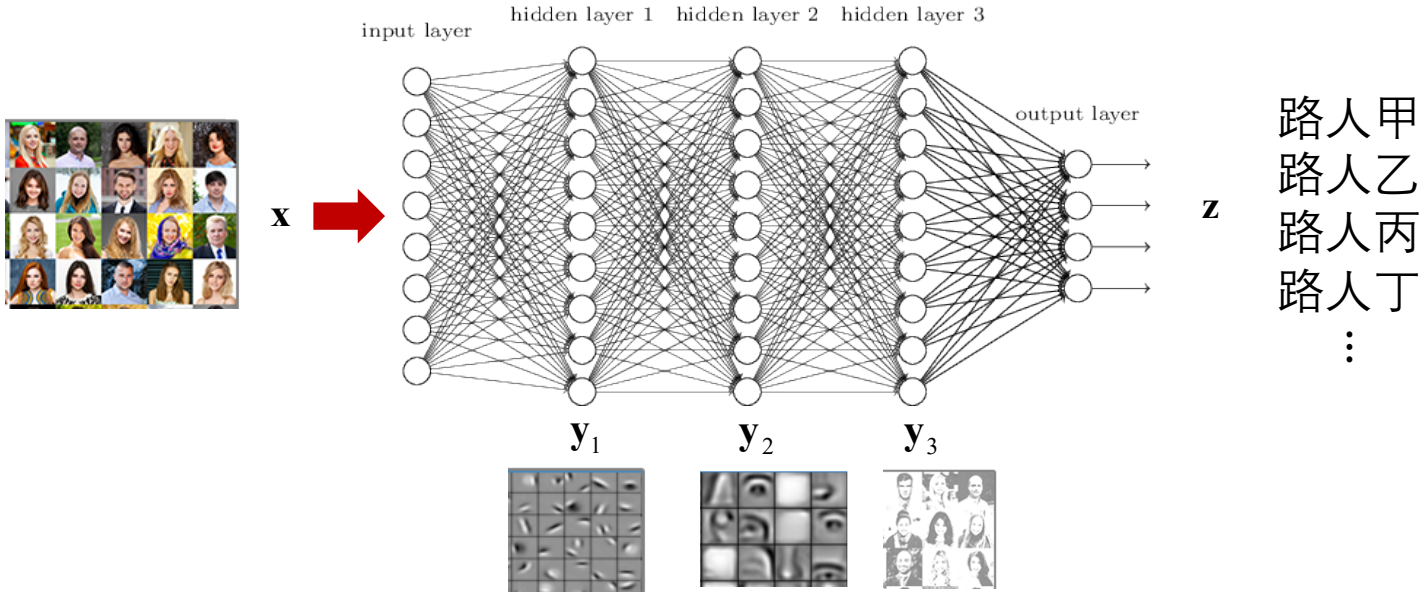
主讲人：屠恩美

《机器学习与知识发现》

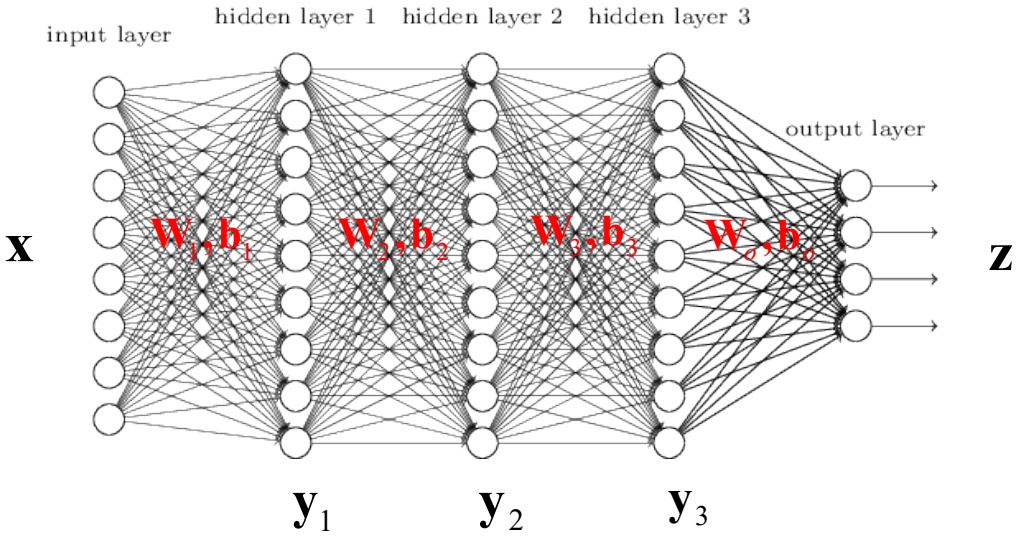
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

什么是深度学习？

- 多层神经网络（通常隐含层≥2个），采用反向传播算法进行训练
- 为什么叫深度学习？
  - 网络结构“深”（多层串联）
  - 隐含层特征深(由低级粗糙到高级抽象)
  - 换个名字好做文章 (NIPS多年不收神经网络论文)



输入输出关系(前馈网络为例)



$$\mathbf{y}_1 = f(\mathbf{W}_1^T\mathbf{x} + \mathbf{b}_1)$$
$$\mathbf{y}_2 = f(\mathbf{W}_2^T\mathbf{y}_1 + \mathbf{b}_2)$$
$$\mathbf{y}_3 = f(\mathbf{W}_3^T\mathbf{y}_2 + \mathbf{b}_3)$$
$$\hat{\mathbf{z}}_2 = f(\mathbf{W}_o^T\mathbf{y}_3 + \mathbf{b}_o)$$

每一层都是前一层输出的函数
因此总输出是多层参数的**复合函数**

$$\mathbf{z} = f\left(\underbrace{\mathbf{W}_L^T f\left(\mathbf{W}_{L-1}^T \underbrace{f(\ldots)}_{L-2} + \mathbf{b}_{L-1}\right)}_{L-1} + \mathbf{b}_L\right)$$

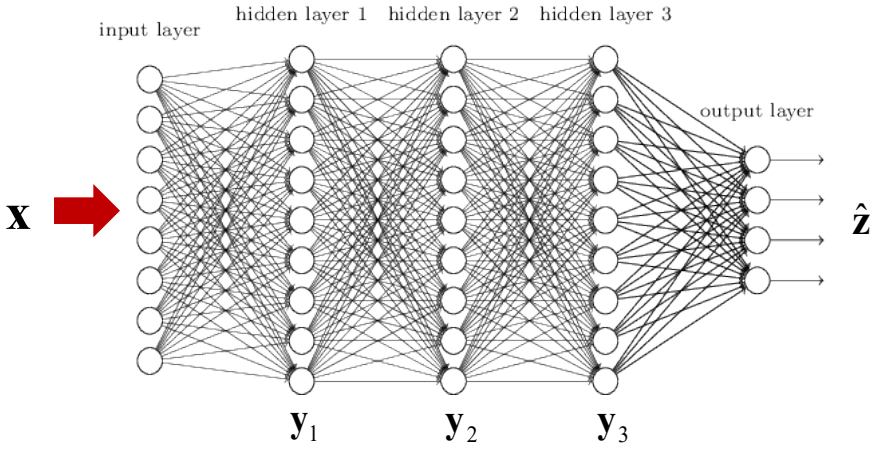
网络训练

- 训练数据 $:\mathcal{X}=\{\mathbf{x}_1,\mathbf{x}_2,...,\mathbf{x}_n\}$ 来自C 个类别，样本标记向量 $\mathcal{Z}=\{\mathbf{z}_1,\mathbf{z}_2,...,\mathbf{z}_n\}$

  单热向量 $\mathbf{z}_i=[0 \quad \cdots \quad 0 \quad 1 \quad 0 \quad \cdots \quad 0]^T \in \mathbb{R}^C$ 对应类别位置1，其他都是0

- 网络输出 $\hat{\mathcal{Z}}=\{\hat{\mathbf{z}}_1,\hat{\mathbf{z}}_2,...,\hat{\mathbf{z}}_n\}$，$\hat{\mathbf{z}}_i=[z_{i1} \quad z_{i2} \quad \cdots \quad z_{ic}]$

- 损失函数：均方误差损失函数: $J(\mathcal{Z},\hat{\mathcal{Z}})=\sum_{i=1}^{n}\|\mathbf{z}_i-\hat{\mathbf{z}}_i\|^2$ ， **交叉熵损失函数**



交叉熵损失函数

- **交叉熵损失** (Cross Entropy Loss)： $J\left(\mathcal{Z},\hat{\mathcal{Z}}\right)=\sum_{i=1}^{n}H(z_i,p(\hat{\mathbf{z}}_i))$

  其中交叉熵 $H(q,p)$ 衡量概率分布q,p的差异性

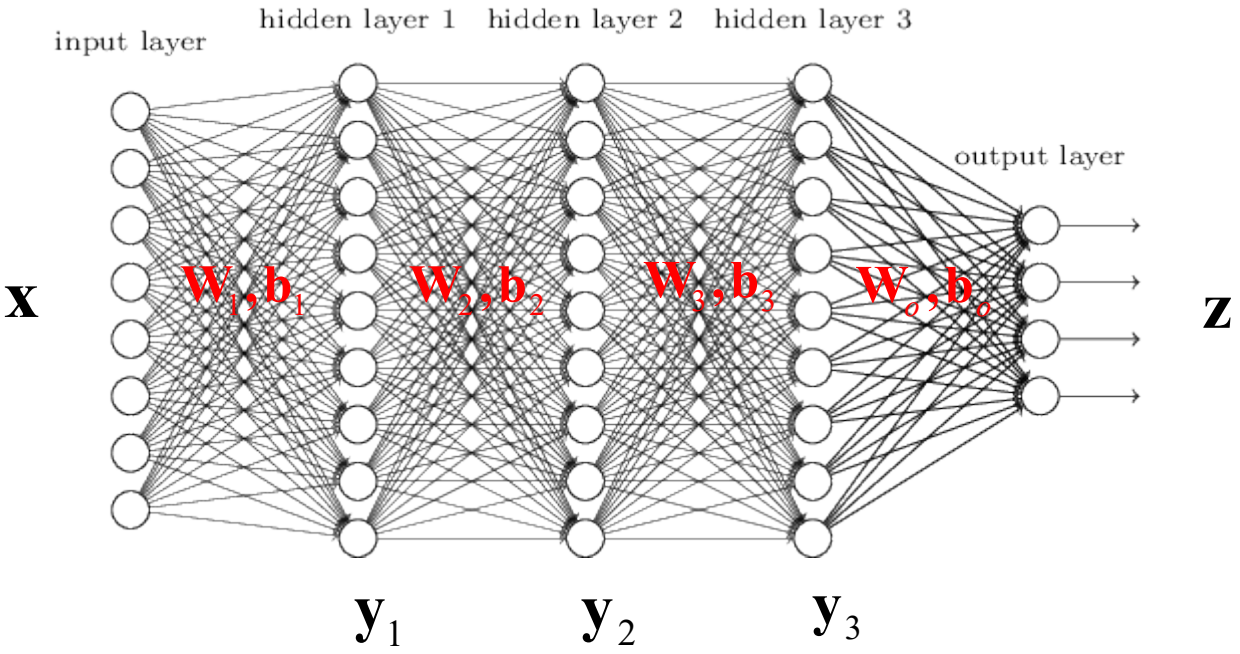
$$H(\mathbf{q},\mathbf{p})=-\sum_{j=1}^{C}q_j\log p_j \qquad H(q,p)=-\int q(x)\log p(x)dx$$

- 如何把网络输出输出 $\hat{\mathbf{z}}$ 转化为类别概率分布 $p(\hat{\mathbf{z}})$ ？softmax函数

$$P(j\mid\hat{\mathbf{z}})=\frac{e^{\hat{z}_j}}{\sum_{k=1}^{C}e^{\hat{z}_k}} \qquad \sum_{j=1}^{C}P(j\mid\hat{\mathbf{z}})=\sum_{j=1}^{C}\frac{e^{\hat{z}_j}}{\sum_{k=1}^{C}e^{\hat{z}_k}}=\frac{1}{\sum_{k=1}^{C}e^{\hat{z}_k}}\sum_{j=1}^{C}e^{\hat{z}_j}=1$$

- softmax函数特点
  - 把任意一串实数转化为概率分布
  - 连续任意阶可导

训练思路



- 如果用θ表示网络参数$(\mathbf{W}_1,\mathbf{W}_2,\mathbf{W}_3,\mathbf{W}_o,\mathbf{b}_1,\mathbf{b}_2,\mathbf{b}_3,\mathbf{b}_o)$中的任意一个
- 记$g=\frac{dJ}{d\theta}$ **误差函数对参数的导数**，则梯度下降可写成$\theta_{t+1}=\theta_t-\lambda\, g_t$
- 不同优化算法$\theta_{t+1}=\theta_t-\lambda\,\Delta\theta_t$关键在于**如何确定梯度变换函数**$\Delta\theta_t=f(g_t)$

深入链式求导法则

- 考虑复合函数

$$\begin{cases} z = f(t) \\ t = g(x, y) \end{cases}$$

- 链式求导法则有

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x} = f'(t)g'_x(x, y)$$

- 同理对 y 求导 $\frac{\partial z}{\partial y} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial y} = f'(t)g'_y(x, y)$

