贾志刚

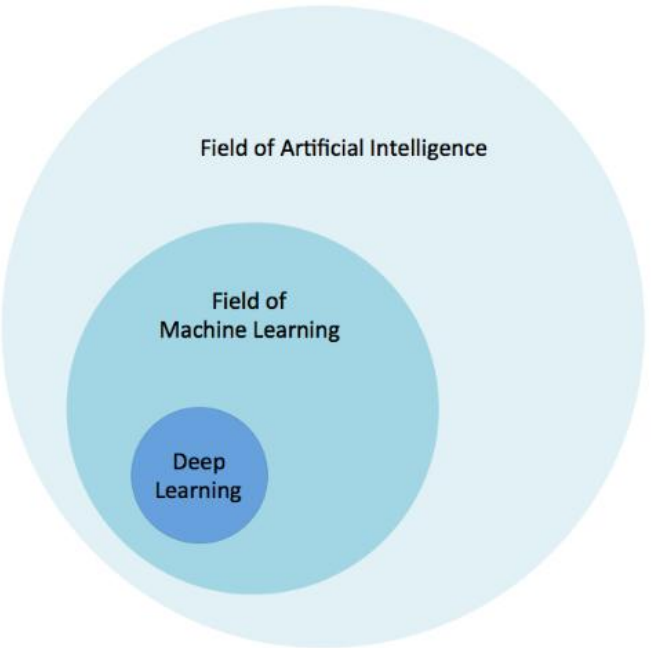
深度学习-CNN与RNN

主要内容

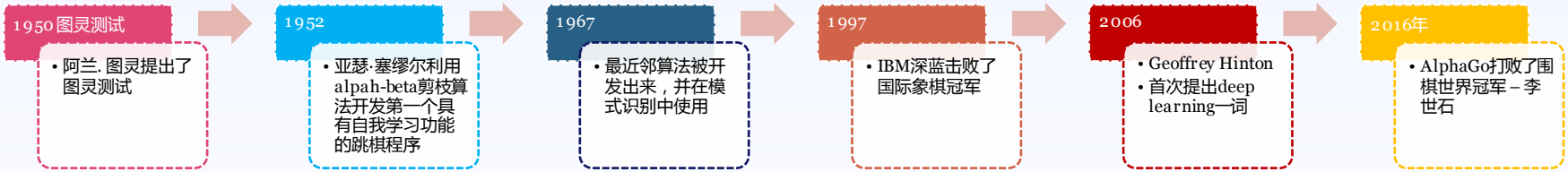
- 深度学习的历史
- 深度学习的应用领域
- 卷积神经网络
- 循环神经网络
- 基于卷积神经网络数字识别

深度学习的历史

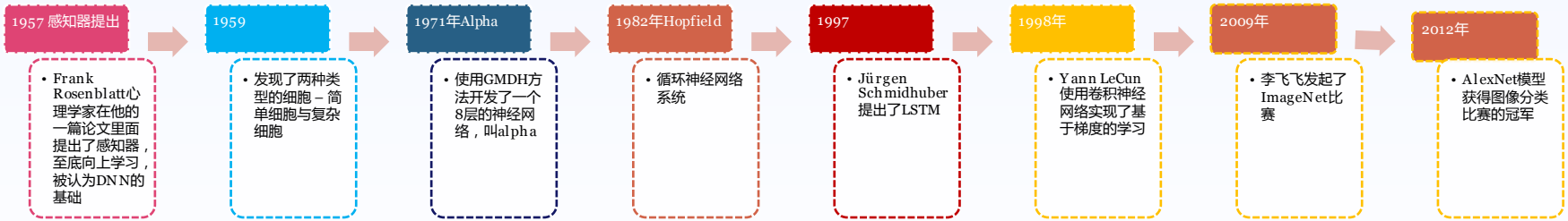

Google DeepMind
Challenge Match
8 - 15 March 2016
AlphaGo
AlphaGo
Lee Sedol


Field of Artificial Intelligence
Field of
Machine Learning
Deep
Learning

机器学习的历史


1950图灵测试
• 阿兰. 图灵提出了图灵测试
1952
• 亚瑟·塞缪尔利用alpah-beta剪枝算法开发第一个具有自我学习功能的跳棋程序
1967
• 最近邻算法被开发出来，并在模式识别中使用
1997
• IBM深蓝击败了国际象棋冠军
2006
• Geoffrey Hinton
• 首次提出deep learning一词
2016年
• AlphaGo打败了围棋世界冠军－李世石

