深度学习之旅

模型、算法与实践

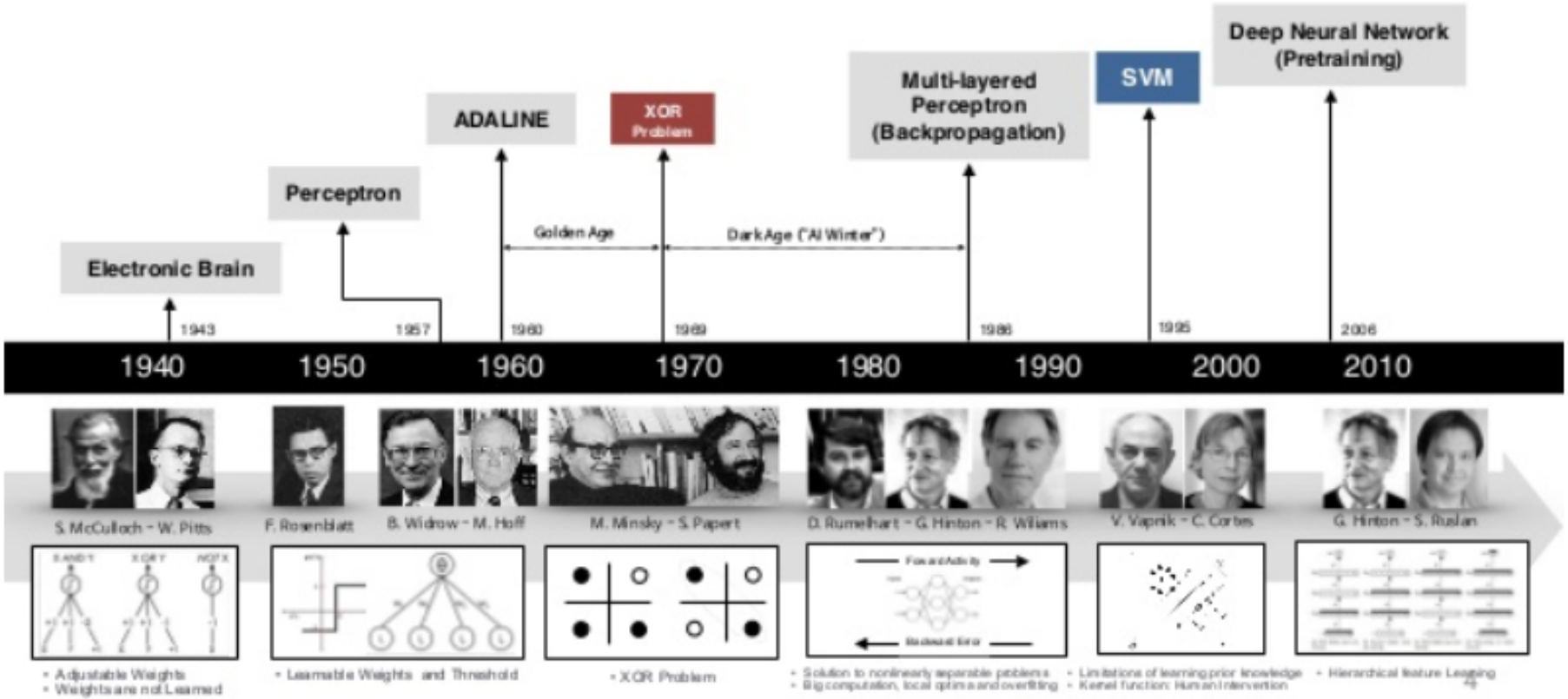
张钟元

2018-01-19

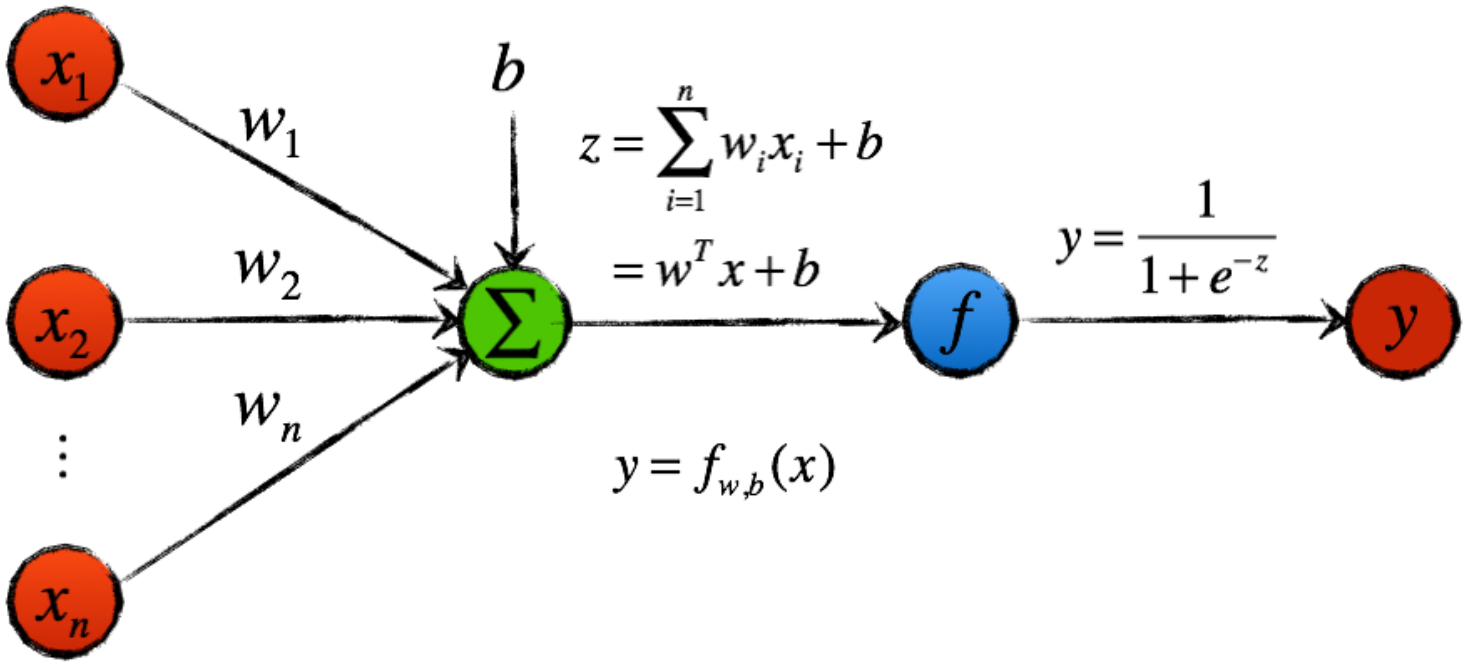
内容

破冰之旅

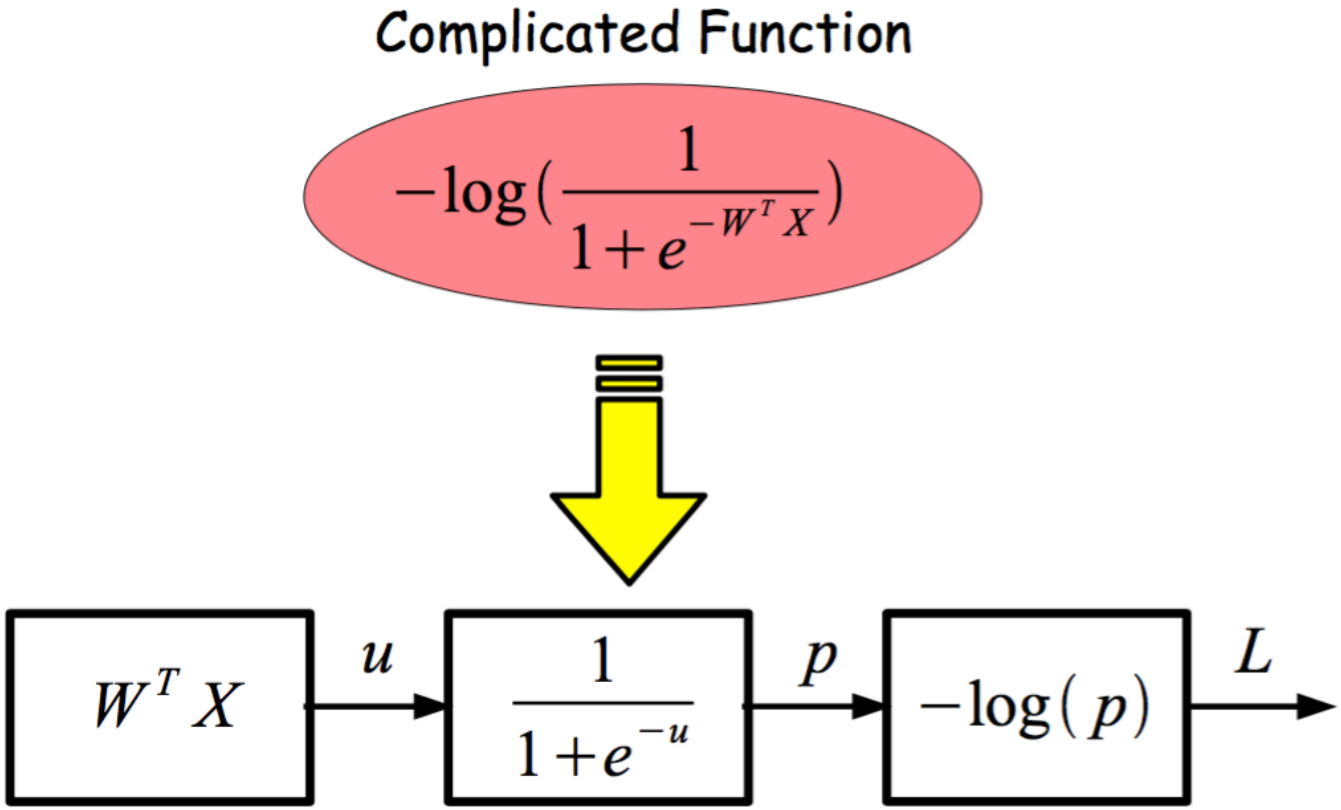
历史


Deep Neural Network
(Pretraining)
SVM
Multi-layered
Perceptron
(Backpropagation)
XOR
Problem
ADALINE
Perceptron
Electronic Brain
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch - W. Pitts
F. Rosenblatt
B. Widrow - M. Hoff
M. Minsky - S. Papert
D. Rumelhart - G. Hinton - R. Wiliams
V. Vapnik - C. Cortes
G. Hinton - S. Ruslan
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human intervention
• Hierarchical feature Learning

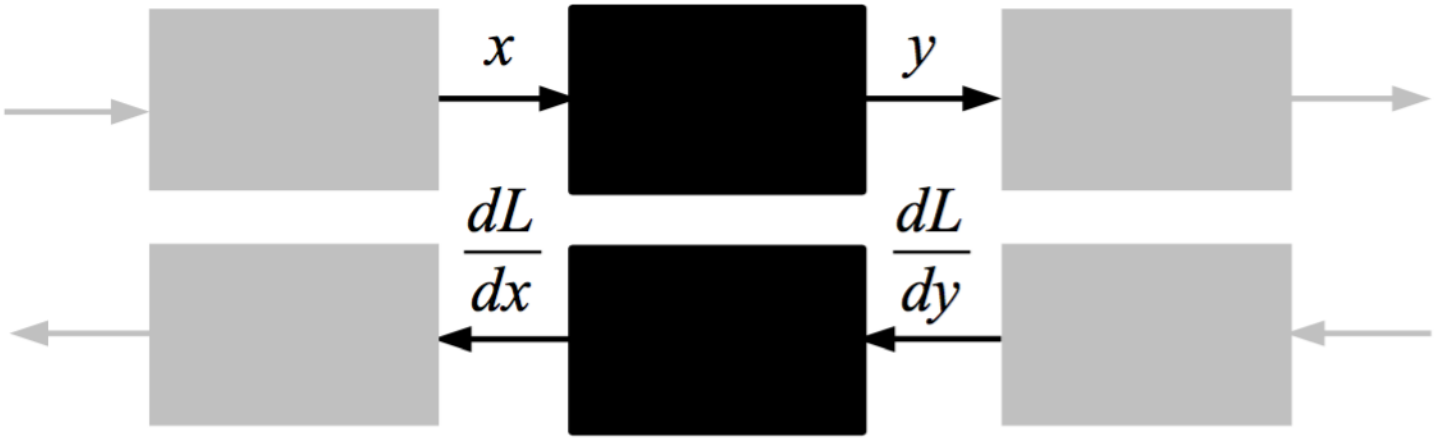
逻辑回归


x_1
x_2
x_n
w_1
w_2
w_n
b
Σ
$z = \sum_{i=1}^{n} w_i x_i + b$
$= w^T x + b$
f
$y = \frac{1}{1+e^{-z}}$
y
$y = f_{w,b}(x)$

计算损失：函数组合


Complicated Function
$-\log(\frac{1}{1+e^{-W^T X}})$
$W^T X$
u
$\frac{1}{1+e^{-u}}$
p
$-\log(p)$
L

链式法则



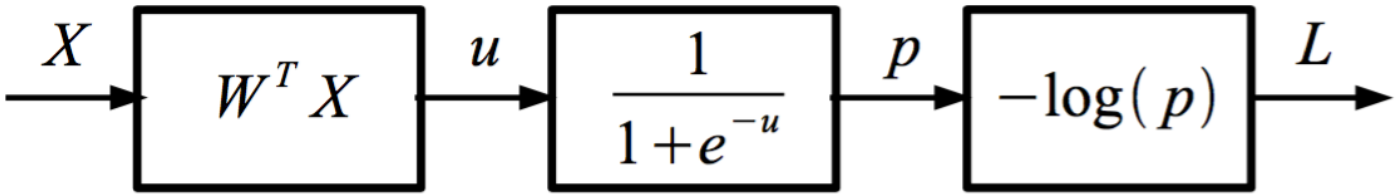
Given $y(x)$ and dL/dy,

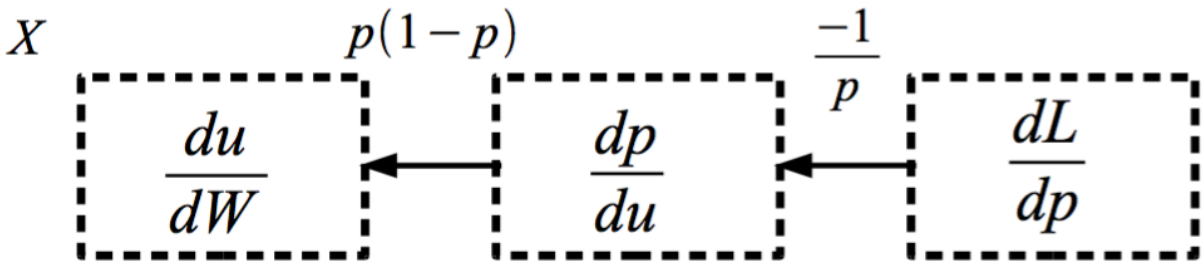
What is dL/dx?

$$\frac{dL}{dx} = \frac{dL}{dy} \cdot \frac{dy}{dx}$$

All needed information is local!

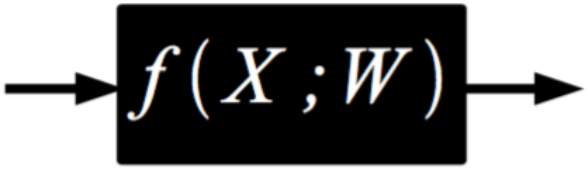
计算梯度



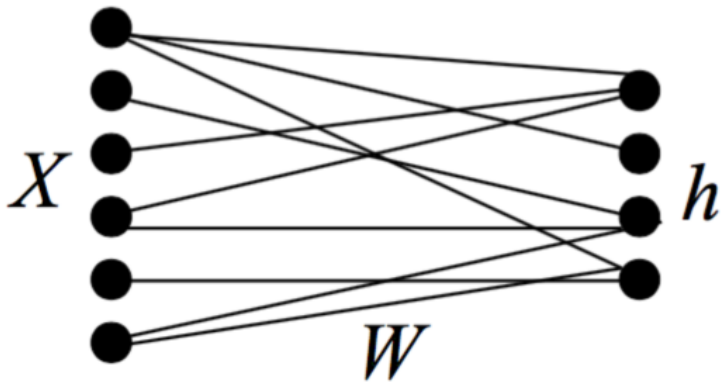


$$\frac{dL}{dW} = \frac{dL}{dp} \cdot \frac{dp}{du} \cdot \frac{du}{dW} = (p-1)X$$

单层网络



is equivalent to

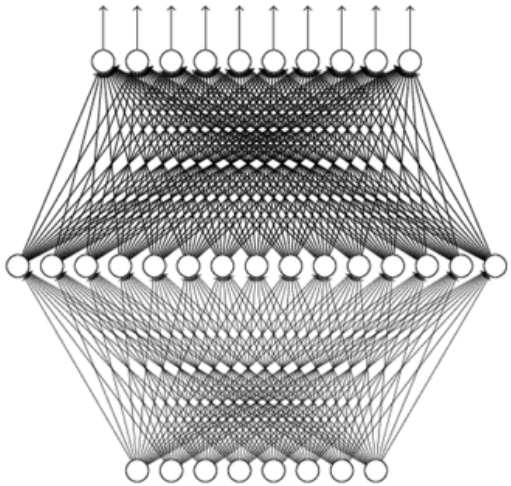


万能近似定理

Any continuous function

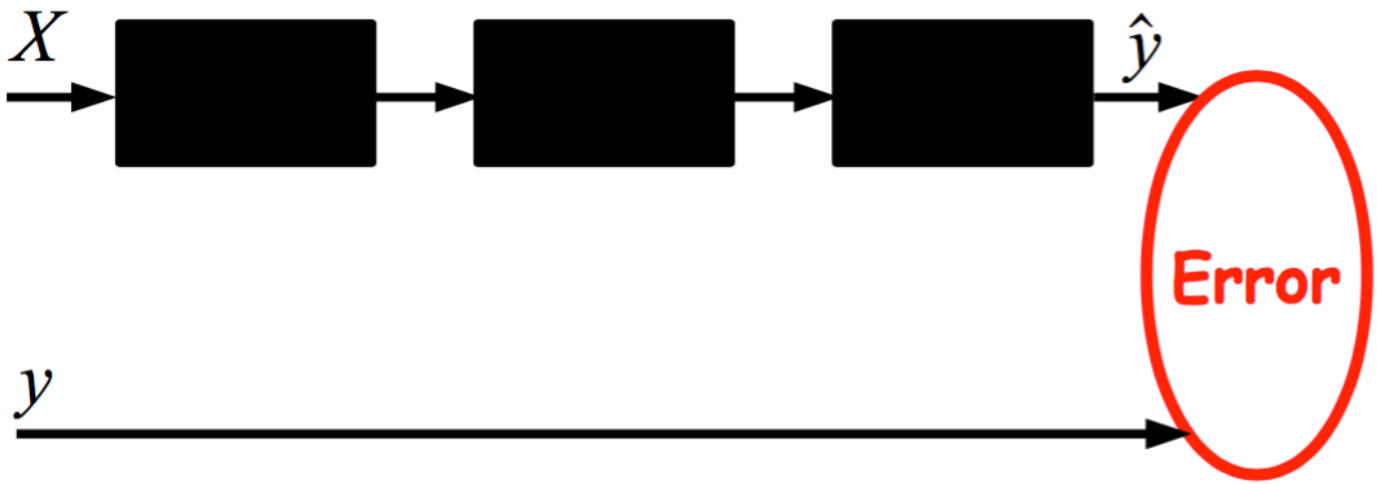
$$f : R^N \rightarrow R^M$$

Can be realized by a network with one hidden layer, given enough hidden neurons)



多层网络：深度学习

回归

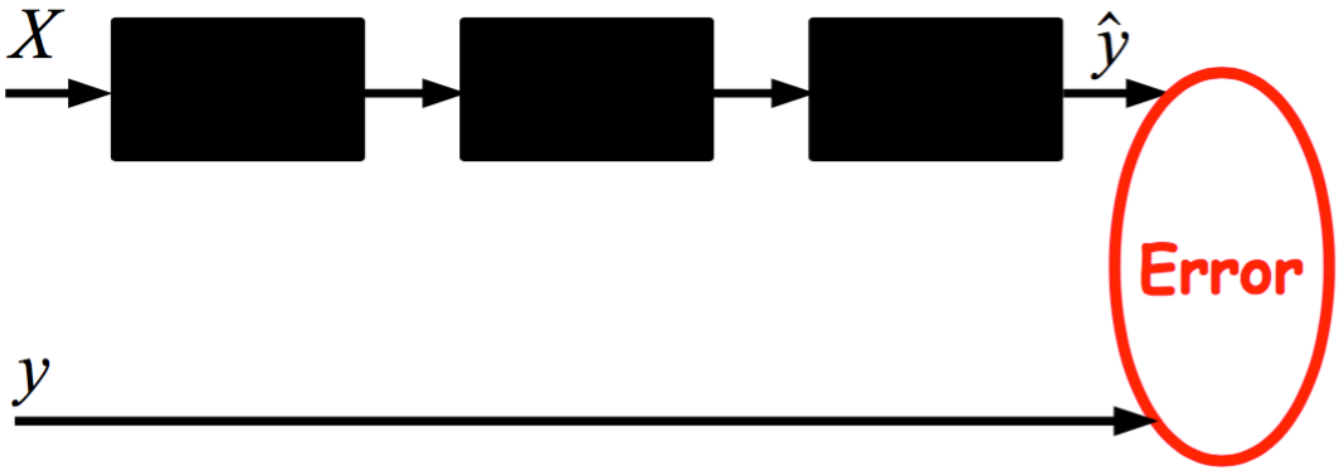


Square Euclidean Distance (regression):

$y, \hat{y} \in R^N$

$$L = \frac{1}{2} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

分类



Cross Entropy (classification):

$$y, \hat{y} \in [0,1]^N, \ \sum_{i=1}^{N} y_i = 1, \ \sum_{i=1}^{N} \hat{y}_i = 1$$

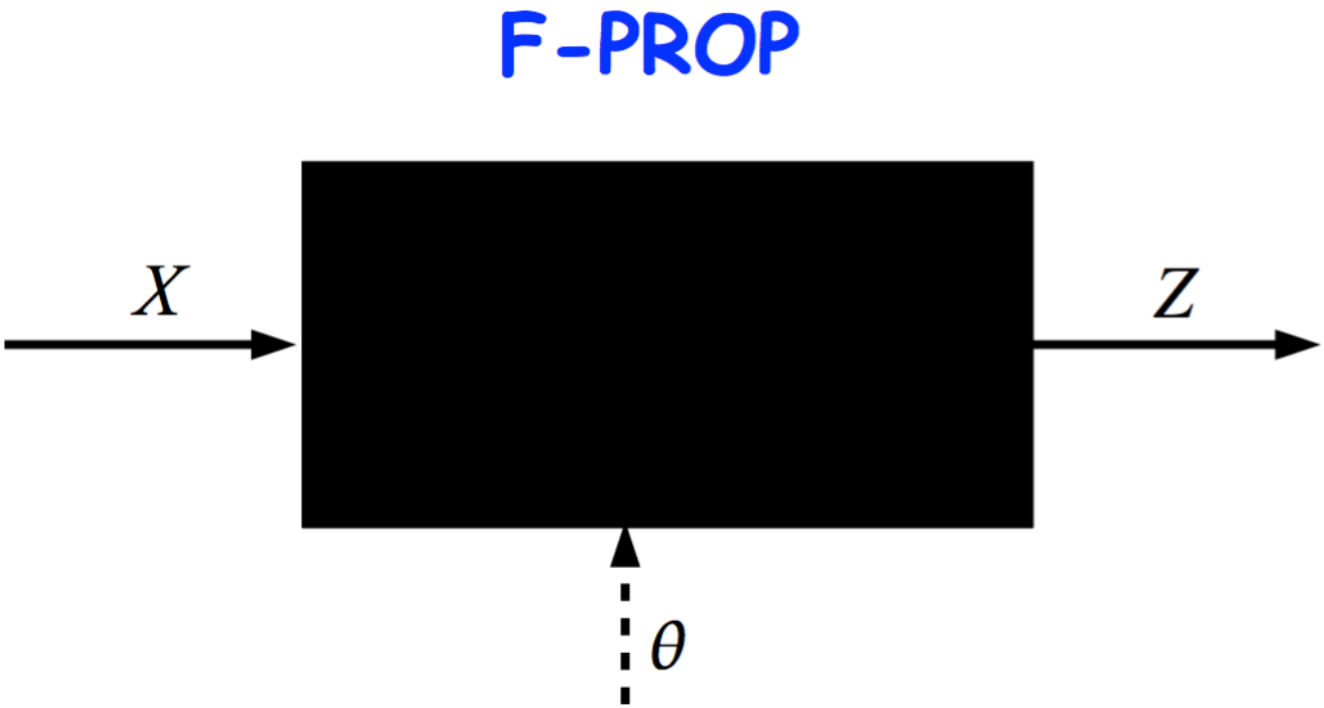
$$L = -\sum_{i=1}^{N} y_i \log \hat{y}_i$$

51

前向计算

A) Compute loss on small mini-batch

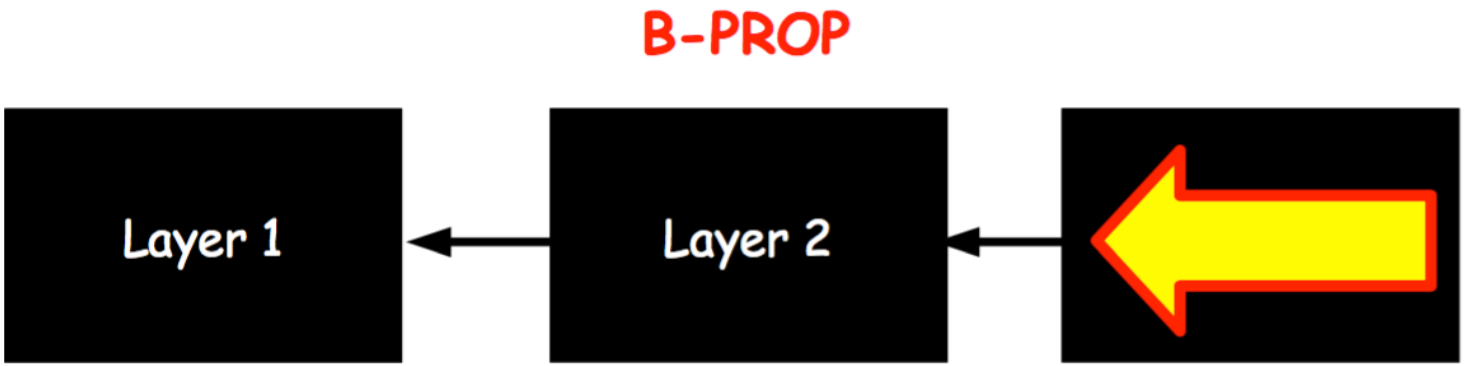
前向计算



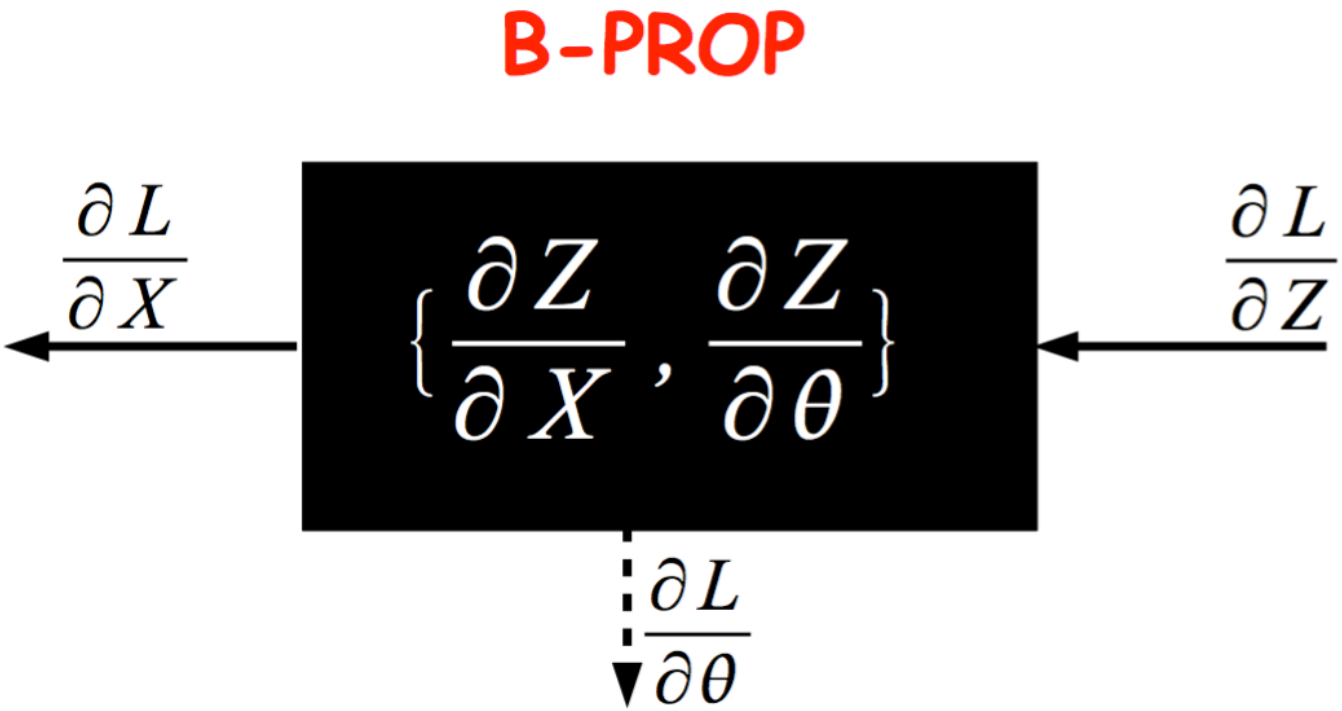
反向传播

A) Compute loss on small mini-batch

B) Compute gradient w.r.t. parameters



反向传播

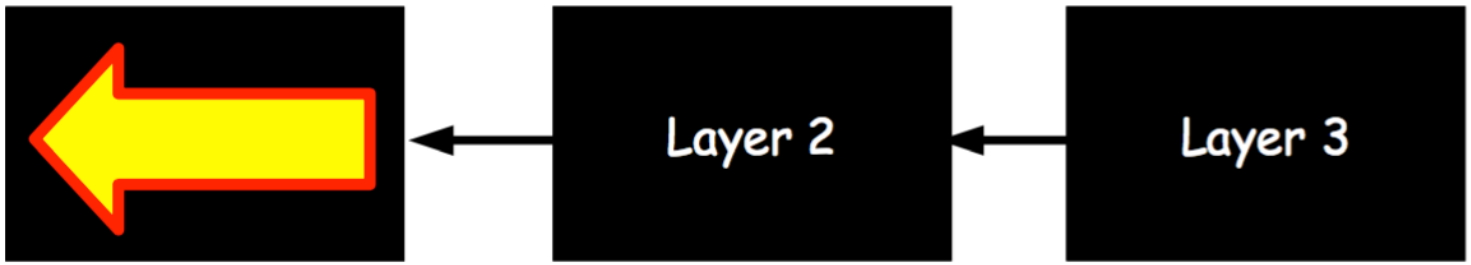

B-PROP
$\frac{\partial L}{\partial X}$
$\{\frac{\partial Z}{\partial X}, \frac{\partial Z}{\partial \theta}\}$
$\frac{\partial L}{\partial Z}$
$\frac{\partial L}{\partial \theta}$