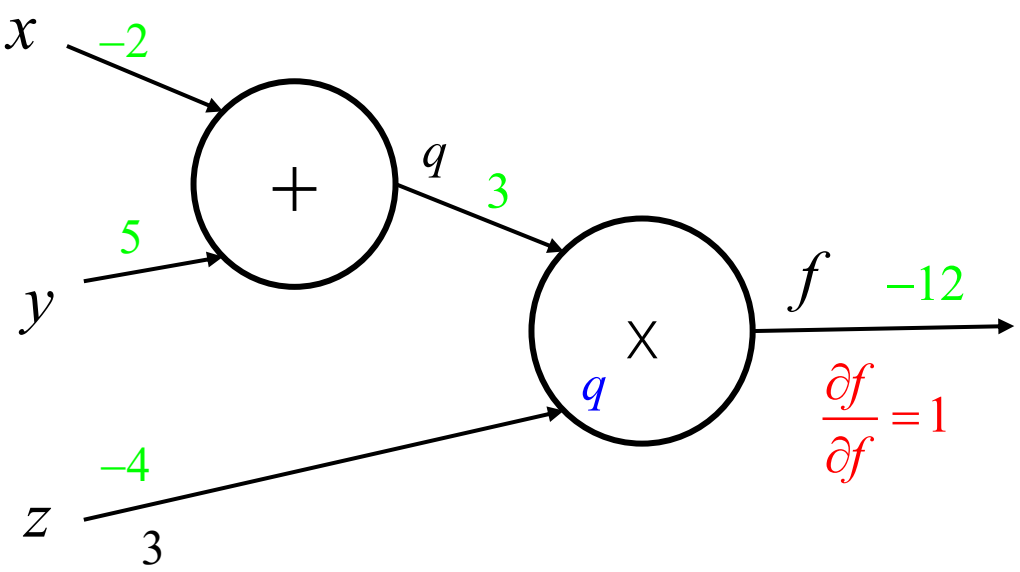
深入链式求导法则

$$f = (x + y)z$$

$$\begin{cases} f = qz \\ q = x + y \end{cases}$$

本地梯度

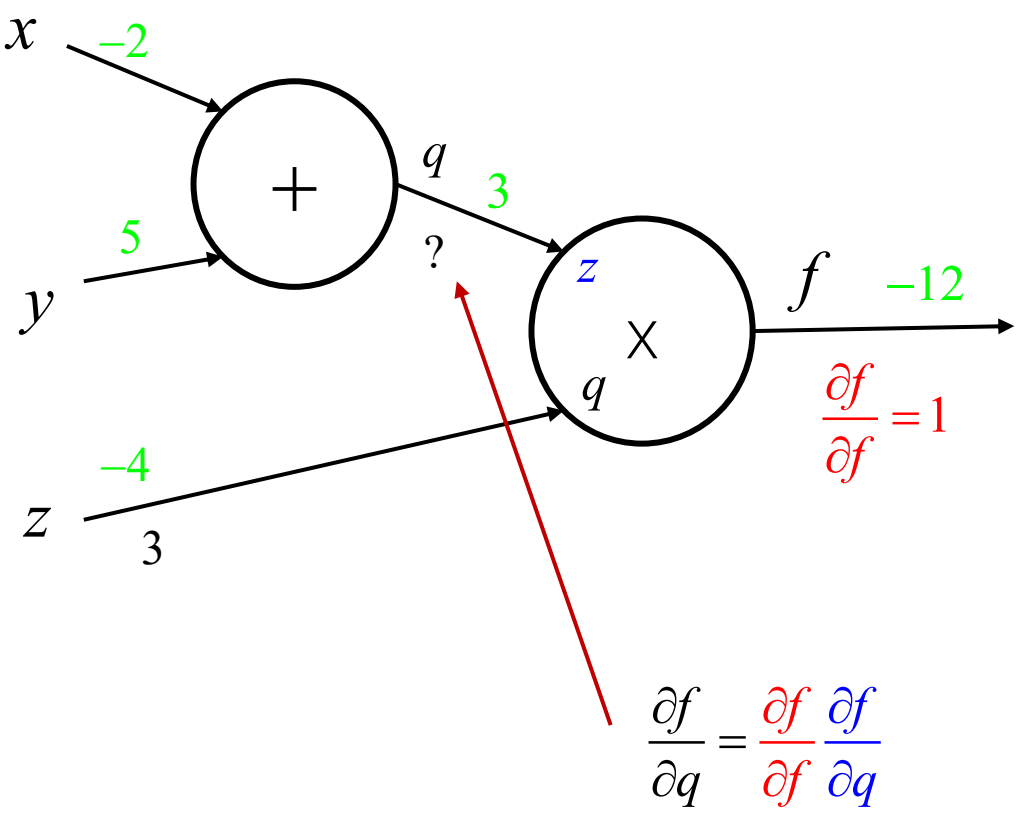
$$\frac{\partial f}{\partial z} = q$$



深入链式求导法则

$$f = (x + y)z$$

$$\begin{cases} f = qz \\ q = x + y \end{cases}$$



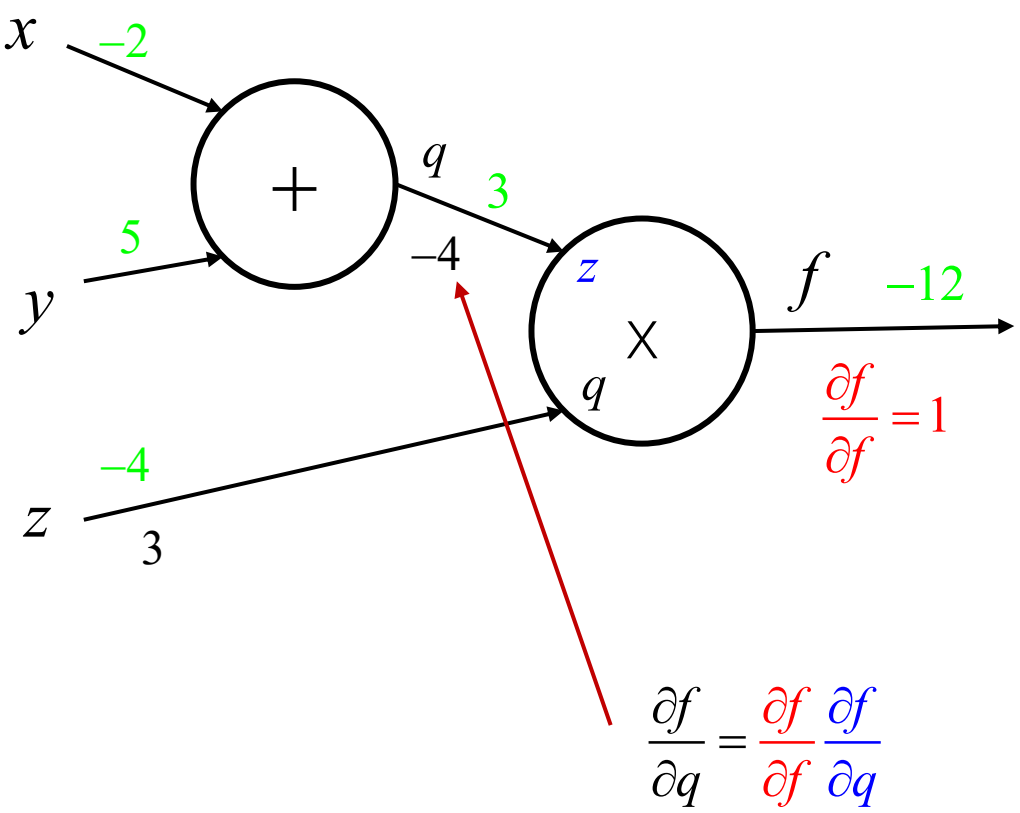
本地梯度

$$\frac{\partial f}{\partial z} = q \qquad \frac{\partial f}{\partial q} = z$$

深入链式求导法则

$$f = (x + y)z$$

$$\begin{cases} f = qz \\ q = x + y \end{cases}$$



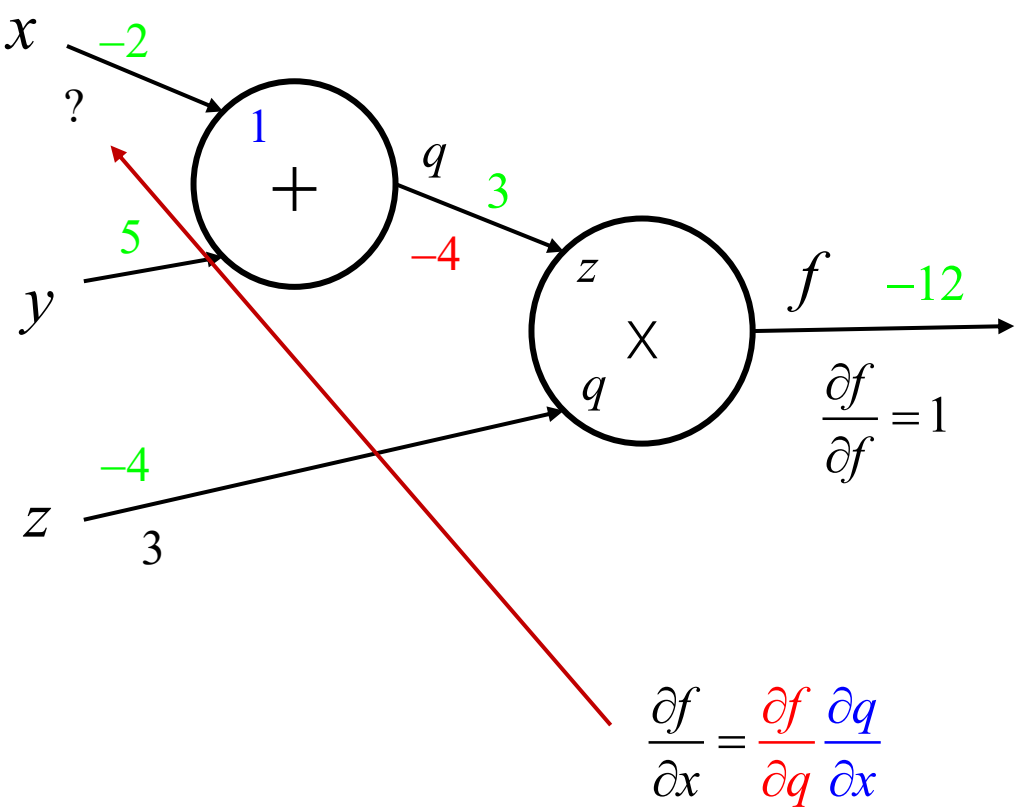
本地梯度

$$\frac{\partial f}{\partial z} = q \qquad \frac{\partial f}{\partial q} = z$$

深入链式求导法则

$$f = (x + y)z$$

$$\begin{cases} f = qz \\ q = x + y \end{cases}$$



本地梯度

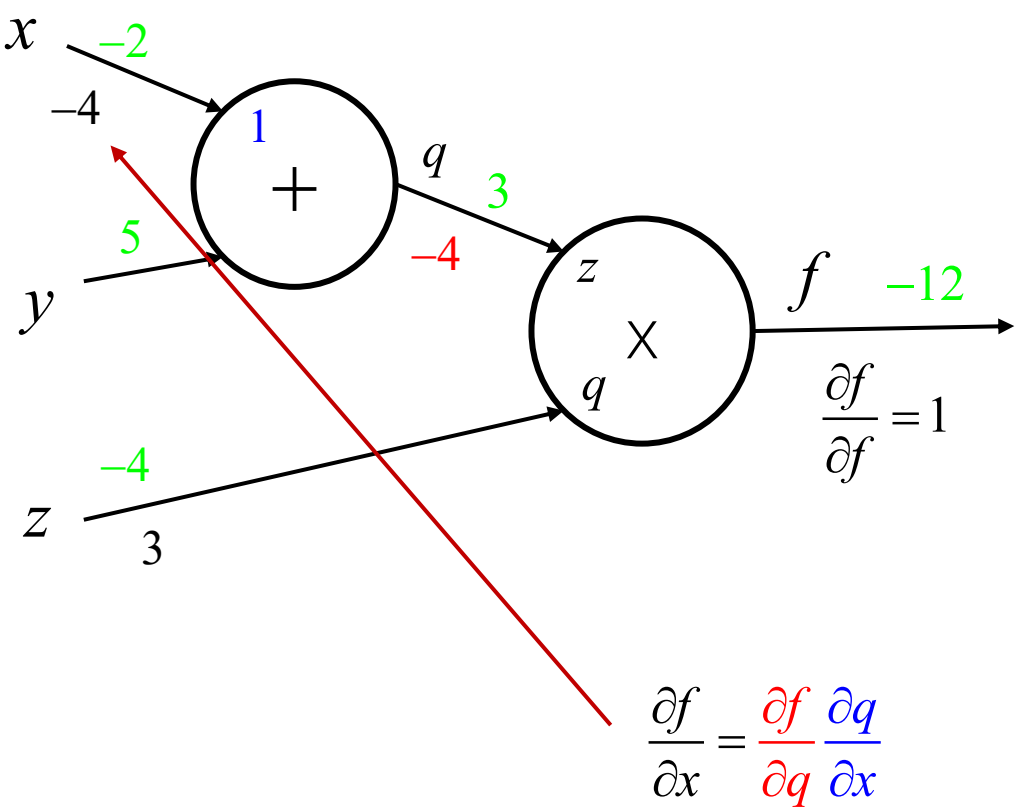
$$\frac{\partial f}{\partial z} = q \qquad \frac{\partial f}{\partial q} = z$$

$$\frac{\partial q}{\partial x} = 1$$

深入链式求导法则

$$f = (x + y)z$$

$$\begin{cases} f = qz \\ q = x + y \end{cases}$$



本地梯度

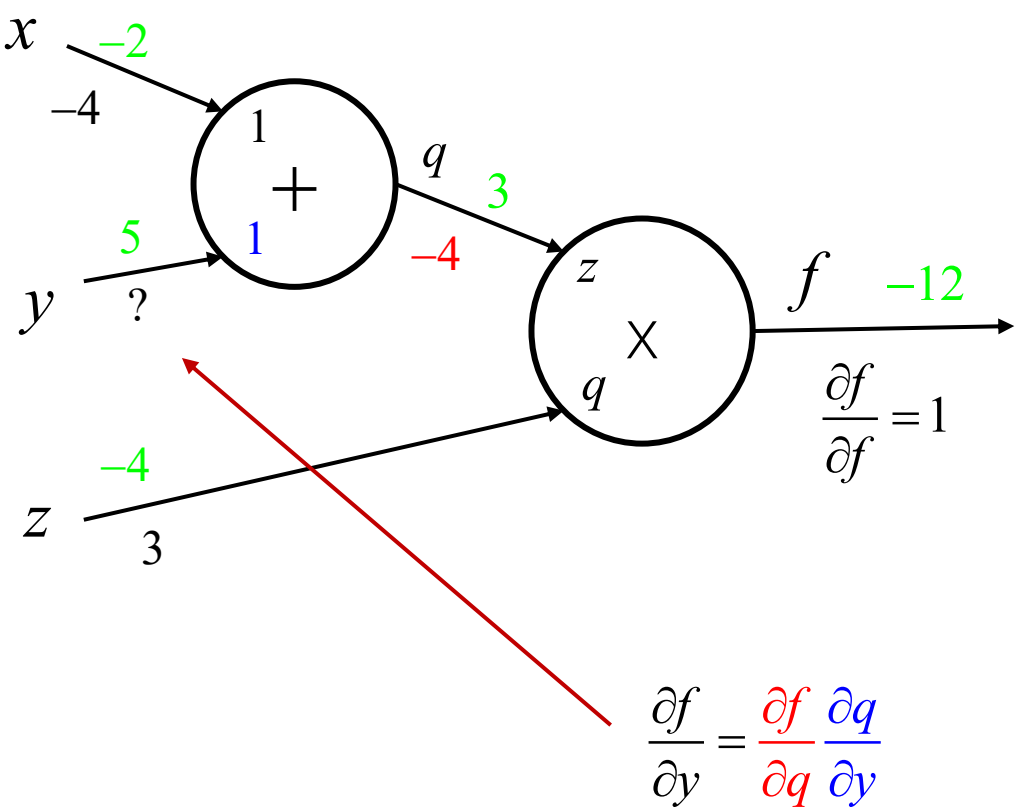
$$\frac{\partial f}{\partial z} = q \qquad \frac{\partial f}{\partial q} = z$$

$$\frac{\partial q}{\partial x} = 1$$

深入链式求导法则

$$f = (x + y)z$$

$$\begin{cases} f = qz \\ q = x + y \end{cases}$$



本地梯度

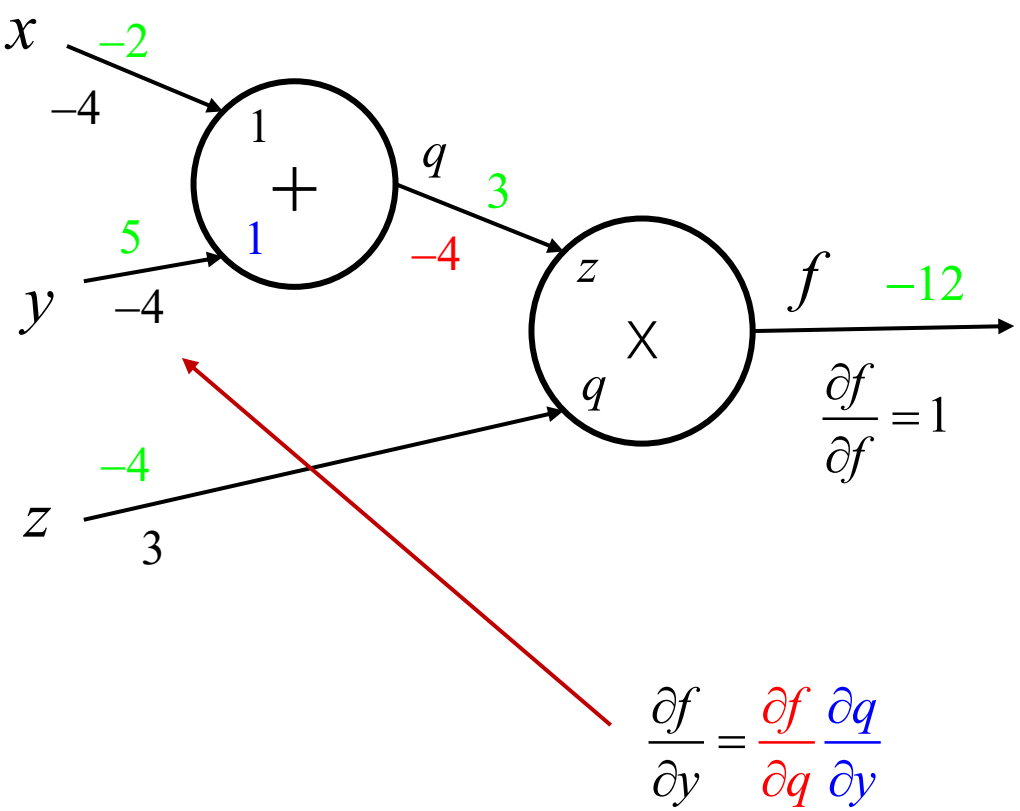
$$\frac{\partial f}{\partial z} = q \qquad \frac{\partial f}{\partial q} = z$$

$$\frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

深入链式求导法则

$$f = (x + y)z$$

$$\begin{cases} f = qz \\ q = x + y \end{cases}$$



本地梯度

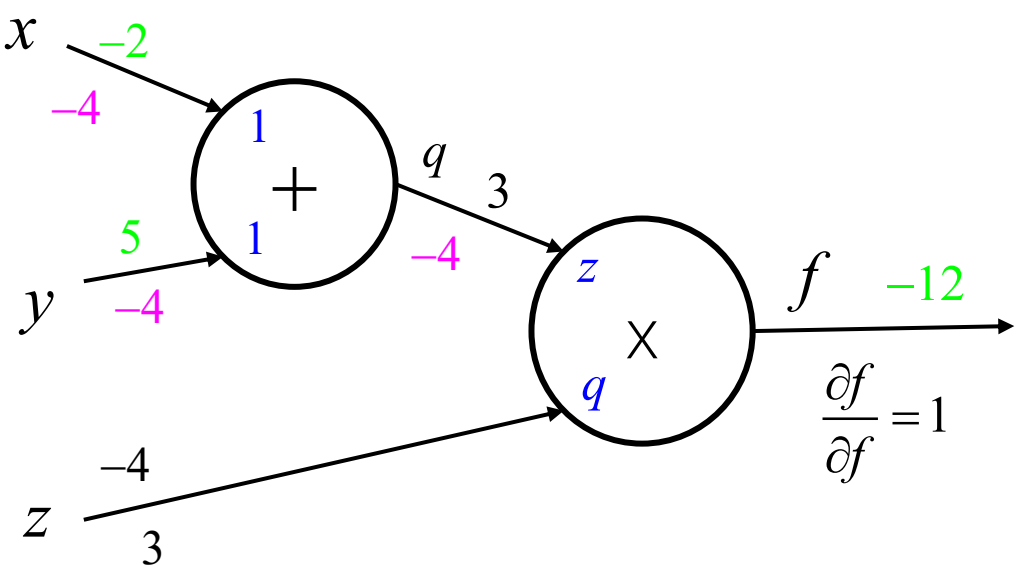
$$\frac{\partial f}{\partial z} = q \qquad \frac{\partial f}{\partial q} = z$$

$$\frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

深入链式求导法则

$$f = (x + y)z$$

$$\begin{cases} f = qz \\ q = x + y \end{cases}$$



本地梯度

$$\frac{\partial f}{\partial z} = q \qquad \frac{\partial f}{\partial q} = z$$

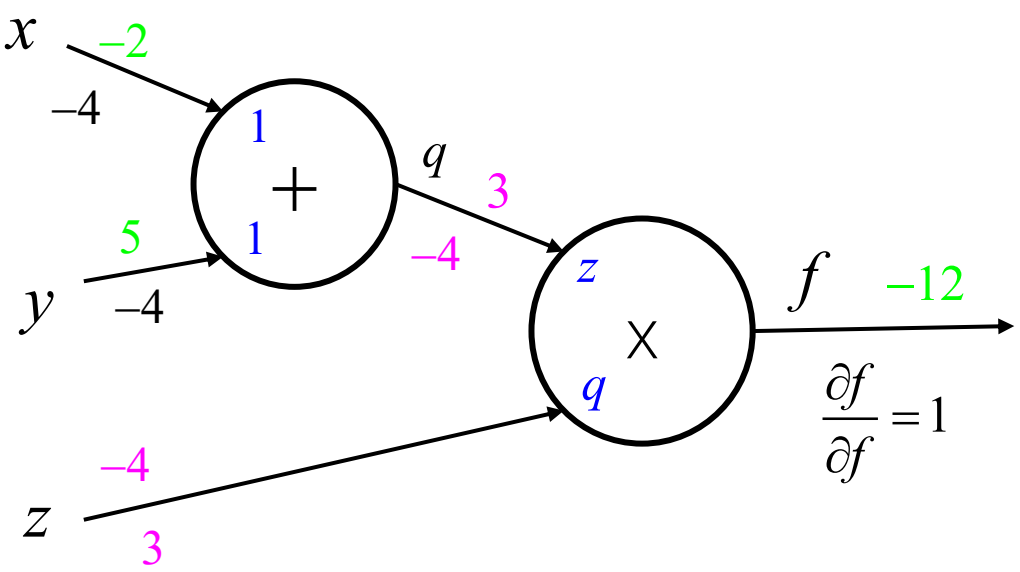
$$\frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

加：梯度分配操作

深入链式求导法则

$$f = (x + y)z$$

$$\begin{cases} f = qz \\ q = x + y \end{cases}$$



本地梯度

$$\frac{\partial f}{\partial z} = q \qquad \frac{\partial f}{\partial q} = z$$

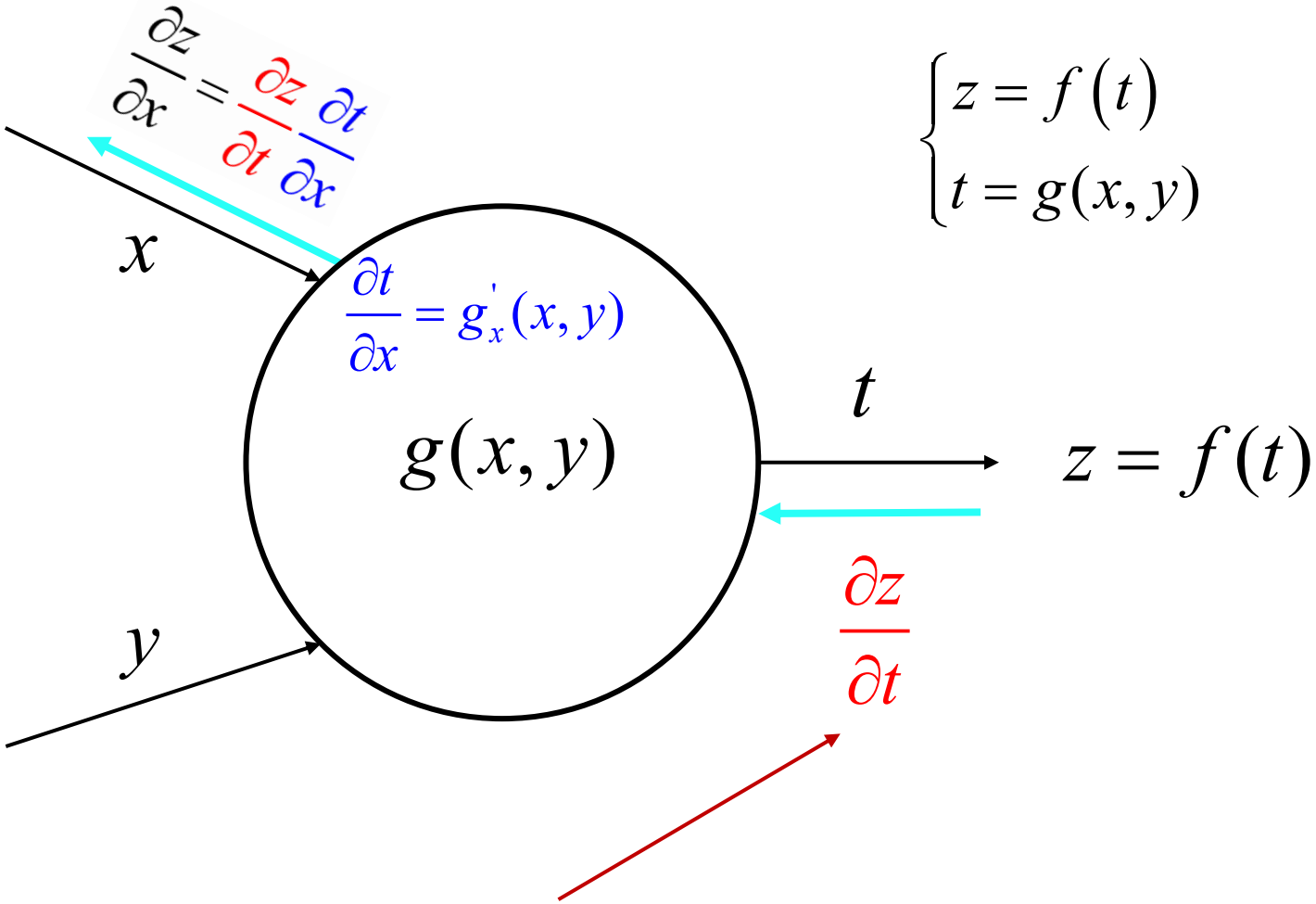
$$\frac{\partial q}{\partial x} = 1 \qquad \frac{\partial q}{\partial y} = 1$$

$$\frac{\partial f}{\partial f} = 1$$

加：梯度分配操作；　乘：梯度-输入交换
Max(a, b): 梯度传送

其他运算求导类似
运算图 (Computational Graph): 节点是运算，边是数据流

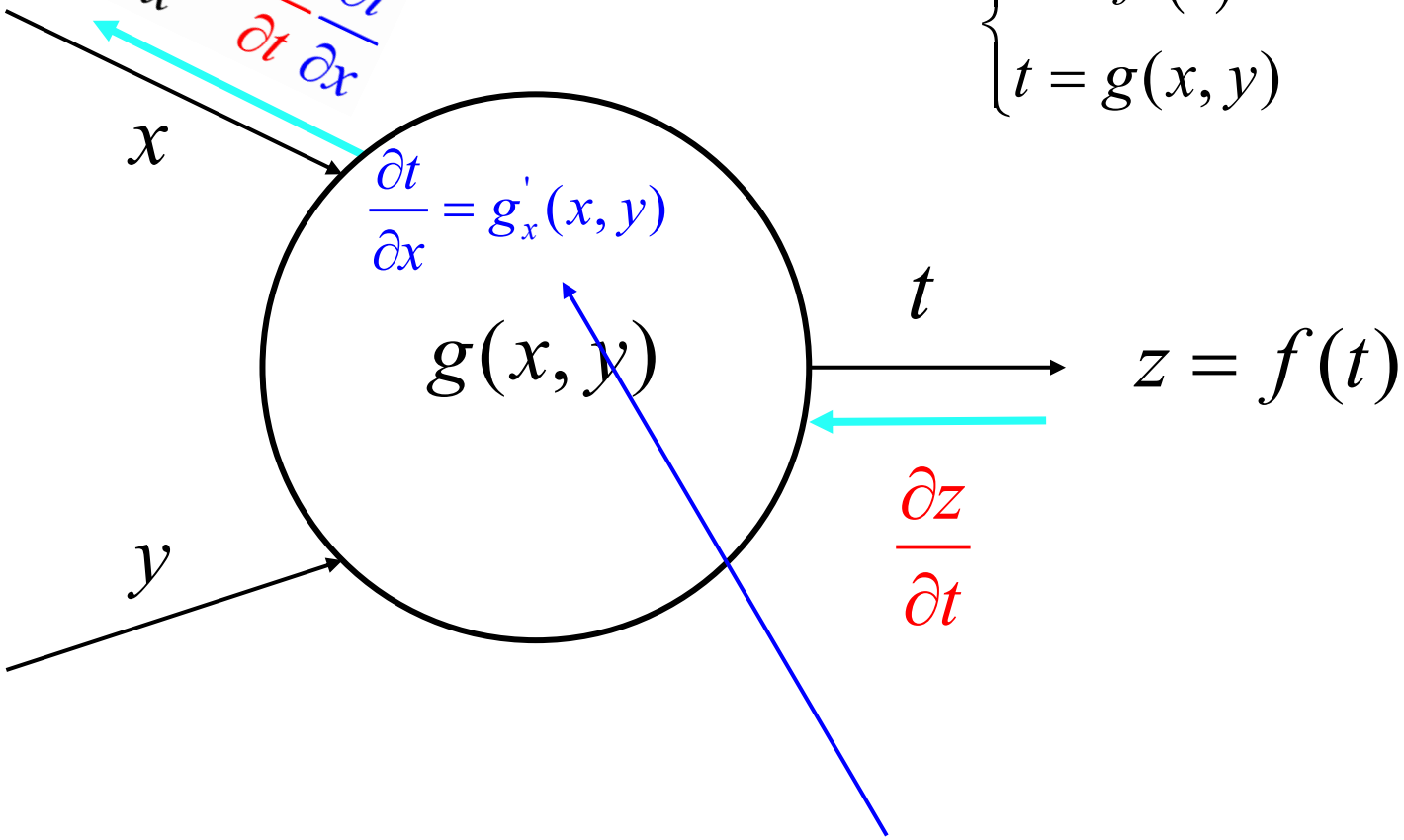
深入链式求导法则


$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x}$
$\begin{cases} z = f(t) \\ t = g(x,y) \end{cases}$
x
$\frac{\partial t}{\partial x} = g_x'(x,y)$
$g(x,y)$
t
$z = f(t)$
y
$\frac{\partial z}{\partial t}$
上游梯度