深度学习/神经网络发展历史



深度学习几个要素

算法 – 模型

- 知识系统
- 专家系统
- 机器学习
- 神经网络
- 深度学习

硬件 – 算力

- CPU
- GPU
- TPU
- NPU
- IPU

存储 – 数据

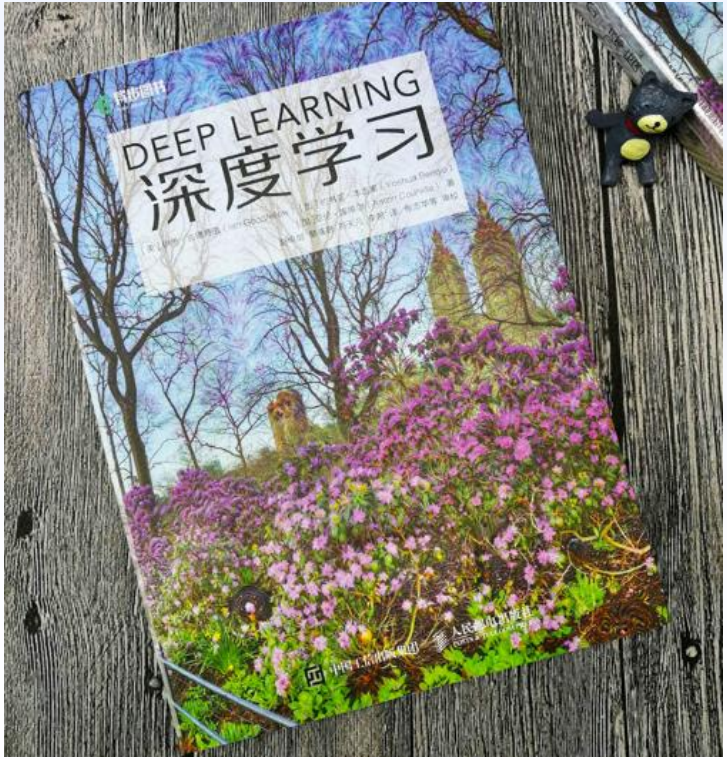
- 大数据清洗处理
- 非关系型数据库
- 各种文本与图像
- 各种视频数据

Geoffrey Hinton (Toronto)

Yann Lecun (NYU)

Youshua Bengio (UdeM)

Andrew Ng (Stanford)