更新参数

A) Compute loss on small mini-batch

B) Compute gradient w.r.t. parameters

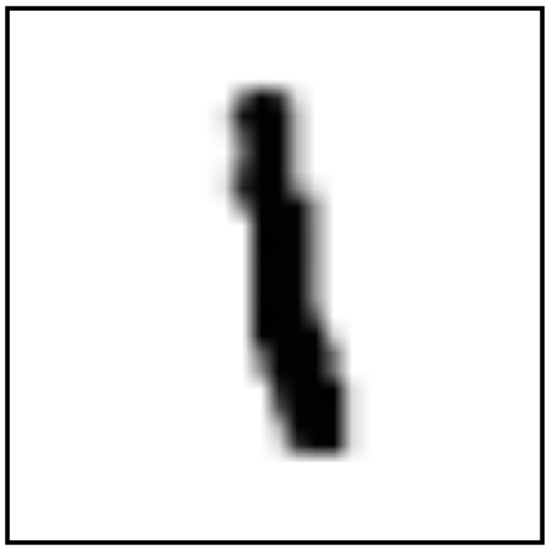
C) Use gradient to update parameters $\theta \leftarrow \theta - \eta \frac{dL}{d\theta}$



前馈神经网络

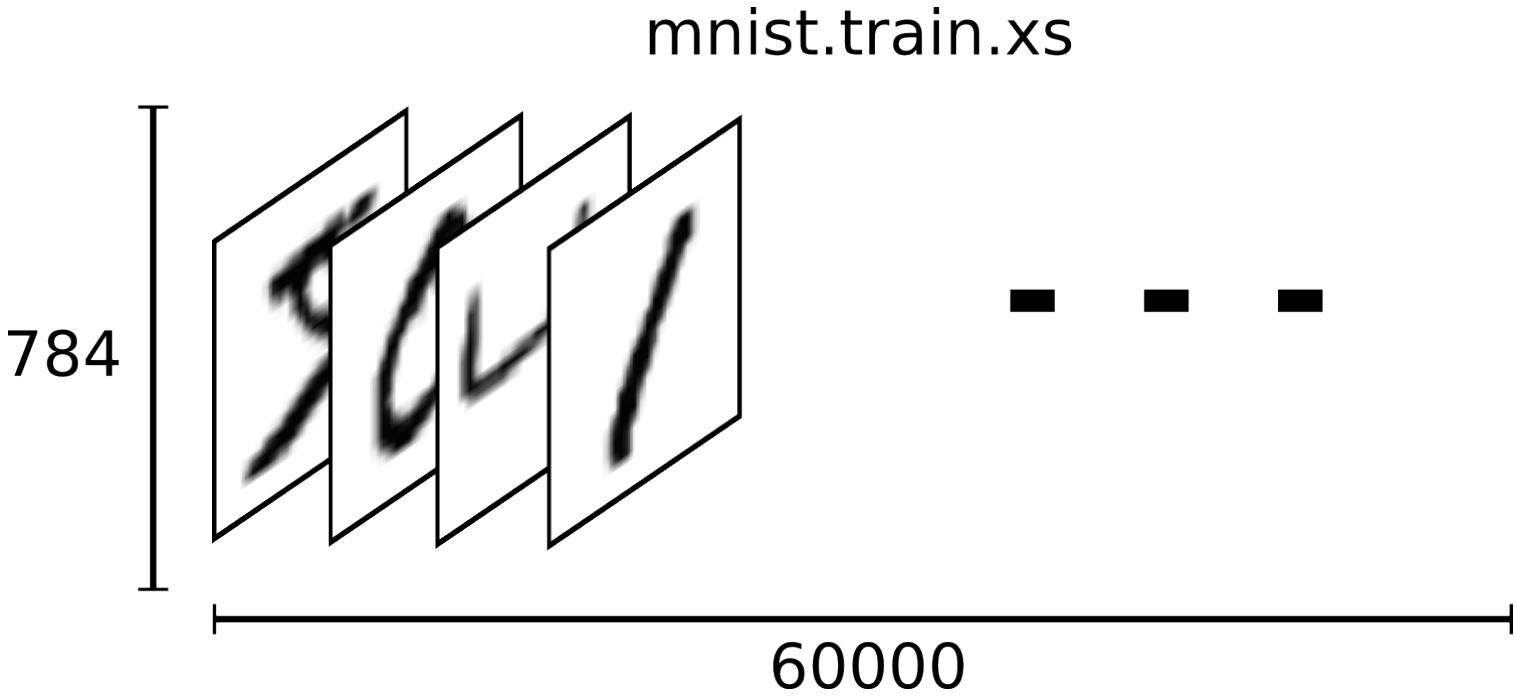
1. 单层前馈网络
2. 多层前馈网络

图片：28x28 = 784

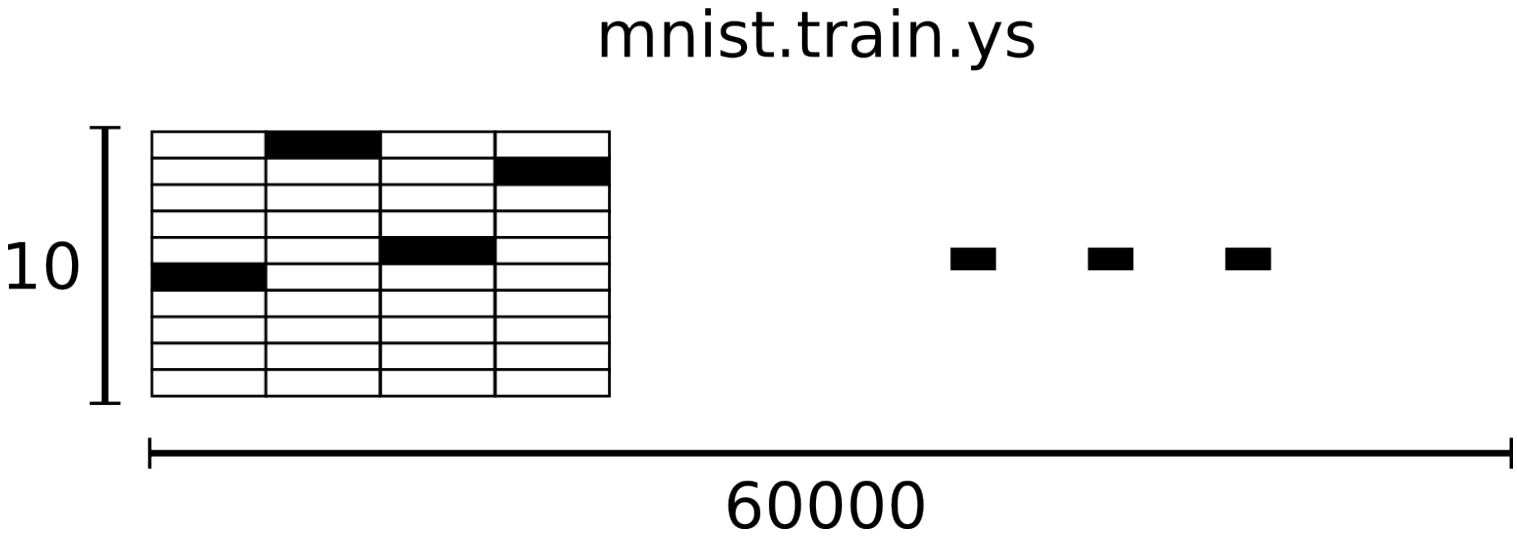


$$\approx \begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .6 & .8 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & .5 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .4 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & .7 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .9 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & .3 & 1 & .1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}$$

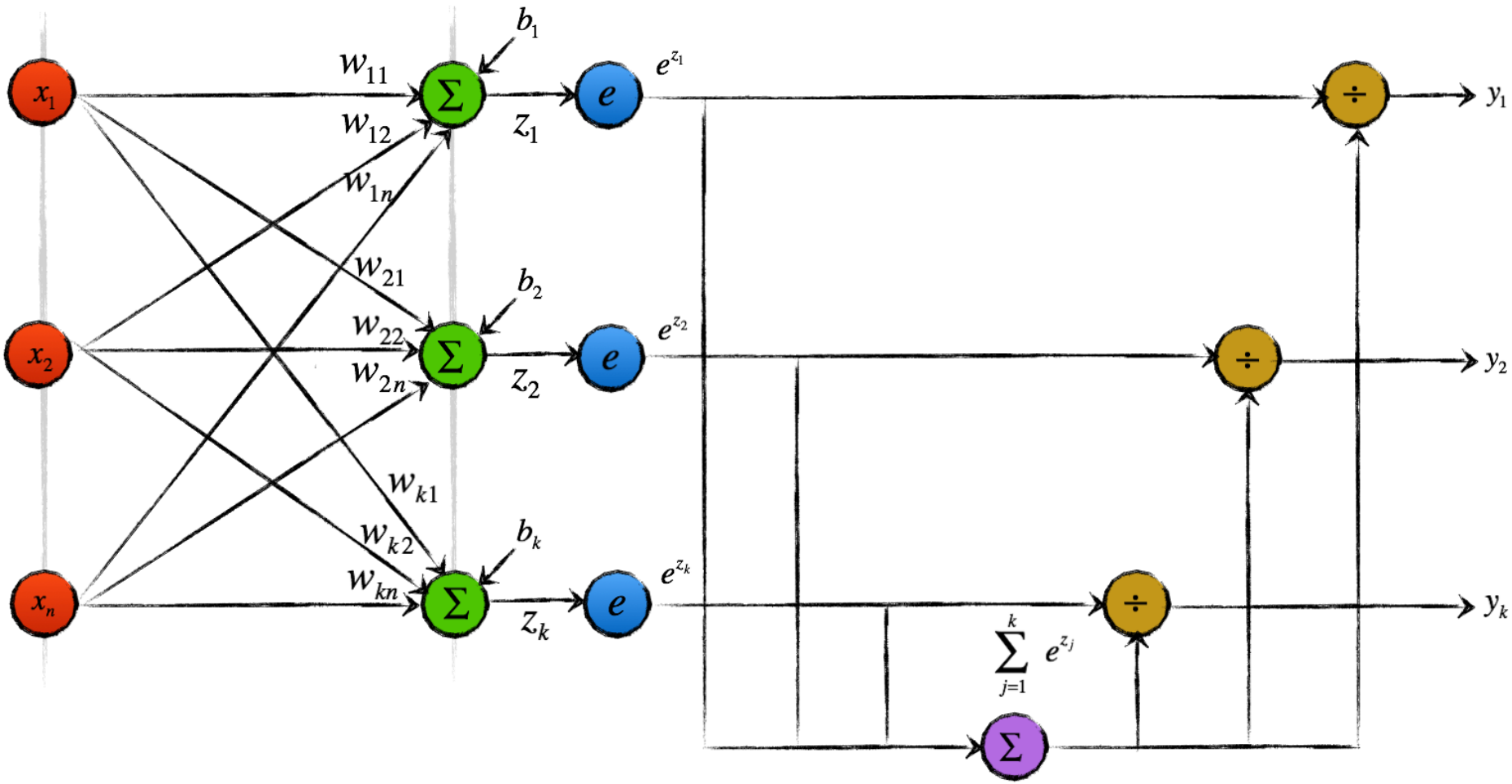
训练数据集输入：[100(*batch_size*), 784]



训练数据集输出：$[100(batch\_size), 10]$



Softmax 模型


x_1
x_2
x_n
w_{11}
w_{12}
w_{1n}
w_{21}
w_{22}
w_{2n}
w_{k1}
w_{k2}
w_{kn}
b_1
b_2
b_k
z_1
z_2
z_k
e^{z_1}
e^{z_2}
e^{z_k}
$\sum_{j=1}^{k} e^{z_j}$
y_1
y_2
y_k

k 分类问题

参数定义

$$W = \begin{pmatrix} w_{11} & \cdots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{k1} & \cdots & w_{kn} \end{pmatrix} \in \mathbb{R}^{k \times n}$$

$$b = (b_1, b_2, ..., b_k)^T \in \mathbb{R}^k$$

$$x = (x_1, x_2, ..., x_n)^T \in \mathbb{R}^n$$

模型定义

$$z = Wx + b \in \mathbb{R}^k$$

$$y = softmax(z) \in \mathbb{R}^k$$

$$softmax(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}, \ i = 1, 2, ..., k$$

损失函数

交叉熵损失函数

$$L(W,b;S) = -\frac{1}{m}\sum_{i=1}^{m} t^{(i)} \odot \log\left(y^{(i)}\right)$$

$$= -\frac{1}{m}\sum_{i=1}^{m}\sum_{j=1}^{k} t_j^{(i)} \log\left(y_j^{(i)}\right)$$

优化问题

$$W^*, b^* = \arg\min_{W,b} L(W,b;S)$$

优化算法

梯度下降

$$W \leftarrow W - \alpha \nabla_W L(W, b)$$
$$b \leftarrow b - \alpha \nabla_b L(W, b)$$

梯度计算

$$\nabla_W L(W, b; x, t) = (y - t) \otimes x$$
$$\nabla_b L(W, b; x, t) = (y - t)$$

SGD & BGD → SGD with MiniBatch

SGD

$$w \leftarrow w - \alpha \left(y^{(i)} - t^{(i)} \right) x$$

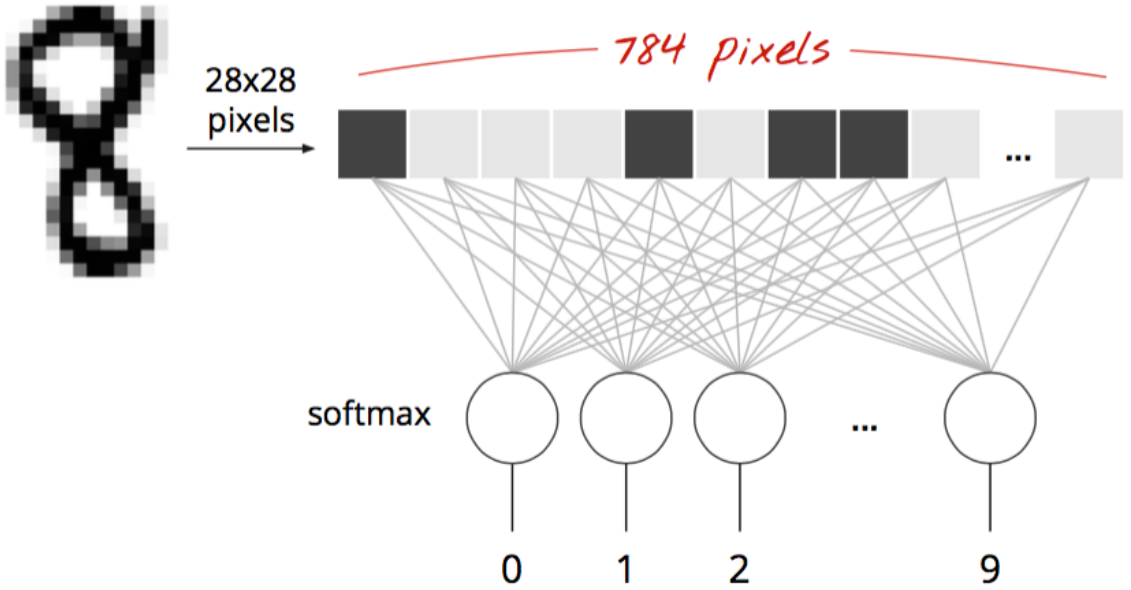
$$b \leftarrow b - \alpha \left(y^{(i)} - t^{(i)} \right)$$

BGD

$$w \leftarrow w - \alpha \left(\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} - t^{(i)} \right) \right) x$$

$$b \leftarrow b - \alpha \left(\frac{1}{m} \sum_{i=1}^{m} \left(y^{(i)} - t^{(i)} \right) \right)$$

单层网络模型



weighted sum of all pixels + bias

$$softmax(L_n) = \frac{e^{L_n}}{||e^L||}$$

neuron outputs

占位符

```
# inputs
x  = tf.placeholder("float", [None, 784])

# labels
t = tf.placeholder("float", [None, 10])
```

- tf.placeholder 定义了一个占位 OP
- None 表示未确定的样本数目 (batch_size)
- Session.run 时提供 feed_dict 提供一个批次的样本数据

变量

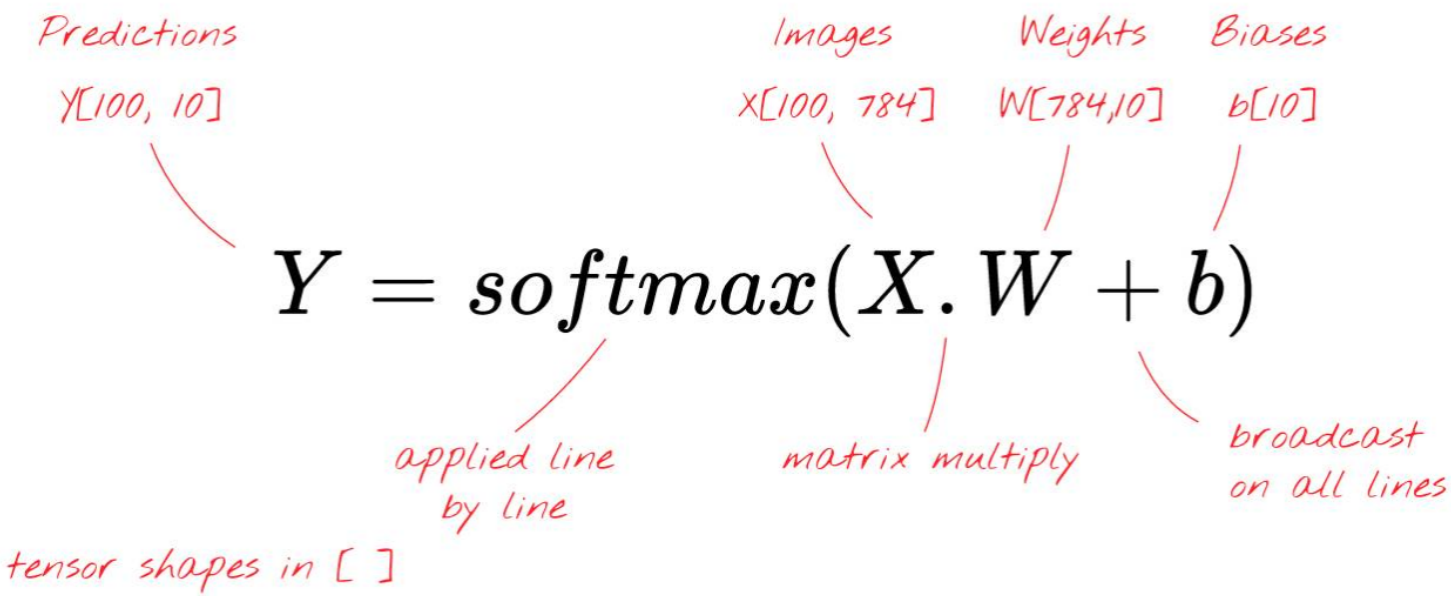
```
# train parameters
W = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))

# init_op
init_op = tf.global_variables_initializer()
```

- 变量：使用 tf.Variable 定义变量，常用于定义模型的权重和偏置
- 初始化：使用初始化 init_op 初始化所有全局变量

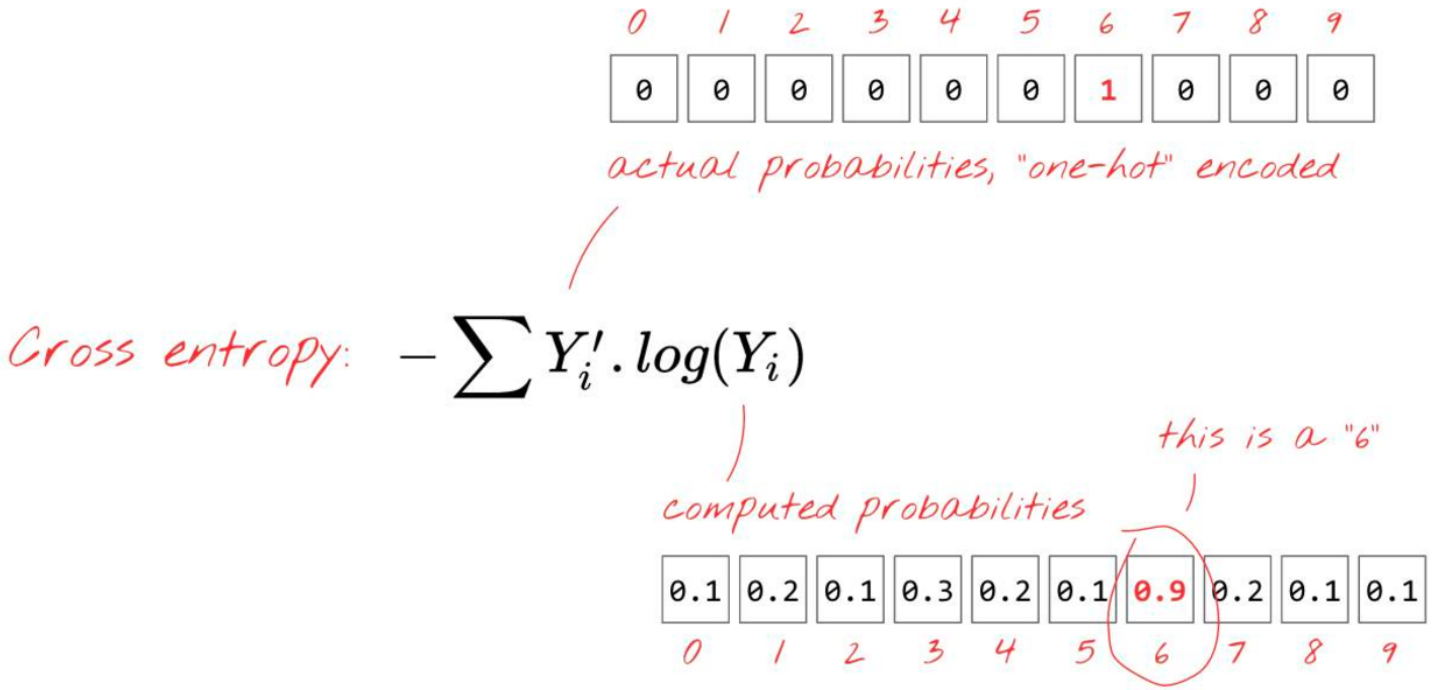
模型: Softmax

```
y = tf.nn.softmax(tf.matmul(x, W) + b)
```

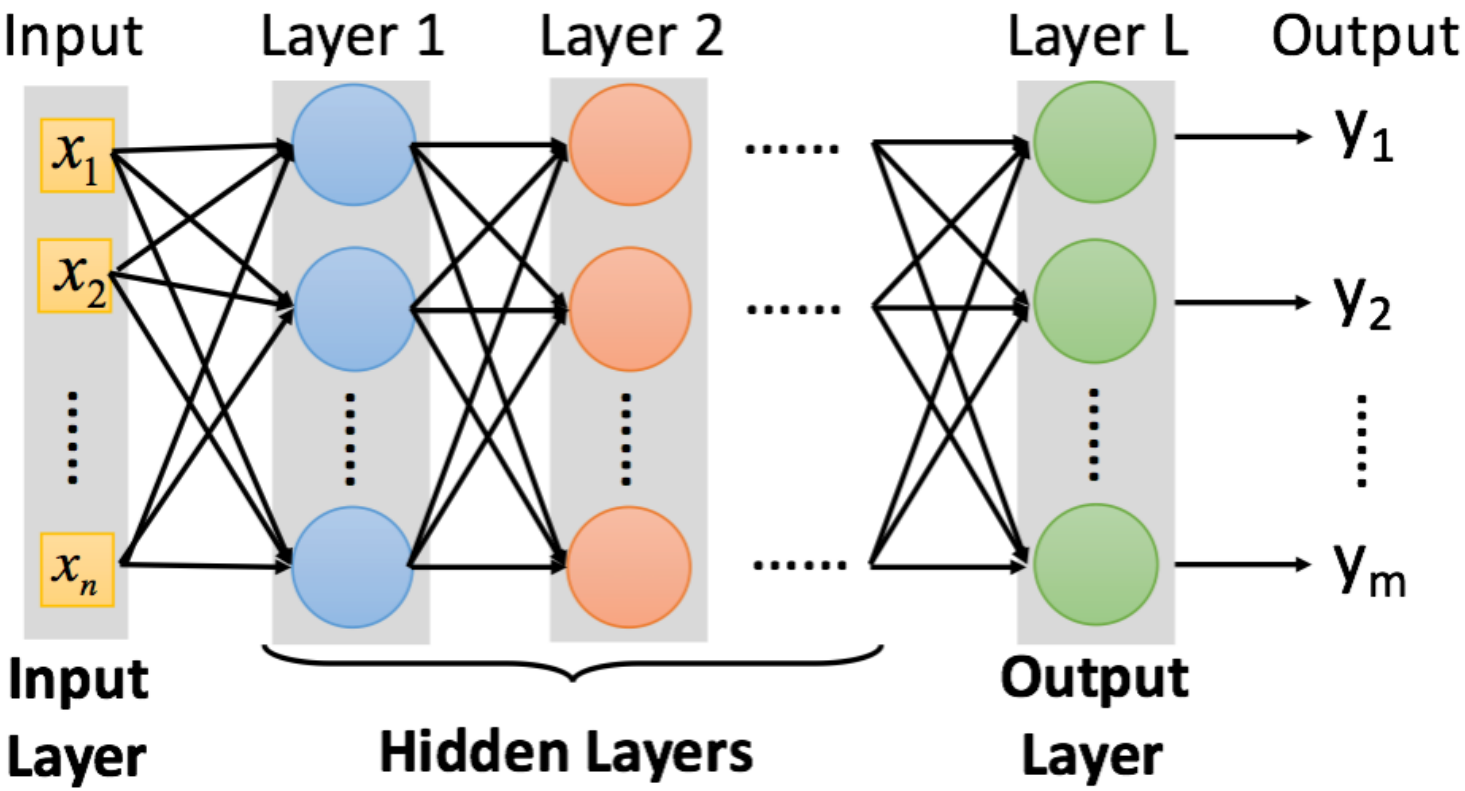


损失函数

```
cross_entropy = -tf.reduce_sum(t * tf.log(y))
```



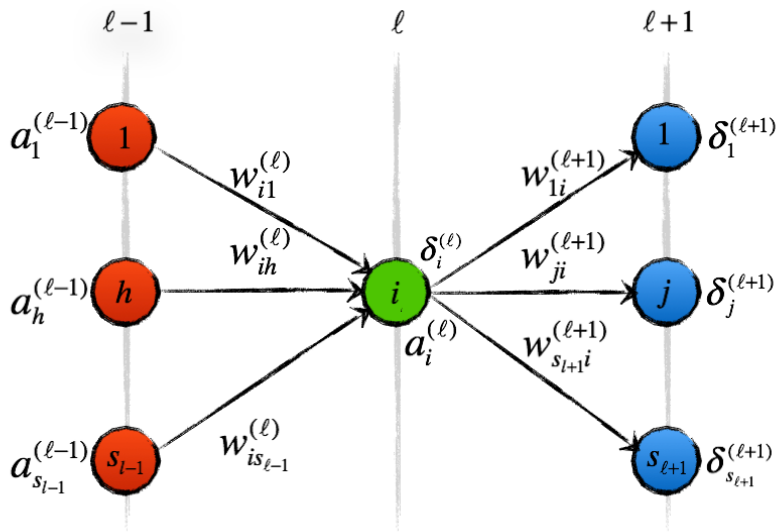
多层前馈网络


Input
Layer 1
Layer 2
Layer L
Output
x_1
x_2
x_n
y_1
y_2
y_m
Input Layer
Hidden Layers
Output Layer

符号

- n_ℓ: 网络层数，其中第 0 层为输入层，第 n_ℓ 层为输出层
- s_ℓ: 第 ℓ 层的节点数，$\ell = 0, 1, ..., n_\ell$
- $w_{ji}^{(\ell)}$: 第 $(\ell - 1)$ 层节点 i 与第 ℓ 层节点 j 之间的权重，$\ell = 1, ..., n_\ell$
- $b_i^{(\ell)}$: 第 ℓ 层节点 i 的偏置项，$\ell = 1, ..., n_\ell$
- $a_i^{(\ell)}$: 第 ℓ 层节点 i 的输出，$\ell = 1, ..., n_\ell, x = a^{(0)}, y = a^{(n_\ell)}$
- $z_i^{(\ell)}$: 第 ℓ 层节点 i 的权重和，$\ell = 1, ..., n_\ell$
- $\delta_i^{(\ell)}$: 第 ℓ 层节点 i 的误差项，$\ell = 1, ..., n_\ell$
- $S = \{(x^{(t)}, y^{(t)}); t = 1, 2, ..., m\}$: 样本空间