深入链式求导法则

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x}$$

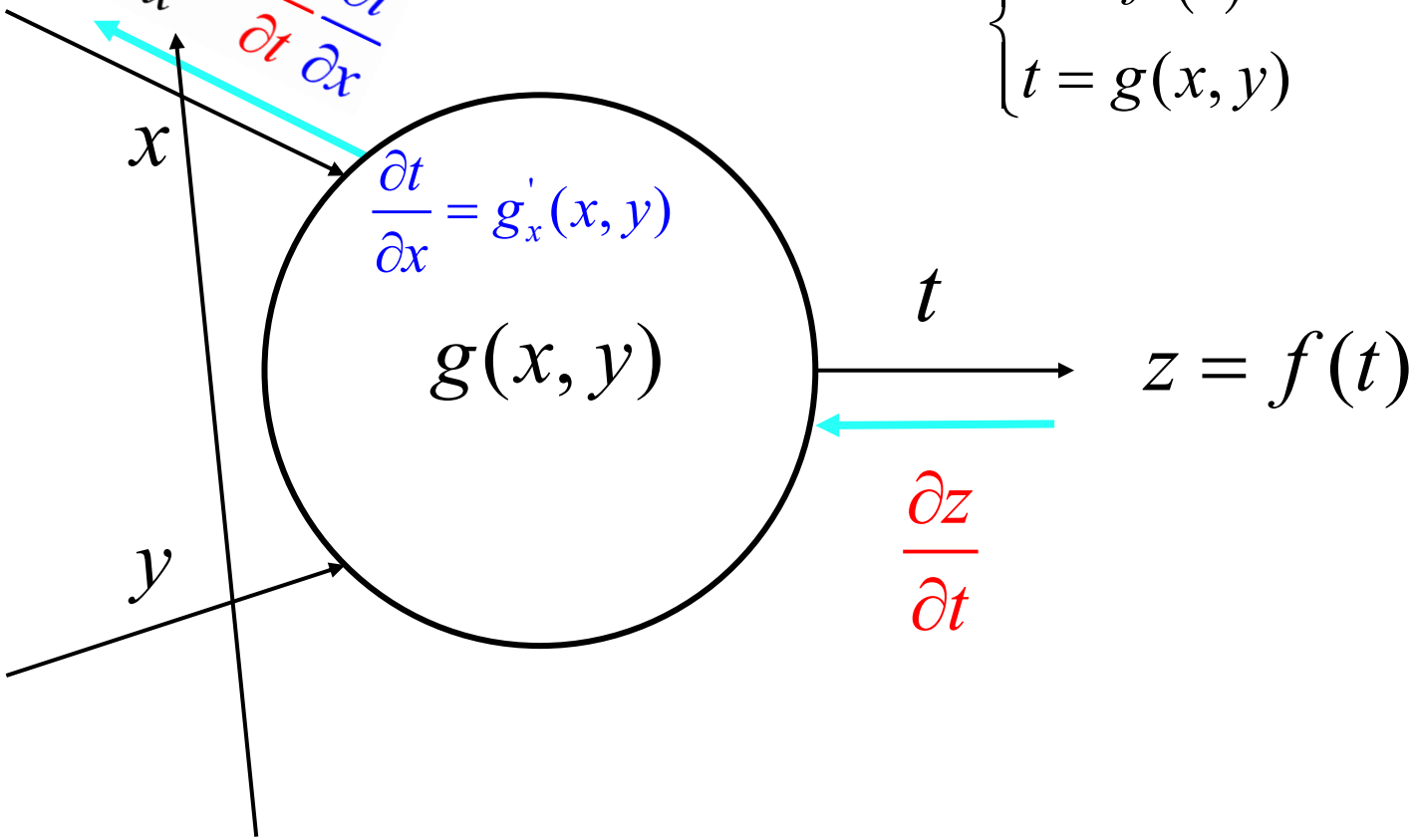
$$\begin{cases} z = f(t) \\ t = g(x, y) \end{cases}$$



深入链式求导法则

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x}$$

$$\begin{cases} z = f(t) \\ t = g(x, y) \end{cases}$$

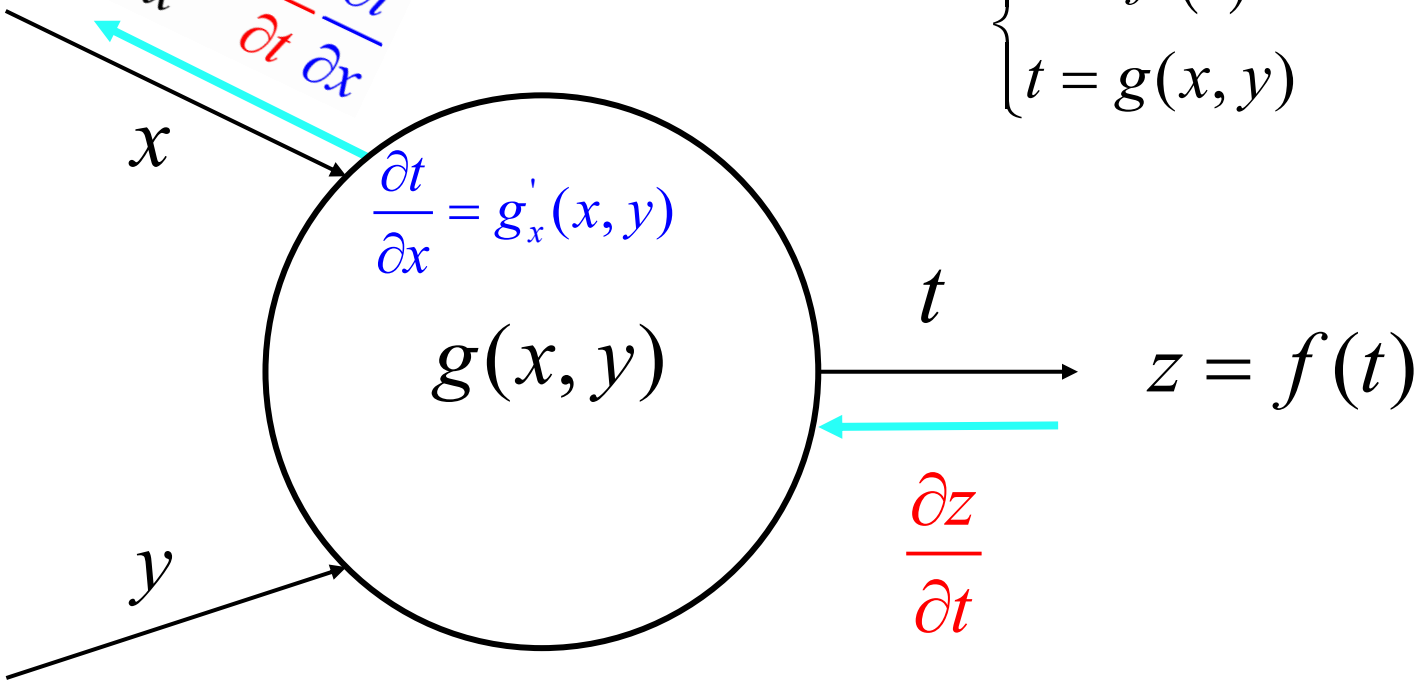


下游梯度　　上游梯度　　本地梯度

深入链式求导法则

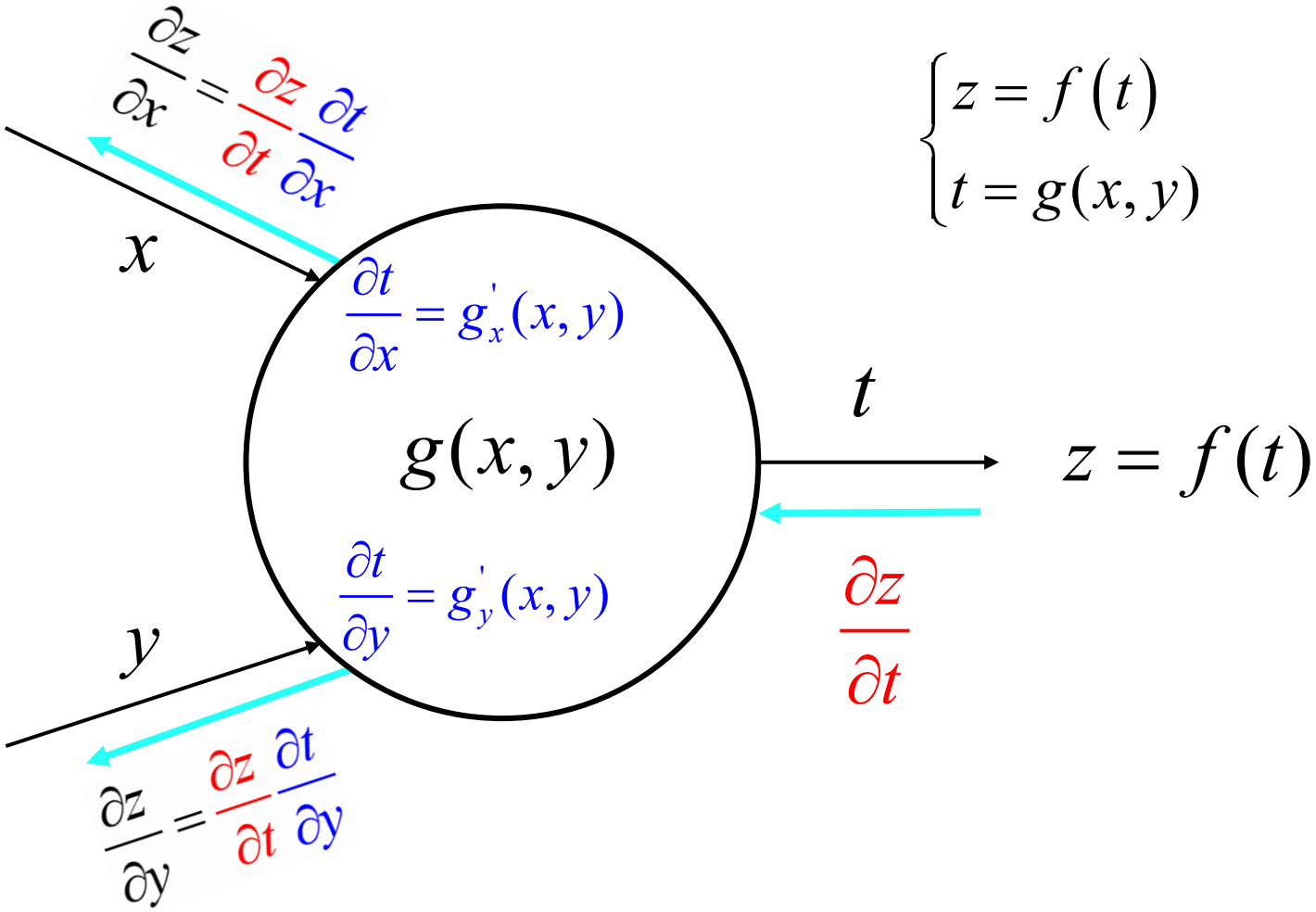
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x}$$

$$\begin{cases} z = f(t) \\ t = g(x, y) \end{cases}$$



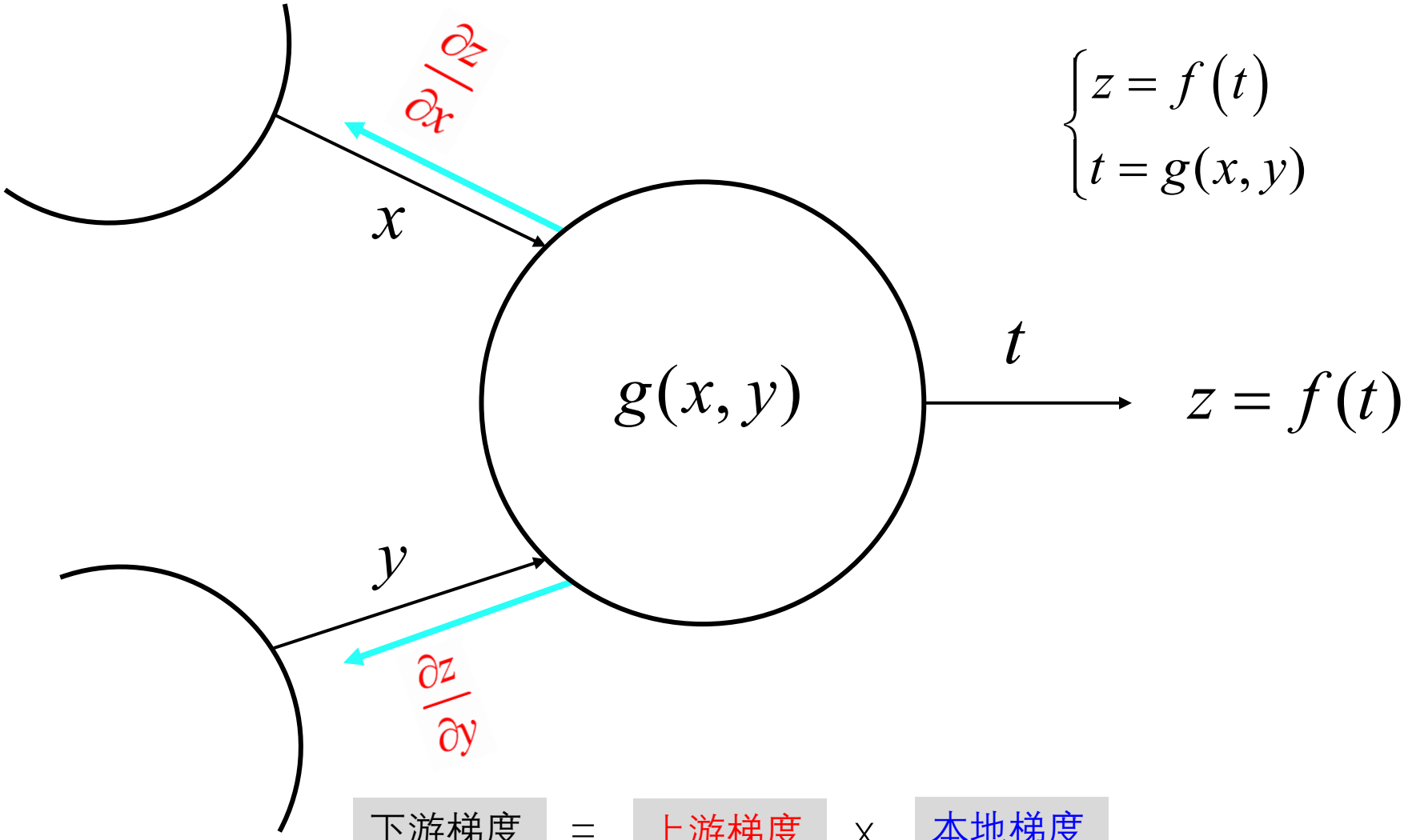
下游梯度 = 上游梯度 X 本地梯度

深入链式求导法则



下游梯度 = 上游梯度 x 本地梯度

深入链式求导法则

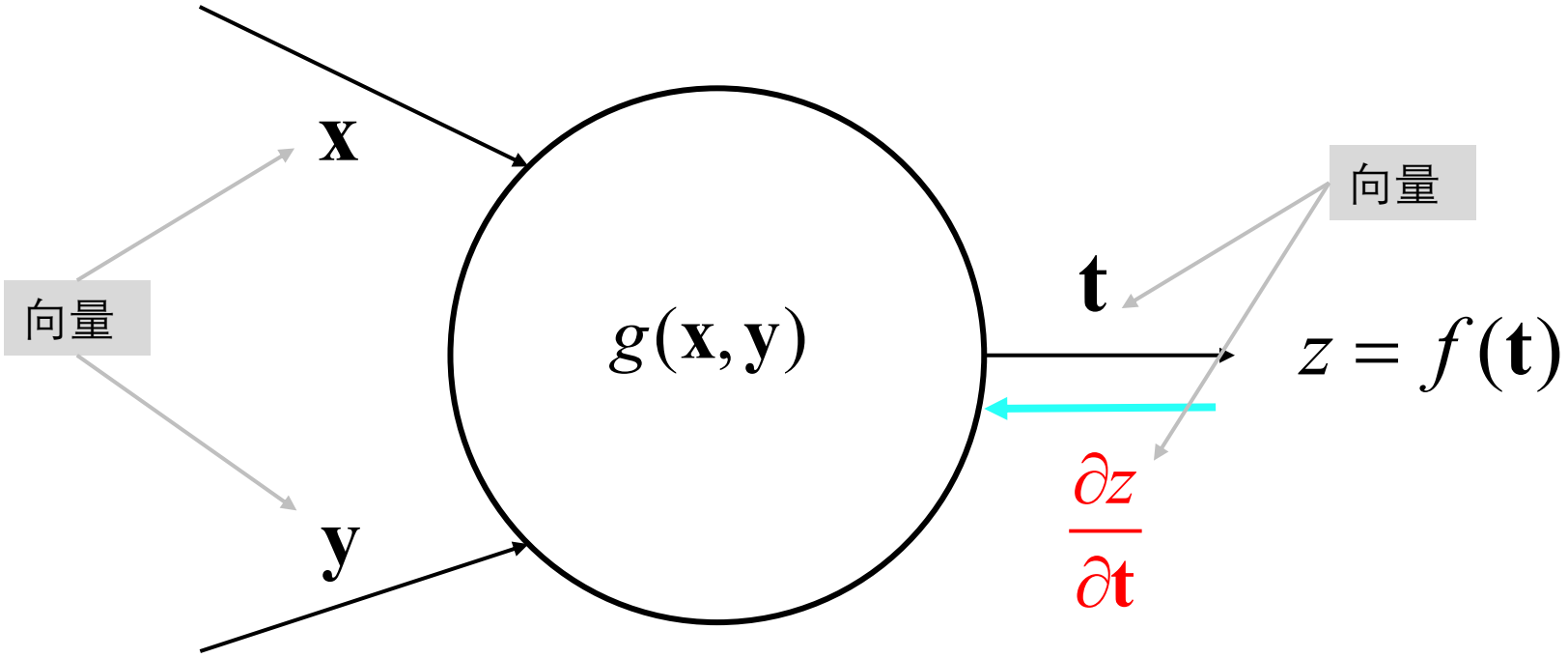


$$\begin{cases} z = f(t) \\ t = g(x,y) \end{cases}$$

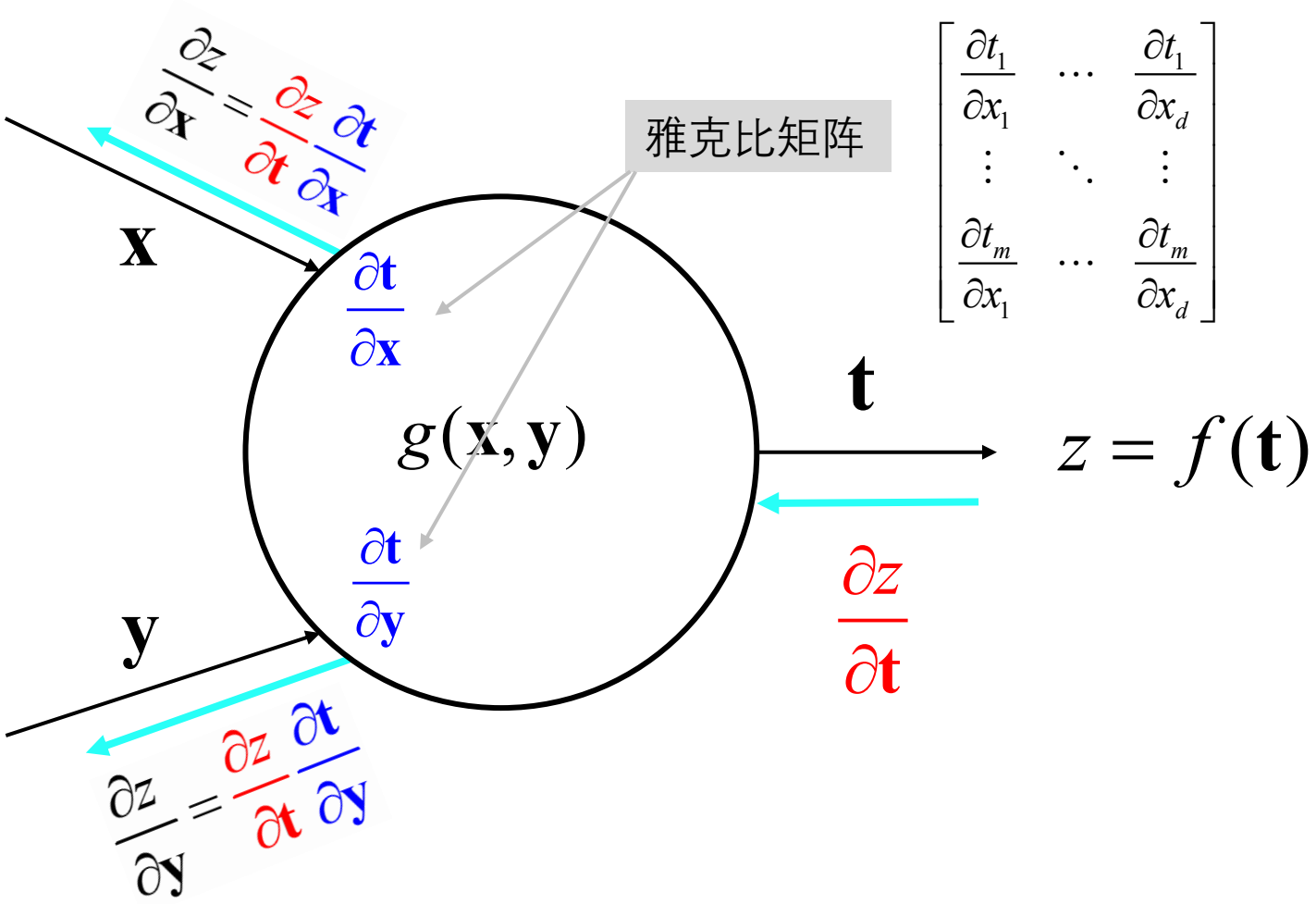
下游梯度 = 上游梯度 x 本地梯度

Examples from Stanford Course

神经网络的反向传播



神经网络的反向传播


$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial t}\frac{\partial t}{\partial x}$
雅克比矩阵
$\begin{bmatrix} \frac{\partial t_1}{\partial x_1} & \cdots & \frac{\partial t_1}{\partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial t_m}{\partial x_1} & \cdots & \frac{\partial t_m}{\partial x_d} \end{bmatrix}$
x
$\frac{\partial \mathbf{t}}{\partial \mathbf{x}}$
$g(\mathbf{x},\mathbf{y})$
t
$z = f(\mathbf{t})$
$\frac{\partial \mathbf{t}}{\partial \mathbf{y}}$
$\frac{\partial z}{\partial \mathbf{t}}$
y
$\frac{\partial z}{\partial \mathbf{y}} = \frac{\partial z}{\partial \mathbf{t}}\frac{\partial \mathbf{t}}{\partial \mathbf{y}}$