DEEP LEARNING
深度学习


机器学习
周志华 著
MACHINE LEARNING
机器学习
清华大学出版社

深度学习的应用领域

- 图像分类
- 图像对象检测与识别
- 图像分割
- 图像标注
- 姿态评估
- 图像生成
- 图像修复与重建
- 风格迁移

分类

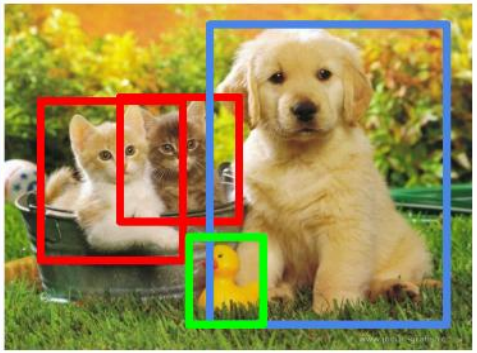
CAT

分类+定位

CAT

单个对象

对象检测

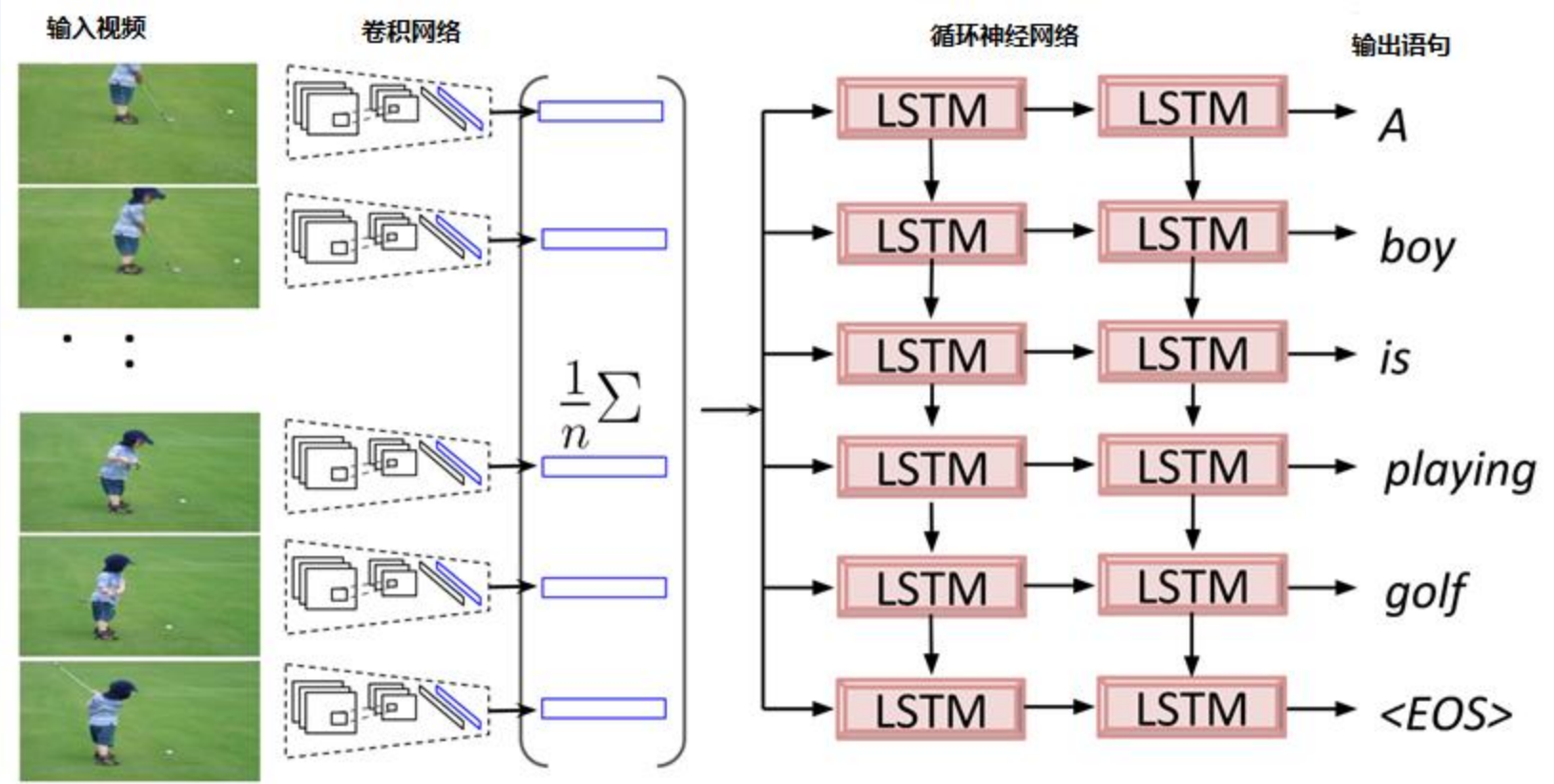