计算梯度 (标量)



$$\nabla_{w_{ih}^{(\ell)}} L(w,b) = \frac{\partial L}{\partial W_{ih}^{(\ell)}} = \frac{\partial L}{\partial z_i^{(\ell)}} \frac{\partial z_i^{(\ell)}}{\partial W_{ih}^{(\ell)}} = \delta_i^{(\ell)} a_h^{(\ell-1)}$$

$$z_i^{(\ell)} = \sum_{h=1}^{s_{\ell-1}} \left(W_{ih}^{(\ell)} a_h^{(\ell-1)} + b_i^{(\ell)} \right)$$

$$\frac{\partial z_i^{(\ell)}}{\partial W_{ih}^{(\ell)}} = a_h^{(\ell-1)}$$

$$a_i^{(\ell)} = f\left(z_i^{(\ell)}\right)$$

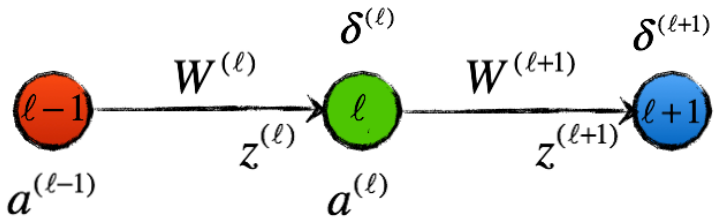
$$z_j^{(\ell+1)} = \sum_{i=1}^{s_\ell} \left(W_{ji}^{(\ell+1)} a_i^{(\ell)} + b_j^{(\ell+1)} \right)$$

$$\frac{\partial L}{\partial z_i^{(\ell)}} = \sum_{j=1}^{s_{\ell+1}} \frac{\partial L}{\partial z_j^{(\ell+1)}} \frac{\partial z_j^{(\ell+1)}}{\partial a_i^{(\ell)}} \frac{\partial a_i^{(\ell)}}{\partial z_i^{(\ell)}}$$

$$= f'\left(z_i^{(\ell)}\right) \sum_{j=1}^{s_{\ell+1}} \frac{\partial L}{\partial z_j^{(\ell+1)}} W_{ji}^{(\ell+1)}$$

$$\delta_i^{(\ell)} = f'\left(z_i^{(\ell)}\right) \sum_{j=1}^{s_{\ell+1}} \delta_j^{(\ell+1)} W_{ji}^{(\ell+1)}$$

计算梯度 (张量)



$$\nabla_{w^{(\ell)}} L(w,b) = \frac{\partial L}{\partial W^{(\ell)}} = \frac{\partial L}{\partial z^{(\ell)}} \frac{\partial z^{(\ell)}}{\partial W^{(\ell)}} = \delta^{(\ell)} \left(a^{(\ell-1)}\right)^T$$

$$z^{(\ell)} = W^{(\ell)} a^{(\ell-1)} + b^{(\ell)}$$

$$a^{(\ell)} = f\left(z^{(\ell)}\right)$$

$$z^{(\ell+1)} = W^{(\ell+1)} a^{(\ell)} + b^{(\ell+1)}$$

$$\frac{\partial z^{(\ell)}}{\partial W^{(\ell)}} = \left(a^{(\ell-1)}\right)^T$$

$$\frac{\partial L}{\partial z^{(\ell)}} = \frac{\partial a^{(\ell)}}{\partial z^{(\ell)}} \frac{\partial z^{(\ell+1)}}{\partial a^{(\ell)}} \frac{\partial L}{\partial z^{(\ell+1)}} = f'\left(z^{(\ell)}\right) \odot \left(W^{(\ell+1)}\right)^T \frac{\partial L}{\partial z^{(\ell+1)}}$$

$$\delta^{(\ell)} = f'\left(z^{(\ell)}\right) \odot \left(W^{(\ell+1)}\right)^T \delta^{(\ell+1)}$$

$$\delta^{(n_\ell)} = f'\left(z^{(\ell)}\right) \odot \nabla_y L(W,b)$$

前向输出

$$z^{(\ell)} = w^{(\ell)} a^{(\ell-1)} + b^{(\ell)}$$
$$a^{(\ell)} = f(z^{(\ell)})$$
$$a^{(0)} = x$$
$$y = a^{(n_\ell)}$$

反向传播

$$\delta^{(\ell)} = \begin{cases} f'(z^{(\ell)}) \odot (w^{(\ell+1)})^T \delta^{(\ell+1)}; & \ell \neq n_\ell \\ f'(z^{(\ell)}) \odot \nabla_y L(W, b); & \ell = n_\ell \end{cases}$$

梯度计算

$$\nabla_{W^{(\ell)}} L(W, b; x, y) = \delta^{(\ell)} \otimes a^{(\ell-1)}$$
$$\nabla_{b^{(\ell)}} L(W, b; x, y) = \delta^{(\ell)}$$
$$\ell = 1, 2, ..., n_\ell$$

变化量

$$\Delta w^{(\ell)} \leftarrow \Delta w^{(\ell)} + \nabla_{w^{(\ell)}} L\left(w, b; x^{(k)}, t^{(k)}\right)$$

$$\Delta b^{(\ell)} \leftarrow \Delta b^{(\ell)} + \nabla_{b^{(\ell)}} L\left(w, b; x^{(k)}, t^{(k)}\right)$$

$$k = 1, 2, ..., m; \ell = 1, 2, ..., n_\ell$$

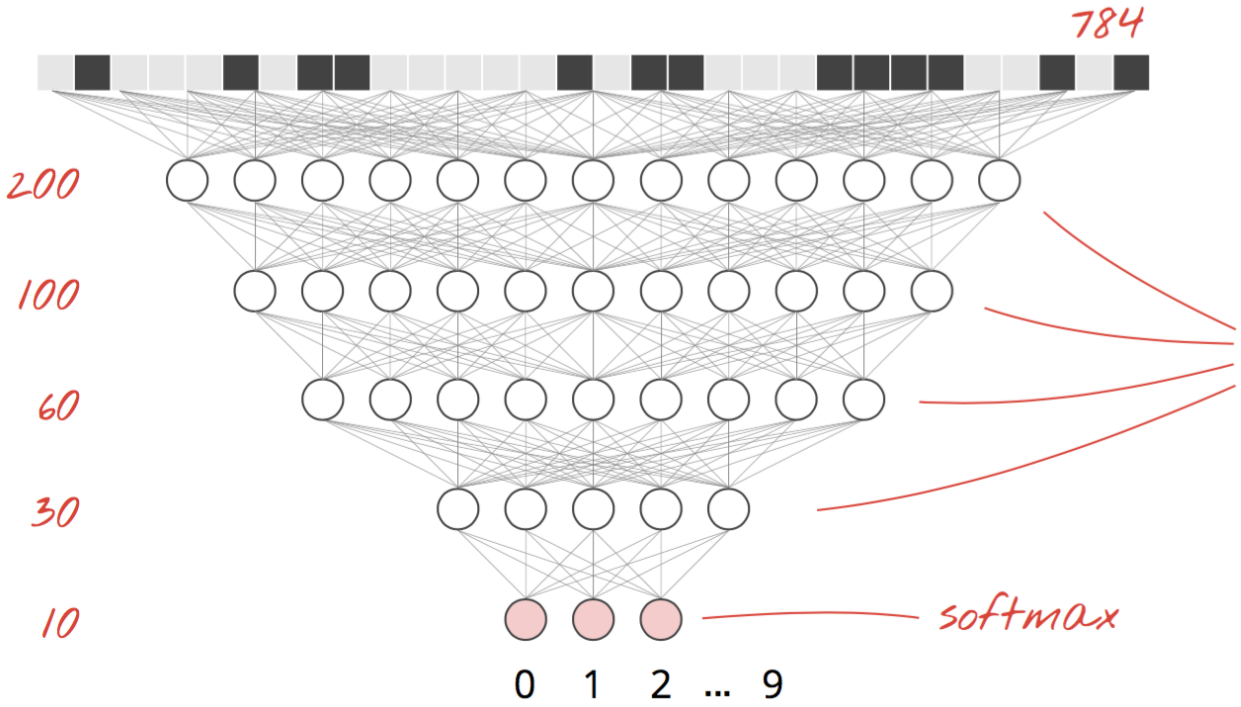
权重更新

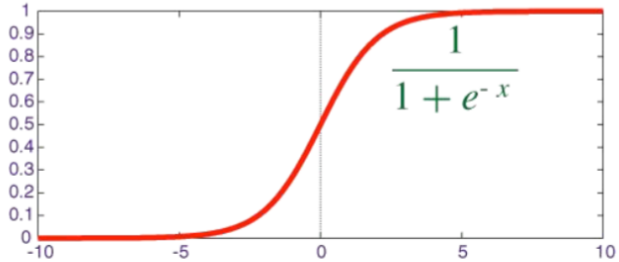
$$w^{(\ell)} \leftarrow w^{(\ell)} - \alpha \left(\frac{\Delta w^{(\ell)}}{m} \right)$$

$$b^{(\ell)} \leftarrow b^{(\ell)} - \alpha \frac{\Delta b^{(\ell)}}{m}$$

$$\ell = 1, 2, ..., n_\ell$$

多层前馈网络


784
200
100
60
30
10
0 1 2 ... 9
softmax


1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
-10
-5
0
5
10
$\frac{1}{1+e^{-x}}$
sigmoid function

定义参数

```
K = 200
L = 100
M = 60
N = 30

W1 = tf.Variable(tf.truncated_normal([28*28, K] ,stddev=0.1))
B1 = tf.Variable(tf.zeros([K]))
W2 = tf.Variable(tf.truncated_normal([K, L], stddev=0.1))
B2 = tf.Variable(tf.zeros([L]))
W3 = tf.Variable(tf.truncated_normal([L, M], stddev=0.1))
B3 = tf.Variable(tf.zeros([M]))
W4 = tf.Variable(tf.truncated_normal([M, N], stddev=0.1))
B4 = tf.Variable(tf.zeros([N]))
W5 = tf.Variable(tf.truncated_normal([N, 10], stddev=0.1))
B5 = tf.Variable(tf.zeros([10]))
```

weights initialised with random values

定义模型

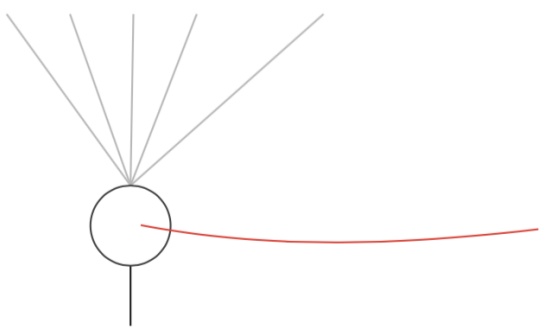
weights and biases

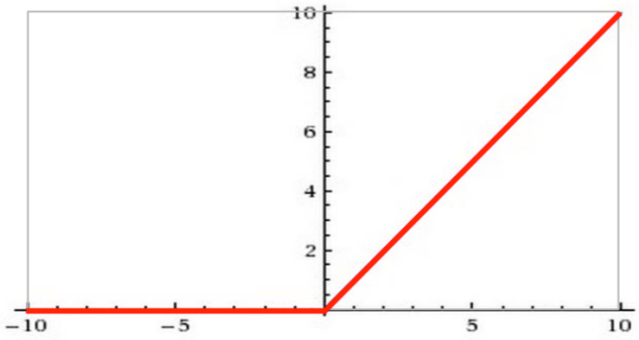
```
X = tf.reshape(X, [-1, 28*28])

Y1 = tf.nn.sigmoid(tf.matmul(X, W1) + B1)
Y2 = tf.nn.sigmoid(tf.matmul(Y1, W2) + B2)
Y3 = tf.nn.sigmoid(tf.matmul(Y2, W3) + B3)
Y4 = tf.nn.sigmoid(tf.matmul(Y3, W4) + B4)
Y = tf.nn.softmax(tf.matmul(Y4, W5) + B5)
```

梯度消失：引入 ReLU

RELU = Rectified Linear Unit



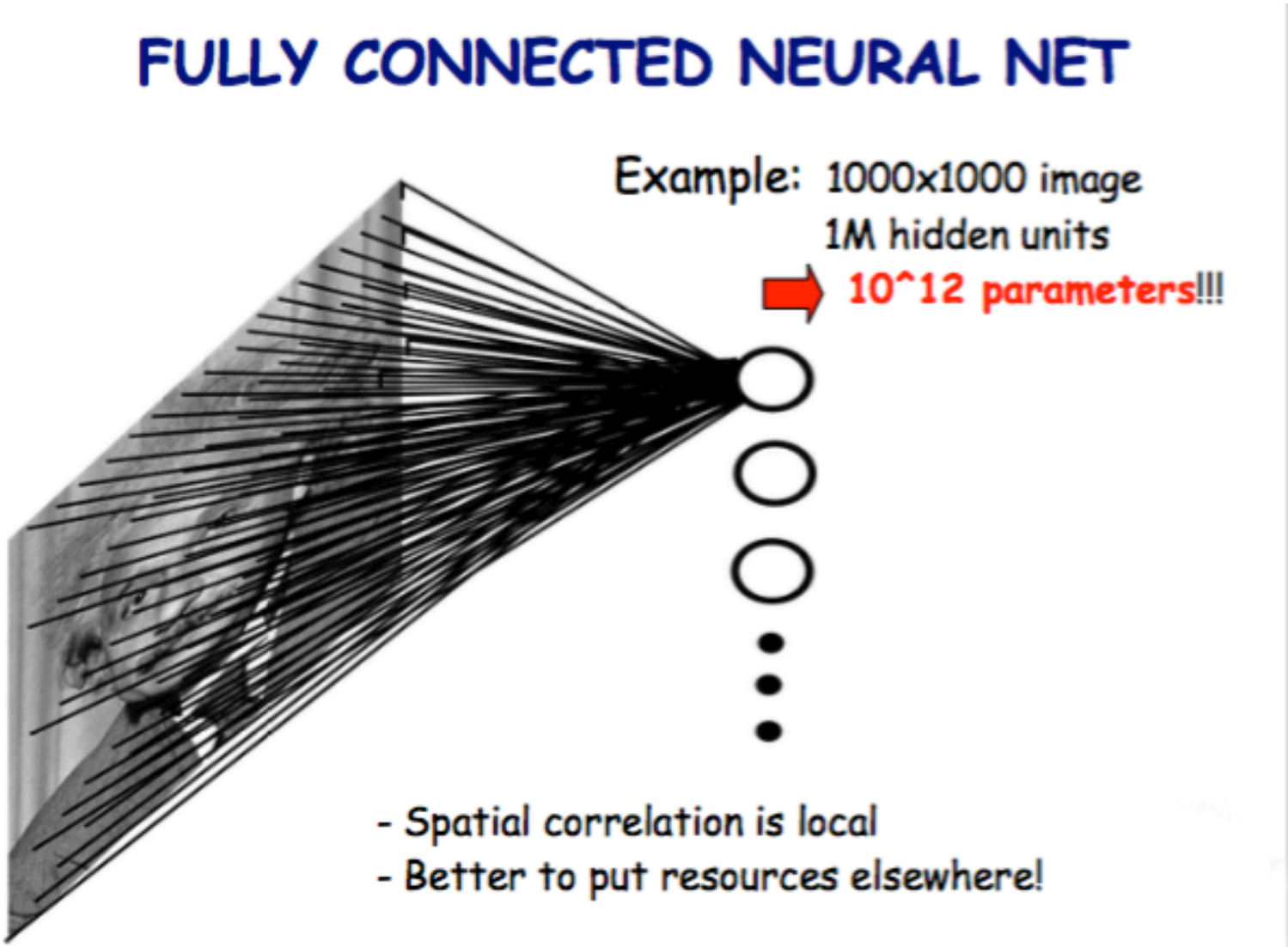



```
Y = tf.nn.relu(tf.matmul(X, W) + b)
```

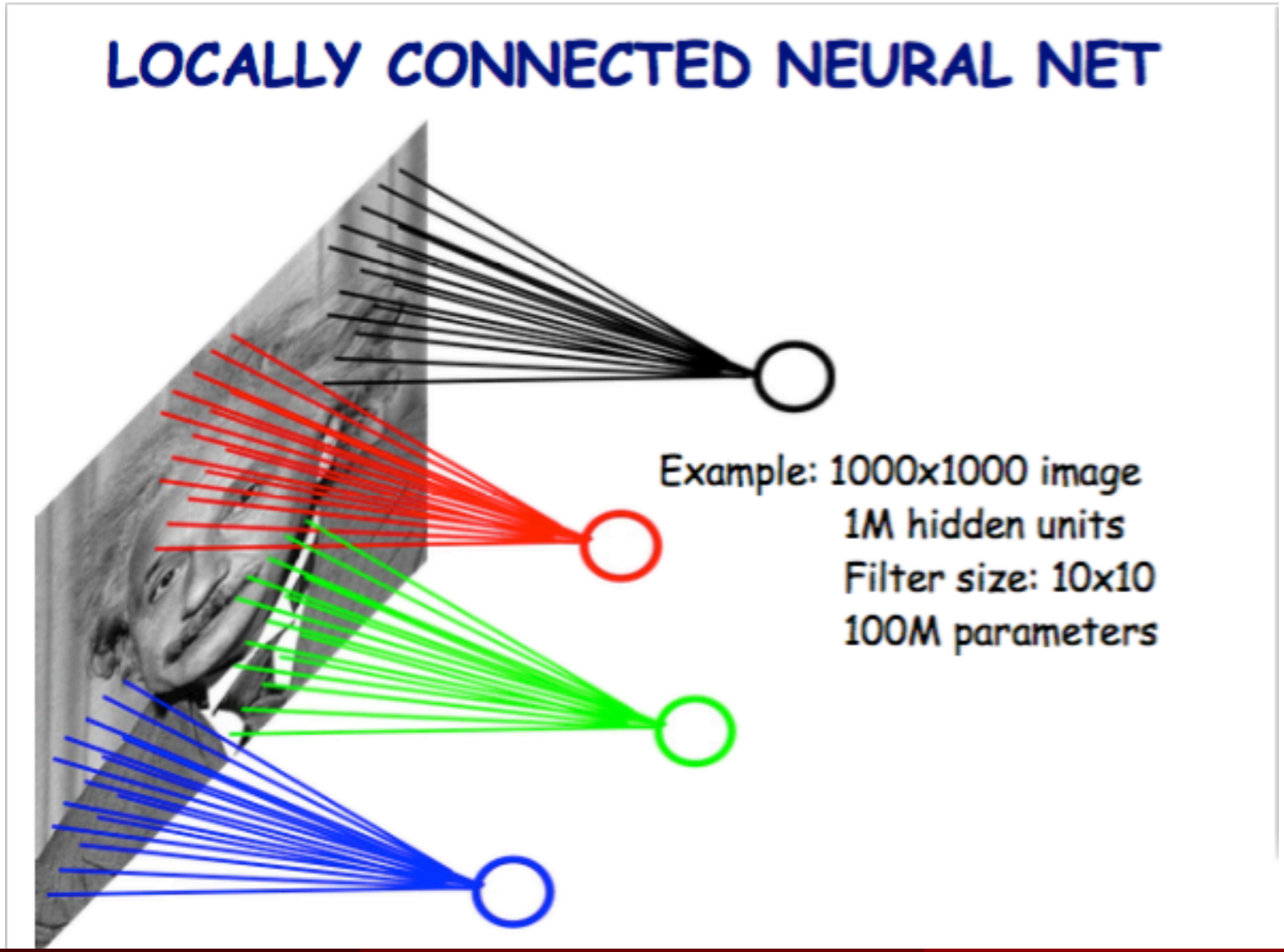
卷积神经网络

1. 动机
2. 卷积层
3. 池化层
4. 计算

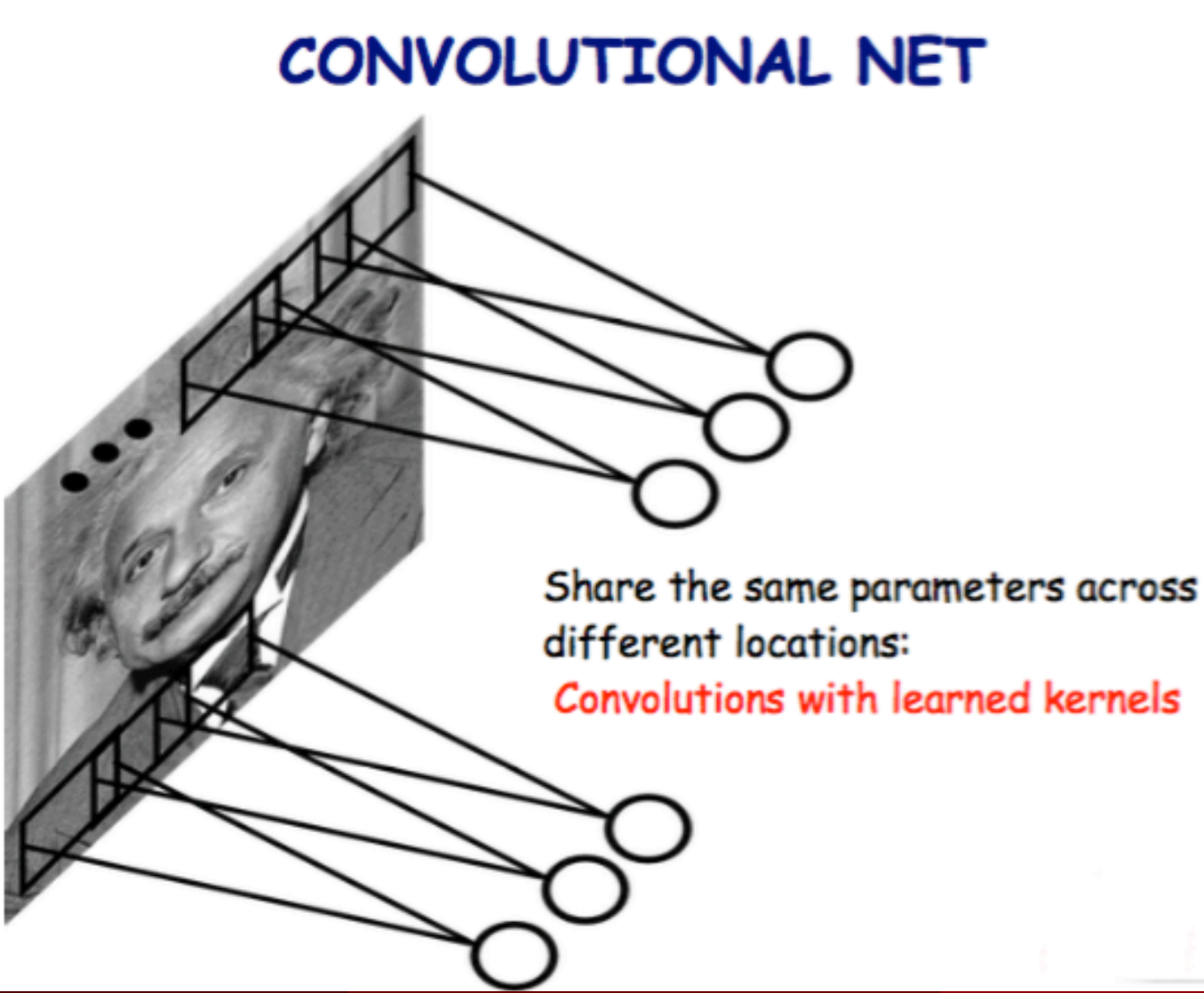
全连接网络


FULLY CONNECTED NEURAL NET
Example: 1000x1000 image
1M hidden units
10^12 parameters!!!
- Spatial correlation is local
- Better to put resources elsewhere!

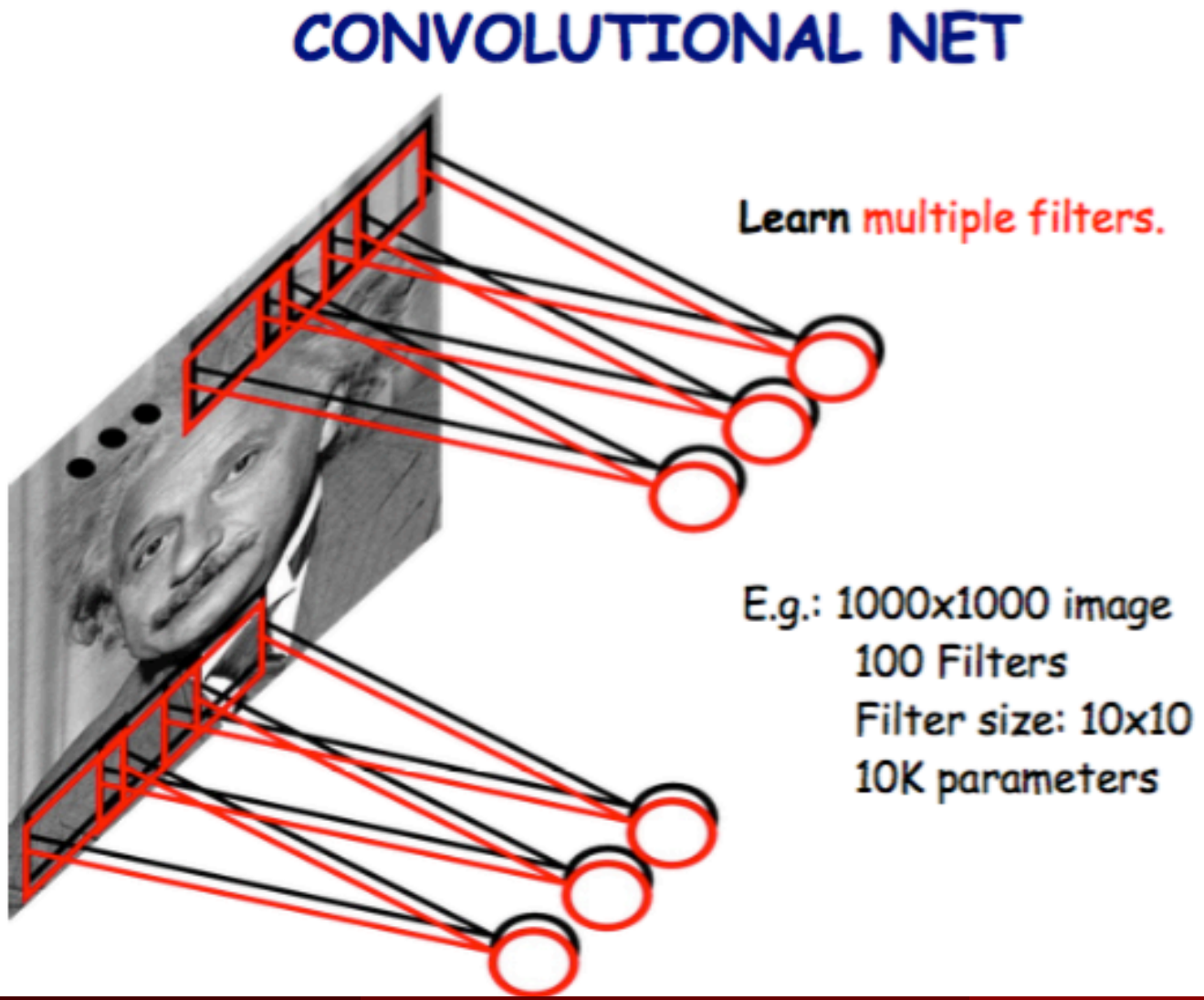
局部连接网络


LOCALLY CONNECTED NEURAL NET
Example: 1000x1000 image
1M hidden units
Filter size: 10x10
100M parameters

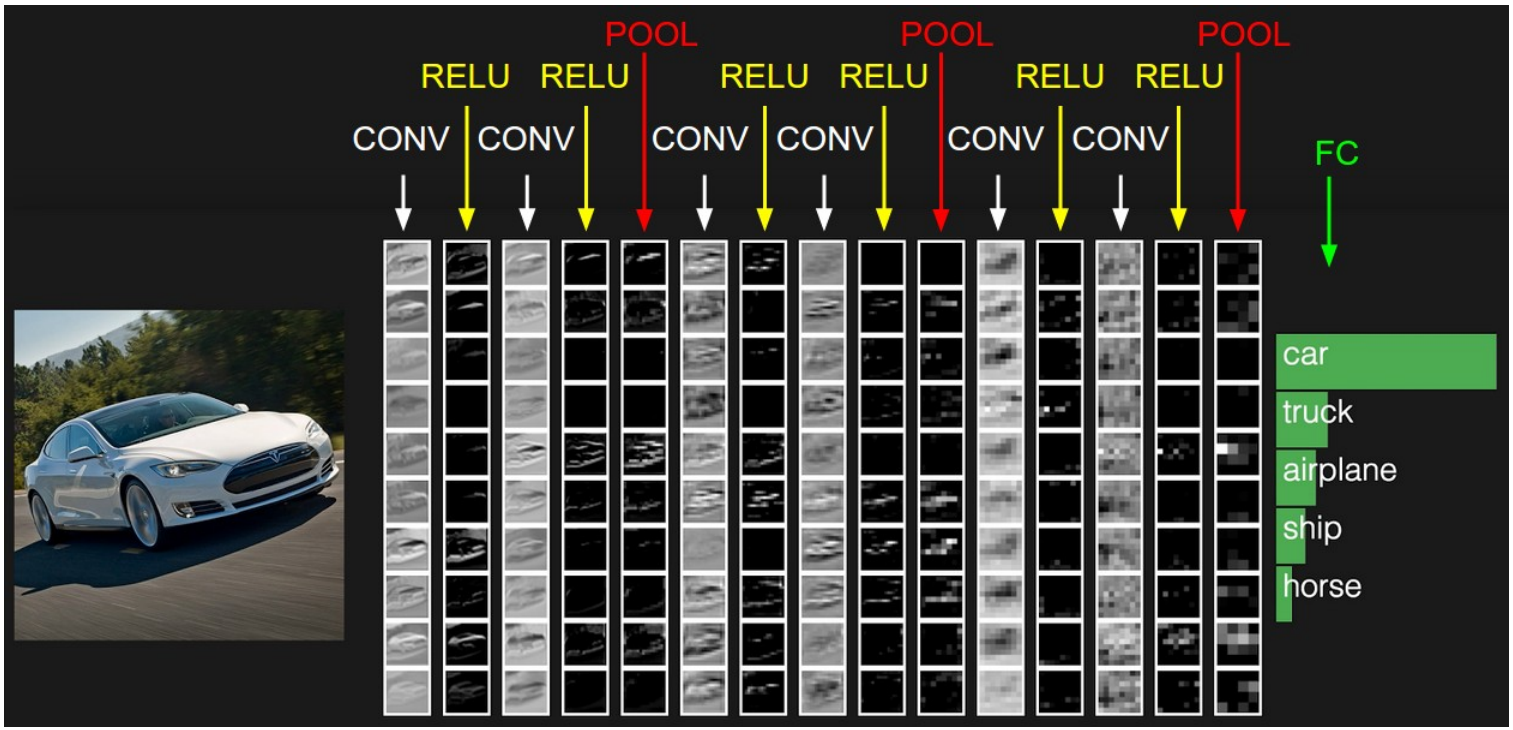
权值共享：一个卷积核


CONVOLUTIONAL NET
Share the same parameters across different locations:
Convolutions with learned kernels