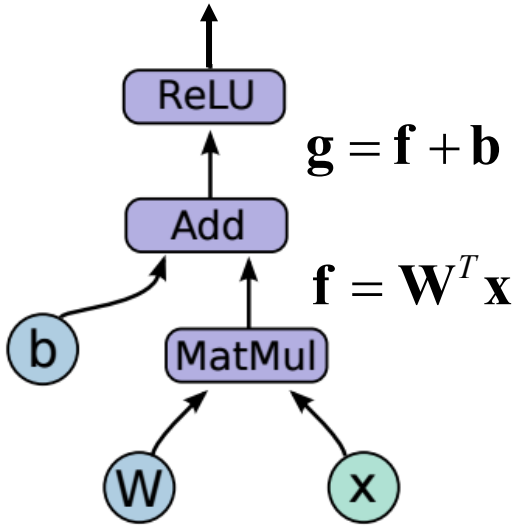
- 目标函数 $J = \| \mathbf{y} - \mathbf{t} \|^2$

$$\mathbf{y} = \max\left(\mathbf{W}^T\mathbf{x} + \mathbf{b},\ 0\right)$$

$$\frac{\partial J}{\partial \mathbf{y}} = 2(\mathbf{y} - \mathbf{t})$$

$$\mathbf{y} = \max(\mathbf{g}, 0)$$

$$\frac{\partial J}{\partial \mathbf{g}} = \frac{\partial J}{\partial \mathbf{y}} \odot \frac{\partial \mathbf{y}}{\partial \mathbf{g}} \longleftarrow \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \quad g_i > 0$$



$$\mathbf{g} = \mathbf{f} + \mathbf{b}$$

$$\mathbf{f} = \mathbf{W}^T\mathbf{x}$$

$$\frac{\partial J}{\partial \mathbf{b}} = \frac{\partial J}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{b}} \longleftarrow \mathbf{I}$$

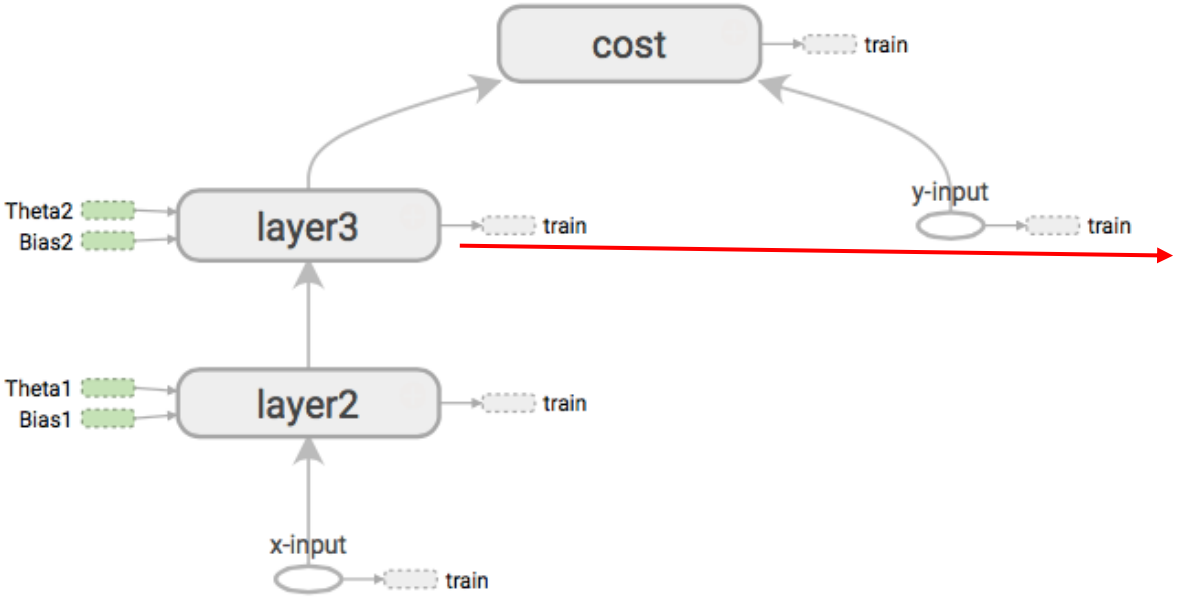
$$\frac{\partial J}{\partial \mathbf{f}} = \frac{\partial J}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{f}} \longleftarrow \mathbf{I}$$

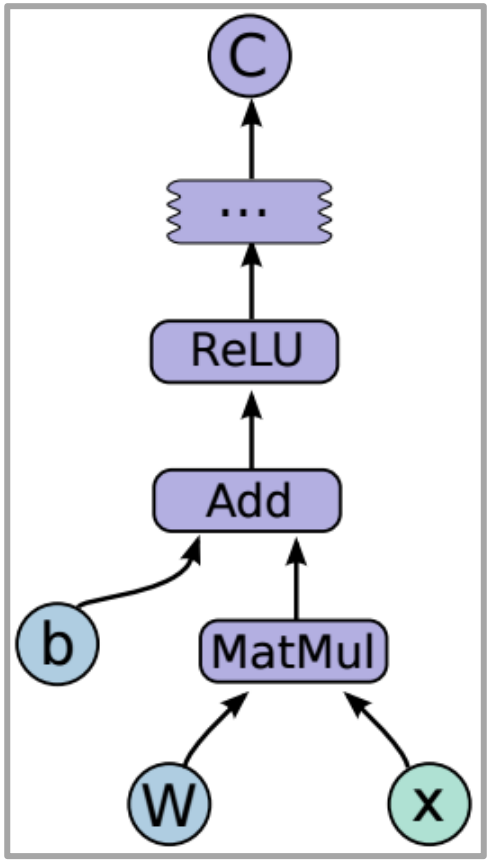
$$\frac{\partial J}{\partial \mathbf{W}} = \frac{\partial J}{\partial \mathbf{f}} \frac{\partial \mathbf{f}}{\partial \mathbf{W}} \longleftarrow \mathbf{x}^T$$

自动求导原理

- 理论上任何运算，只要给定输入输出和上游导数，就可以求下游导数，即**梯度反向传播**
- 自动求导的原理(深度学习库如Tensorflow…)

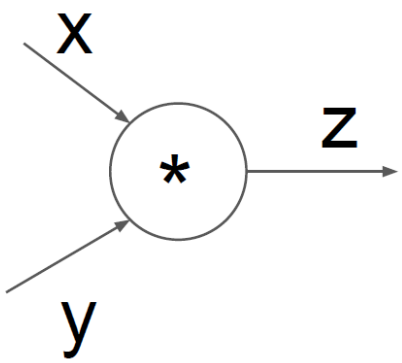
$$\mathbf{t} = \max\left(\mathbf{W}^T\mathbf{x} + \mathbf{b},\ 0\right)$$





自动求导原理

- 深度学习库通常按网络层类型采用模块化编程



(x,y,z are scalars)

```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

前向运算

逆向求导

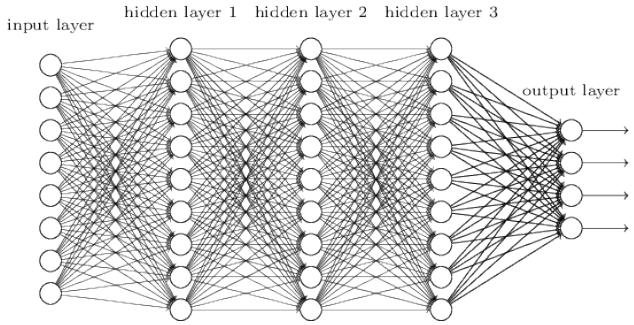
Local gradient

Upstream gradient variable

深度学习挑战性

- 网络函数复杂（非凸函数的多层嵌套）

$$\mathbf{z} = f\left(\underbrace{\mathbf{W}_L^T f\left(\mathbf{W}_{L-1}^T \underbrace{f(...)}_{L-2} + \mathbf{b}_{L-1}\right)}_{L-1} + \mathbf{b}_L\right)$$



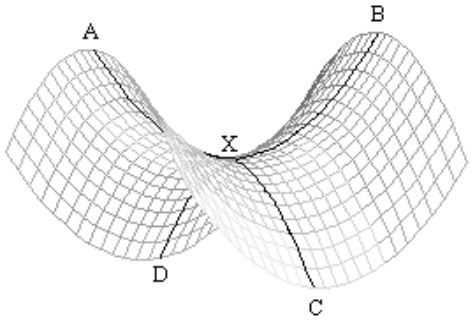
- 梯度消失或 “爆炸” （链式求导的累积效应）

$$\Delta \mathbf{W}_k = \frac{dJ}{d\hat{z}} \frac{\partial \hat{z}}{\partial \mathbf{y}_{L-1}} \frac{\partial \mathbf{y}_{L-1}}{\partial \mathbf{y}_{L-2}} \cdots \frac{\partial \mathbf{y}_k}{\partial \mathbf{W}}$$

深度学习挑战性

- 网络函数复杂（非凸函数的多层嵌套）

$$\mathbf{z} = f\left(\underbrace{\mathbf{W}_L^T f\left(\mathbf{W}_{L-1}^T \underbrace{f(...)}_{L-2} + \mathbf{b}_{L-1}\right)}_{L-1} + \mathbf{b}_L\right)$$



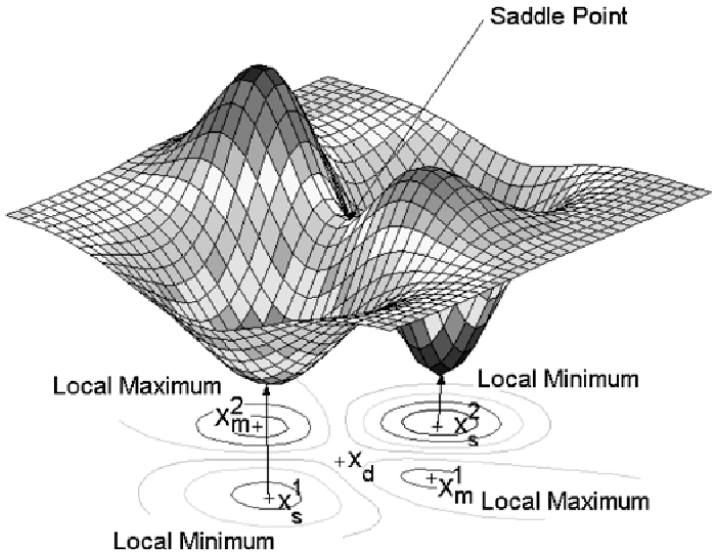
- 梯度消失或 “爆炸” （链式求导的累积效应）

$$\Delta \mathbf{W}_k = \frac{dJ}{d\hat{z}} \frac{\partial \hat{z}}{\partial \mathbf{y}_{L-1}} \frac{\partial \mathbf{y}_{L-1}}{\partial \mathbf{y}_{L-2}} \cdots \frac{\partial \mathbf{y}_k}{\partial \mathbf{W}}$$

- 高维空间中常见的鞍点（梯度为0非极值点）

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \lambda \Delta \mathbf{W}$$

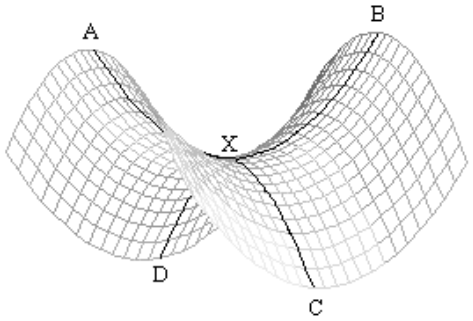
- 大量质量参差不齐的局部次优解（非凸性）
- 海量标记样本和巨大计算量 (网络参数多)



深度学习挑战性

- 网络函数复杂（非凸函数的多层嵌套）

$$\mathbf{z} = f\left(\underbrace{\mathbf{W}_L^T f\left(\mathbf{W}_{L-1}^T \underbrace{f(...)}_{L-2} + \mathbf{b}_{L-1}\right)}_{L-1} + \mathbf{b}_L\right)$$

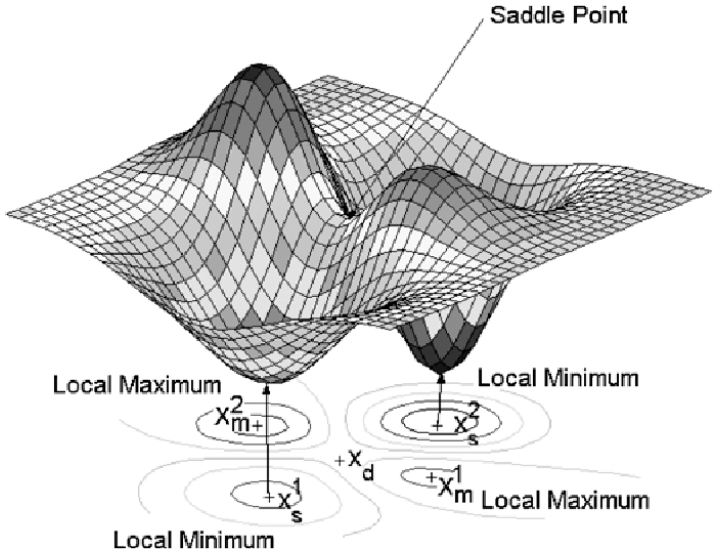


- 梯度消失或 “爆炸” （链式求导的累积效应）

$$\Delta \mathbf{W}_k = \frac{dJ}{d\hat{z}} \frac{\partial \hat{z}}{\partial \mathbf{y}_{L-1}} \frac{\partial \mathbf{y}_{L-1}}{\partial \mathbf{y}_{L-2}} \cdots \frac{\partial \mathbf{y}_k}{\partial \mathbf{W}}$$

- 高维空间中常见的鞍点（梯度为0非极值点）

$$\mathbf{W}^{(t+1)} = \mathbf{W}^{(t)} - \lambda \Delta \mathbf{W}$$

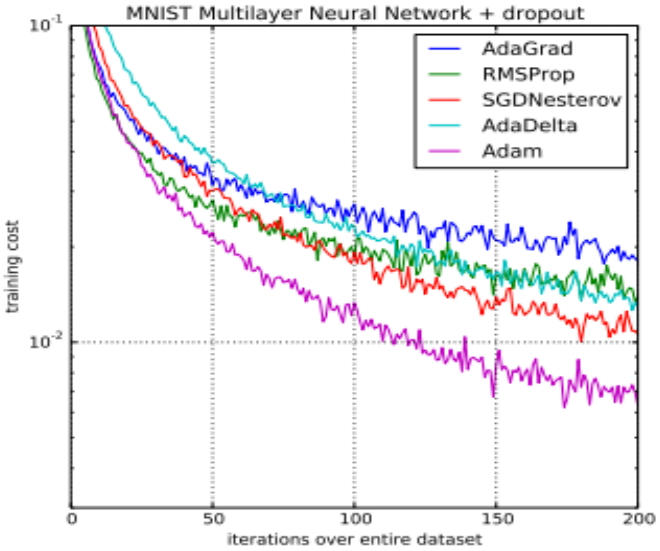


- 大量质量参差不齐的局部次优解（非凸性）
- 海量标记样本和巨大计算量 (网络参数多)

关键：优化算法！

优化算法

Name	Update Rule
SGD	$\Delta\theta_t = -\alpha g_t$
Momentum	$m_t = \gamma m_{t-1} + (1-\gamma) g_t,$ $\Delta\theta_t = -\alpha\, m_t$
Adagrad	$G_t = G_{t-1} + g_t^2,$ $\Delta\theta_t = -\alpha\, g_t G_t^{-1/2}$
Adadelta	$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$ $\Delta\theta_t = -\alpha g_t v_t^{-1/2} D_{t-1}^{1/2},$ $D_t = \beta_1 D_{t-1} + (1-\beta_1)(\Delta\theta_t/\alpha)^2$
RMSprop	$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$ $\Delta\theta_t = -\alpha\, g_t v_t^{-1/2}$