权值共享：多个卷积核

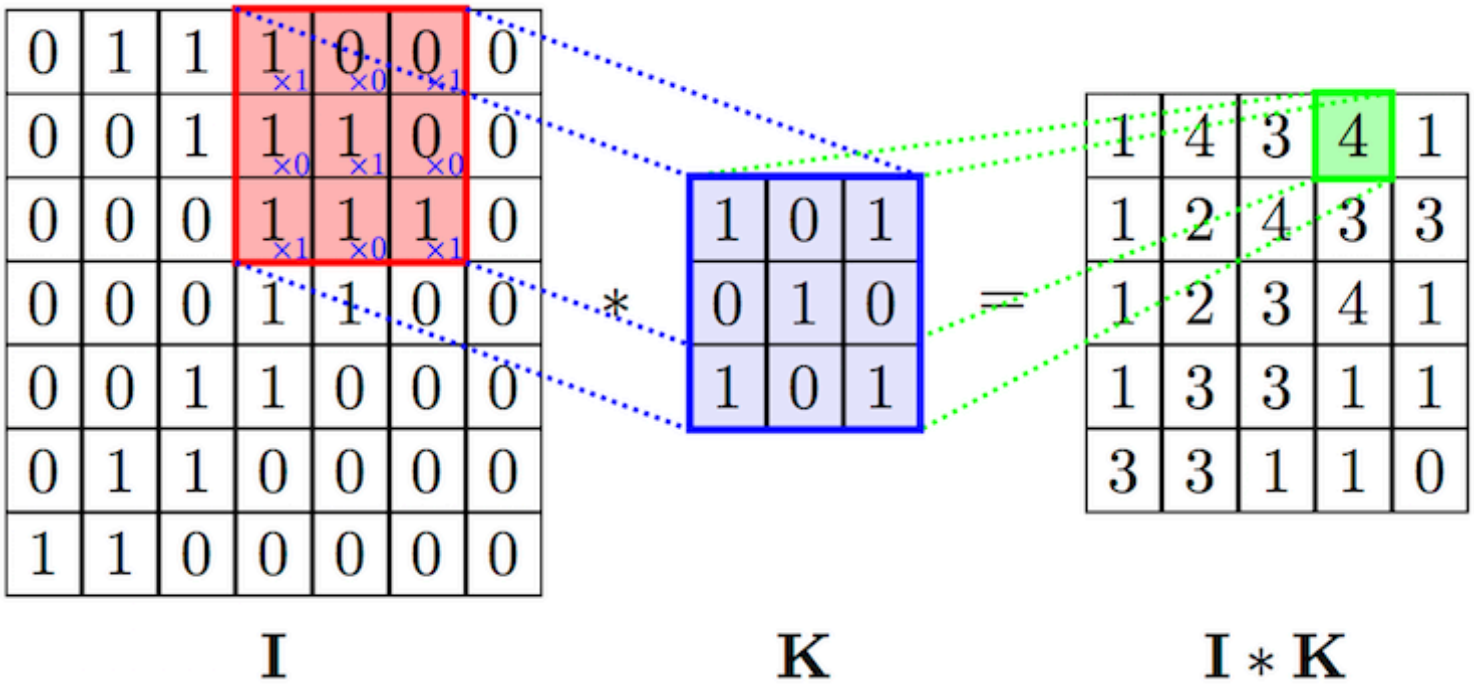

CONVOLUTIONAL NET
Learn multiple filters.
E.g.: 1000x1000 image
100 Filters
Filter size: 10x10
10K parameters

架构模式

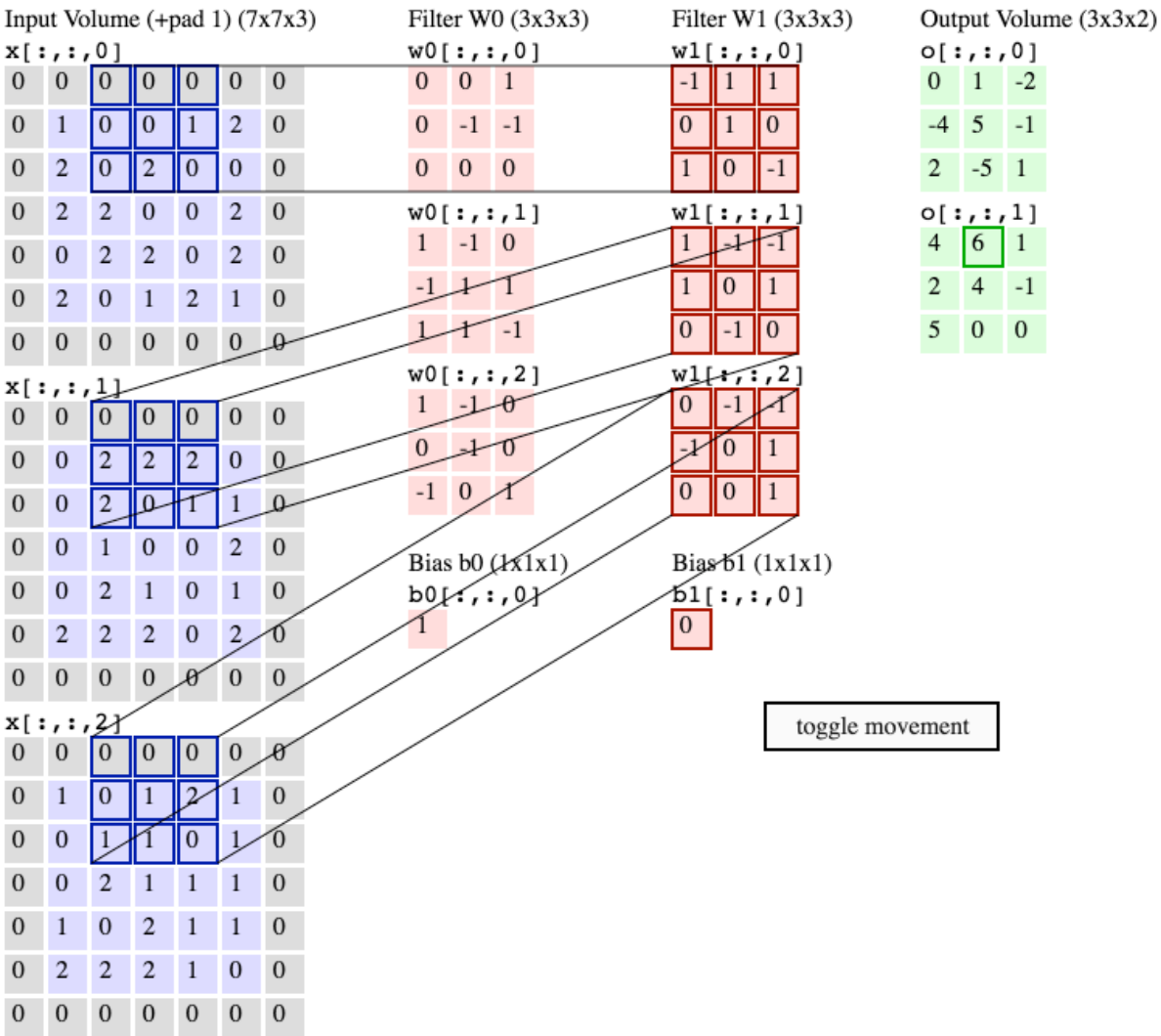



```
INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC
```

卷积运算


0 1 1 1 0 0 0
0 0 1 1 1 0 0
0 0 0 1 1 1 0
0 0 0 1 1 0 0
0 0 1 1 0 0 0
0 1 1 0 0 0 0
1 1 0 0 0 0 0
×1 ×0 ×1
×0 ×1 ×0
×1 ×0 ×1
*
1 0 1
0 1 0
1 0 1
=
1 4 3 4 1
1 2 4 3 3
1 2 3 4 1
1 3 3 1 1
3 3 1 1 0
I
K
I * K

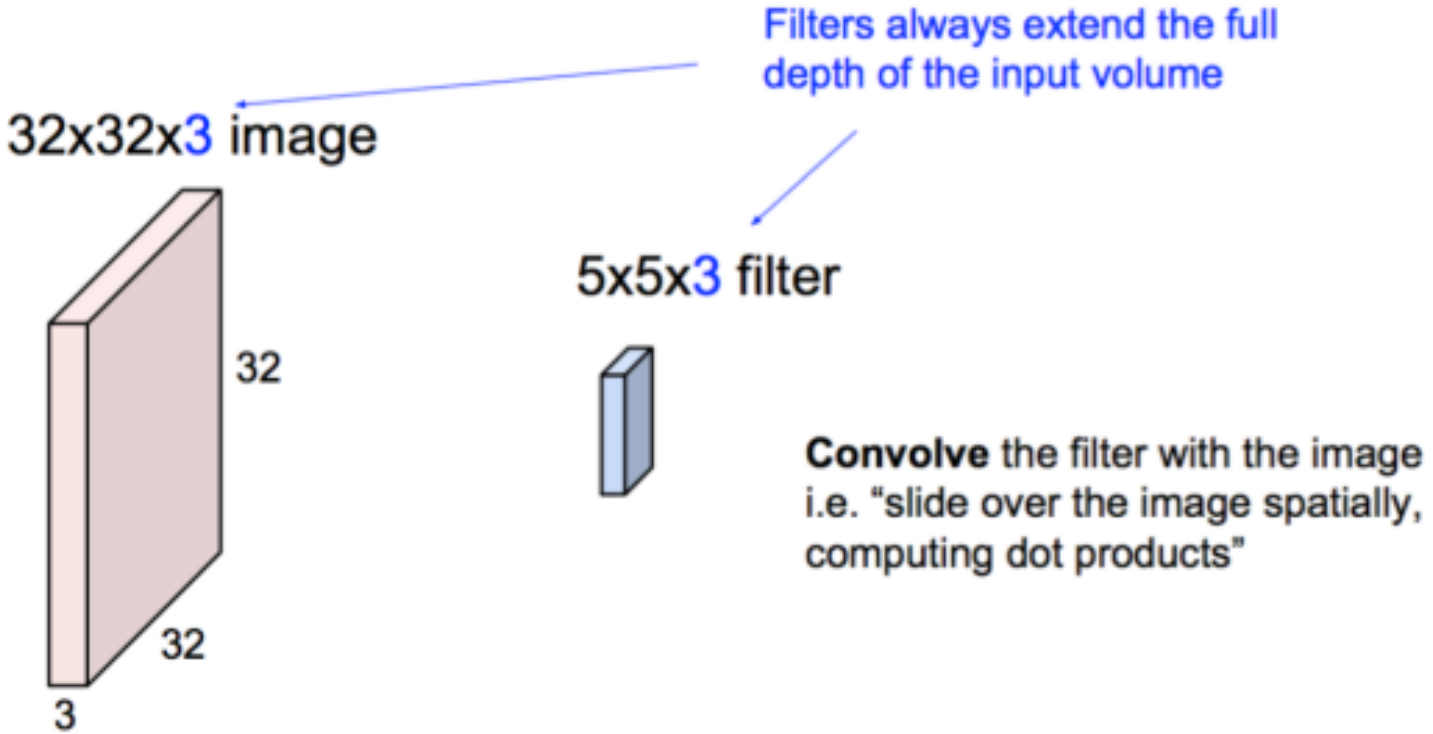
卷积运算


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 0 0 1 2 0
0 2 0 2 0 0 0
0 2 2 0 0 2 0
0 0 2 2 0 2 0
0 2 0 1 2 1 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 0 2 2 2 0 0
0 0 2 0 1 1 0
0 0 1 0 0 2 0
0 0 2 1 0 1 0
0 2 2 2 0 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 1 0 1 2 1 0
0 0 1 1 0 1 0
0 0 2 1 1 1 0
0 1 0 2 1 1 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 1
0 -1 -1
0 0 0
w0[:,:,1]
1 -1 0
-1 1 1
1 1 -1
w0[:,:,2]
1 -1 0
0 -1 0
-1 0 1
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 1 1
0 1 0
1 0 -1
w1[:,:,1]
1 -1 -1
1 0 1
0 -1 0
w1[:,:,2]
0 -1 -1
-1 0 1
0 0 1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
0 1 -2
-4 5 -1
2 -5 1
o[:,:,1]
4 6 1
2 4 -1
5 0 0
toggle movement

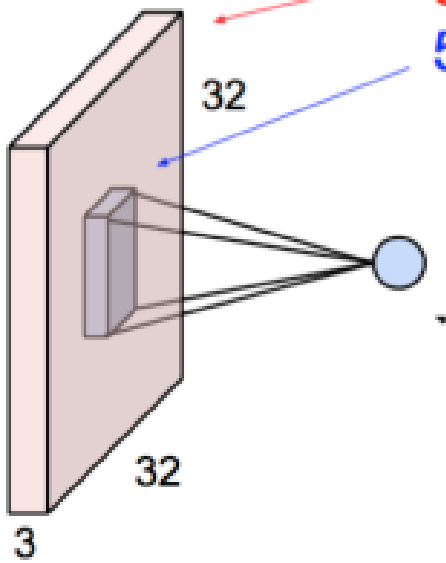
卷积运算

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

卷积运算


Filters always extend the full depth of the input volume
32x32x3 image
32
32
3
5x5x3 filter
Convolve the filter with the image
i.e. "slide over the image spatially,
computing dot products"

卷积运算



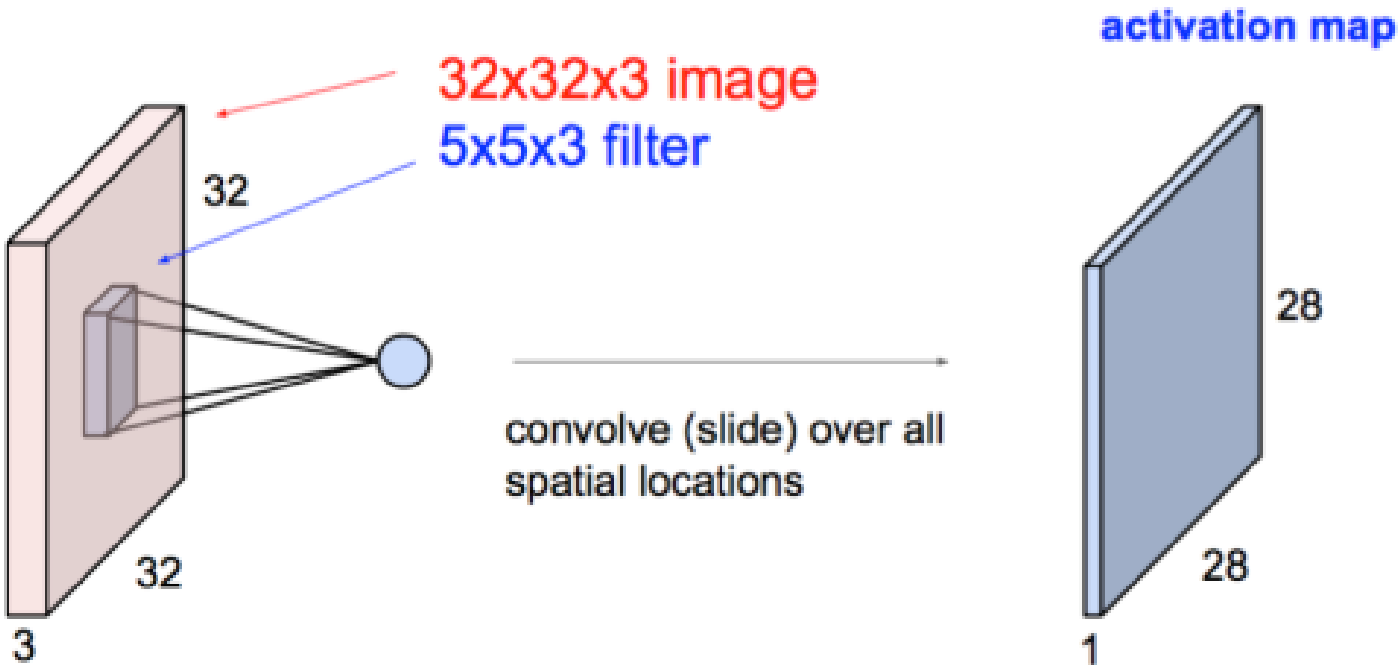
32x32x3 image

5x5x3 filter w

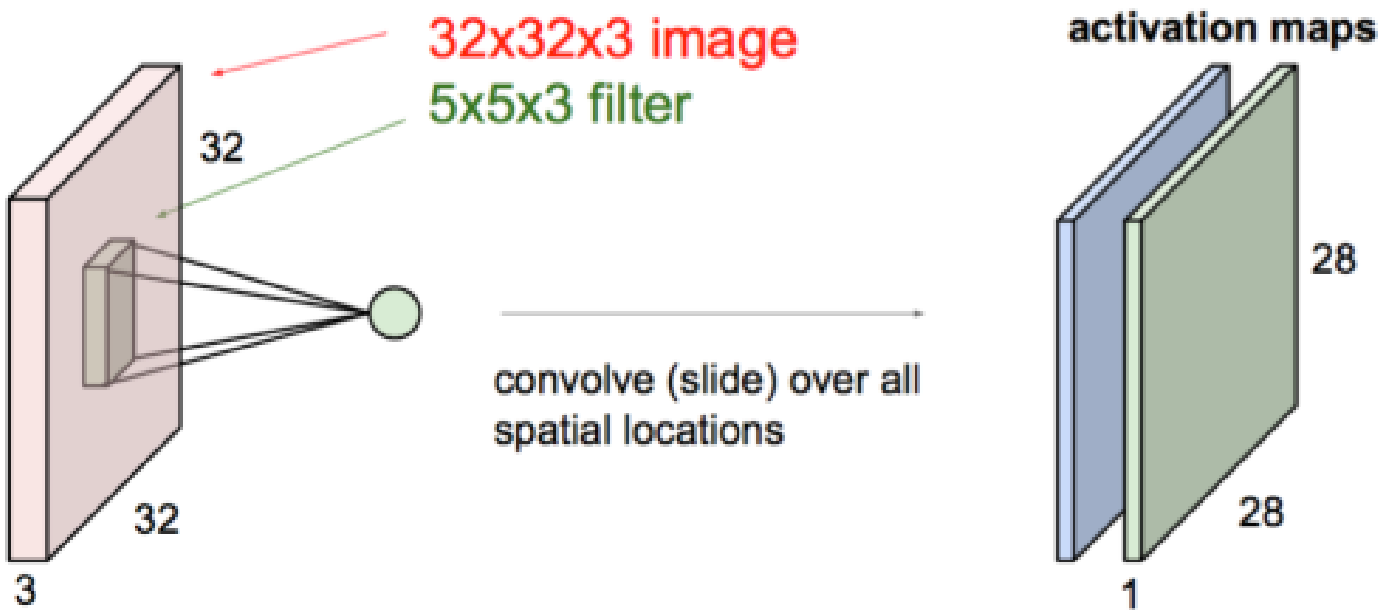
1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

卷积运算

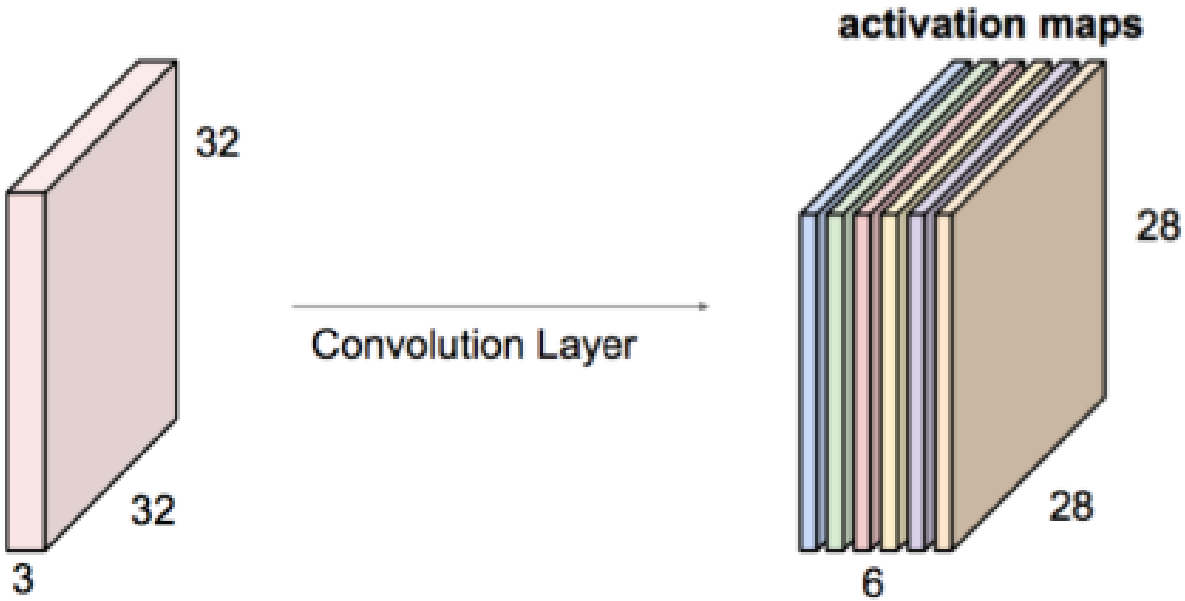

32x32x3 image
5x5x3 filter
32
32
3
convolve (slide) over all
spatial locations
activation map
28
28
1

卷积运算


32x32x3 image
5x5x3 filter
32
32
3
convolve (slide) over all
spatial locations
activation maps
28
28
1

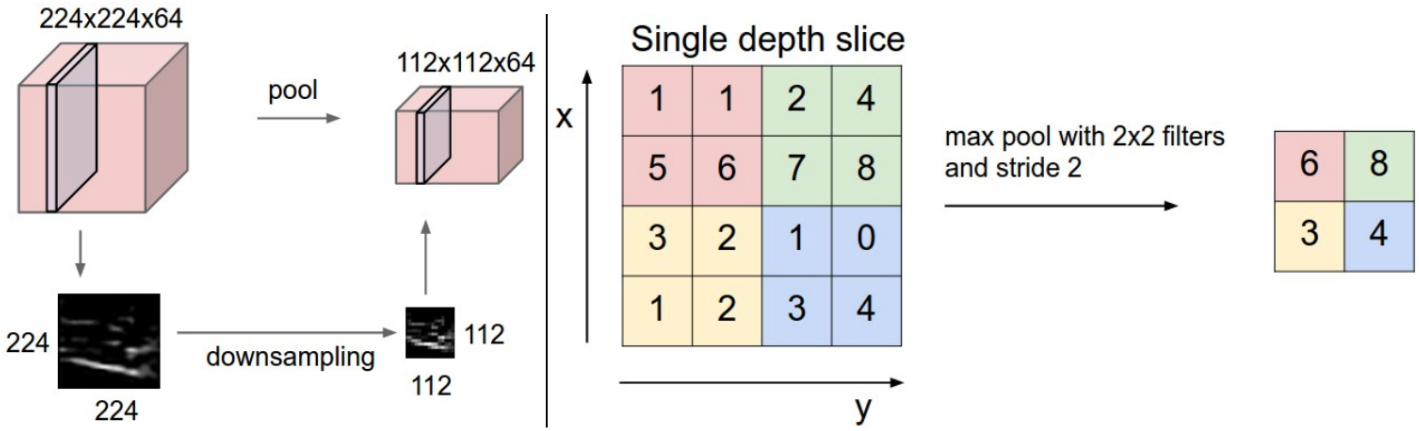
卷积运算

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



We stack these up to get a "new image" of size 28x28x6!

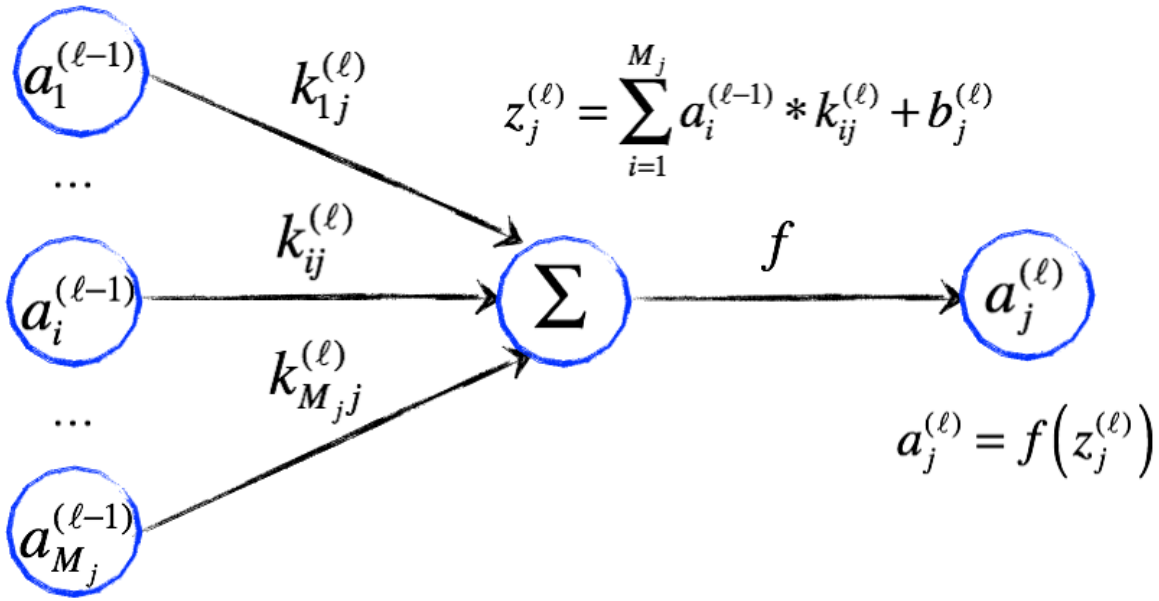
下采样：Max Pooling



下采样

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires two hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

前向计算：卷积

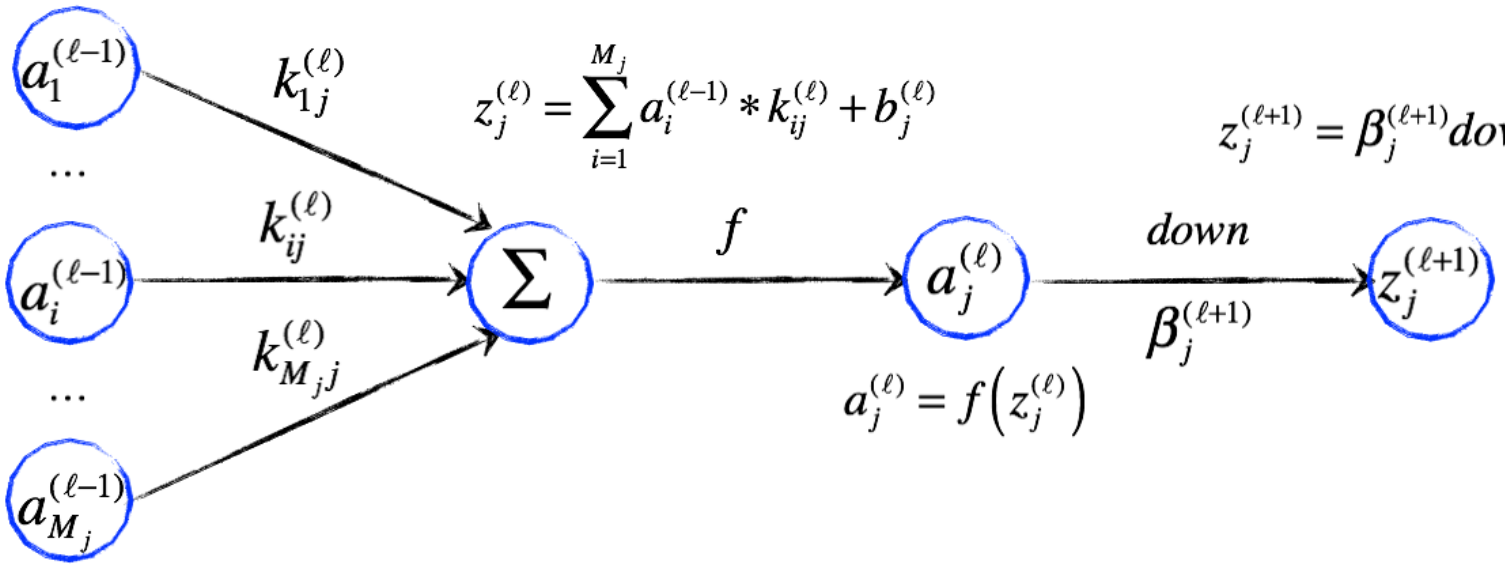


$$\frac{\partial L}{\partial k_{ij}^{(\ell)}} = a_i^{(\ell-1)} * \delta_j^{(\ell)}$$

$$\frac{\partial L}{\partial k_{ij}^{(\ell)}} = rot180\left(conv2\left(a_i^{(\ell-1)}, rot180\left(\delta_j^{(\ell)}\right), 'valid'\right)\right)$$

$$\frac{\partial L}{\partial b_j^{(\ell)}} = \sum_{u,v}\left(\delta_j^{(\ell)}\right)_{u,v}$$

反向传播：下采样



$$z_j^{(\ell)} = \sum_{i=1}^{M_j} a_i^{(\ell-1)} * k_{ij}^{(\ell)} + b_j^{(\ell)}$$

$$z_j^{(\ell+1)} = \beta_j^{(\ell+1)} down\left(a_j^{(\ell)}\right) + b_j^{(\ell+1)}$$

$$a_j^{(\ell)} = f\left(z_j^{(\ell)}\right)$$

$$\delta_j^{(\ell)} = \beta_j^{(\ell+1)}\left(f'\left(z_j^{(\ell)}\right) \odot up\left(\delta_j^{(\ell+1)}\right)\right)$$