Adam	$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t,$ $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$ $\hat{m}_t = m_t/(1-\beta_1^t),$ $\hat{v}_t = v_t/(1-\beta_2^t),$ $\Delta\theta_t = -\alpha\, \hat{m}_t \hat{v}_t^{-1/2}$

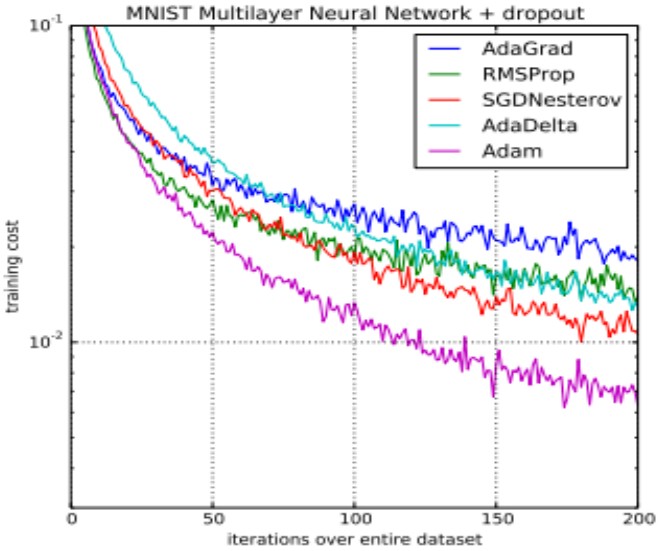


$$\theta_{t+1} = \theta_{t+1} - \lambda \bullet f(g_t)$$

不同优化算法的差异地方：
梯度的变换函数不同

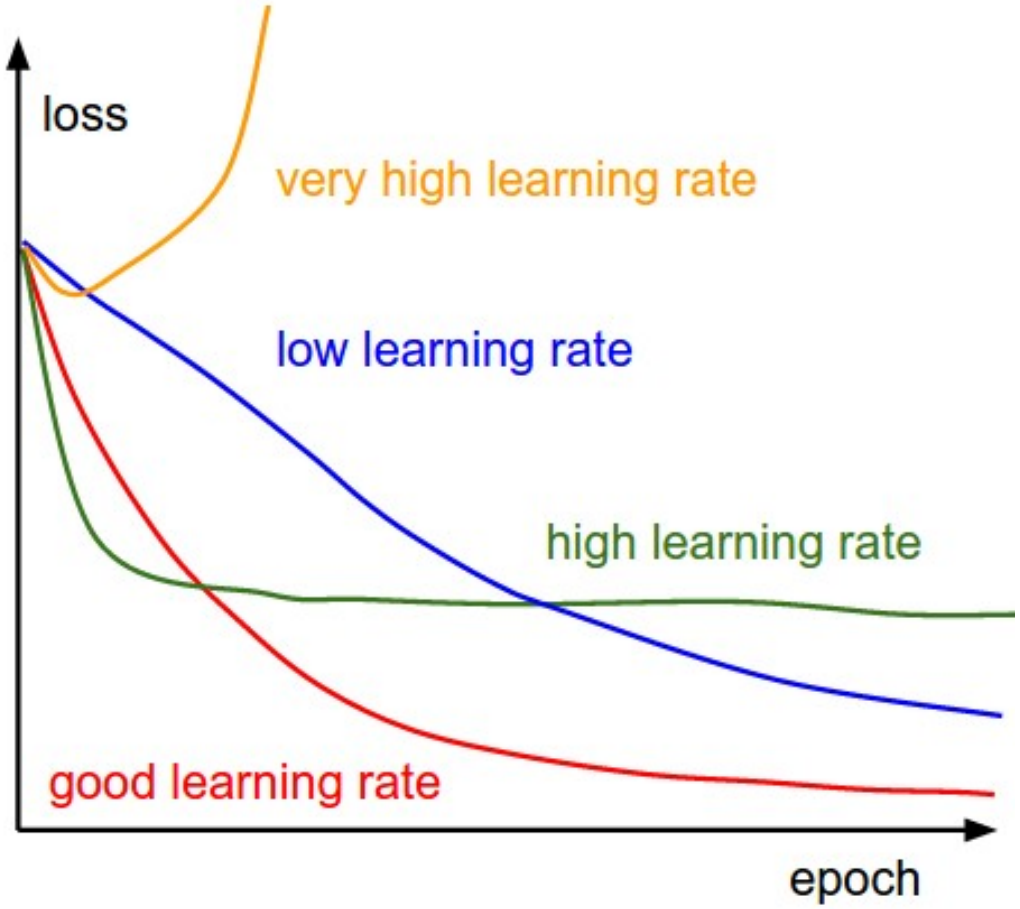
优化算法

Name	Update Rule
SGD	$\Delta\theta_t = -\alpha g_t$
Momentum	$m_t = \gamma m_{t-1} + (1-\gamma) g_t,$ $\Delta\theta_t = -\alpha\, m_t$
Adagrad	$G_t = G_{t-1} + g_t^2,$ $\Delta\theta_t = -\alpha\, g_t G_t^{-1/2}$
Adadelta	$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$ $\Delta\theta_t = -\alpha g_t v_t^{-1/2} D_{t-1}^{1/2},$ $D_t = \beta_1 D_{t-1} + (1-\beta_1)(\Delta\theta_t/\alpha)^2$
RMSprop	$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$ $\Delta\theta_t = -\alpha\, g_t v_t^{-1/2}$
Adam	$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t,$ $v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2,$ $\hat{m}_t = m_t/(1-\beta_1^t),$ $\hat{v}_t = v_t/(1-\beta_2^t),$ $\Delta\theta_t = -\alpha\, \hat{m}_t \hat{v}_t^{-1/2}$



优化算法

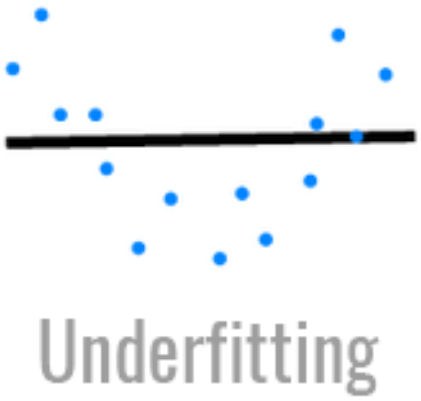
不同学习率对算法收敛影响也较大



欠拟合与过拟合

- 稠密的连接使得网络具有强大的拟合能力，容易导致过拟合发生

欠拟合



模型复杂度<数据分布

正拟合

模型复杂度=数据分布

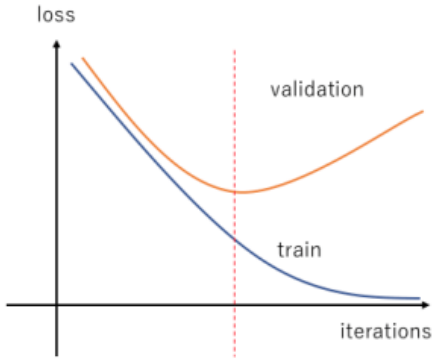
过拟合

模型复杂度>数据分布

避免策略

- 避免欠拟合常用策略：
  - 增加网络宽度和深度
  - 减小正则化项权重
- 避免过拟合常用策略：
  - 减小网络宽带或深度
  - 扩充训练数据（新增数据或者变换已有数据）
  - 引入dropout
  - 增大正则化项权重
  - 交叉验证，提前终止训练
  - ……

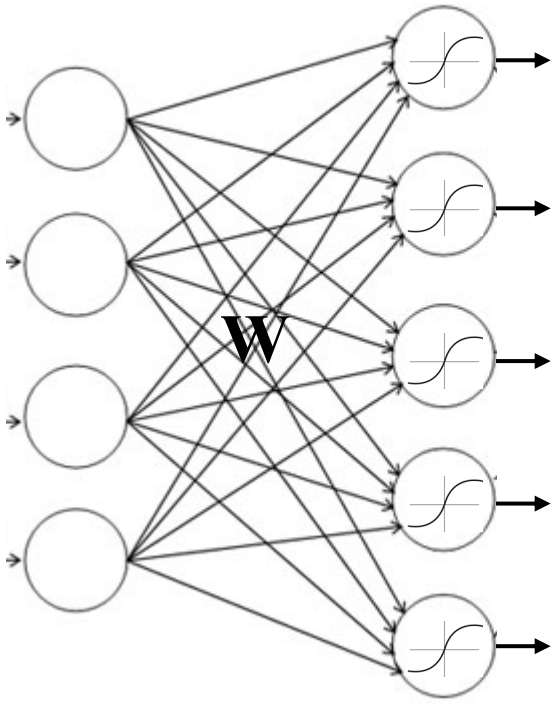
更严重



基本网络层类型

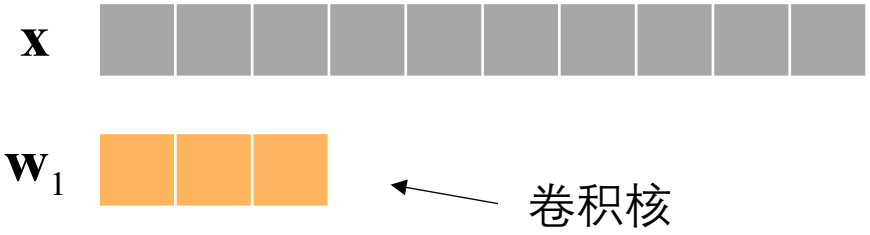
全连接，卷积，聚集，dropout，归一化

全连接层



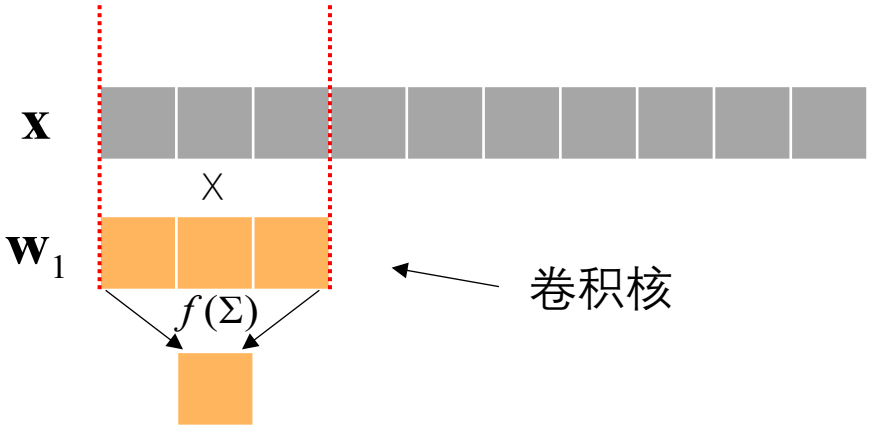
$$\mathbf{y} = f(\mathbf{W}^T\mathbf{x} + \mathbf{b})$$

卷积神经网络 – 卷积



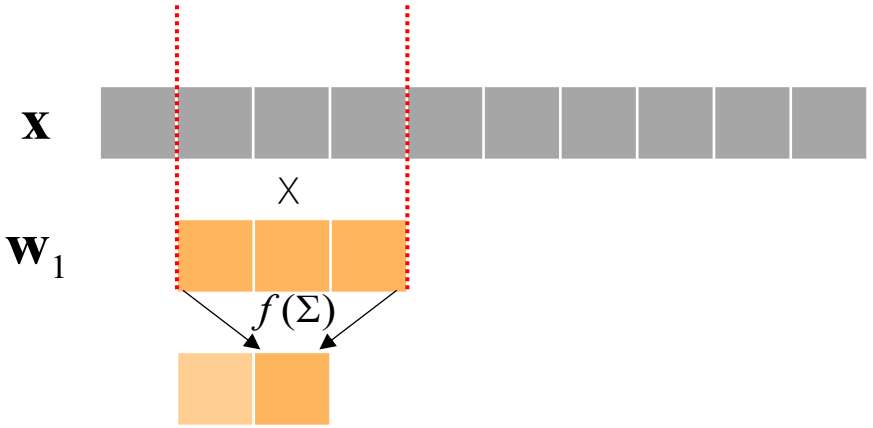
一维

卷积神经网络 – 卷积



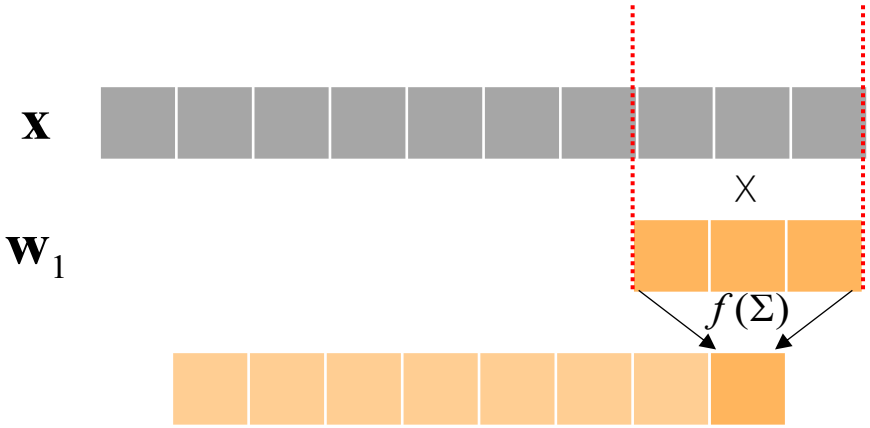
一维

卷积神经网络 – 卷积



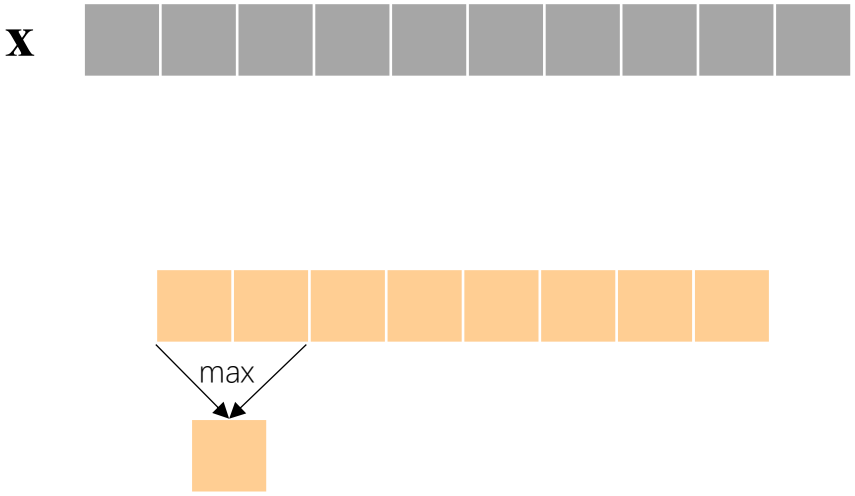
一维

卷积神经网络 – 卷积



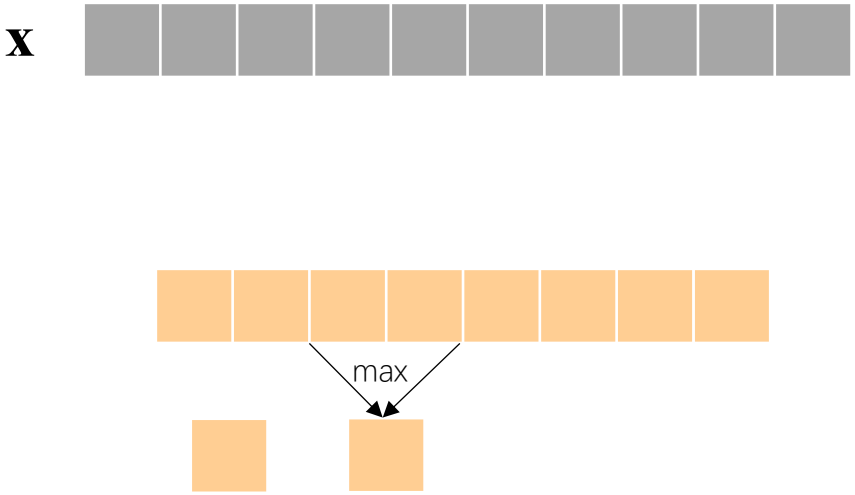
一维

卷积神经网络 – 聚集



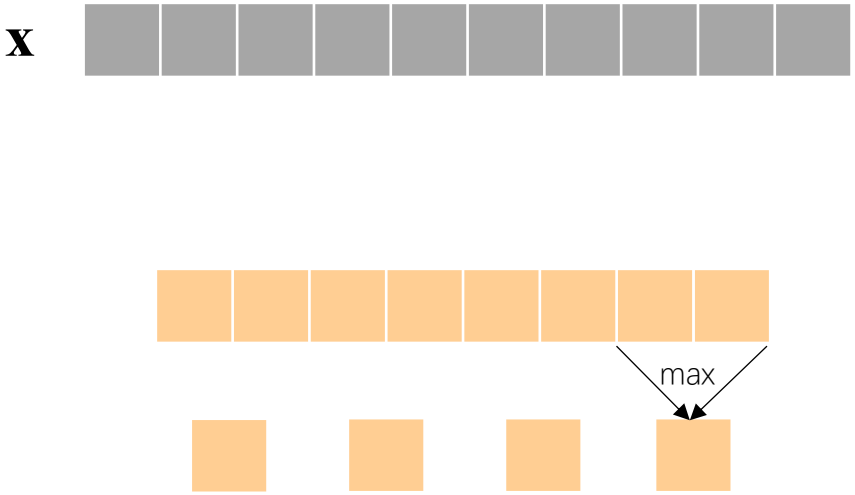
一维

卷积神经网络 – 聚集



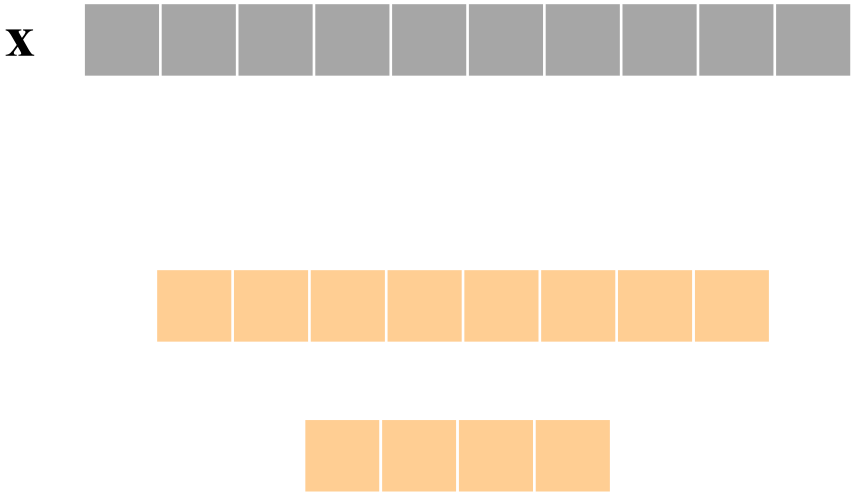
一维

卷积神经网络 – 聚集



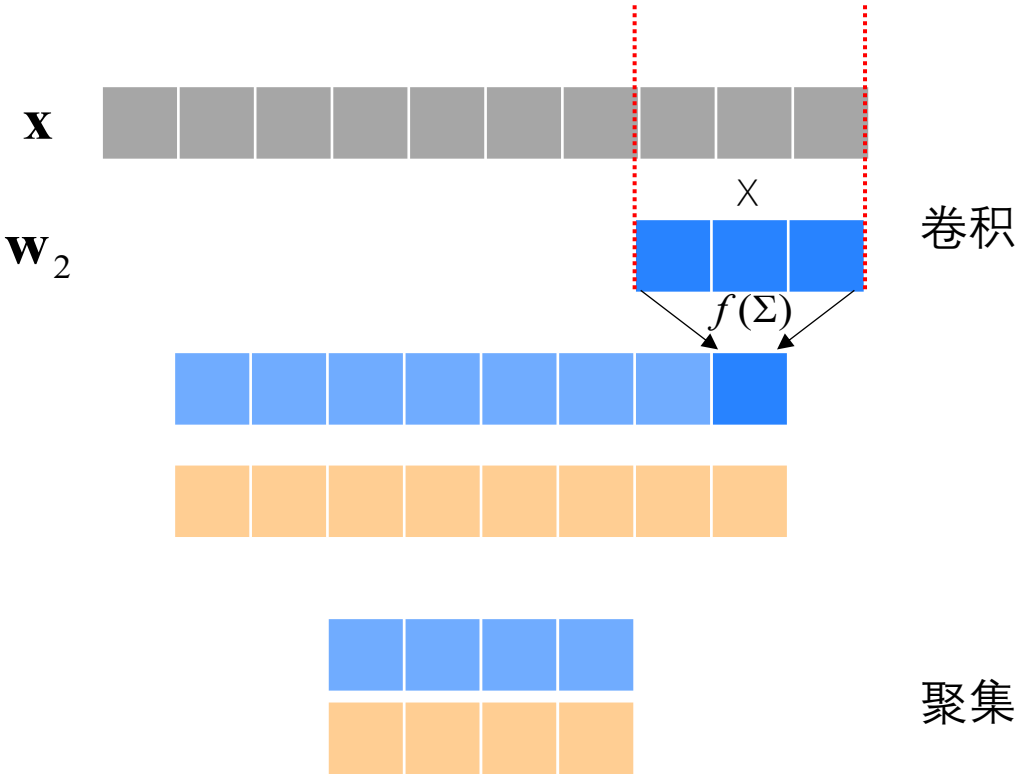
一维

卷积神经网络 – 聚集

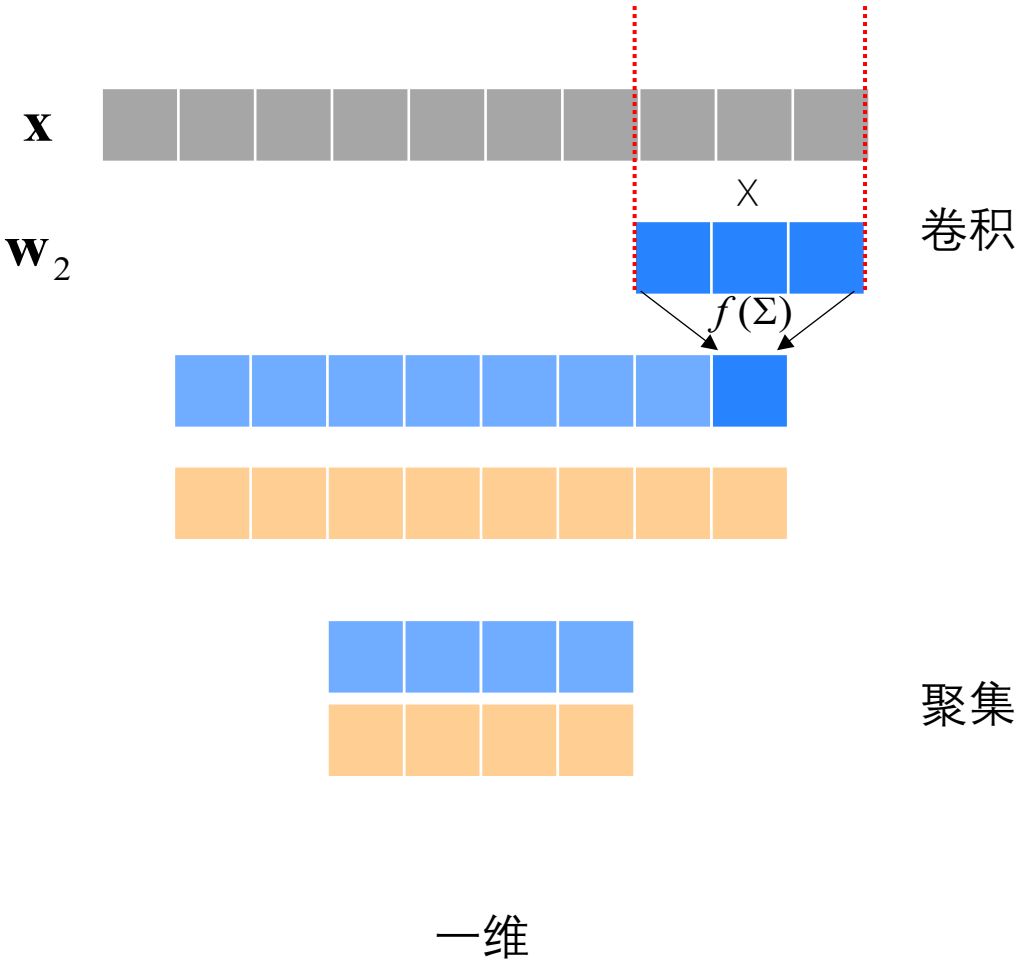


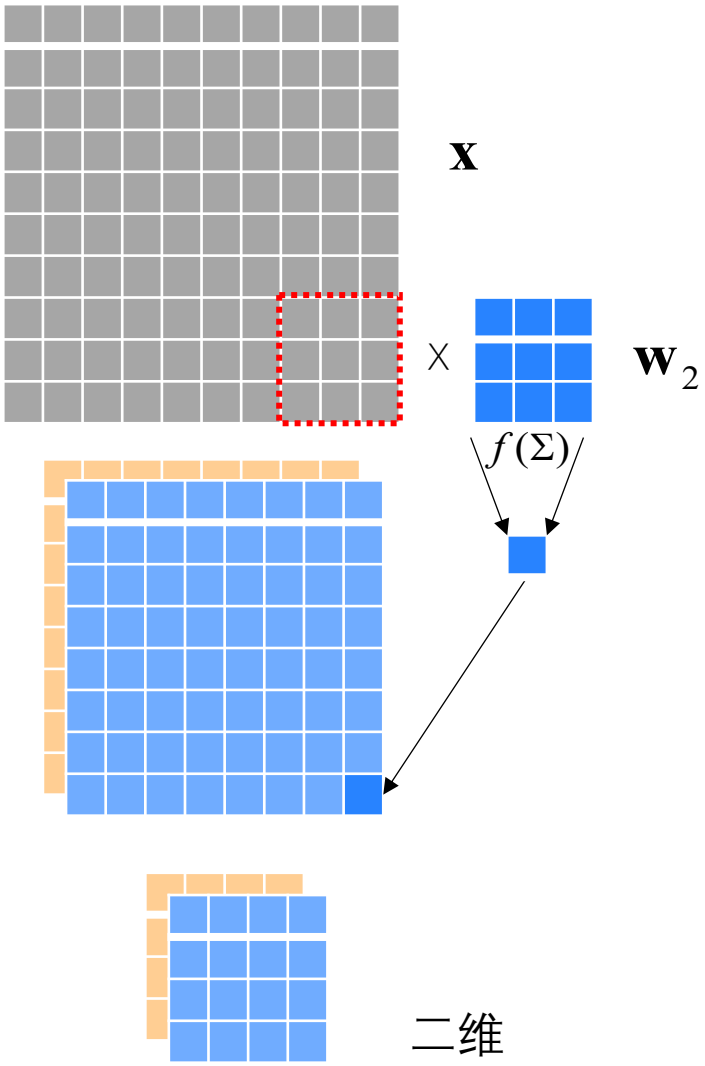
一维

卷积神经网络



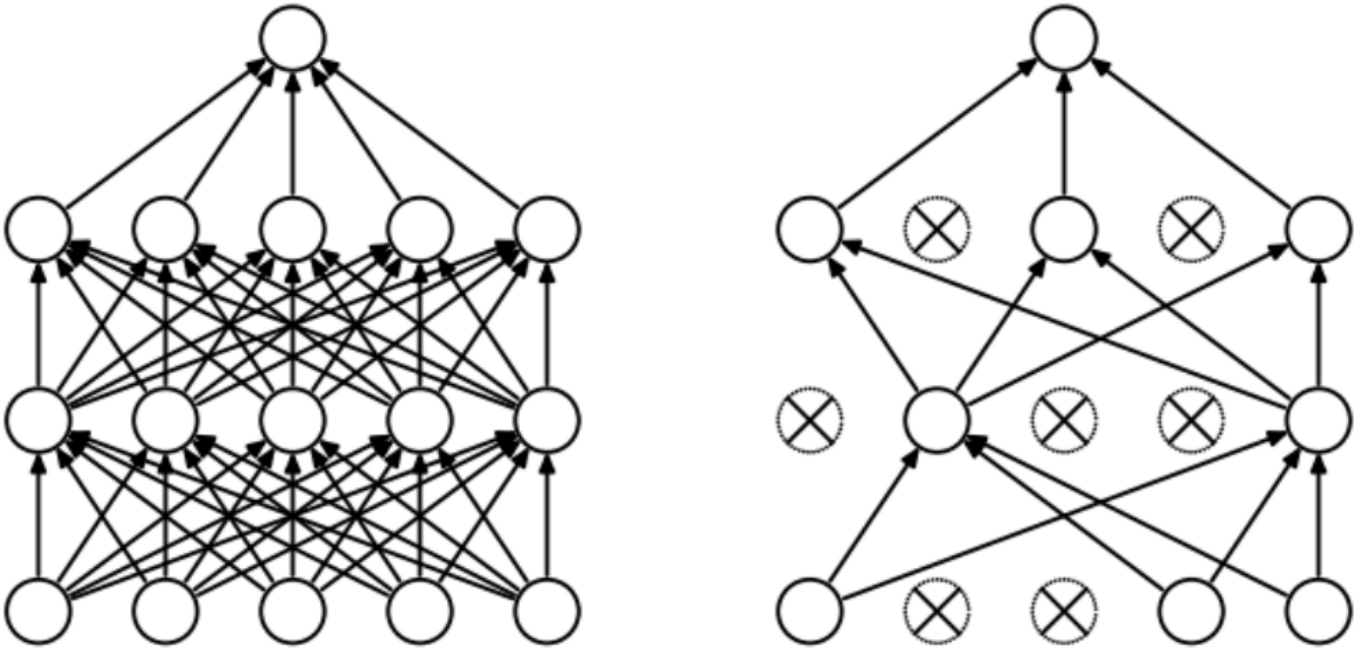
卷积神经网络





Dropout

- 为了减小过拟合，训练中随机丢掉一些连接（测试时全使用）

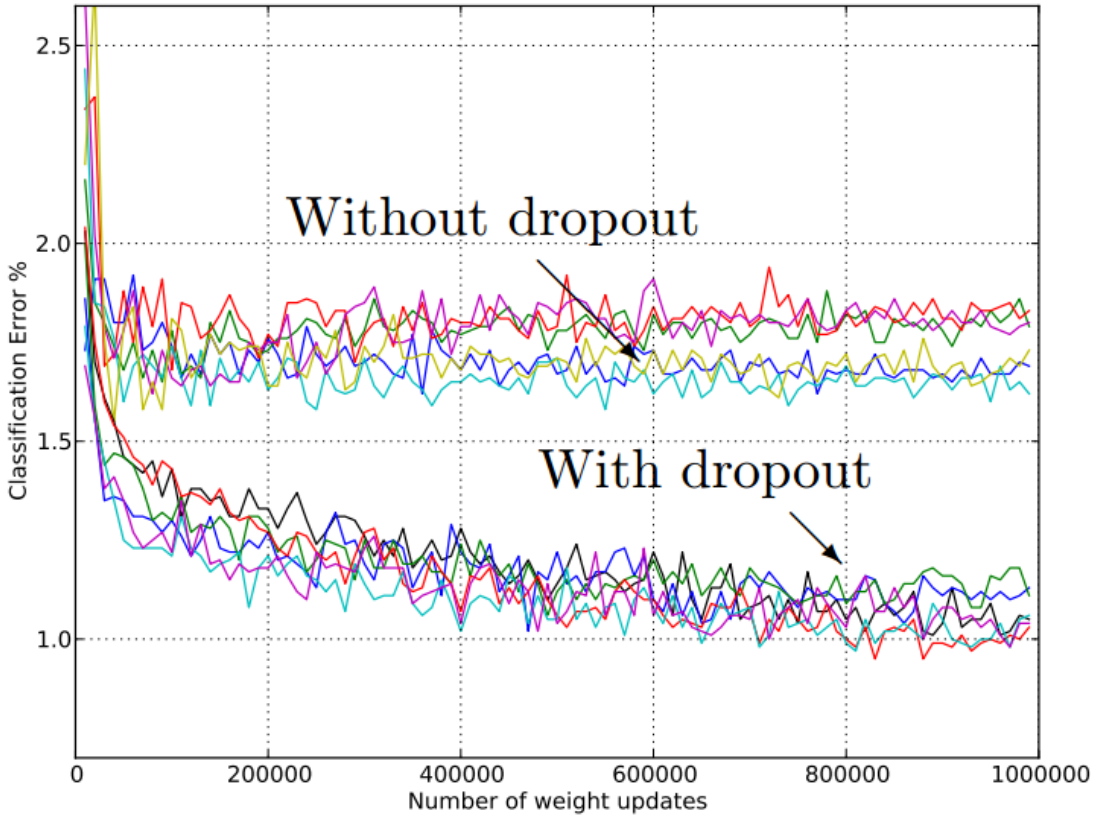


(a) Standard Neural Net

(b) After applying dropout.

Dropout

- 为了减小过拟合，训练中随机丢掉一些连接（测试时全使用）



归一化层

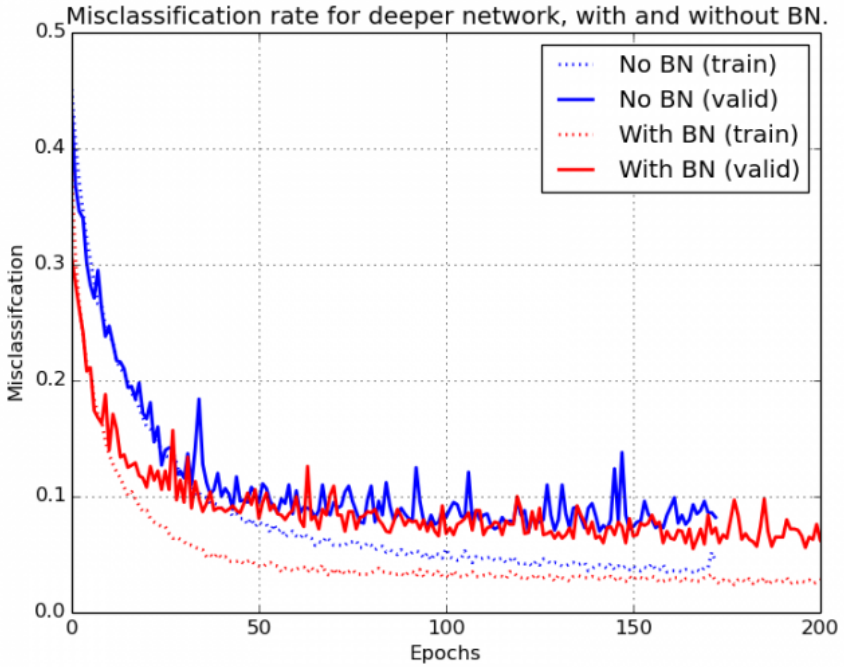
Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

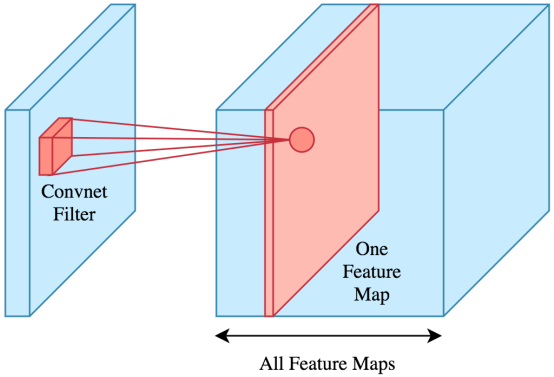