前向计算：下采样

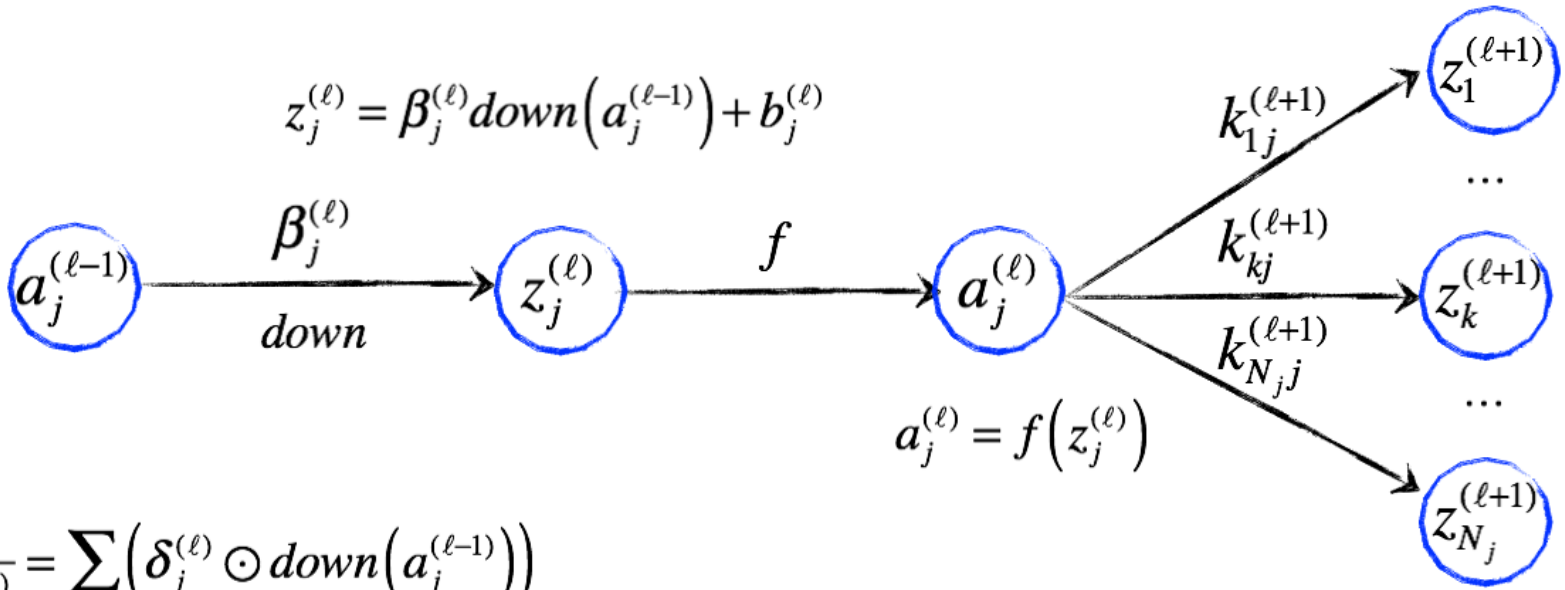
$$z_j^{(\ell)} = \beta_j^{(\ell)} down\left(a_j^{(\ell-1)}\right) + b_j^{(\ell)} \qquad a_j^{(\ell)} = f\left(z_j^{(\ell)}\right)$$

$$\frac{\partial L}{\partial \beta_j^{(\ell)}} = \sum_{u,v}\left(\delta_j^{(\ell)} \odot down\left(a_j^{(\ell-1)}\right)\right)_{u,v}$$

$$\frac{\partial L}{\partial b_j^{(\ell)}} = \sum_{u,v}\left(\delta_j^{(\ell)}\right)_{u,v}$$

反向传播：卷积

$$z_j^{(\ell)} = \beta_j^{(\ell)} down\left(a_j^{(\ell-1)}\right) + b_j^{(\ell)}$$



$$\frac{\partial L}{\partial \beta_j^{(\ell)}} = \sum_{u,v}\left(\delta_j^{(\ell)} \odot down\left(a_j^{(\ell-1)}\right)\right)_{u,v}$$

$$\frac{\partial L}{\partial b_j^{(\ell)}} = \sum_{u,v}\left(\delta_j^{(\ell)}\right)_{u,v}$$

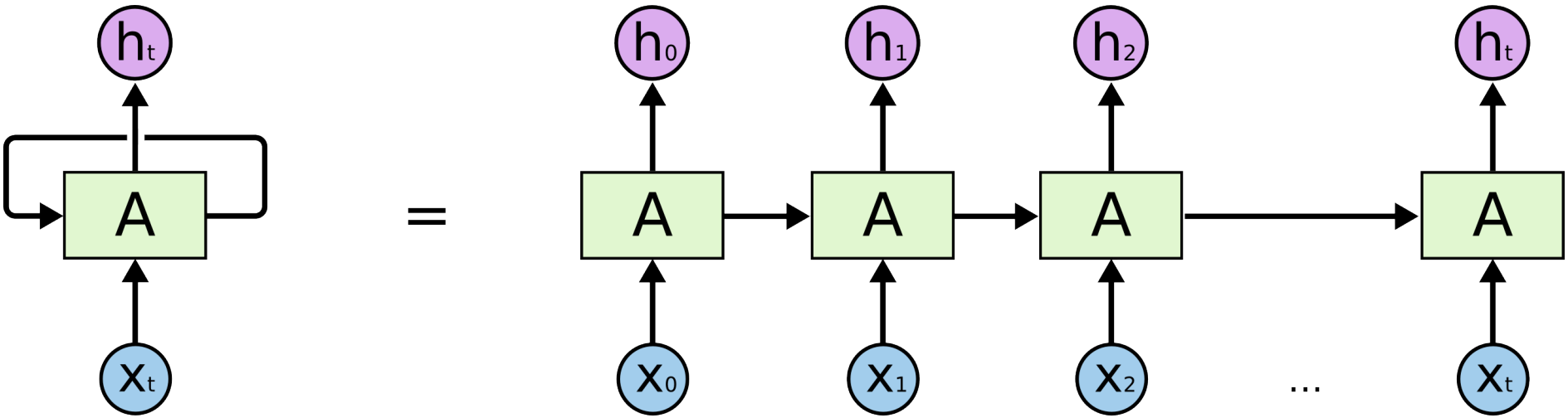
$$\delta_j^{(\ell)} = f'\left(z_j^{(\ell)}\right) \odot \sum_{k=1}^{N_j} padding\left(\delta_k^{(\ell+1)}\right) * rot180\left(k_{kj}^{(\ell+1)}\right)$$

$$= f'\left(z_j^{(\ell)}\right) \odot \sum_{k=1}^{N_j} conv2\left(\delta_k^{(\ell+1)}, rot180\left(k_{kj}^{(\ell+1)}\right), 'full'\right)$$

循环神经网络

1. BPTT 算法
2. 双向 RNN
3. 深层 RNN

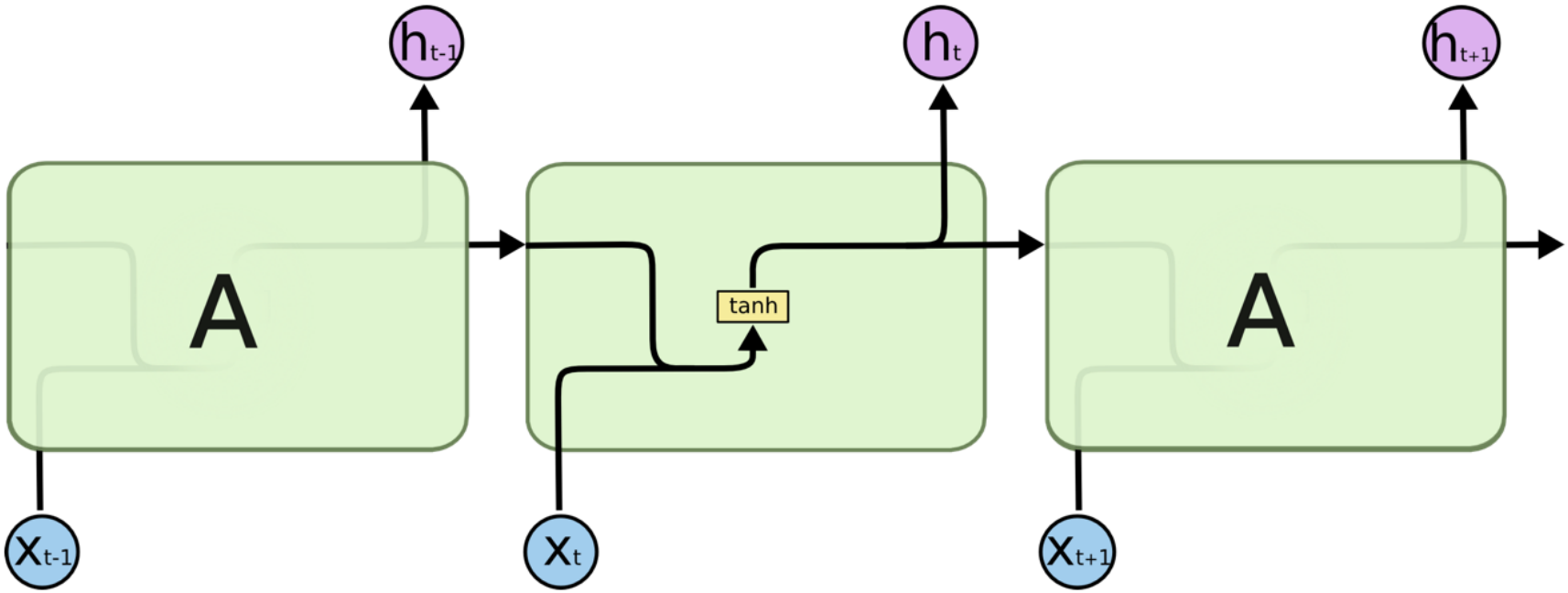
网络架构



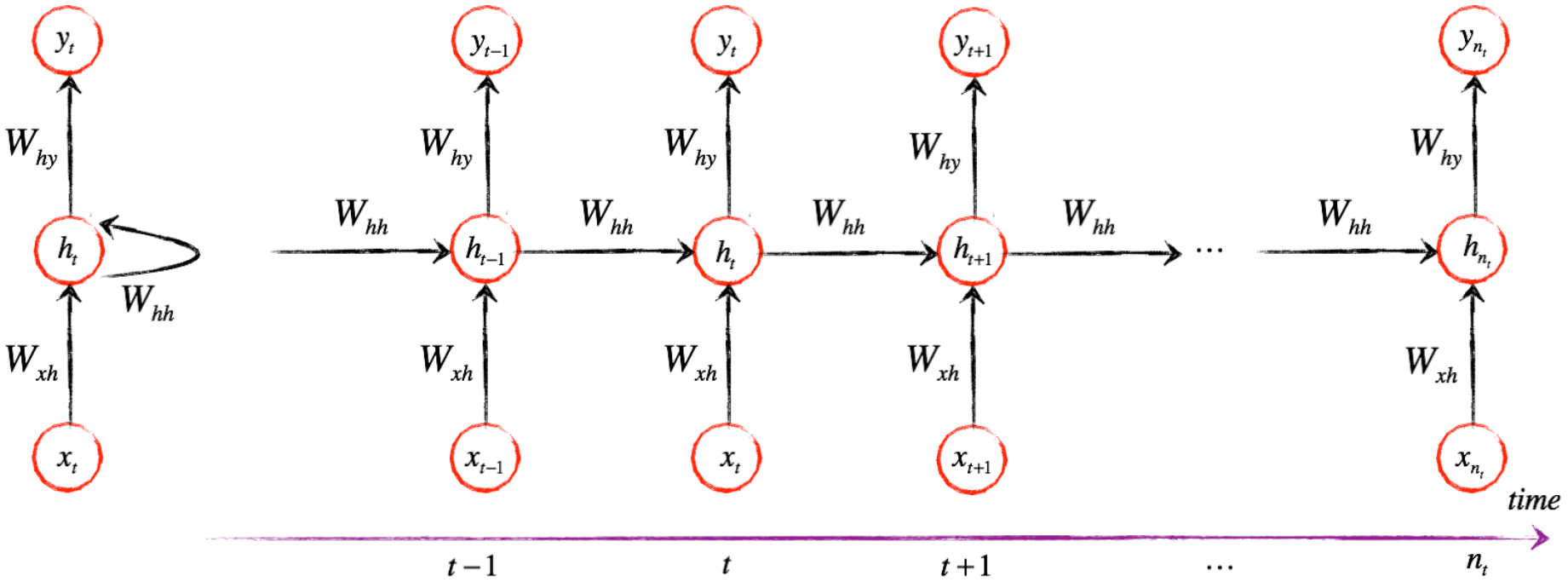
网络架构

$$u_t = W\left[x_t, h_{t-1}\right]$$

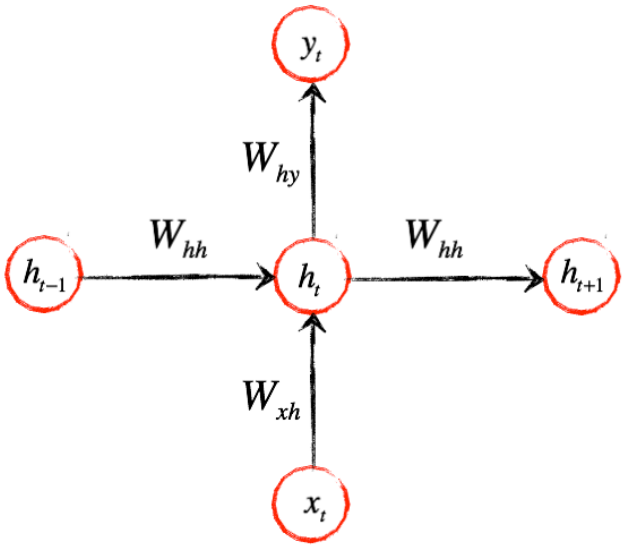
$$h_t = \tanh\left(u_t\right)$$



网络架构


y_t
W_{hy}
h_t
W_{hh}
W_{xh}
x_t
y_{t-1}
h_{t-1}
x_{t-1}
y_{t+1}
h_{t+1}
x_{t+1}
y_{n_t}
h_{n_t}
x_{n_t}
time
$t-1$
t
$t+1$
n_t

前向计算



$$u_t = W_{xh}x_t + W_{hh}h_{t-1}$$
$$h_t = f(u_t)$$
$$v_t = W_{hy}h_t$$
$$y_t = g(v_t)$$

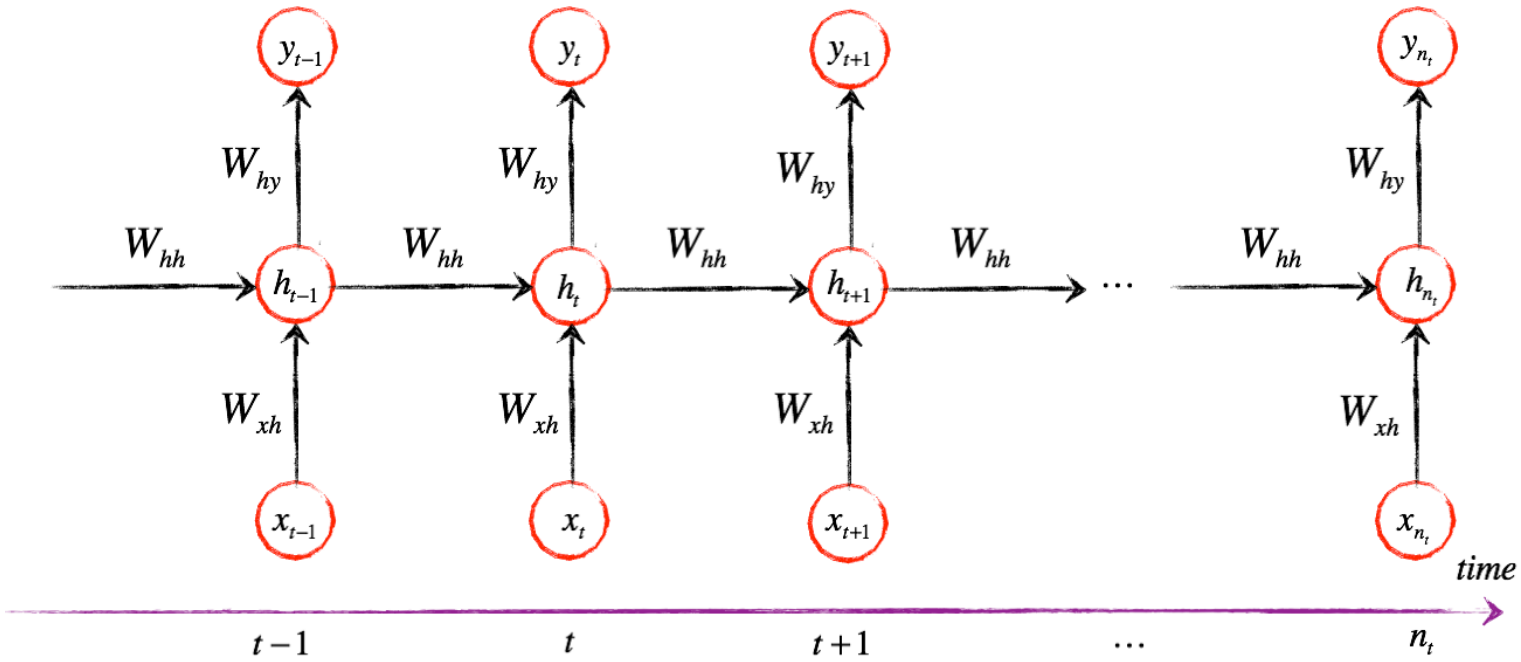
$$\begin{aligned} u_t &= W_{xh}x_t + W_{hh}h_{t-1} \\ &= W[x_t, h_{t-1}] \\ &= \begin{pmatrix} W_{xh} & W_{hh} \end{pmatrix} \begin{pmatrix} x_t \\ h_{t-1} \end{pmatrix} \end{aligned}$$

$$\nabla_{W_{hy}} L = \sum_{t=1}^{n_t} \nabla_{W_{hy}} L_t \qquad \nabla_{W_{hy}} L_t = \frac{\partial L_t}{\partial W_{hy}} = \frac{\partial L_t}{\partial v_t}\frac{\partial v_t}{\partial W_{hy}} = \delta_t^y \otimes h_t$$

$$\nabla_{W_{hh}} L = \sum_{t=1}^{n_t} \nabla_{W_{hh}} L_t \qquad \nabla_{W_{hh}} L_t = \frac{\partial L_t}{\partial W_{hh}} = \frac{\partial L_t}{\partial u_t}\frac{\partial u_t}{\partial W_{hh}} = \delta_t^h \otimes h_{t-1}$$

$$\nabla_{W_{xh}} L = \sum_{t=1}^{n_t} \nabla_{W_{xh}} L_t \qquad \nabla_{W_{xh}} L_t = \frac{\partial L_t}{\partial W_{xy}} = \frac{\partial L_t}{\partial u_t}\frac{\partial u_t}{\partial W_{xh}} = \delta_t^h \otimes x_t$$

反向传播：BPTT



$$t = n_t - 1, n_t - 2, \ldots, 1$$

$$\delta_t^y = \left(\nabla_{y_t} L_t\right) L \odot g'\left(v_t\right)$$

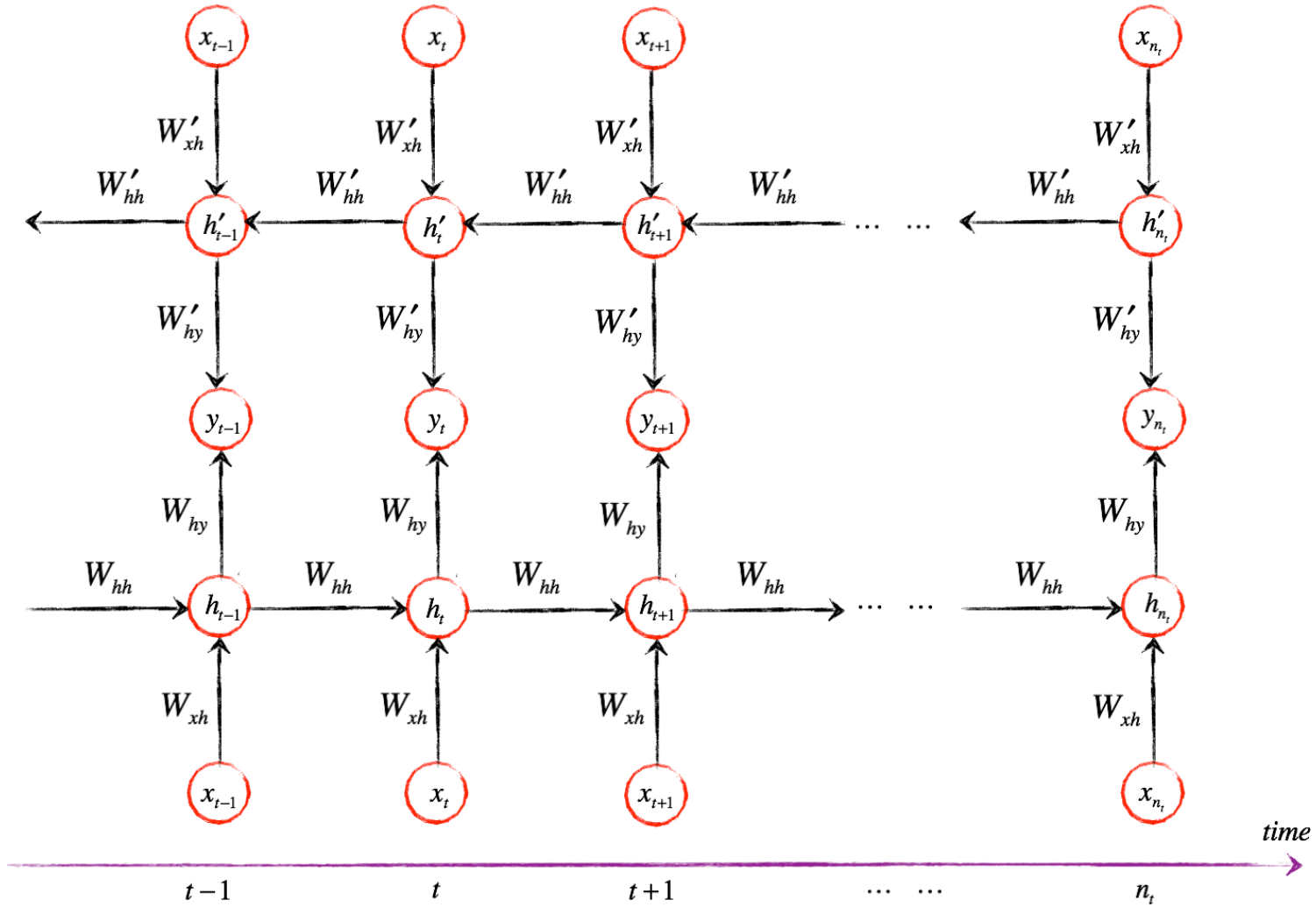
$$\delta_t^h = \left[\left(W_{hy}\right)^T \delta_t^y + \left(W_{hh}\right)^T \delta_{t+1}^h\right] \odot f'\left(u_t\right)$$

$$t = n_t$$

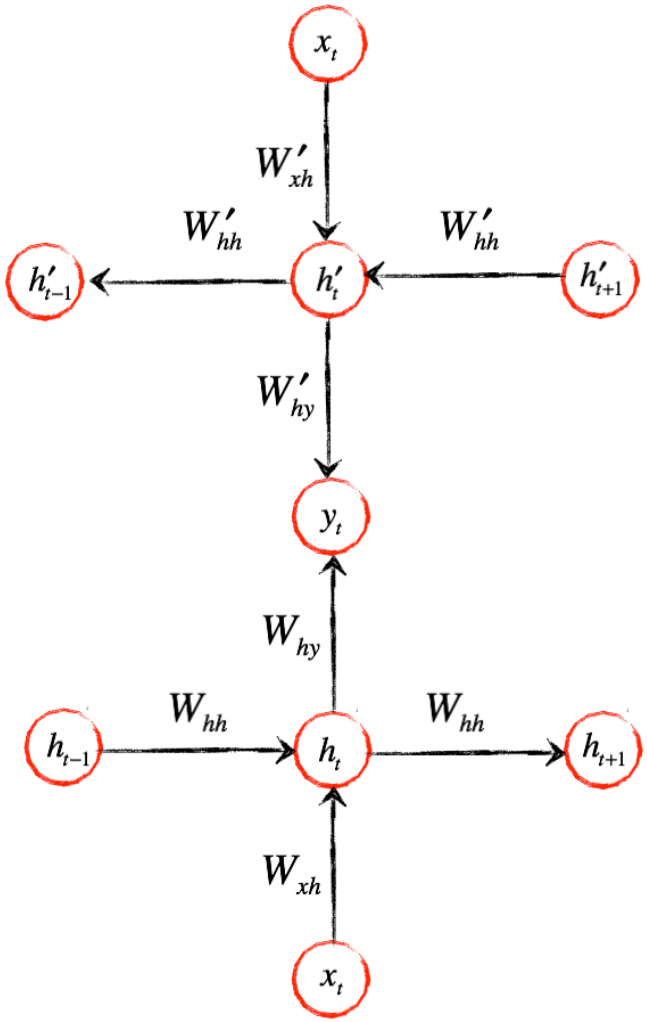
$$\delta_{n_t}^y = \left(\nabla_{y_{n_t}} L_{n_t}\right) \odot g'\left(v_{n_t}\right)$$

$$\delta_{n_t}^h = \left(W_{hy}\right)^T \delta_{n_t}^y \odot f'\left(u_{n_t}\right)$$

网络架构


x_{t-1} x_t x_{t+1} x_{n_t}
W'_{xh} W'_{hh}
h'_{t-1} h'_t h'_{t+1} h'_{n_t}
W'_{hy}
y_{t-1} y_t y_{t+1} y_{n_t}
W_{hy} W_{hh}
h_{t-1} h_t h_{t+1} h_{n_t}
W_{xh}
x_{t-1} x_t x_{t+1} x_{n_t}
time
$t-1$ t $t+1$ n_t

前向计算



$$
\begin{aligned}
u_t &= W_{xh}x_t + W_{hh}h_{t-1} \\
u'_t &= W'_{xh}x_t + W'_{hh}h_{t+1} \\
h_t &= f(u_t) \\
h'_t &= f(u'_t) \\
v_t &= W_{hy}h_t \\
v'_t &= W'_{hy}h_t \\
s_t &= v_t + v'_t \\
y_t &= g(s_t)
\end{aligned}
$$

$$
\begin{aligned}
\nabla_{W_{hy}} L_t &= \frac{\partial L_t}{\partial W_{hy}} = \frac{\partial L_t}{\partial v_t}\frac{\partial v_t}{\partial W_{hy}} = \delta_t^y \otimes h_t \\
\nabla_{W'_{hy}} L_t &= \frac{\partial L_t}{\partial W'_{hy}} = \frac{\partial L_t}{\partial v'_t}\frac{\partial v'_t}{\partial W'_{hy}} = \delta_t'^y \otimes h'_t \\
\nabla_{W_{hh}} L_t &= \frac{\partial L_t}{\partial W_{hh}} = \frac{\partial L_t}{\partial u_t}\frac{\partial u_t}{\partial W_{hh}} = \delta_t^h \otimes h_{t-1} \\
\nabla_{W'_{hh}} L_t &= \frac{\partial L_t}{\partial W'_{hh}} = \frac{\partial L_t}{\partial u'_t}\frac{\partial u'_t}{\partial W'_{hh}} = \delta_t'^h \otimes h'_{t+1} \\
\nabla_{W_{xh}} L_t &= \frac{\partial L_t}{\partial W_{xy}} = \frac{\partial L_t}{\partial u_t}\frac{\partial u_t}{\partial W_{xh}} = \delta_t^h \otimes x_t \\
\nabla_{W'_{xh}} L_t &= \frac{\partial L_t}{\partial W'_{xy}} = \frac{\partial L_t}{\partial u'_t}\frac{\partial u'_t}{\partial W'_{xh}} = \delta_t'^h \otimes x_t
\end{aligned}
$$

反向传播：BPTT

$t=1$

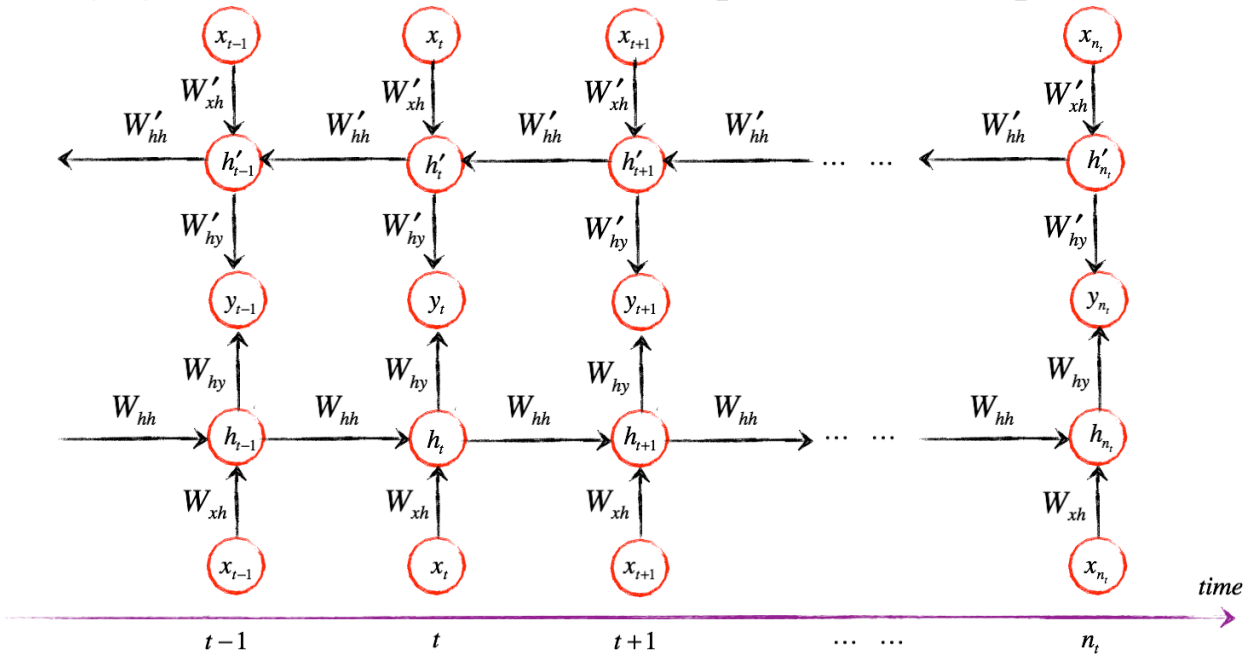
$$\delta_1^y=\left(\nabla_{y_1}L_1\right)\odot g'\left(s_1\right)$$

$$\delta_1'^h=\left(W_{hy}'\right)^T\delta_1^y\odot f'\left(u_1'\right)$$

$t=2,3,\ldots,n_t$

$$\delta_t^y=\left(\nabla_{y_t}L_t\right)\odot g'\left(s_t\right)$$

$$\delta_t'^h=\left[\left(W_{hy}'\right)^T\delta_t^y+\left(W_{hh}'\right)^T\delta_{t-1}'^h\right]\odot f'\left(u_t'\right)$$



$t=n_t-1,n_t-2,\ldots,1$

$$\delta_t^y=\left(\nabla_{y_t}L_t\right)\odot g'\left(s_t\right)$$

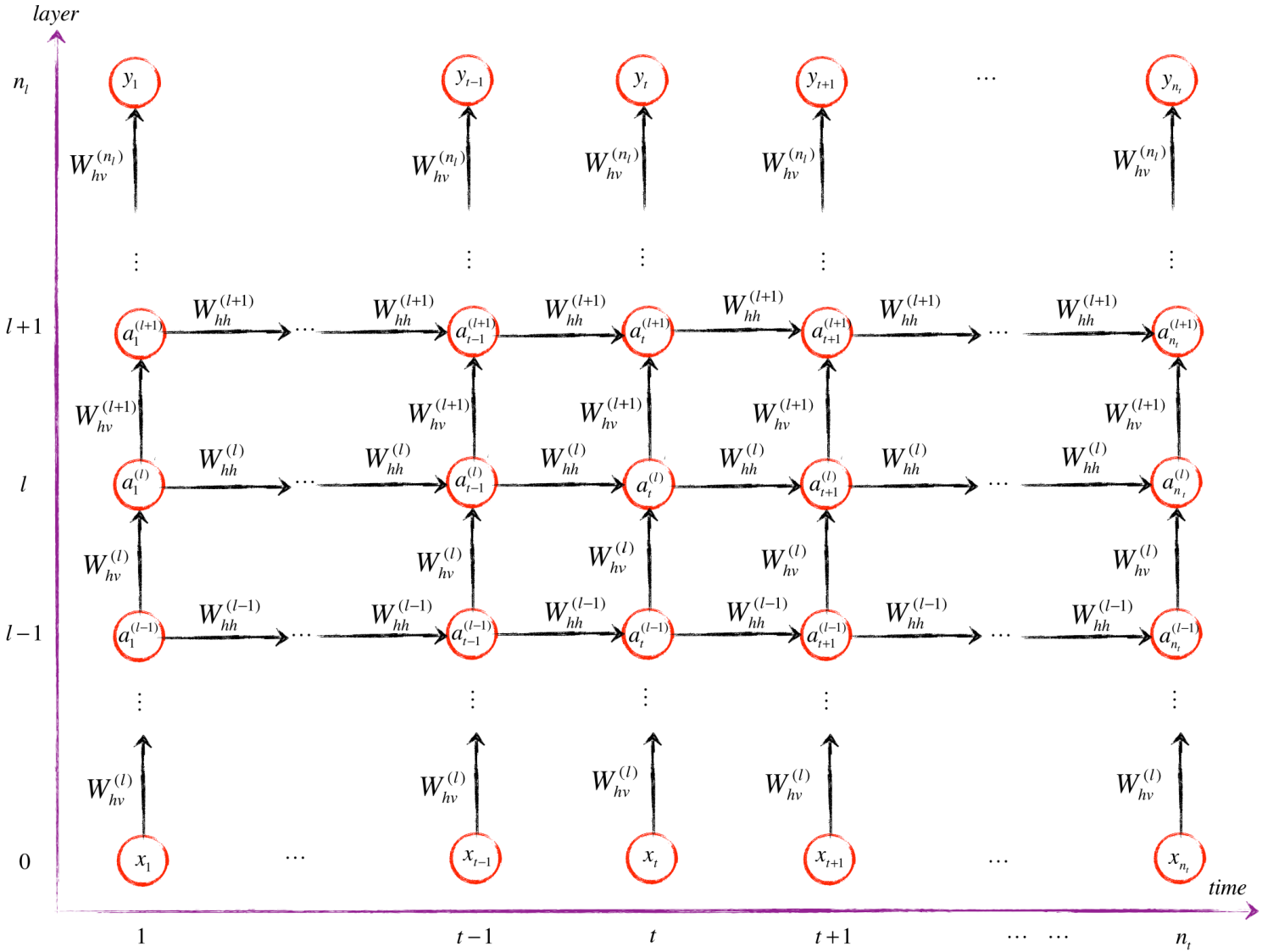
$$\delta_t^h=\left[\left(W_{hy}\right)^T\delta_t^y+\left(W_{hh}\right)^T\delta_{t+1}^h\right]\odot f'\left(u_t\right)$$

$t=n_t$

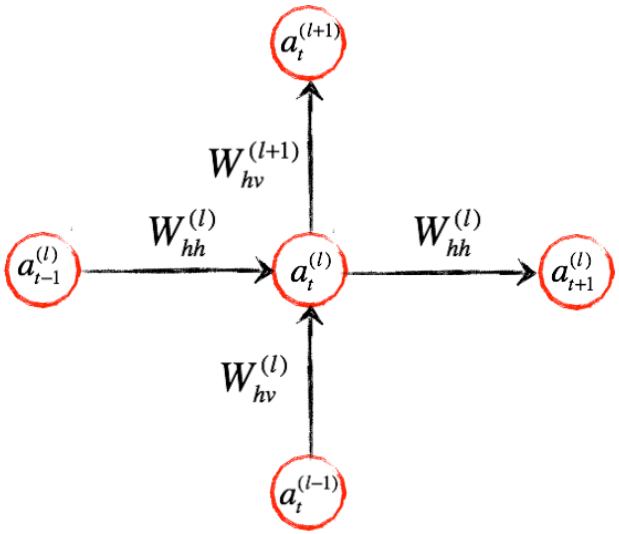
$$\delta_{n_t}^y=\left(\nabla_{y_{n_t}}L_{n_t}\right)\odot g'\left(s_{n_t}\right)$$

$$\delta_{n_t}^h=\left(W_{hy}\right)^T\delta_{n_t}^y\odot f'\left(u_{n_t}\right)$$

网络架构



前向计算



$$u_t^{(l)} = W_{hv}^{(l)} a_t^{(l-1)} + W_{hh}^{(l)} a_{t-1}^{(l)}$$

$$a_t^{(l)} = f\left(u_t^{(l)}\right)$$

$$a_0^{(l)} = \vec{0}$$

$$l = 1,2,\ldots,n_t$$

$$a_t^{(0)} = x_t$$

$$t = 1,2,\ldots,n_t$$

$$y_t = a_t^{(n_l)}$$

$$t = 1,2,\ldots,n_t$$

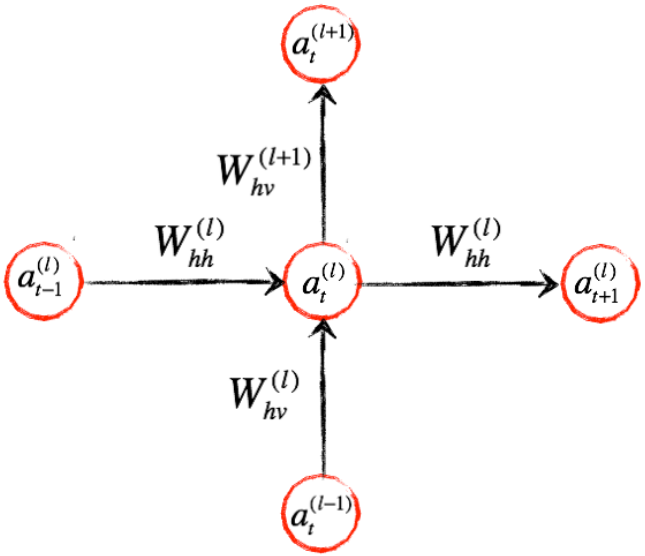
$$\nabla_{W_{hh}^{(l)}} L_t = \frac{\partial L_t}{W_{hh}^{(l)}} = \frac{\partial L_t}{u_t^{(l)}} \frac{u_t^{(l)}}{W_{hh}^{(l)}} = \delta_t^{(l)} \otimes a_{t-1}^{(l)}$$

$$\nabla_{W_{hv}^{(l)}} L_t = \frac{\partial L_t}{W_{hv}^{(l)}} = \frac{\partial L_t}{u_t^{(l)}} \frac{u_t^{(l)}}{W_{hv}^{(l)}} = \delta_t^{(l)} \otimes a_t^{(l-1)}$$

$$\nabla_{W_{hh}^{(l)}} L = \sum_{t=1}^{n_t} \nabla_{W_{hh}^{(l)}} L_t$$

$$\nabla_{W_{hv}^{(l)}} L = \sum_{t=1}^{n_t} \nabla_{W_{hv}^{(l)}} L_t$$

反向传播：BPTT



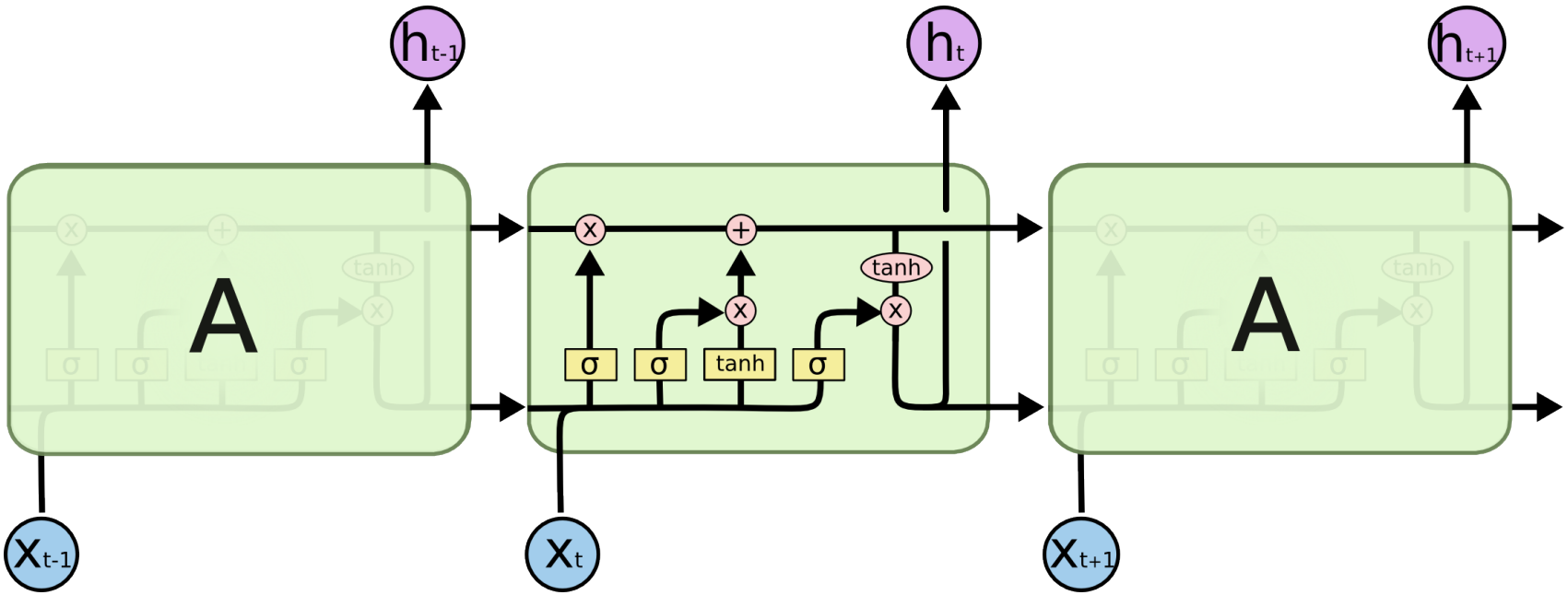
$$\delta_t^{(l)} = \vec{0} \qquad \delta_t^{(n_l)} = \nabla_{y_t} L_t \odot f'\left(u_t^{(n_l)}\right)$$
$$t > n_t \qquad l = n_l$$

$$\delta_t^{(l)} = \left[\left(W_{hv}^{(l+1)}\right)^T \delta_t^{(l+1)} + \left(W_{hh}^{(l)}\right)^T \delta_{t+1}^{(l)}\right] \odot f'\left(u_t^{(l)}\right)$$
$$l = n_l - 1, n_l - 2, \ldots, 2, 1$$
$$t = n_t, n_t - 1, \ldots, 2, 1$$

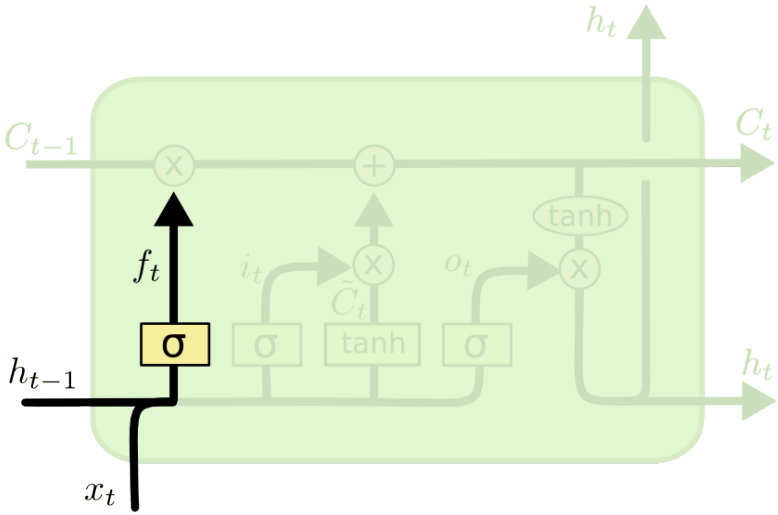
长短时记忆网络

1. 网络架构
2. 网络变种

网络架构

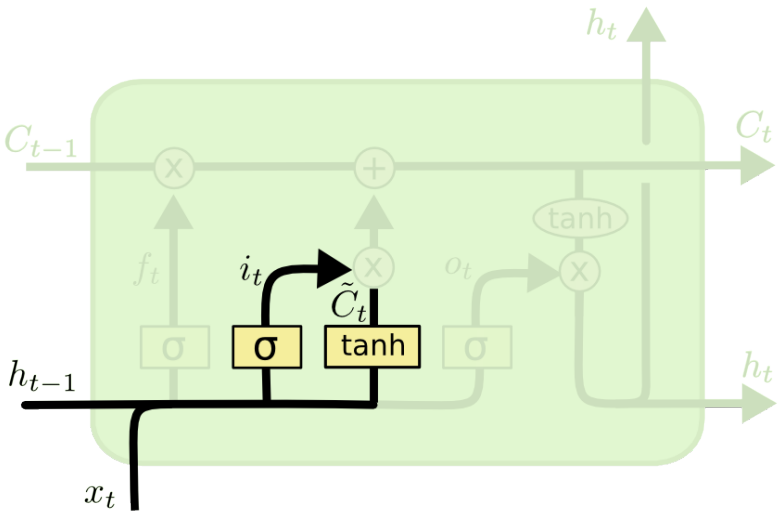


遗忘门



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

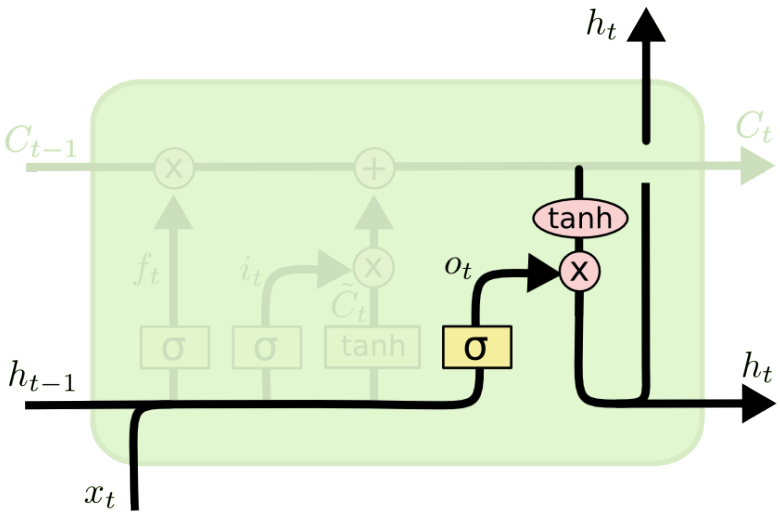
输入门



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

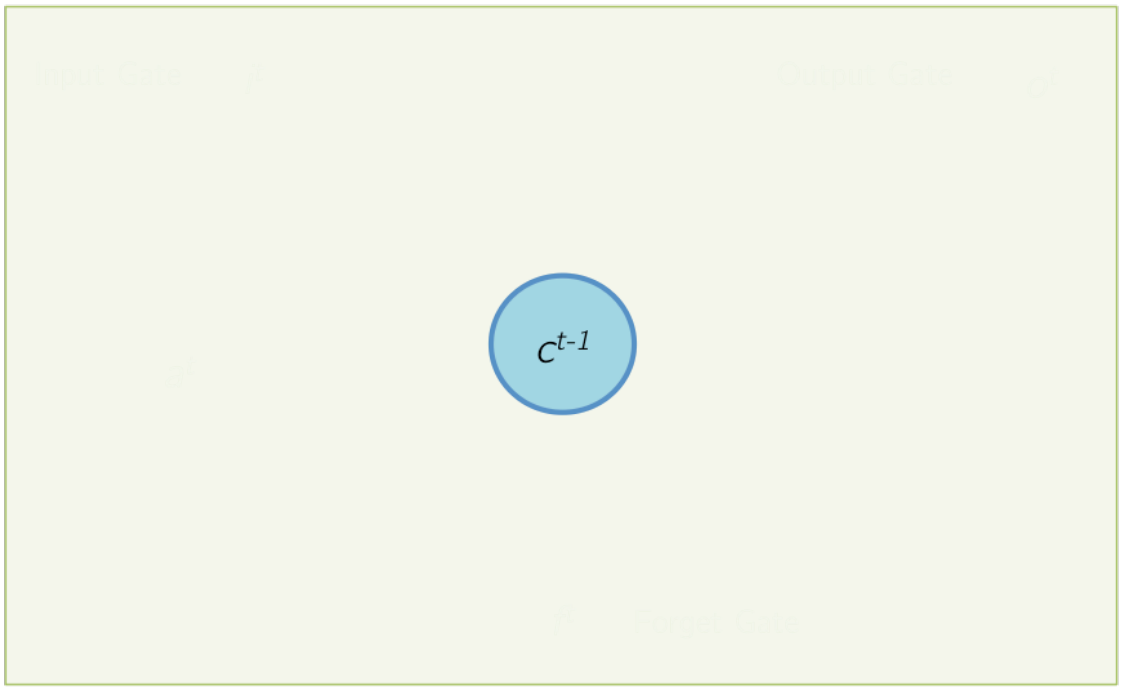
输出门



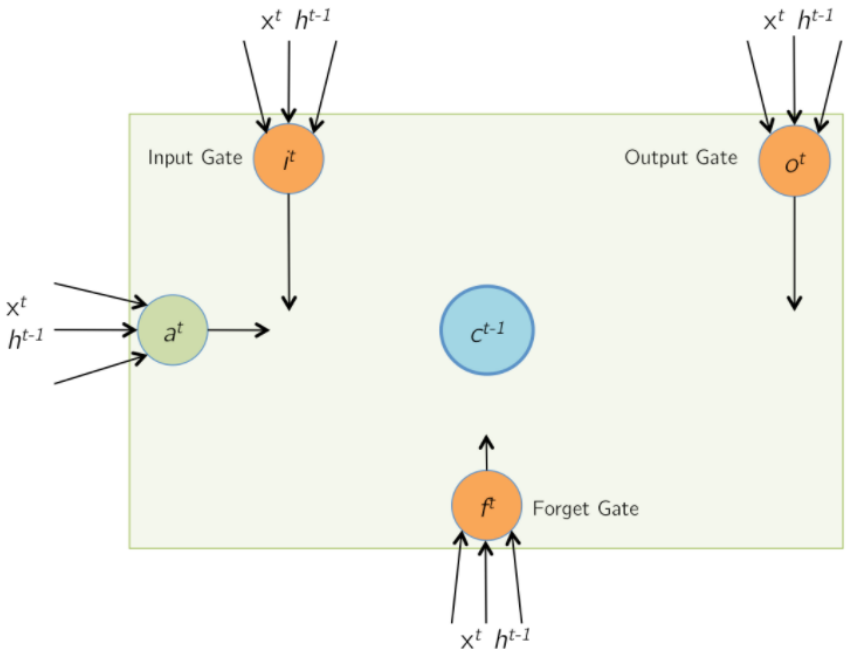
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

初始状态: $t-1$ 时刻



前向传播：输入与门计算

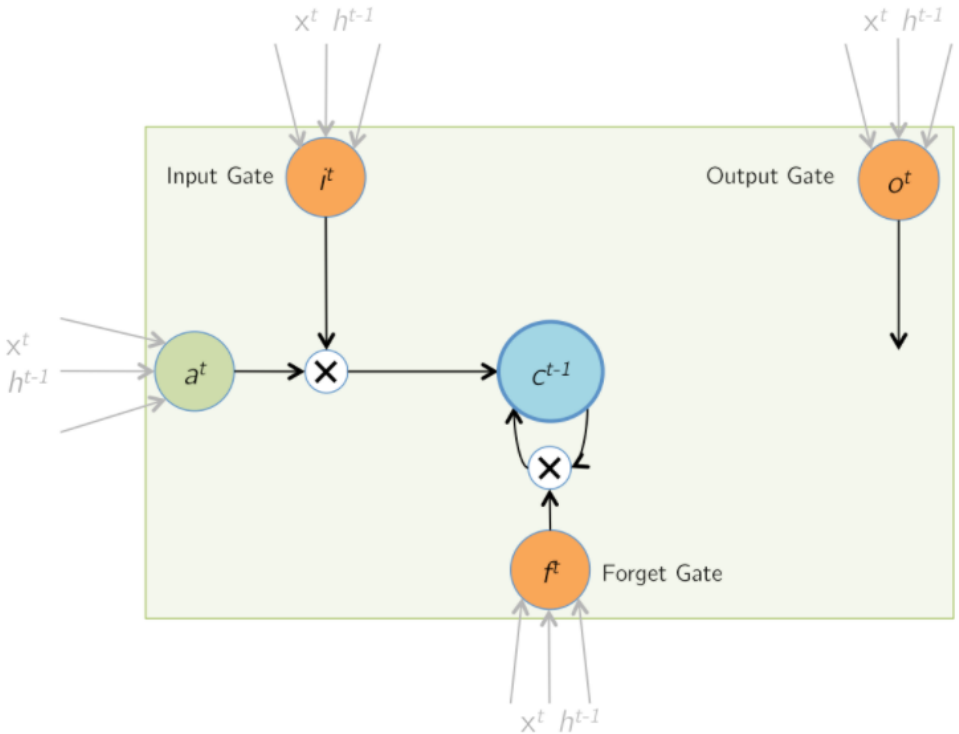


$$
\begin{aligned}
a^t &= \tanh(W_c x^t + U_c h^{t-1}) = \tanh(\hat{a}^t) \\
i^t &= \sigma(W_i x^t + U_i h^{t-1}) = \sigma(\hat{i}^t) \\
f^t &= \sigma(W_f x^t + U_f h^{t-1}) = \sigma(\hat{f}^t) \\
o^t &= \sigma(W_o x^t + U_o h^{t-1}) = \sigma(\hat{o}^t)
\end{aligned}
$$

Ignoring the non-linearities,

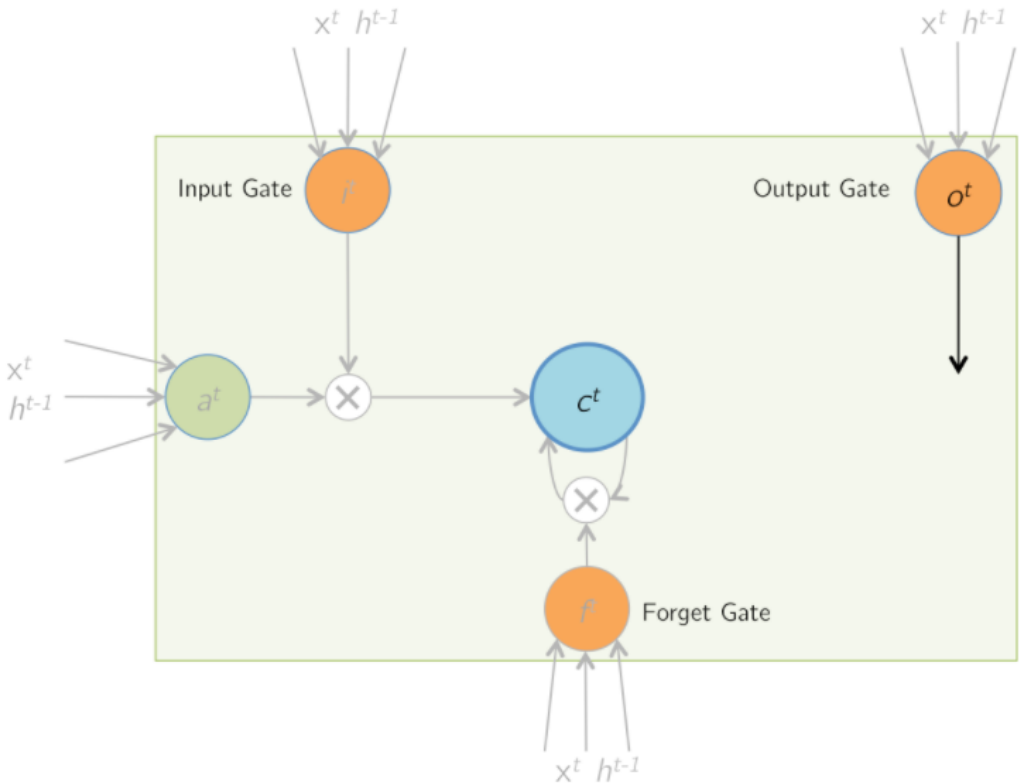
$$
z^t = \begin{bmatrix} \hat{a}^t \\ \hat{i}^t \\ \hat{f}^t \\ \hat{o}^t \end{bmatrix} = \begin{bmatrix} W^c & U^c \\ W^i & U^i \\ W^f & U^f \\ W^o & U^o \end{bmatrix} \times \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix} = W \times I^t
$$

前向传播：状态更新



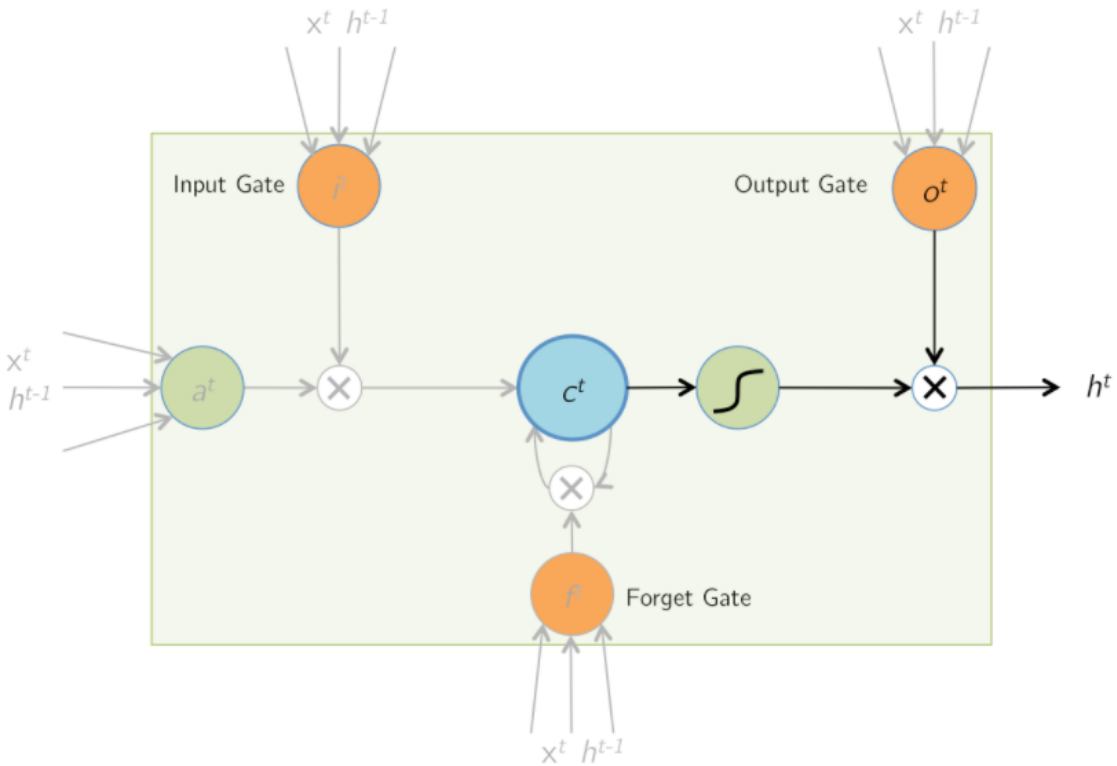
$$c^t = i^t \odot a^t + f^t \odot c^{t-1}$$