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

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

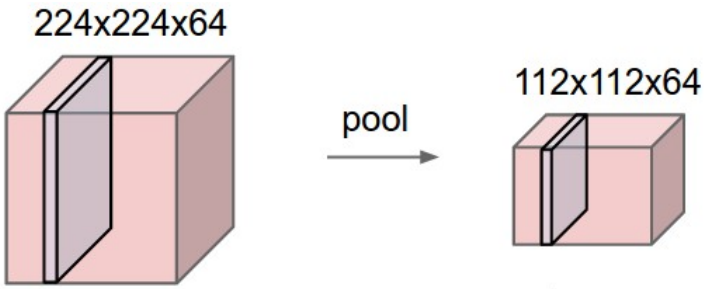
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$



基本网络层类型

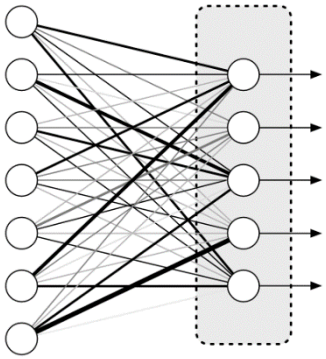


卷积层(convolutional layer)

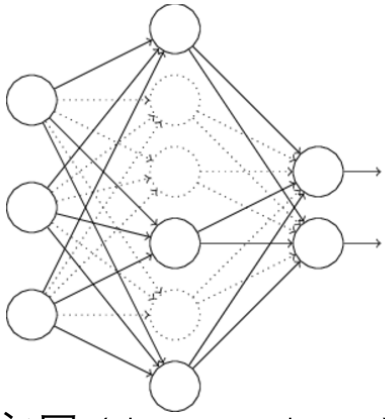


聚集层(Pooling layer)

注：pooling多数中文翻译成“池化” 具有一定误导，
这里本人倾向于根据意义翻译成“聚集”

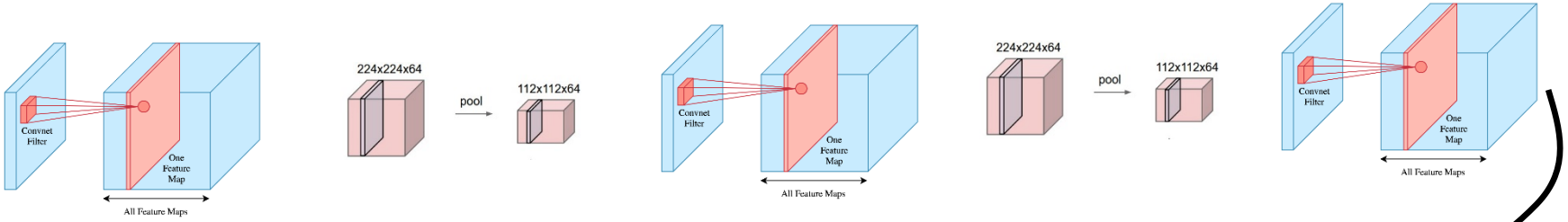


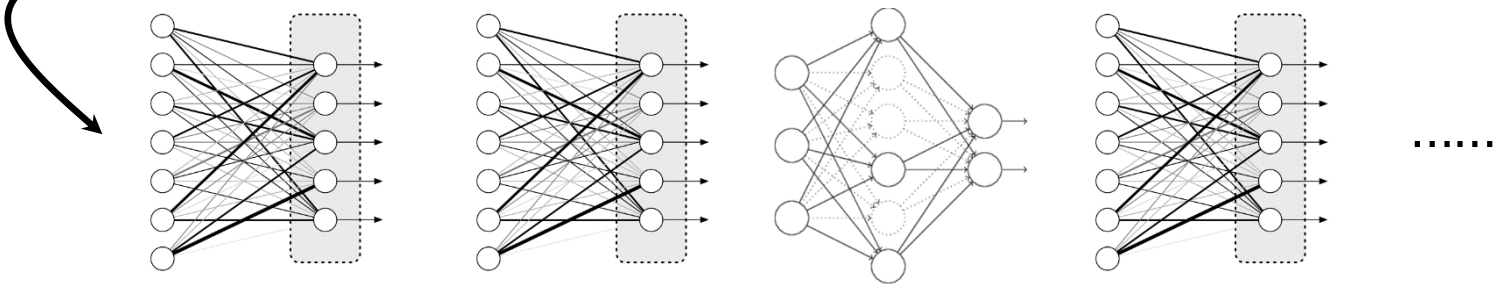
全连接层(fully connected layer)



丢弃层 (dropout layer)

深度网络 = 垒积木





三无三费：无规律，无理论，无保证；费时间，费体力，费资源

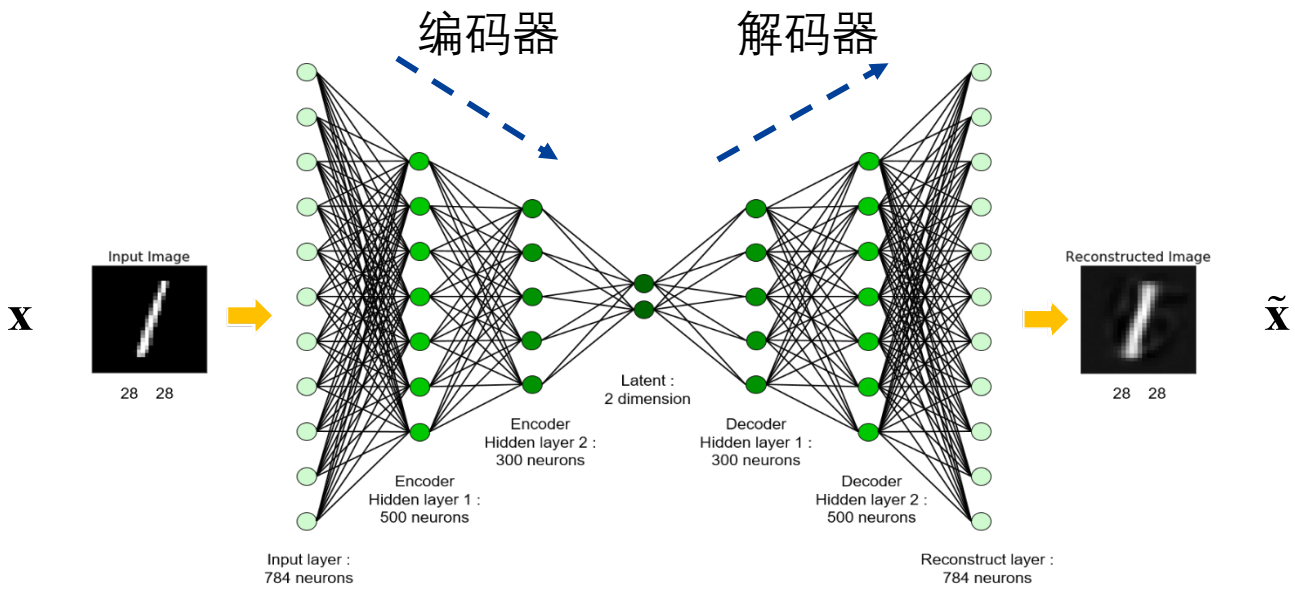
成功窍门：多尝试（努力，努力，再努力！）

为什么还要研究：通常找对了模型精度就碾压 “传统” 方法，值得探索完善！

典型网络架构

AutoEncoder, LeNet, AlexNet, VGG, GoogleNet, ResNet, Unet, GAN

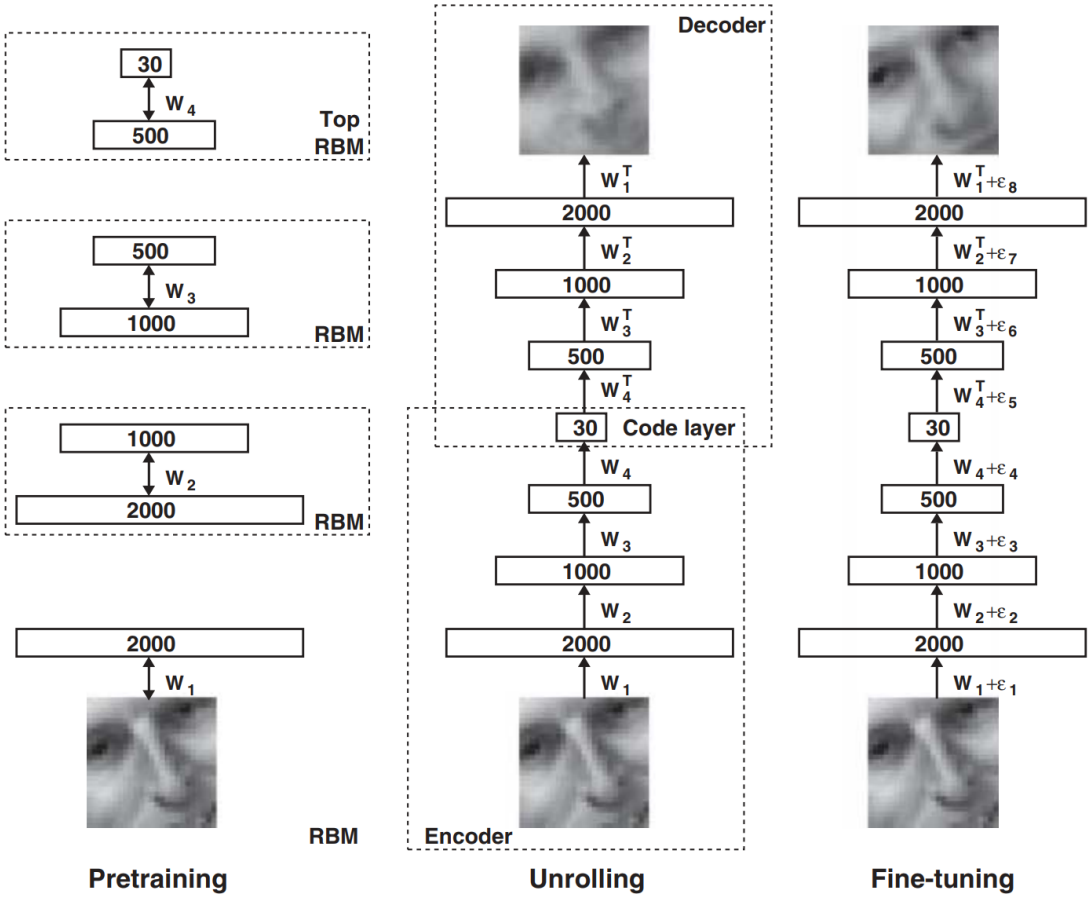
前馈自编码网络



- 目标函数 $J(x) = \|\mathbf{x} - \tilde{\mathbf{x}}\|^2$
- 用途：特征学习、数据可视化、去燥、半监督学习等

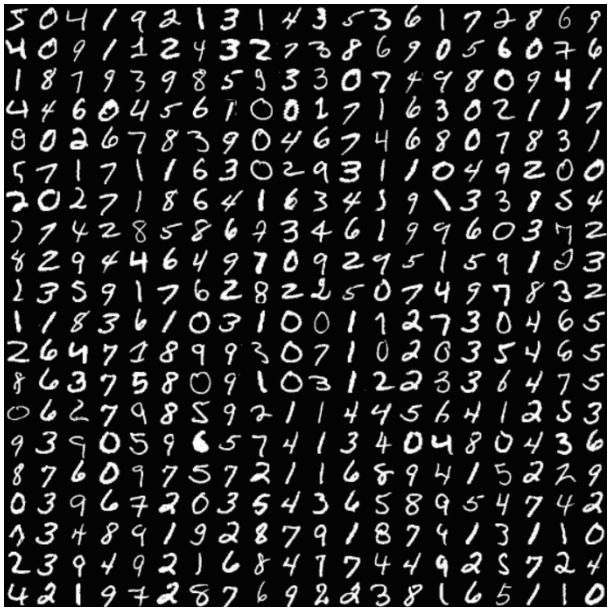
前馈自编码网络

- Science, Hinton & Salakhutdinov, 2006

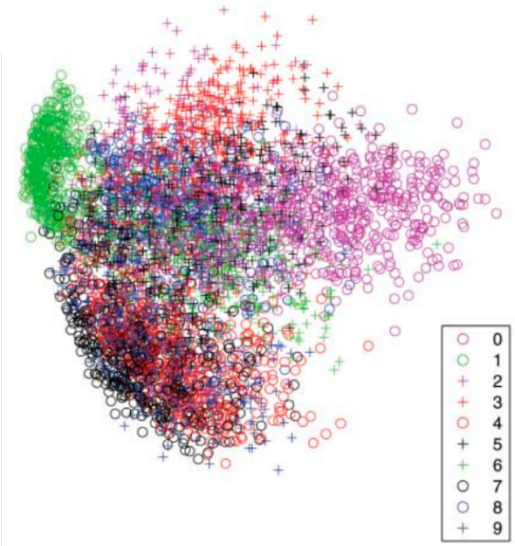


前馈自编码网络

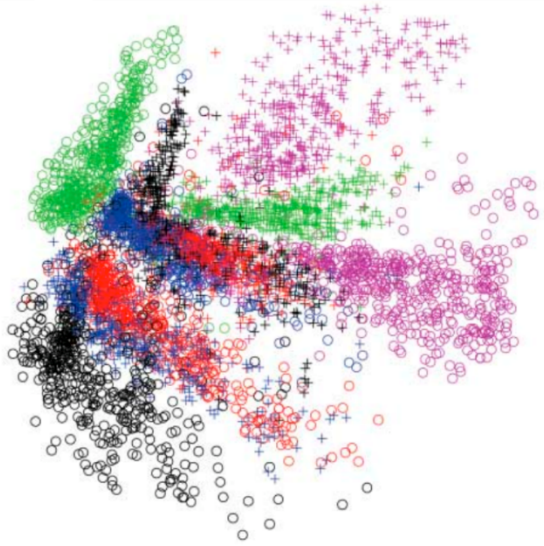
- MNIST手写数字可视化比较（PCA和自编码网络）



MNIST手写数字



PCA



自编码器

典型网络架构 - LeNet

- 最早的深度学习网络架构

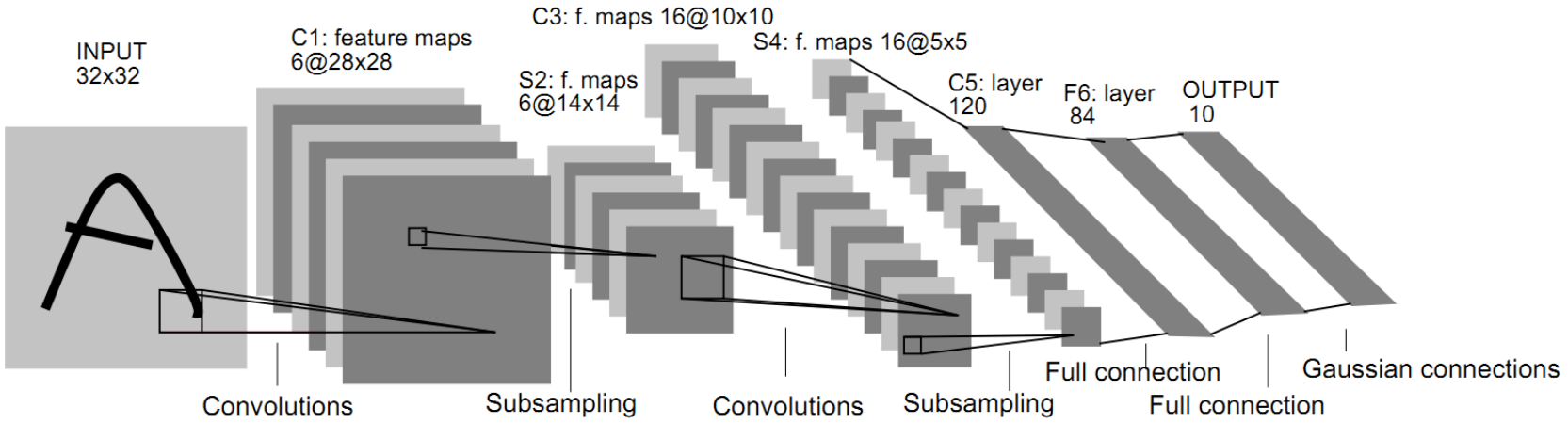
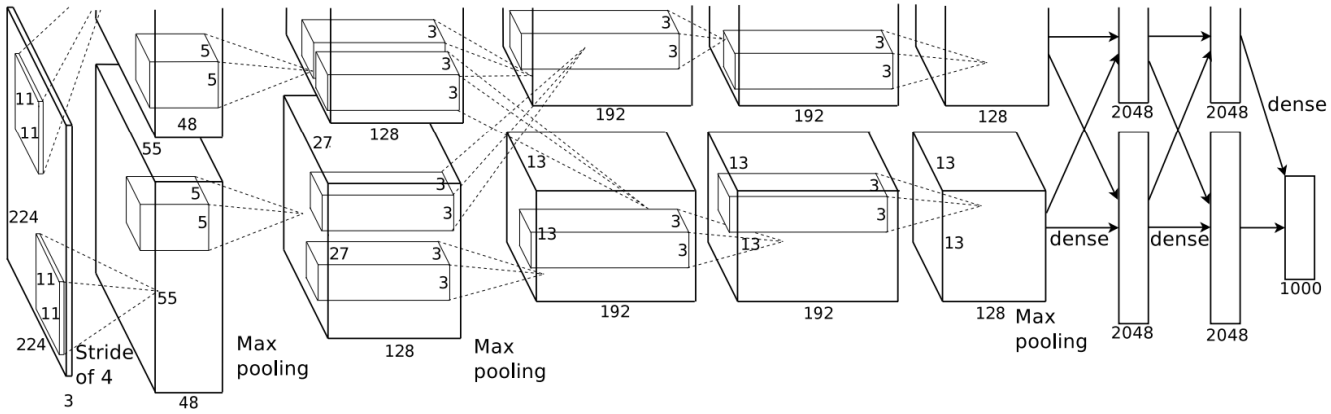


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

- LeCun, Yann, Léon Bottou, Yoshua Bengio, and Patrick Haffner. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE* 86, no. 11 (1998): 2278-2324.

典型网络架构 - AlexNet

- 影响最大的深度学习网络架构（ReLu, GPU, Normlization）



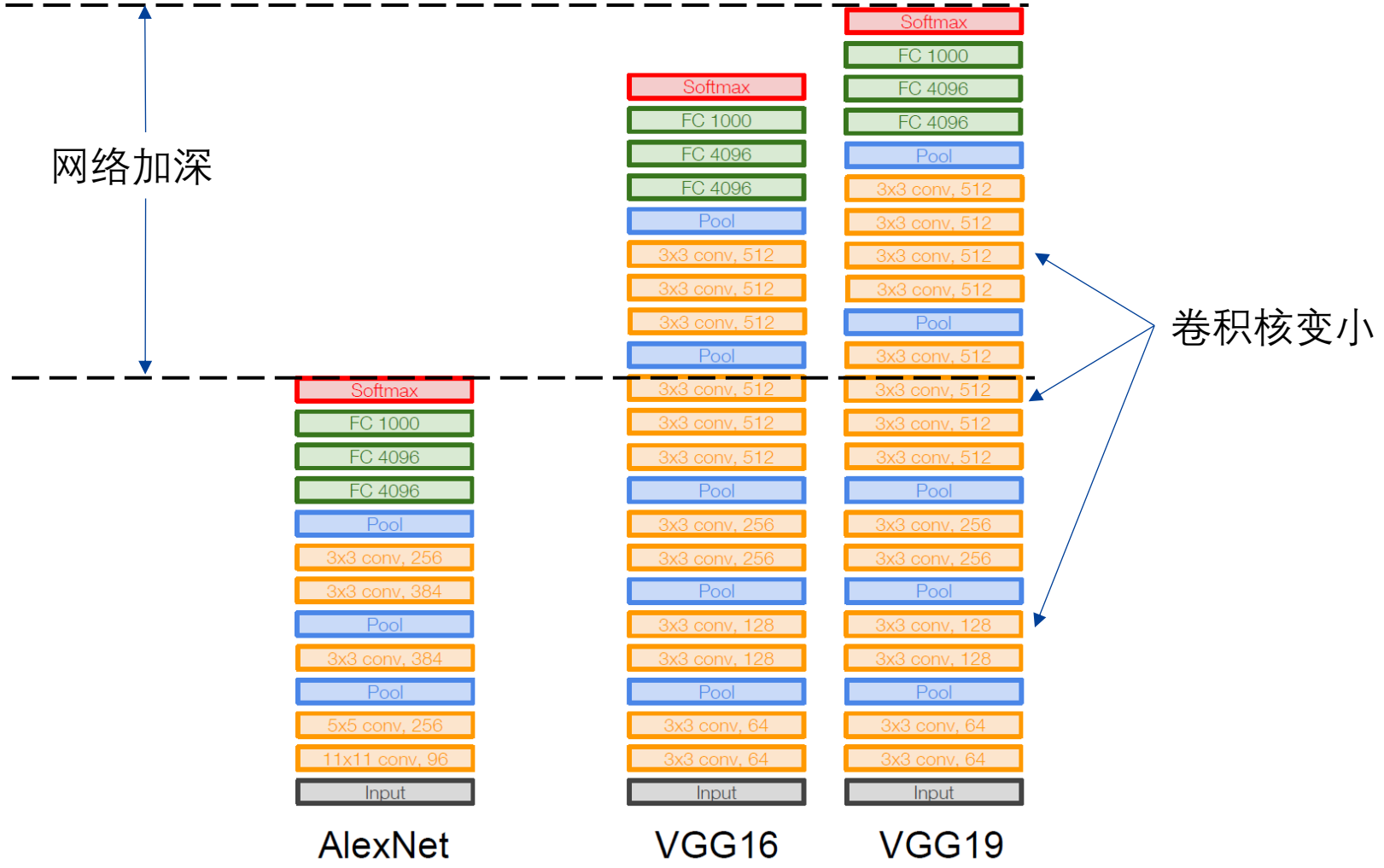
Model	Top-1	Top-5
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**

Table 1: Comparison of results on ILSVRC-2010 test set. In *italics* are best results achieved by others.

- Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." In *Advances in neural information processing systems*, pp. 1097-1105. 2012.

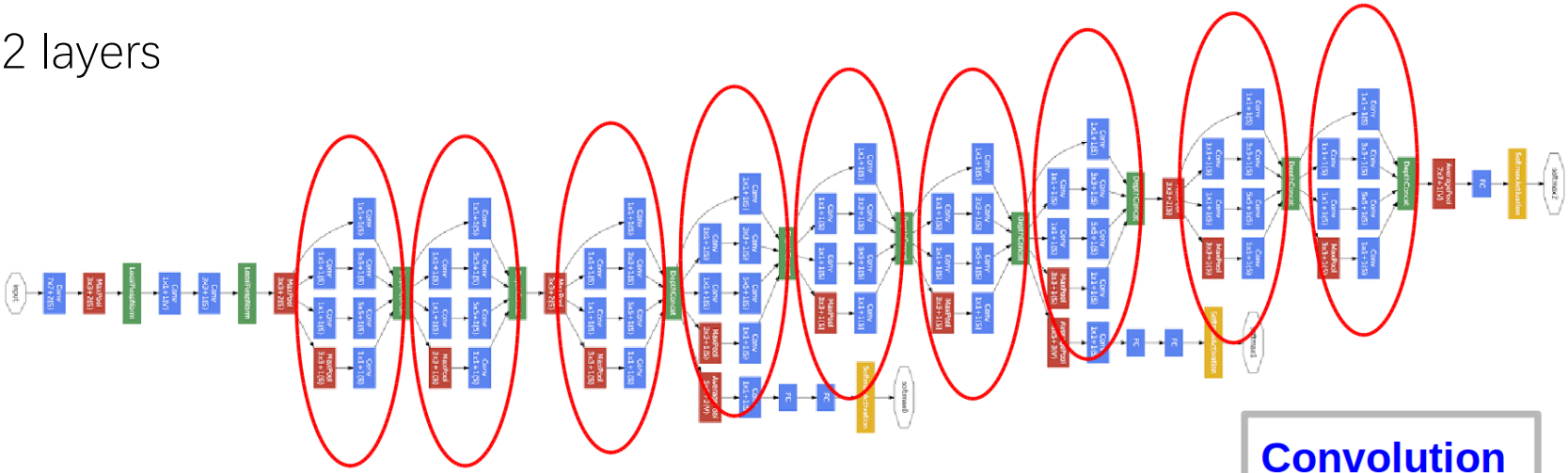
典型网络架构 – VGG Net

典型网络架构 – VGG Net

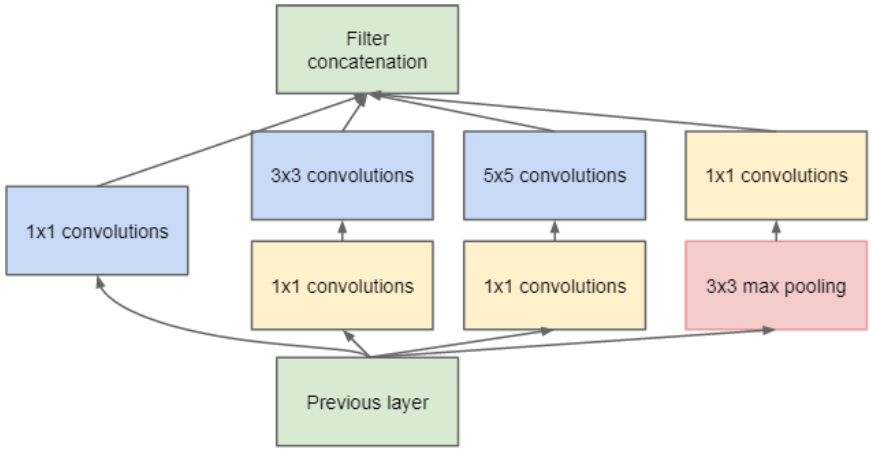


典型网络架构 – Google Net

- 22 layers



Convolution
Pooling
Softmax
Concat/Normalize



(b) Inception module with dimension reductions

典型网络架构 – ResNet

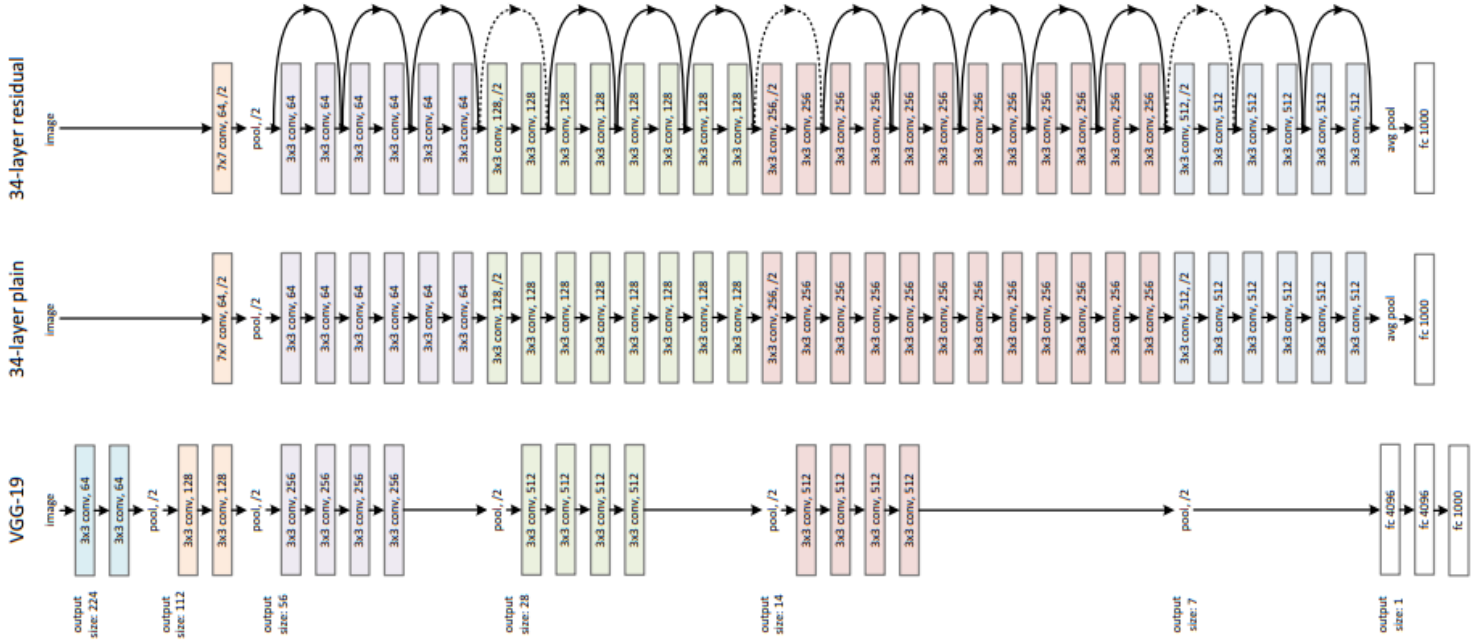


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [40] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.

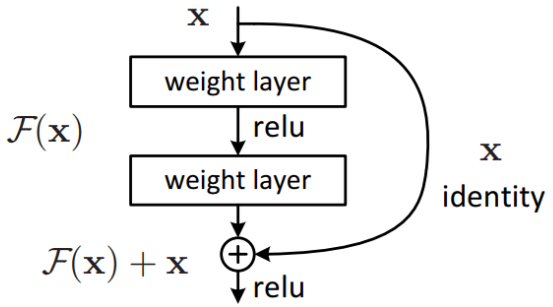
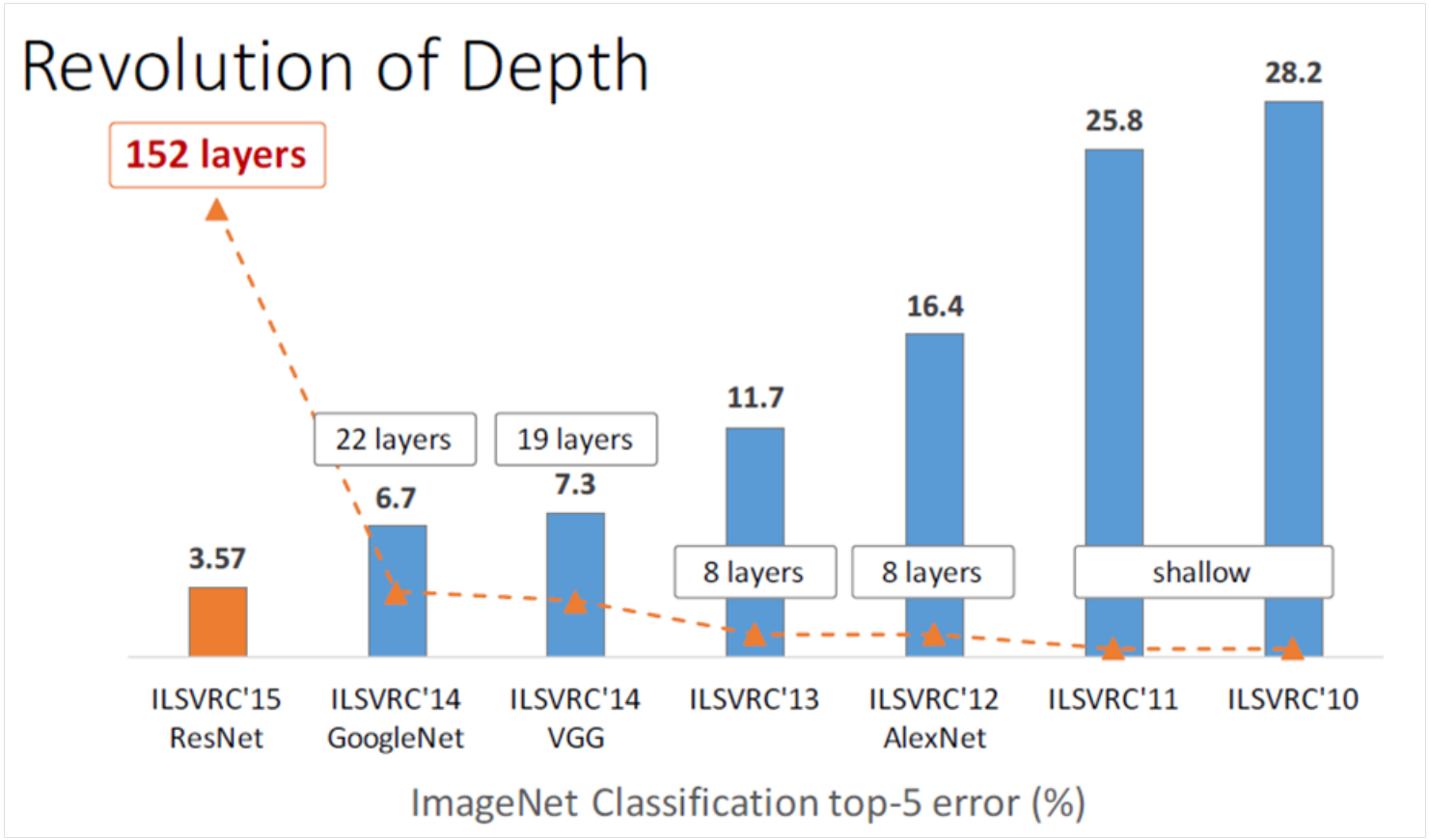


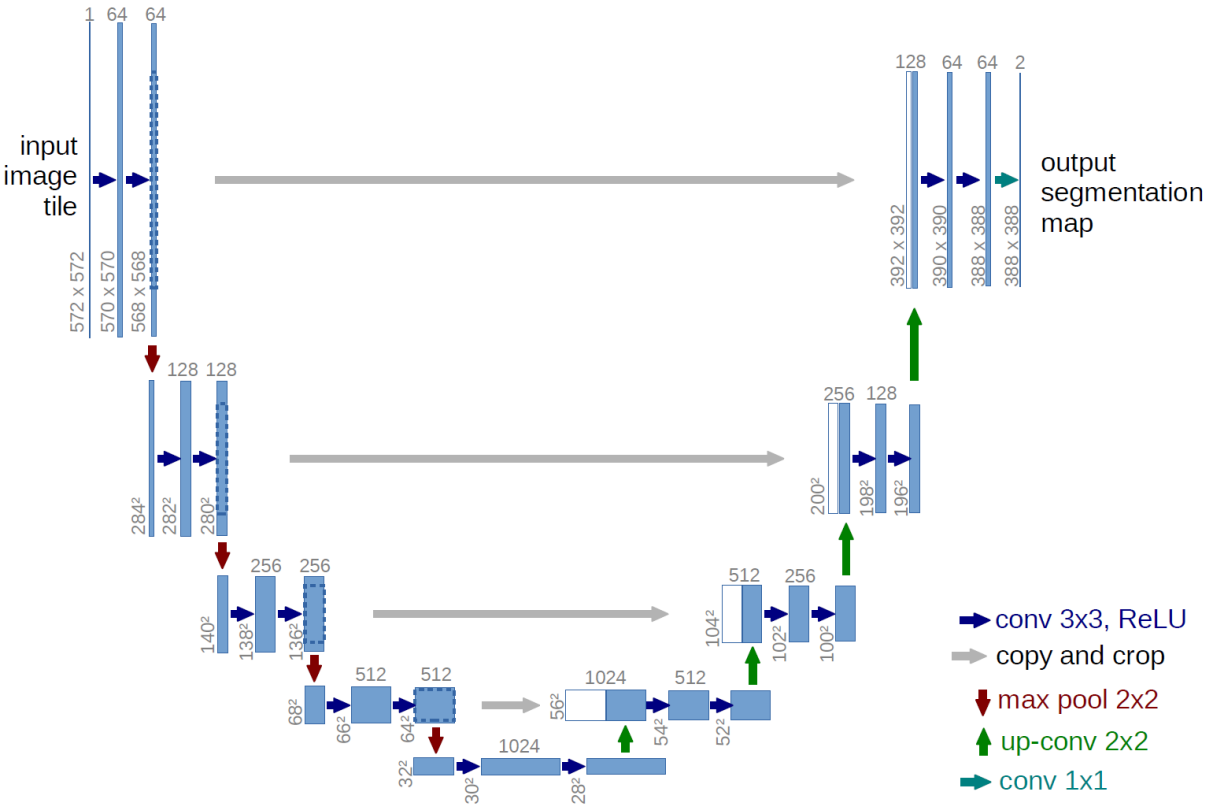
Figure 2. Residual learning: a building block.

典型网络架构 – 比较

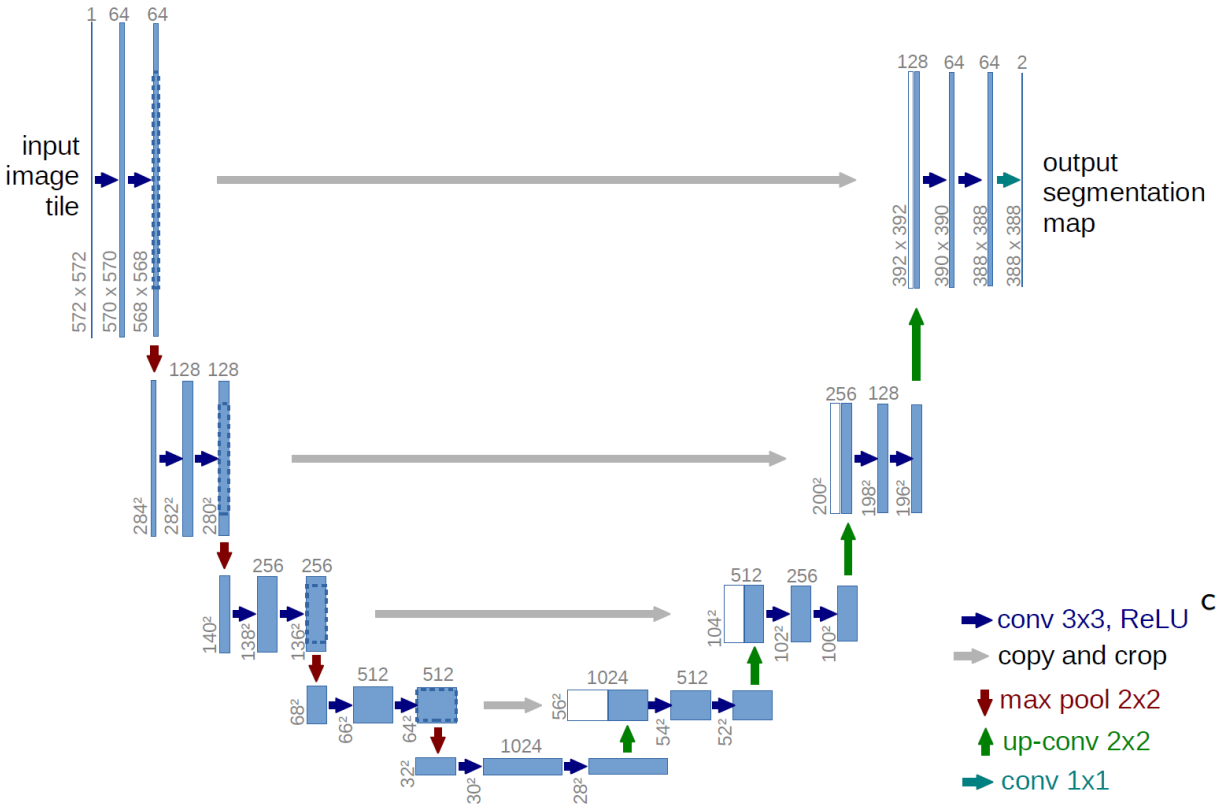


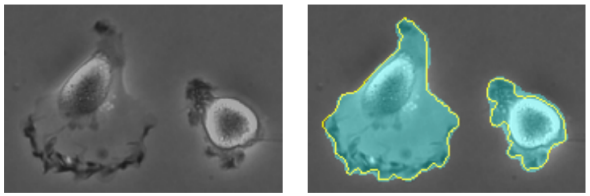
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