前向传播：状态更新



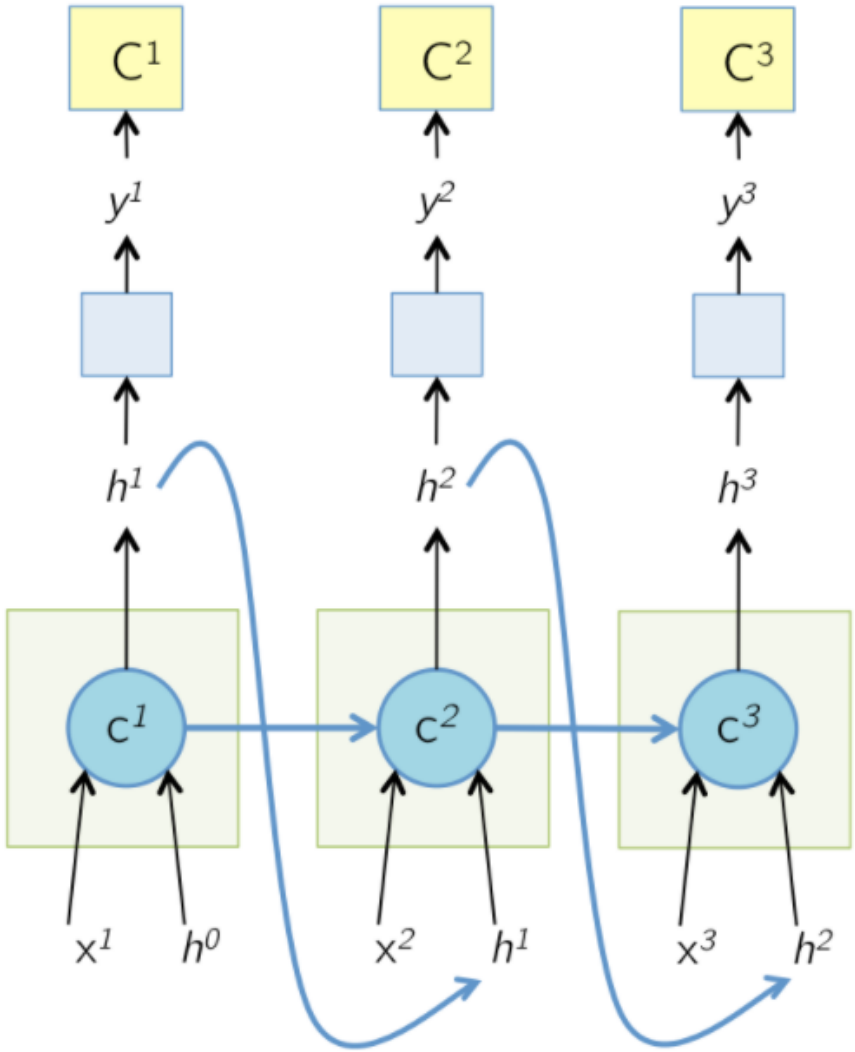
$$c^{t-1} \rightarrow c^t$$

前向传播：输出

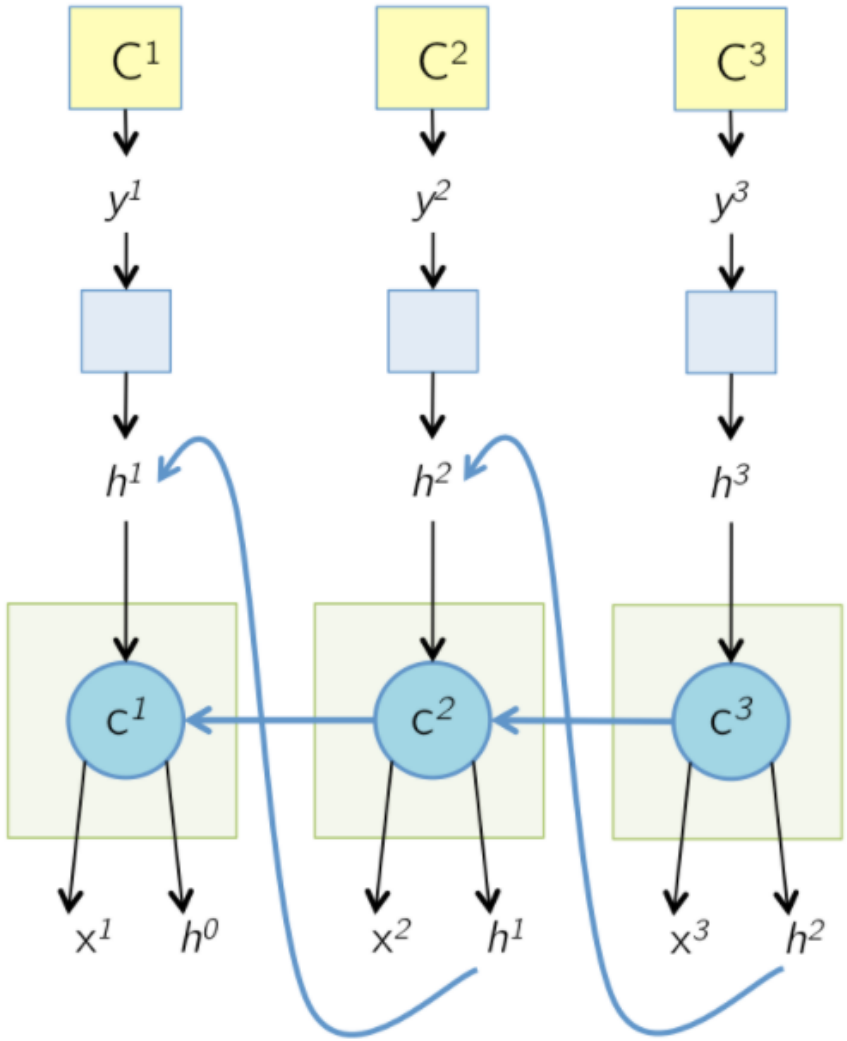


$$h^t = o^t \odot \tanh(c^t)$$

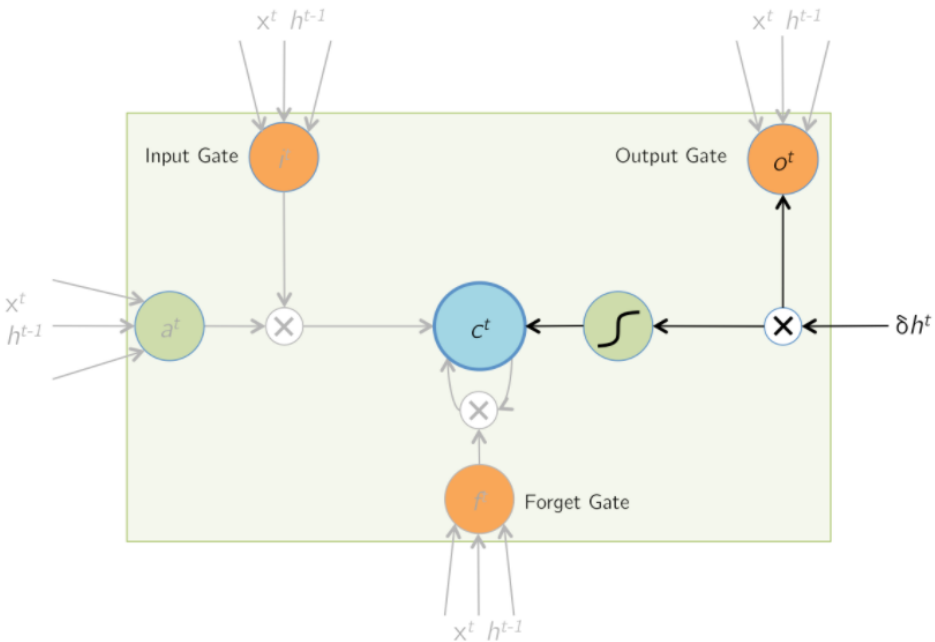
前向传播



反向传播



反向传播



Forward Pass: $h^t = o^t \odot \tanh(c^t)$

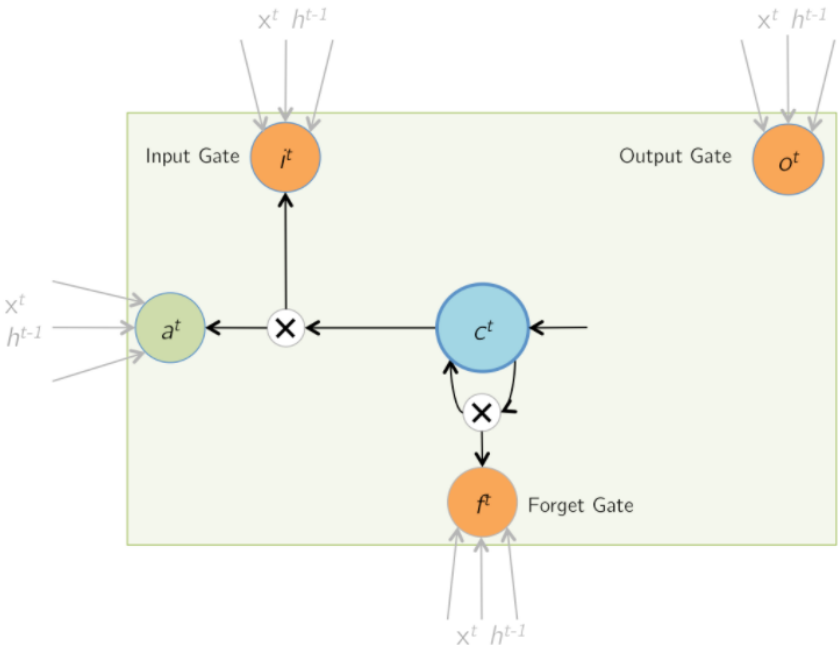
Given $\delta h^t = \frac{\partial E}{\partial h^t}$, find $\delta o^t, \delta c^t$

$$\begin{aligned}\frac{\partial E}{\partial o_i^t} &= \frac{\partial E}{\partial h_i^t} \cdot \frac{\partial h_i^t}{\partial o_i^t} \\ &= \delta h_i^t \cdot \tanh(c_i^t) \\ \therefore \delta o^t &= \delta h^t \odot \tanh(c^t)\end{aligned}$$

$$\begin{aligned}\frac{\partial E}{\partial c_i^t} &= \frac{\partial E}{\partial h_i^t} \cdot \frac{\partial h_i^t}{\partial c_i^t} \\ &= \delta h_i^t \cdot o_i^t \cdot (1 - \tanh^2(c_i^t)) \\ \therefore \delta c^t &+= \delta h^t \odot o^t \odot (1 - \tanh^2(c^t))\end{aligned}$$

Note that the $+=$ above is so that this gradient is added to gradient from time step $(t+1)$ (calculated on next slide, refer to the gradient accumulation mentioned in the previous slide)

反向传播



Forward Pass: $c^t = i^t \odot a^t + f^t \odot c^{t-1}$

Given $\delta c^t = \frac{\partial E}{\partial c^t}$, find $\delta i^t, \delta a^t, \delta f^t, \delta c^{t-1}$

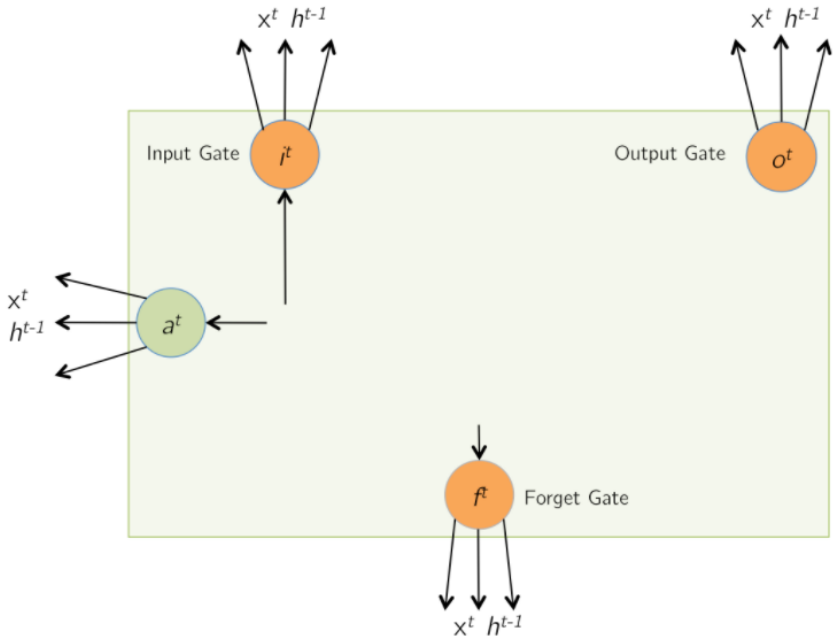
$$\frac{\partial E}{\partial i_i^t} = \frac{\partial E}{\partial c_i^t} \cdot \frac{\partial c_i^t}{\partial i_i^t} = \delta c_i^t \cdot a_i^t \quad \therefore \delta i^t = \delta c^t \odot a^t$$

$$\frac{\partial E}{\partial f_i^t} = \frac{\partial E}{\partial c_i^t} \cdot \frac{\partial c_i^t}{\partial f_i^t} = \delta c_i^t \cdot c_i^{t-1} \quad \therefore \delta f^t = \delta c^t \odot c^{t-1}$$

$$\frac{\partial E}{\partial a_i^t} = \frac{\partial E}{\partial c_i^t} \cdot \frac{\partial c_i^t}{\partial a_i^t} = \delta c_i^t \cdot i_i^t \quad \therefore \delta a^t = \delta c^t \odot i^t$$

$$\frac{\partial E}{\partial c_i^{t-1}} = \frac{\partial E}{\partial c_i^t} \cdot \frac{\partial c_i^t}{\partial c_i^{t-1}} = \delta c_i^t \cdot f_i^t \quad \therefore \delta c^{t-1} = \delta c^t \odot f^t$$

反向传播



Forward Pass: $z^t = W \times I^t$

Given δz^t, find $\delta W^t, \delta h^{t-1}$

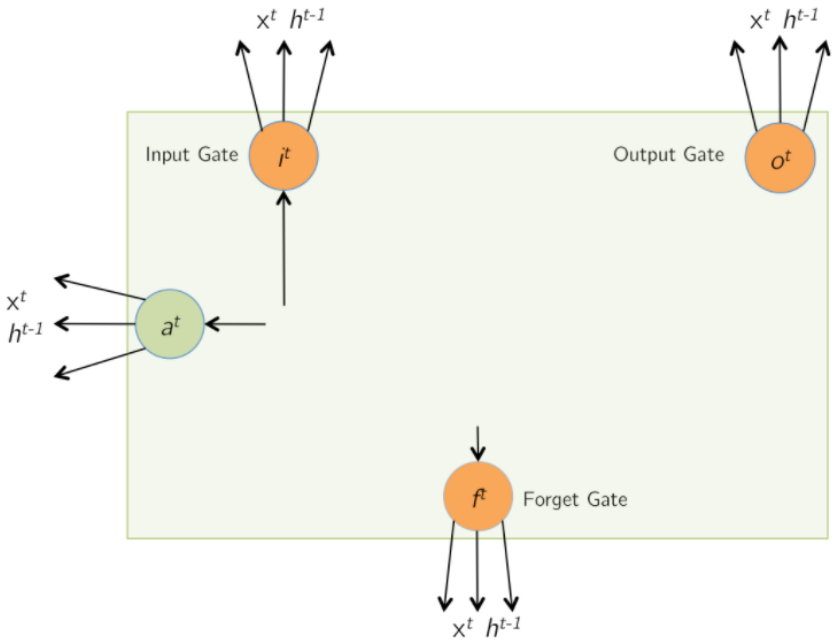
$$\delta I^t = W^T \times \delta z^t$$

$$\text{As } I^t = \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix},$$

δh^{t-1} can be retrieved from δI^t

$$\delta W^t = \delta z^t \times (I^t)^T$$

反向传播



Forward Pass: $z^t = W \times I^t$

Given δz^t, find $\delta W^t, \delta h^{t-1}$

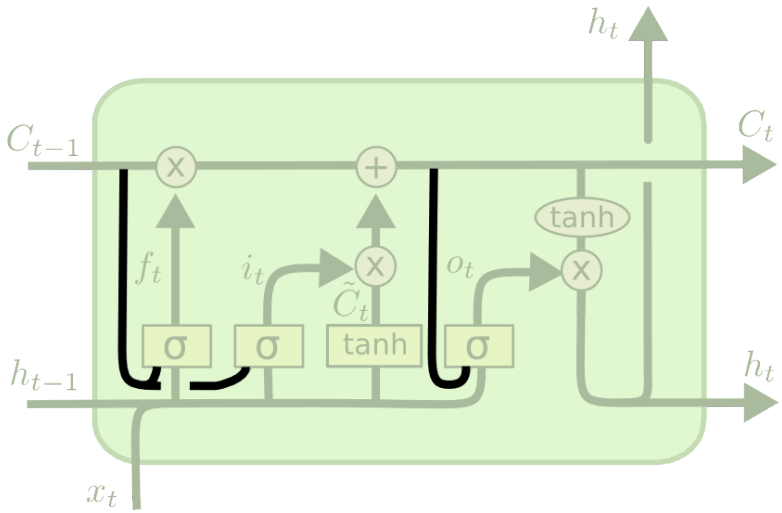
$$\delta I^t = W^T \times \delta z^t$$

$$\text{As } I^t = \begin{bmatrix} x^t \\ h^{t-1} \end{bmatrix},$$

δh^{t-1} can be retrieved from δI^t

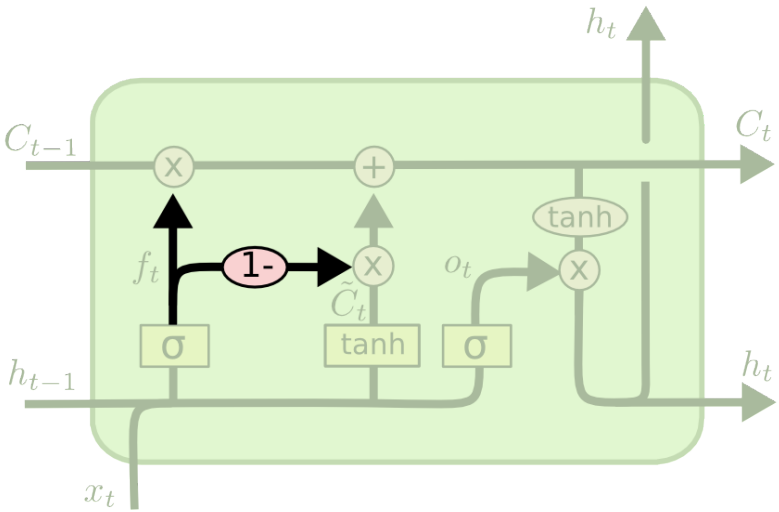
$$\delta W^t = \delta z^t \times (I^t)^T$$

变种 1



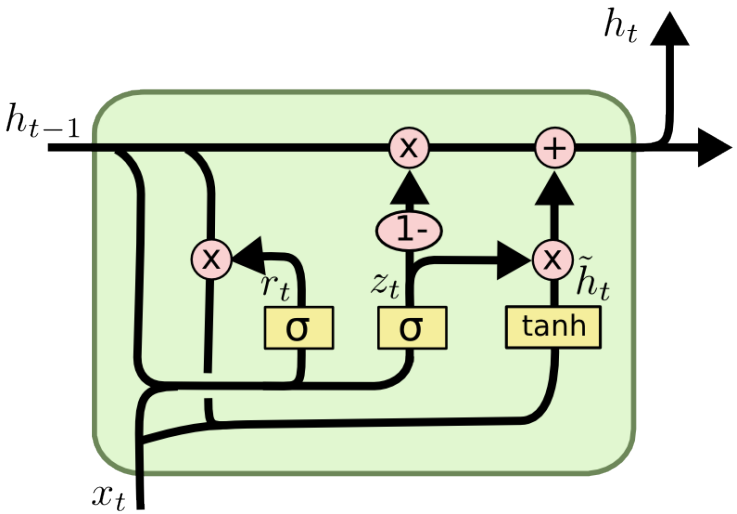
$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] + b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] + b_o\right)$$

变种 2



$$C_t = f_t * C_{t-1} + (1 - \boldsymbol{f_t}) * \tilde{C}_t$$

变种 3：GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

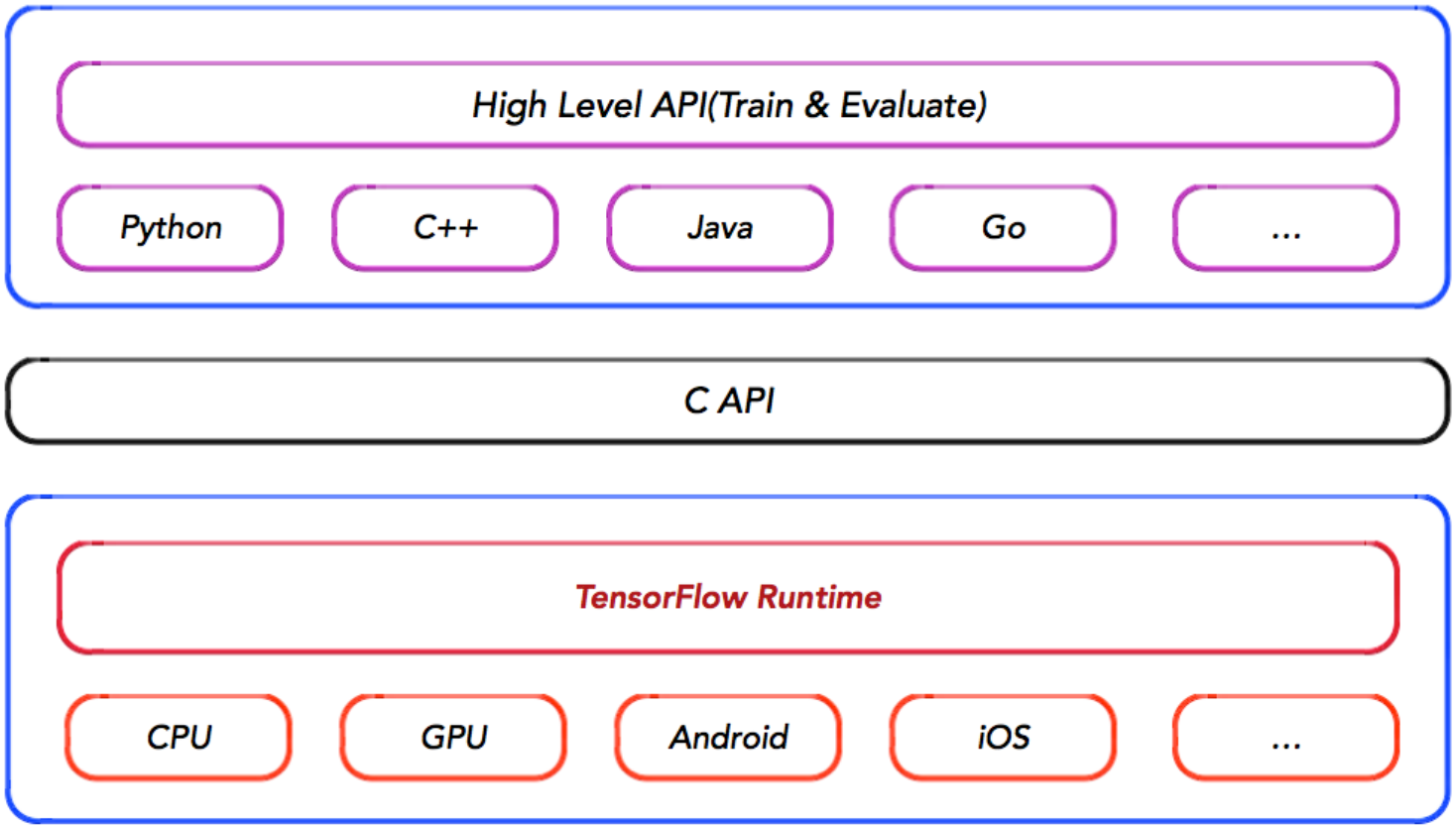
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

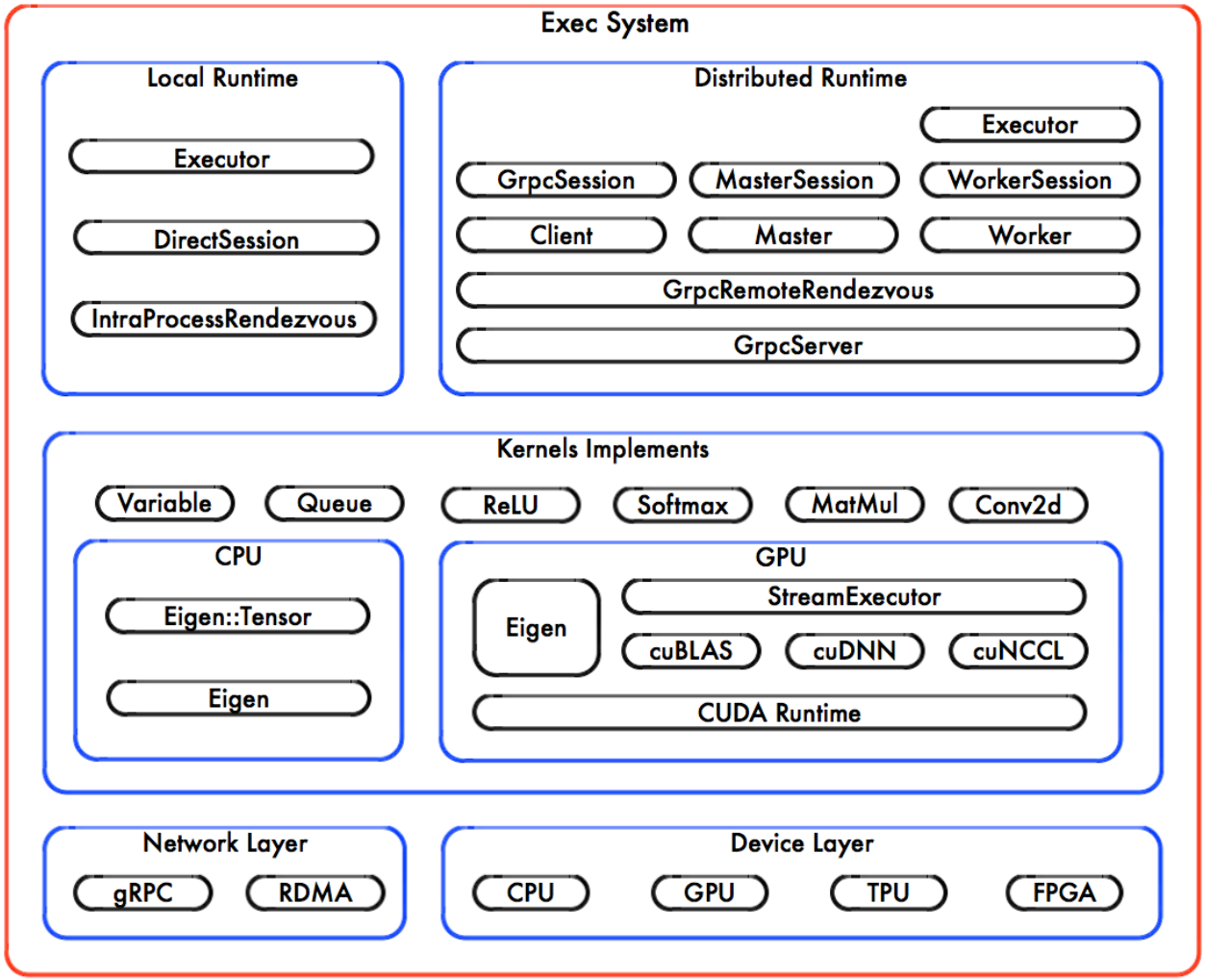
TensorFlow 实现

1. 系统架构
2. 自动微分

架构


High Level API(Train & Evaluate)
Python
C++
Java
Go
...
C API
TensorFlow Runtime
CPU
GPU
Android
iOS
...

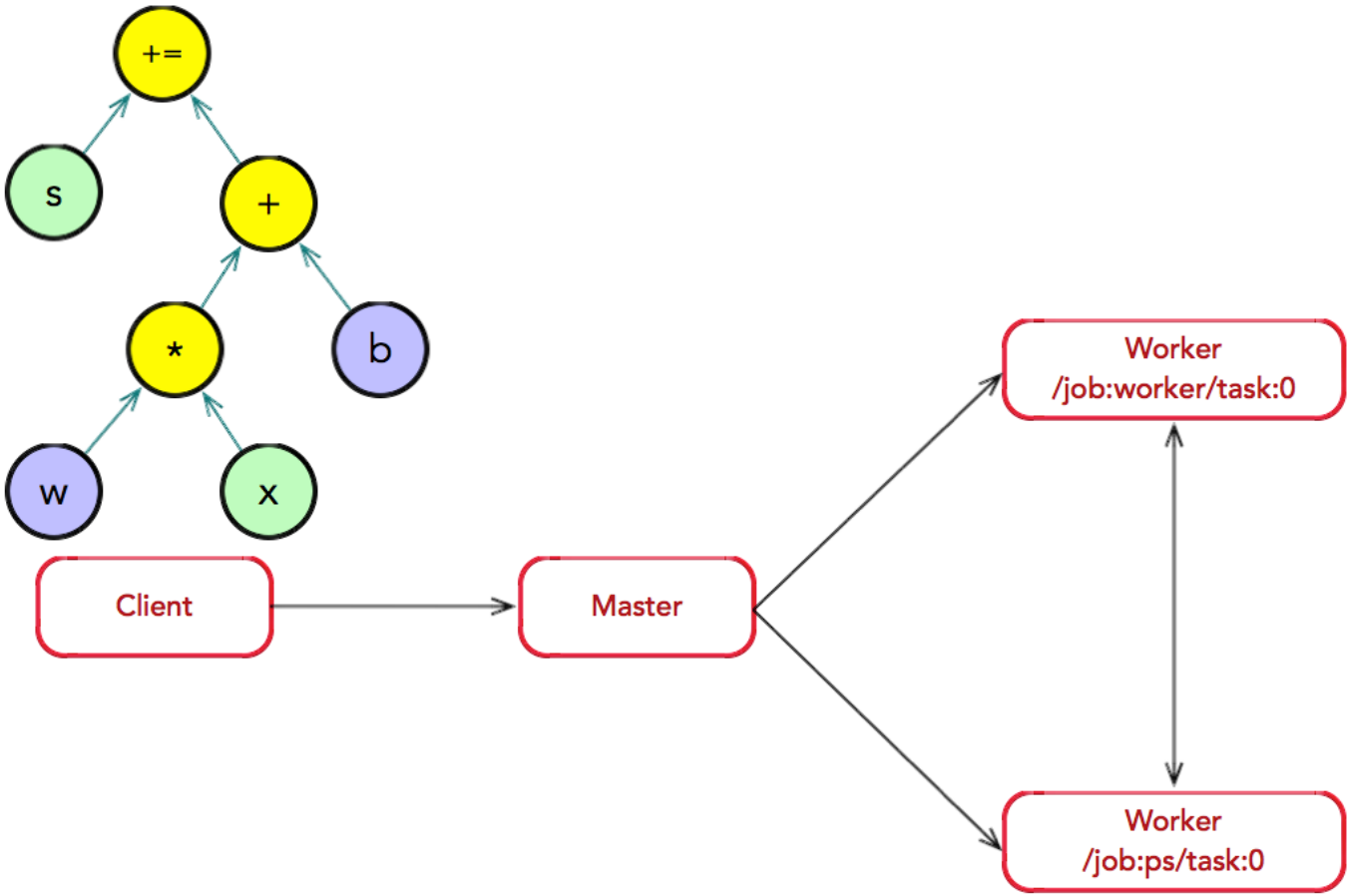
运行时


Exec System
Local Runtime
Executor
DirectSession
IntraProcessRendezvous
Distributed Runtime
Executor
GrpcSession
MasterSession
WorkerSession
Client
Master
Worker
GrpcRemoteRendezvous
GrpcServer
Kernels Implements
Variable
Queue
ReLU
Softmax
MatMul
Conv2d
CPU
Eigen::Tensor
Eigen
GPU
Eigen
StreamExecutor
cuBLAS
cuDNN
cuNCCL
CUDA Runtime
Network Layer
gRPC
RDMA
Device Layer
CPU
GPU
TPU
FPGA

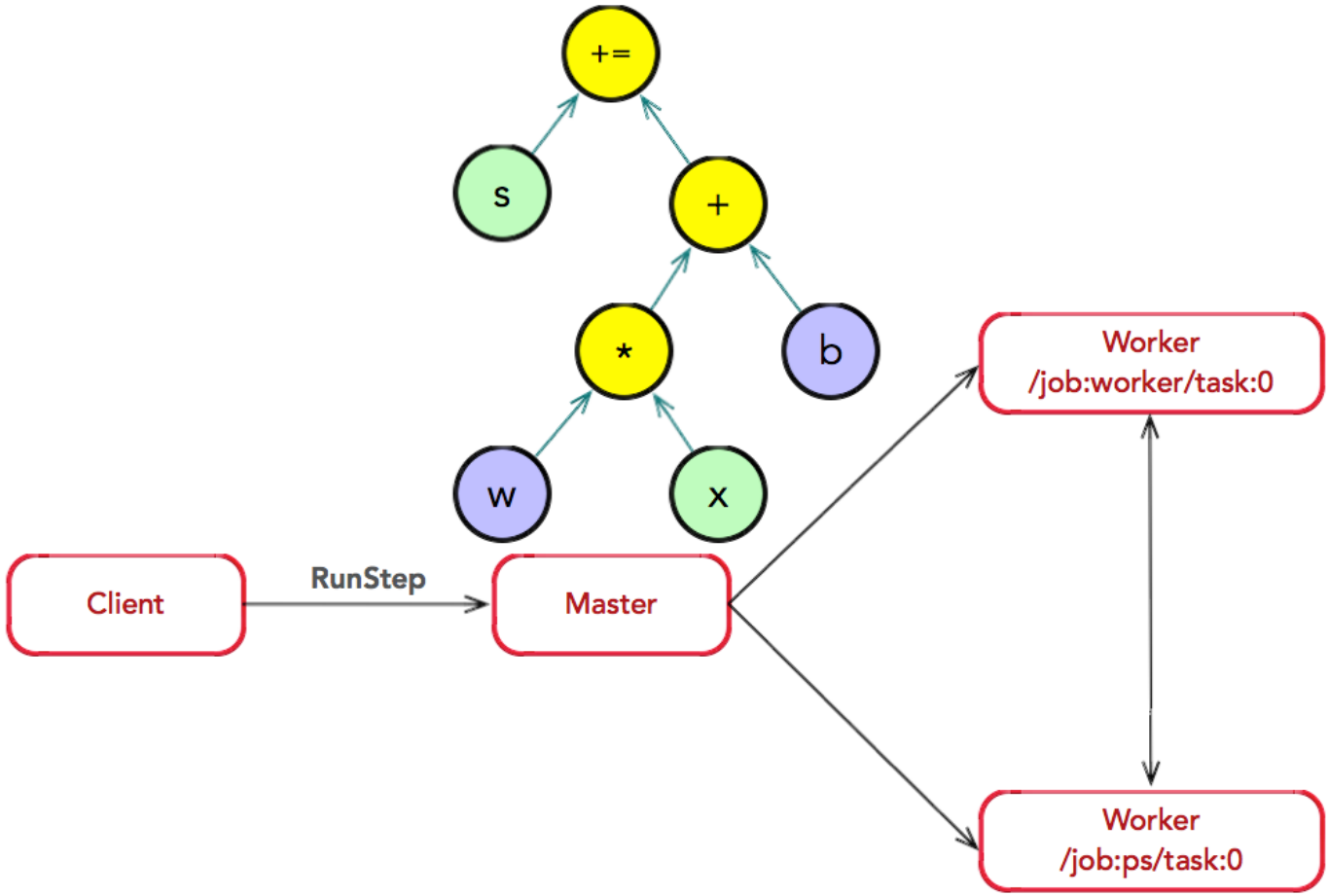
设计原则

- Deferred Execution
- Primitive OP
- Common Abstraction for Heterogeneous Accelerators

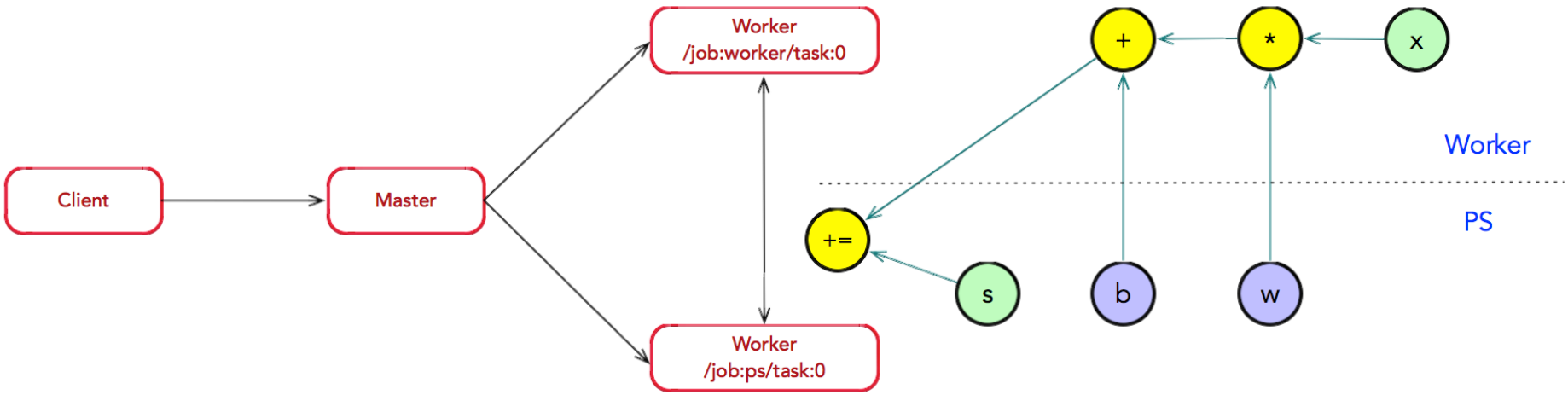
图构造


+=
s
+
*
b
w
x
Client
Master
Worker
/job:worker/task:0
Worker
/job:ps/task:0

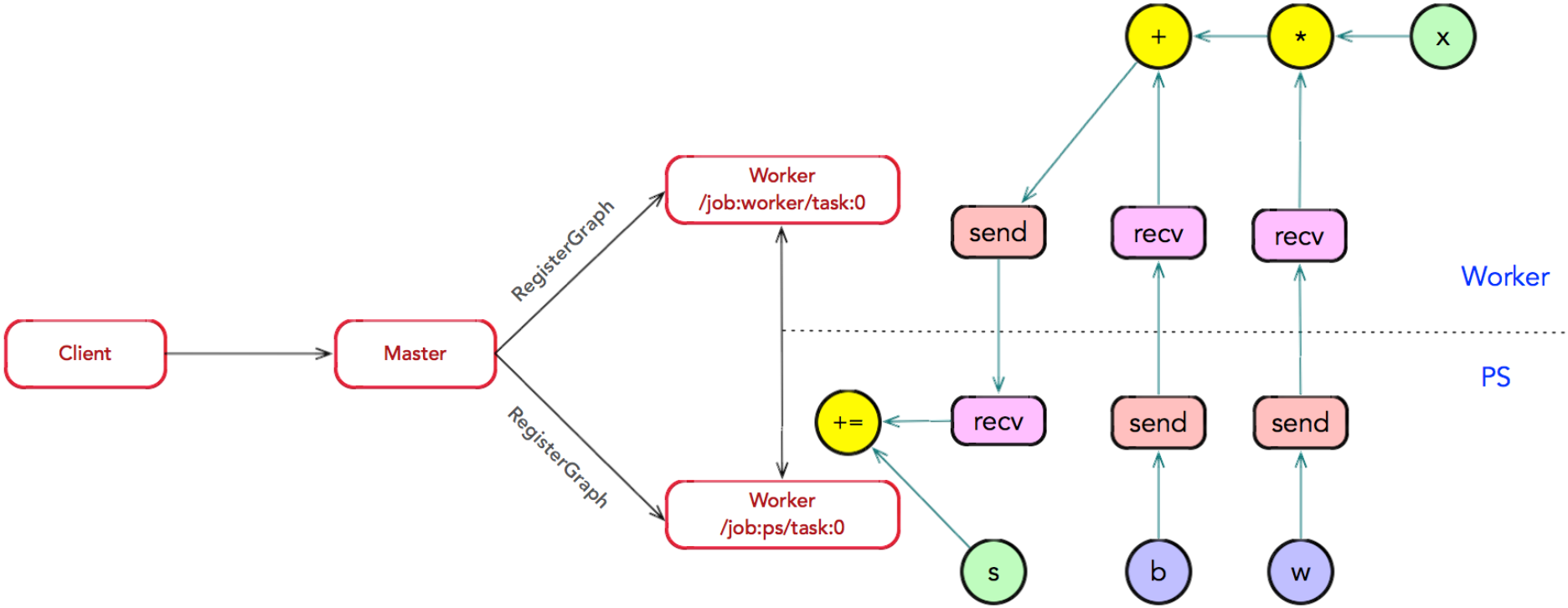
图执行


+=
s
+
*
b
w
x
Client
RunStep
Master
Worker
/job:worker/task:0
Worker
/job:ps/task:0

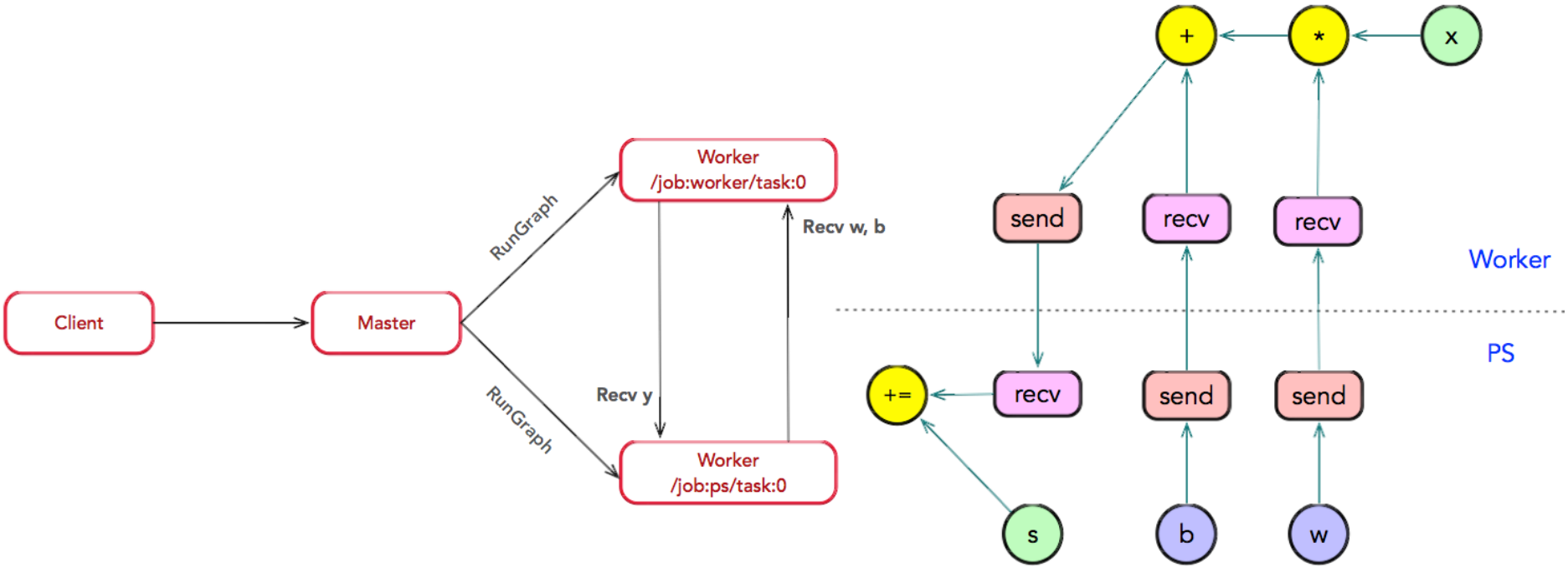
图分裂


Client
Master
Worker
/job:worker/task:0
Worker
/job:ps/task:0
+
*
x
+=
s
b
w
Worker
PS

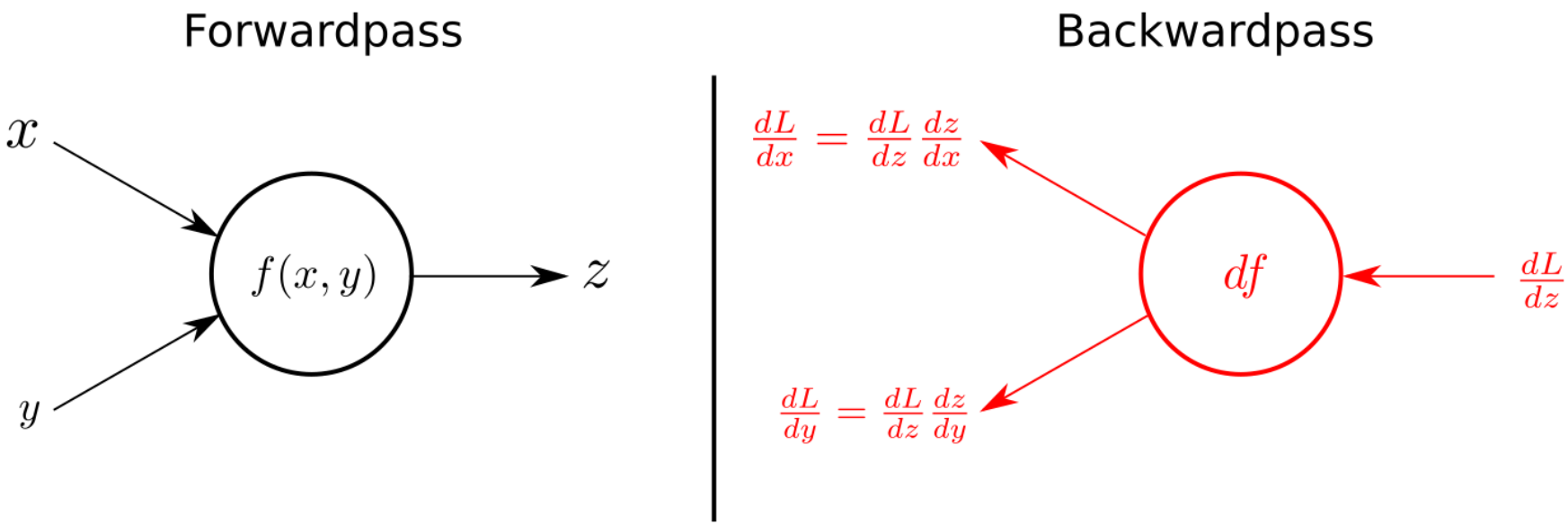
注册图


Client
Master
RegisterGraph
RegisterGraph
Worker
/job:worker/task:0
Worker
/job:ps/task:0
+
*
x
send
recv
recv
Worker
PS
+=
recv
send
send
s
b
w

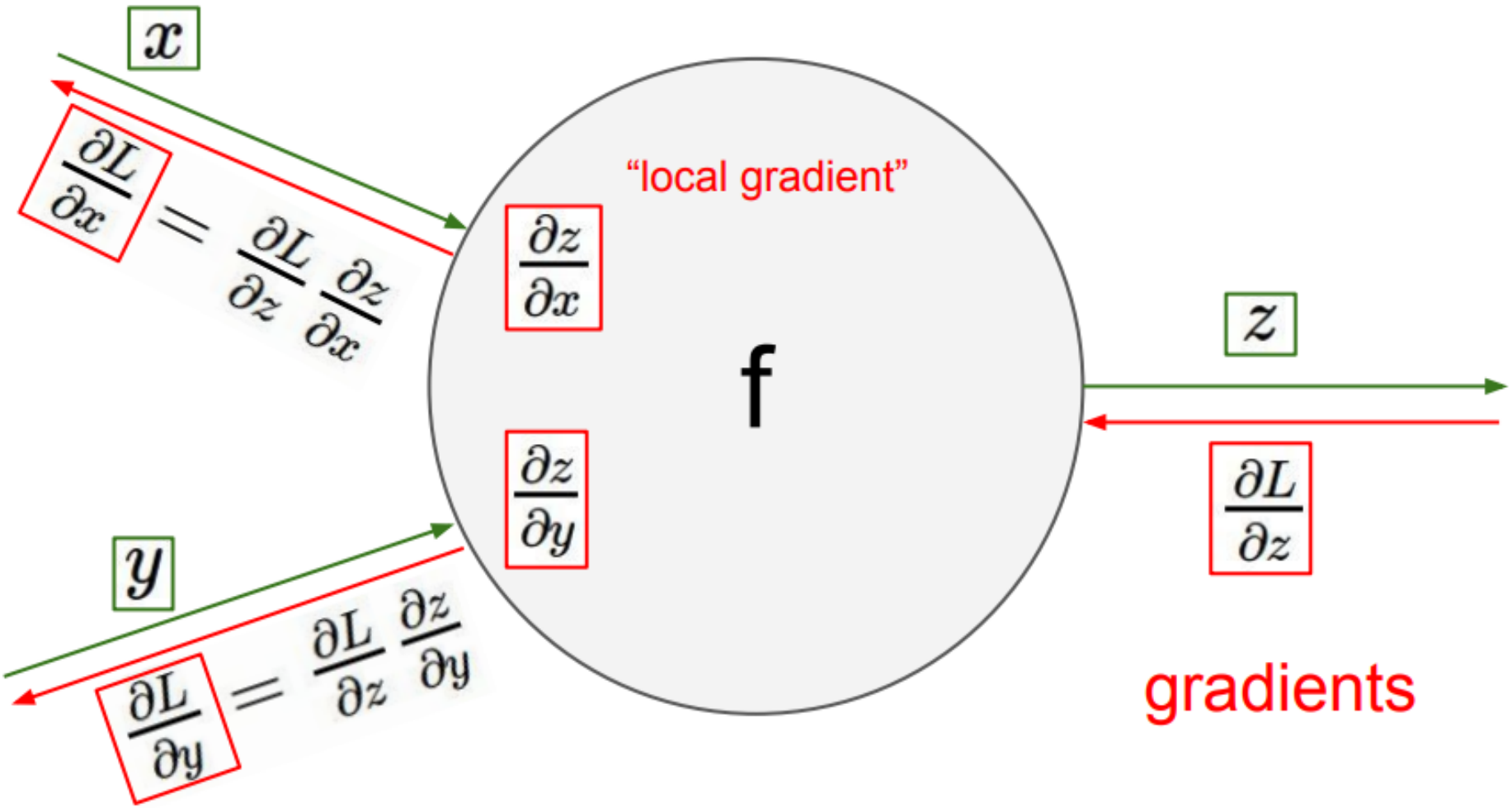
执行图


Client
Master
RunGraph
RunGraph
Worker
/job:worker/task:0
Worker
/job:ps/task:0
Recv w, b
Recv y
+
*
x
send
recv
recv
Worker
PS
+=
recv
send
send
s
b
w

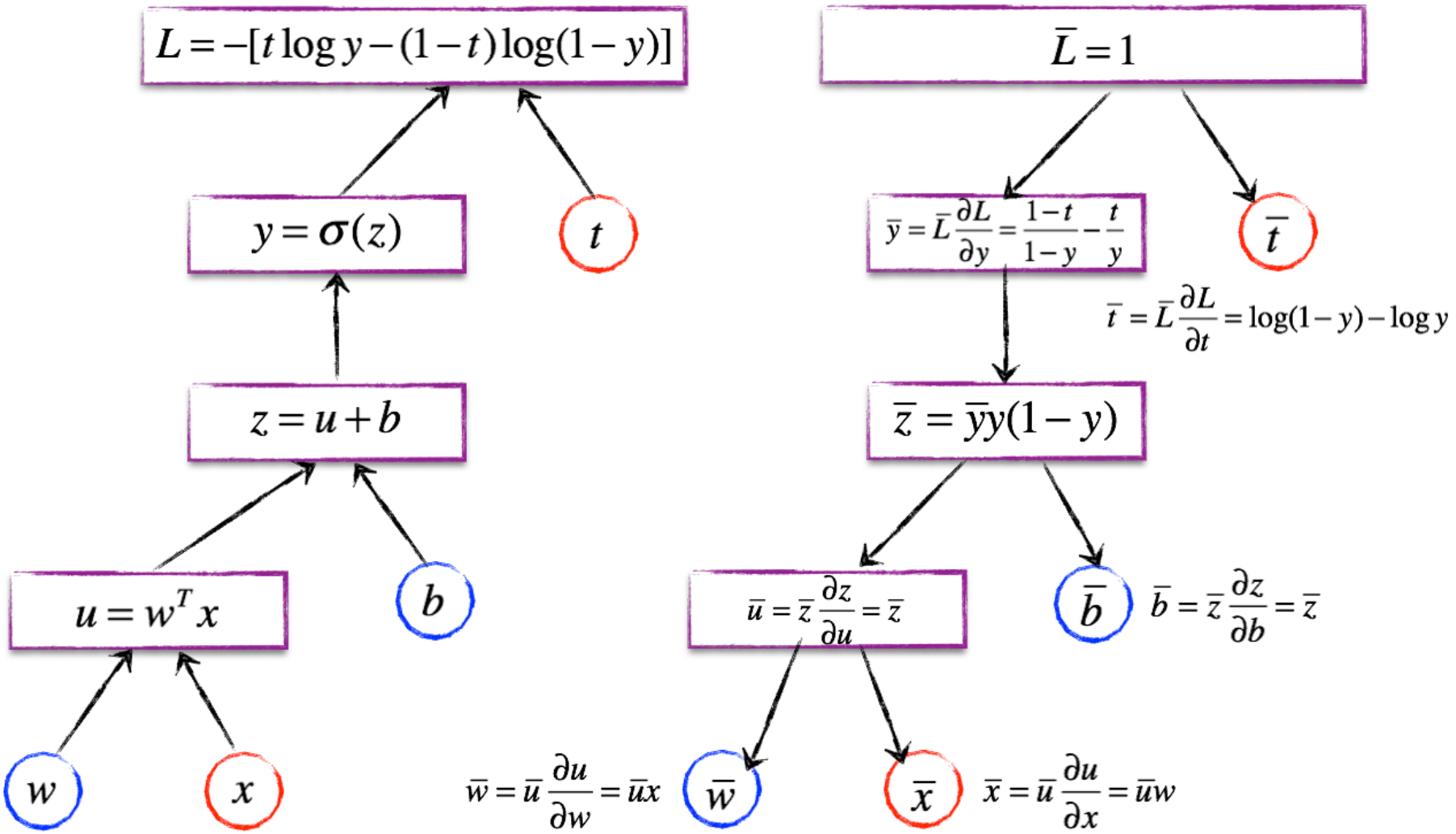
链式法则


Forwardpass
Backwardpass
x
y
$f(x,y)$
z
$\frac{dL}{dx} = \frac{dL}{dz}\frac{dz}{dx}$
$\frac{dL}{dy} = \frac{dL}{dz}\frac{dz}{dy}$
df
$\frac{dL}{dz}$

局部梯度


x
y
z
f
"local gradient"
$\frac{\partial z}{\partial x}$
$\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial x}$
$\frac{\partial L}{\partial y} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial y}$
$\frac{\partial L}{\partial z}$
gradients

例子：逻辑回归


$L=-[t\log y-(1-t)\log(1-y)]$
$y=\sigma(z)$
t
$z=u+b$
$u=w^Tx$
b
w
x
$\bar{L}=1$
$\bar{y}=\bar{L}\frac{\partial L}{\partial y}=\frac{1-t}{1-y}-\frac{t}{y}$
$\bar{t}$
$\bar{t}=\bar{L}\frac{\partial L}{\partial t}=\log(1-y)-\log y$
$\bar{z}=\bar{y}y(1-y)$
$\bar{u}=\bar{z}\frac{\partial z}{\partial u}=\bar{z}$
$\bar{b}$
$\bar{b}=\bar{z}\frac{\partial z}{\partial b}=\bar{z}$
$\bar{w}=\bar{u}\frac{\partial u}{\partial w}=\bar{u}x$
$\bar{w}$
$\bar{x}$
$\bar{x}=\bar{u}\frac{\partial u}{\partial x}=\bar{u}w$

优化器

```
class Optimizer(object):
  def minimize(self, loss, var_list=None, global_step=None):
    grads_and_vars = self.compute_gradients(
      loss, var_list=var_list)
    return self.apply_gradients(
      grads_and_vars,
      global_step=global_step)

  def compute_gradients(loss, var_list):
    grads = gradients(loss, var_list, grad)
    return list(zip(grads, var_list))

  def gradients(loss, var_list, grads=1):
    ops_and_grads = {}
    for op in reversed_graph(loss).topological_sort():
      grad = op.grad_fn(grad)
      ops_and_grads[op] = grad
    return [ops_and_grads.get(var) for var in var_list]
```

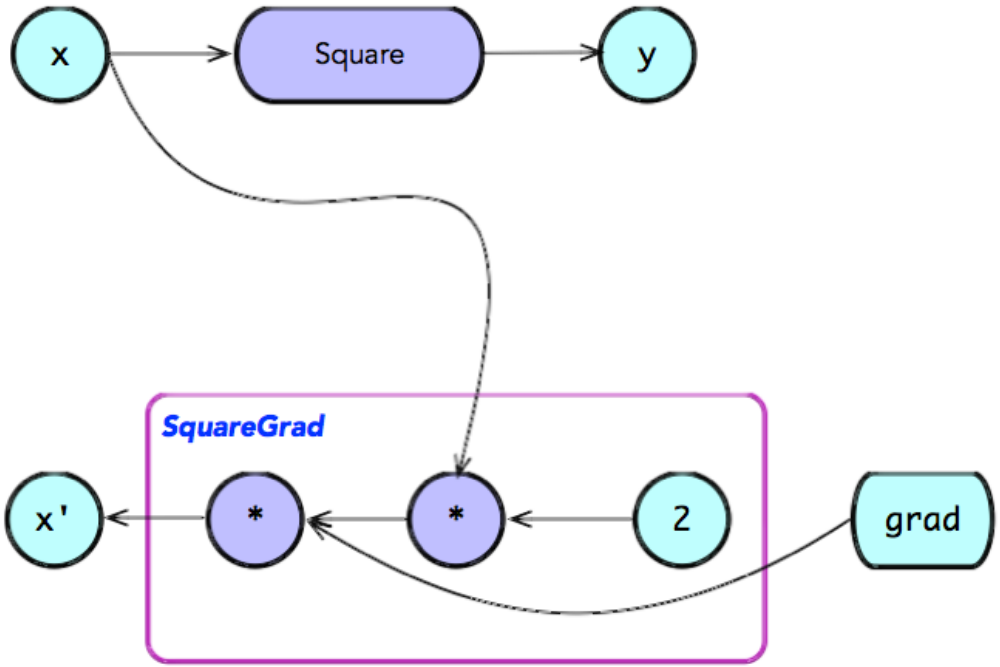
梯度函数

```
@ops.RegisterGradient("op_name")
def grad_func(op, grad):
  """construct gradient subgraph for an op type.
  Returns:
    A list of gradients, one per each input of op.
  """
  return cons_grad_subgraph(op, grad)
```

$$(y_1, y_2, ..., y_m) = f(x_1, x_2, ..., x_n)$$
$$(\partial L/\partial x_1, \partial L/\partial x_2, ..., \partial L/\partial x_n) = g\left(x_1, x_2, ..., x_n; \partial L/\partial y_1, \partial L/\partial y_2, ..., \partial L/\partial y_n\right)$$

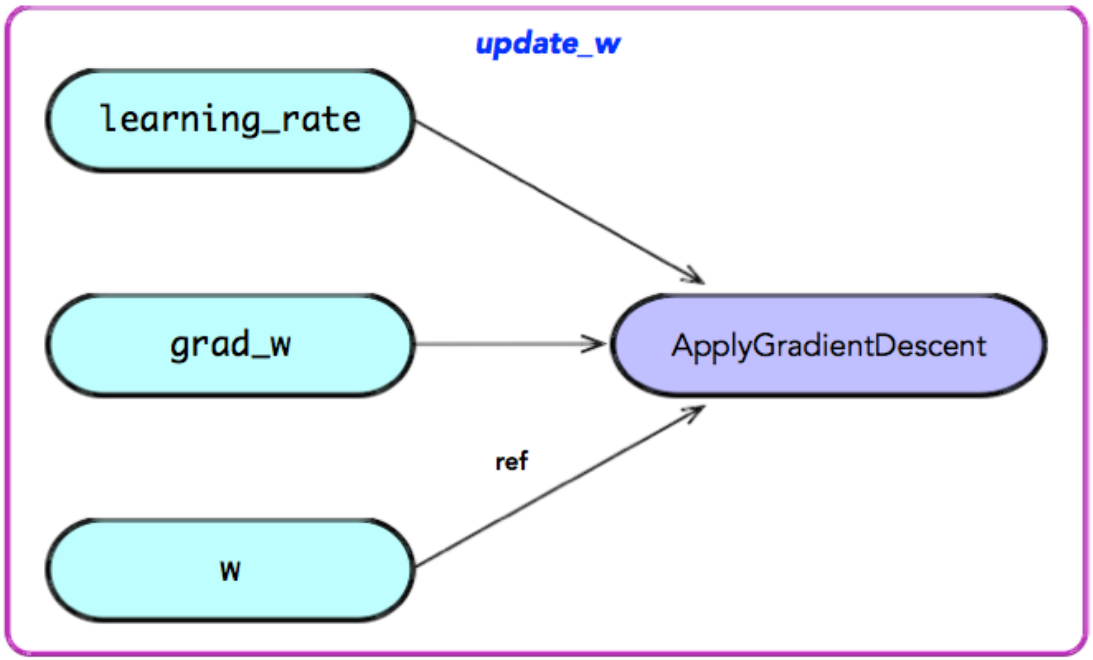
例子：square

```
@ops.RegisterGradient("Square")
def SquareGrad(op, grad):
  x = op.inputs[0]
  with ops.control_dependencies([grad.op]):
    return grad * (2.0 * x)
```

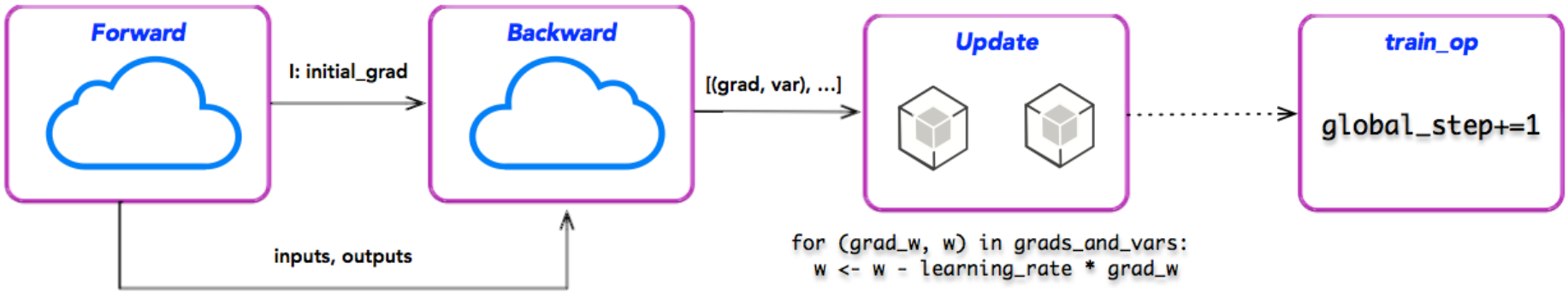


应用梯度

```
def apply_gradients(grads_and_vars, learning_rate):
  for (grad, var) in grads_and_vars:
    apply_gradient_descent(learning_rate, grad, var)
```



关键路径: RunStep



答疑

谢谢