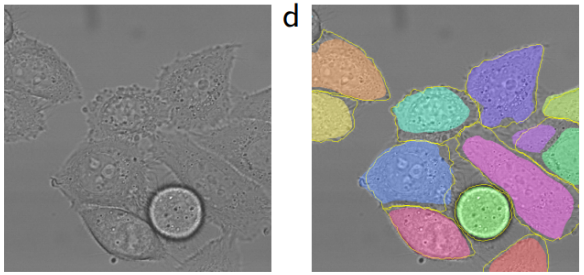
典型网络架构 – UNet


input image tile
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

典型网络架构 – UNet

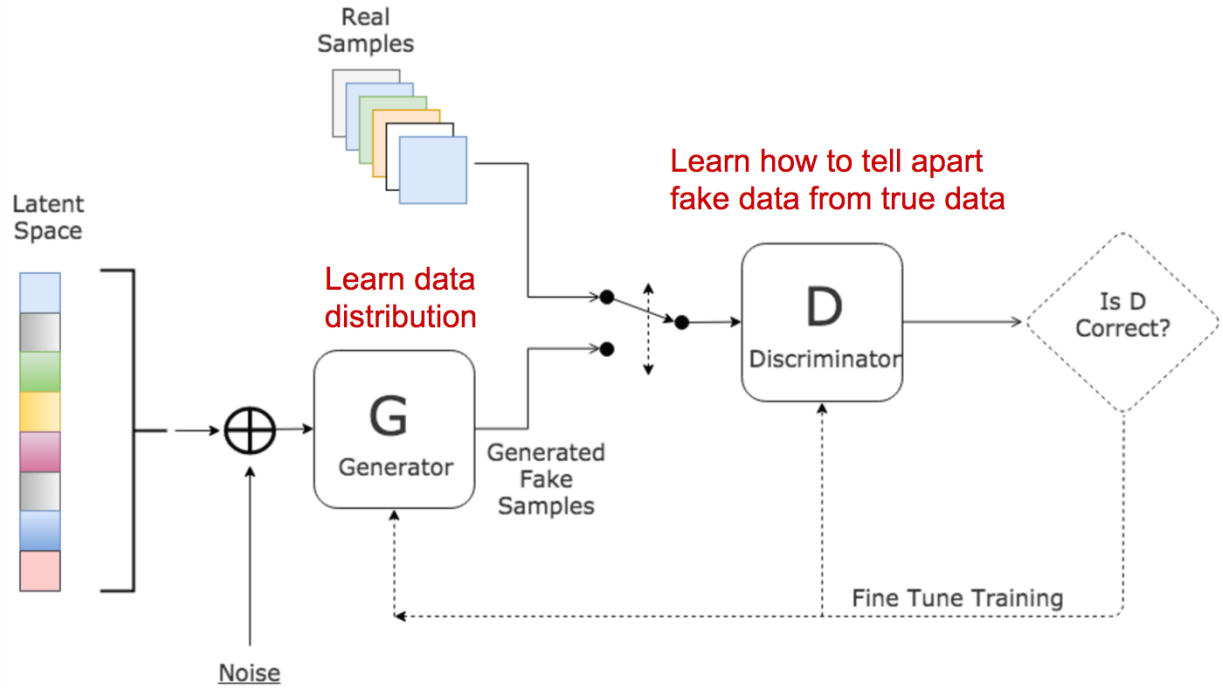

input image tile
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1




c
d

典型网络架构 – 生成对抗网络

- 博弈论思想：生成器和判决器进行博弈



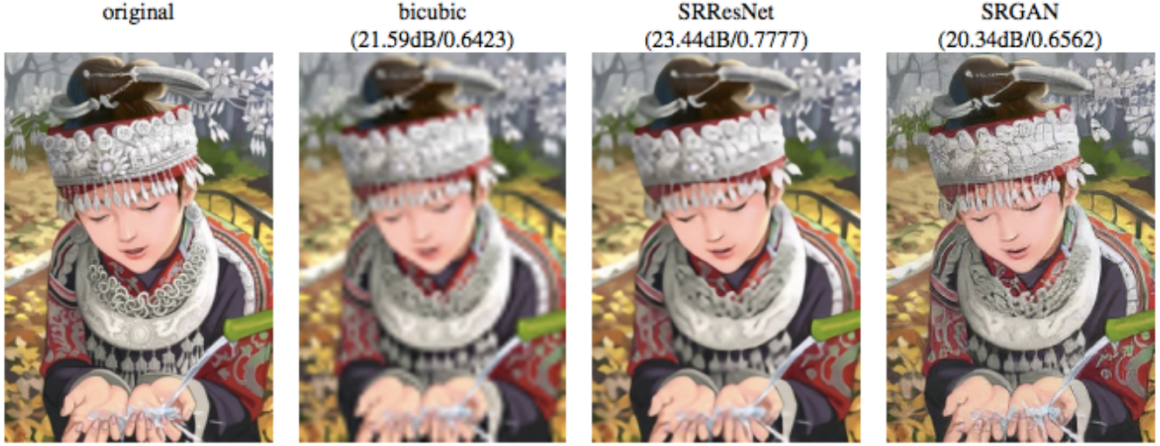
典型网络架构 – 生成对抗网络

典型网络架构 – 生成对抗网络

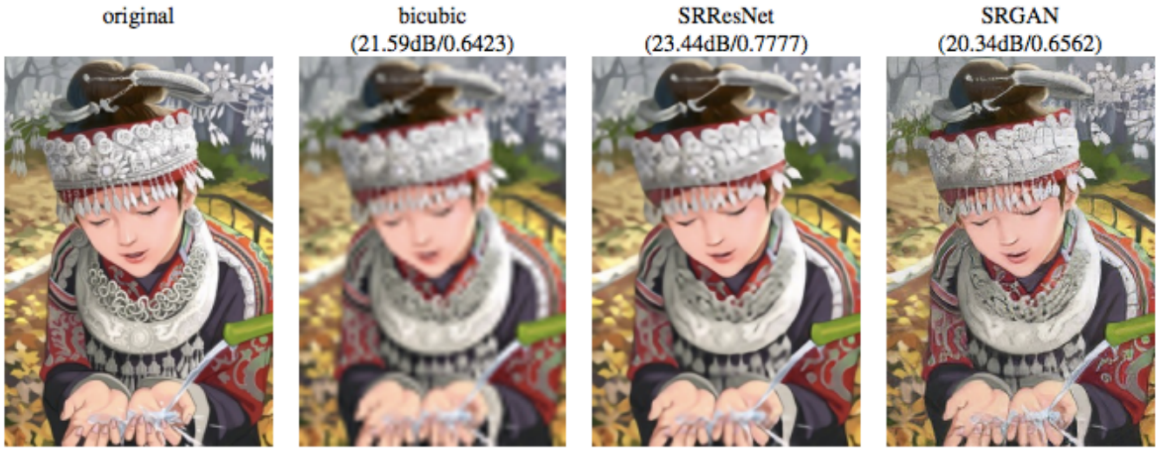
典型网络架构 – 生成对抗网络

缺陷样本！

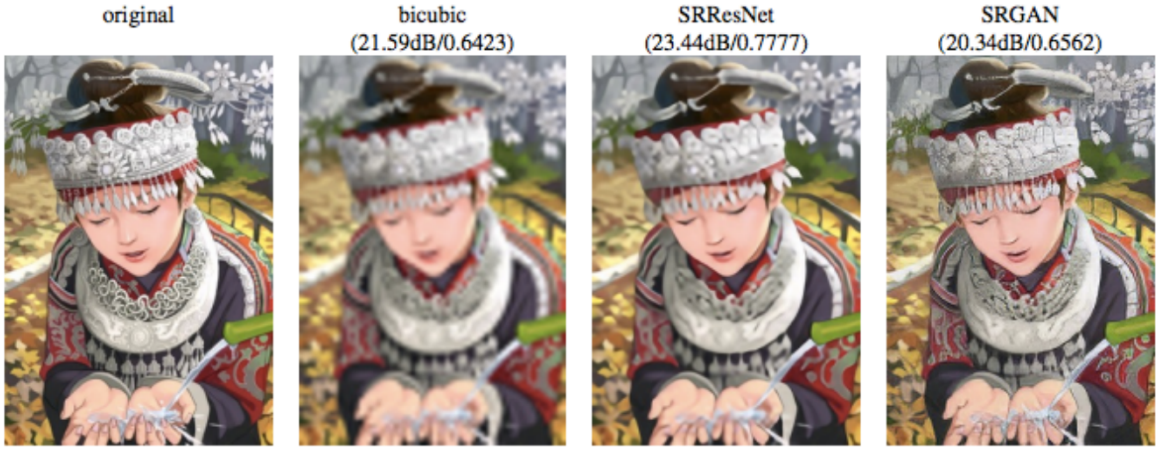
典型网络架构 – 生成对抗网络



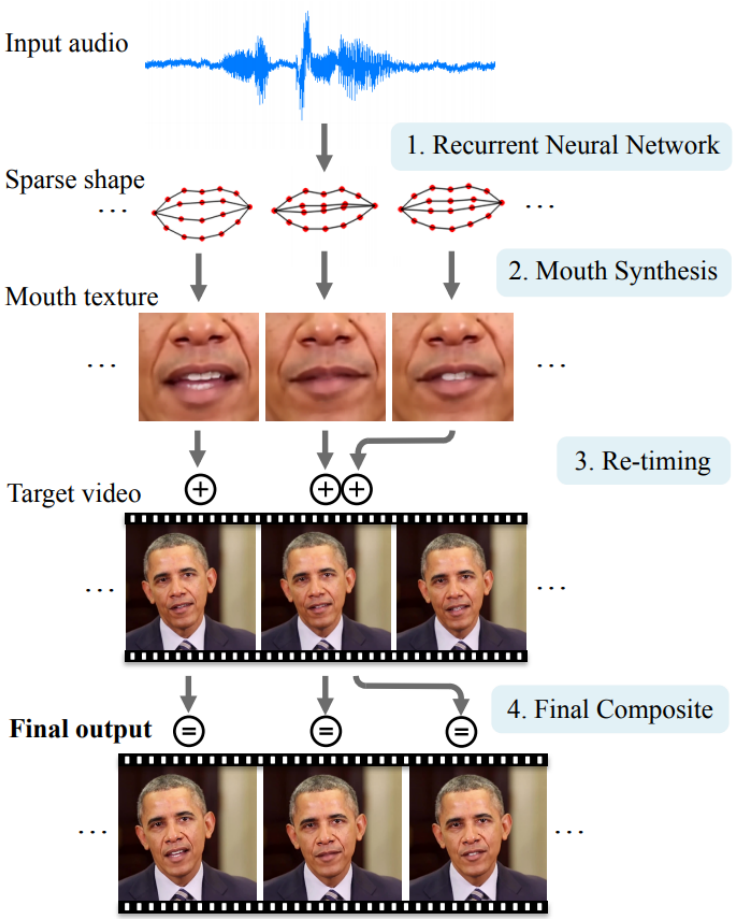
典型网络架构 – 生成对抗网络



典型网络架构 – 生成对抗网络

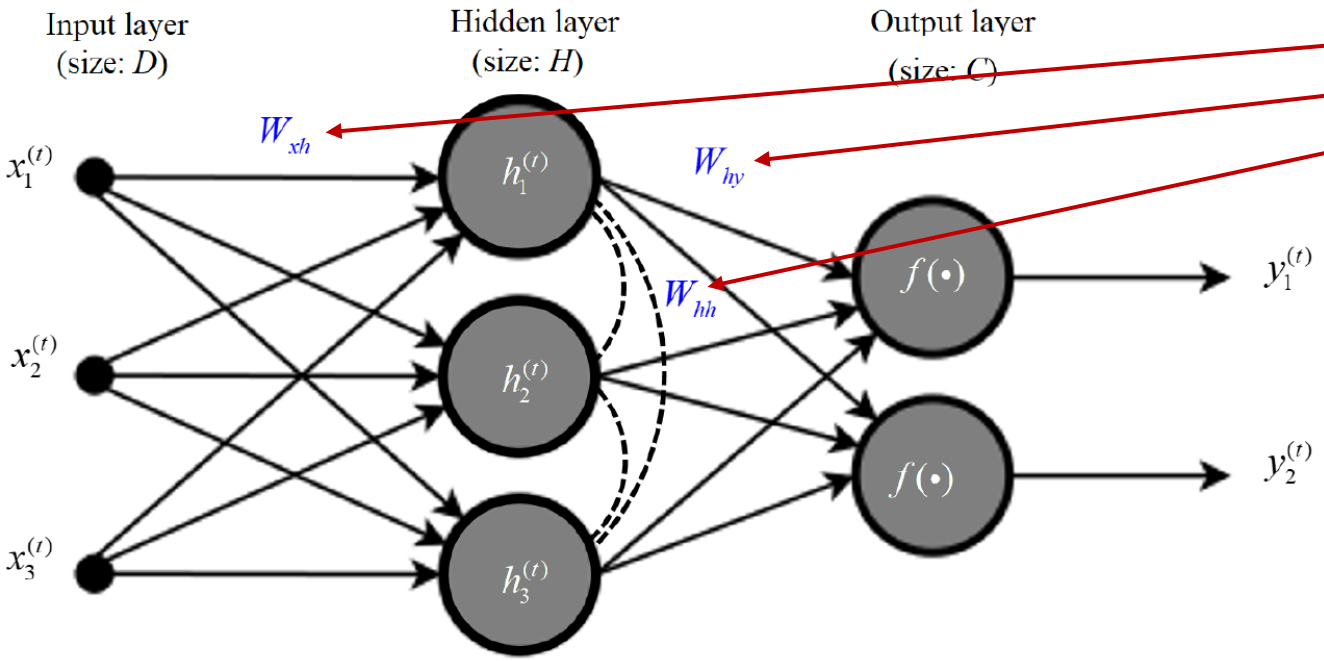


应用例子


Input audio
1. Recurrent Neural Network
Sparse shape
2. Mouth Synthesis
Mouth texture
3. Re-timing
Target video
4. Final Composite
Final output

循环网络

- 循环网络(Recurrent Neural Network, RNN)主要用于处理序列数据，例如语音识别、文本预测、自动翻译等



RNN structure (D: input dim, H:# of hidden neurons, C: # of outputs)

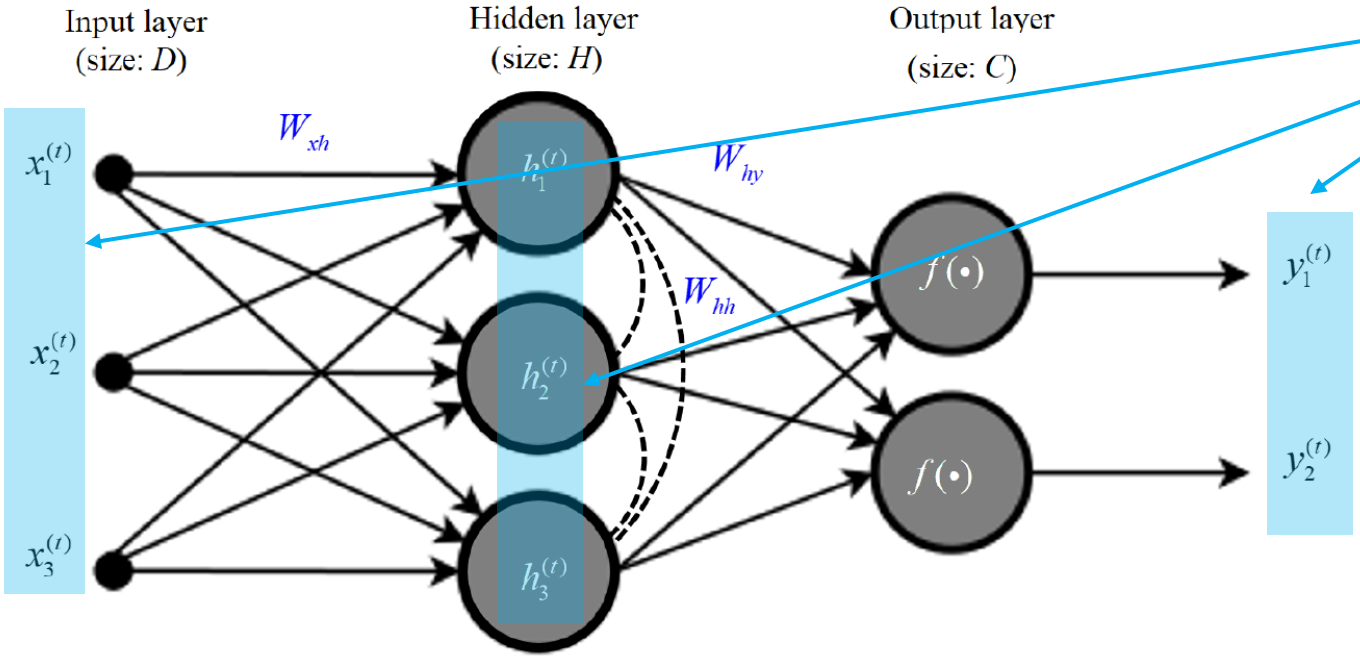
四个重要的静态网络参数：

- 输入层-隐含层
- 隐含层-输出层
- 隐含层-隐含层
- 隐含层偏移

循环网络

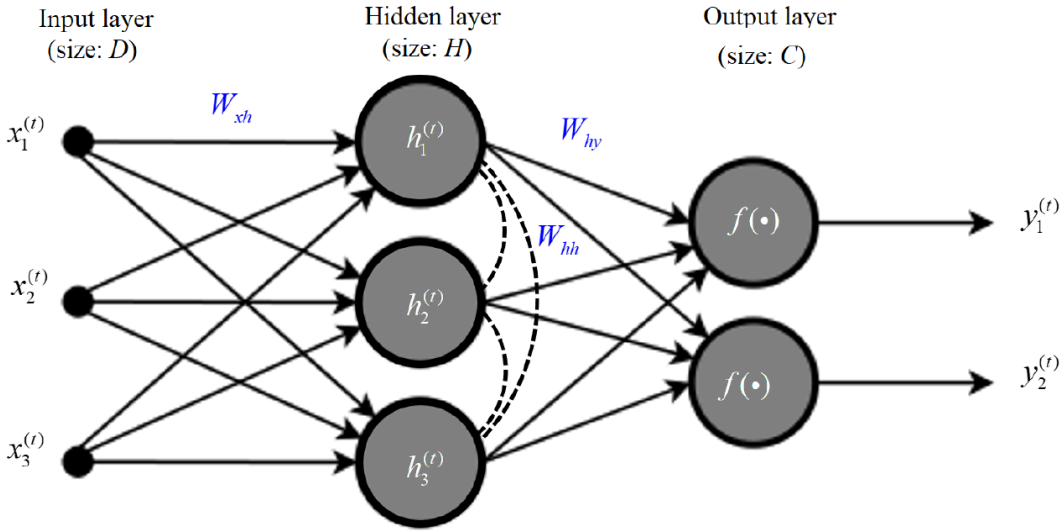
- 循环网络(Recurrent Neural Network, RNN)主要用于处理序列数据，例如语音识别、文本预测、自动翻译等

三个重要的动态序列数据
- 输入数据
- 隐含变量
- 输出数据



RNN structure (D: input dim, H:# of hidden neurons, C: # of outputs)

循环网络



RNN structure (D: input dim, H:# of hidden neurons, C: # of outputs)

输入输出关系

$$\begin{cases} \mathbf{y}^{(t)} = \mathbf{W}_{hy}\mathbf{h}^{(t)} \\ \mathbf{h}^{(t)} = f\left(\mathbf{W}_{xh}\mathbf{x}^{(t)} + \mathbf{W}_{hh}\mathbf{h}^{(t-1)} + \mathbf{b}\right) \end{cases}$$

动态序列数据有上标t

循环网络 – 语句预测

给定一个输入序列 $(\mathbf{x}^{(1)},\mathbf{z}^{(1)}),(\mathbf{x}^{(2)},\mathbf{z}^{(2)}),\ldots,(\mathbf{x}^{(T)},\mathbf{z}^{(T)})$

例如输入预测：“今天 | 下午 | 我 | 要 | 去 | 学校 | 。”，$T=7$

$\mathbf{x}^{(1)}=$ 今天，$\mathbf{x}^{(2)}=$ 下午，$\mathbf{x}^{(3)}=$ 我，$\mathbf{x}^{(4)}=$ 要，$\mathbf{x}^{(5)}=$ 去，$\mathbf{x}^{(6)}=$ 学校

$\mathbf{z}^{(1)}=$ 下午，$\mathbf{z}^{(2)}=$ 我，$\mathbf{z}^{(3)}=$ 要，$\mathbf{z}^{(4)}=$ 去，$\mathbf{z}^{(5)}=$ 学校，$\mathbf{z}^{(6)}=$ 。

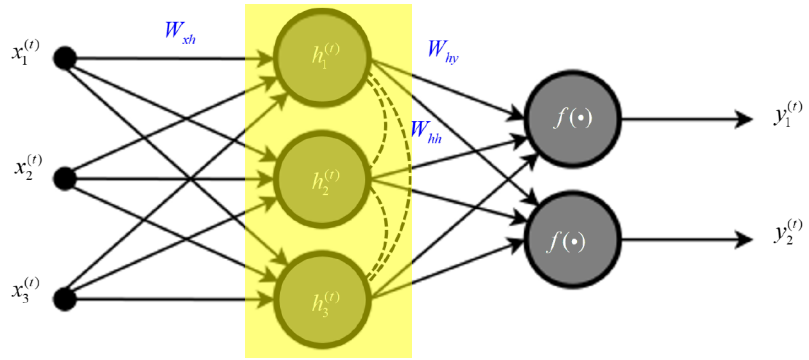
定义损失函数 $J=\sum_{t=1}^{T}\left\|\mathbf{z}^{(t)}-\mathbf{y}^{(t)}\right\|$

采用BP算法进行训练

$$\mathbf{W}(k+1)=\mathbf{W}(k)-\lambda\Delta\mathbf{W}$$

$$\Delta\mathbf{W}=\frac{\partial J}{\partial\mathbf{W}}=2\sum_{t=1}^{T}\left(\mathbf{z}^{(t)}-\mathbf{y}^{(t)}\right)\frac{\partial\mathbf{y}^{(t)}}{\partial\mathbf{W}}$$

循环网络 – 等效展开



$$\begin{cases} \mathbf{y}^{(t)} = \mathbf{W}_{hy}\mathbf{h}^{(t)} \\ \mathbf{h}^{(t)} = f\left(\mathbf{W}_{xh}\mathbf{x}^{(t)} + \mathbf{W}_{hh}\mathbf{h}^{(t-1)} + \mathbf{b}\right) \end{cases}$$