CAT, DOG, DUCK

实例分割

CAT, DOG, DUCK

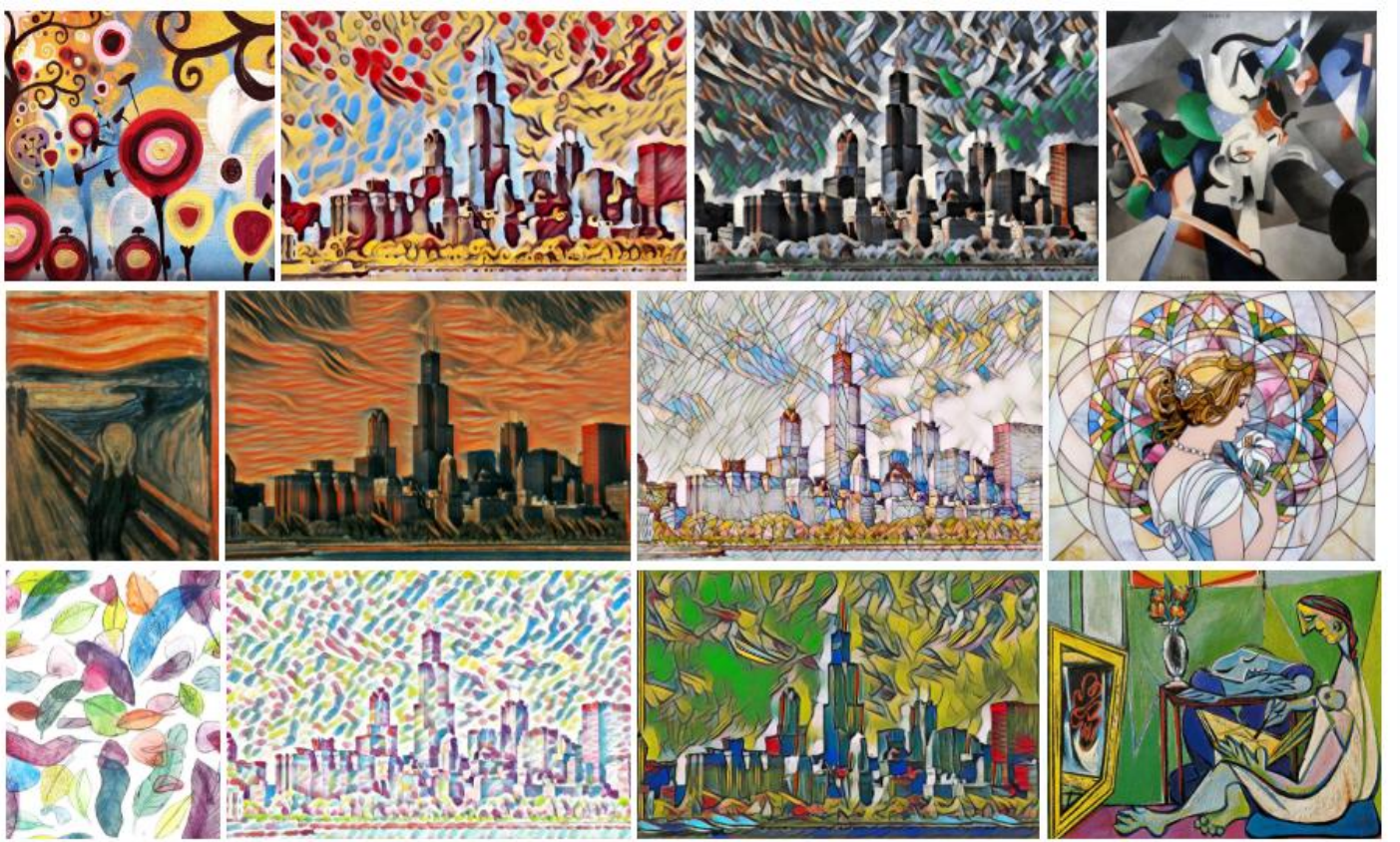
多个对象


输入视频
卷积网络
循环神经网络
输出语句
$\frac{1}{n}\sum$
LSTM
A
boy
is
playing
golf
<EOS>

http://t.cn/RnZ38uB?m=4217785863560097&u=1242677540

我不是梵高，我是AI

-CNN风格迁移网络
-模拟艺术风格



应用领域 – 创作

清华大学团队 使用深度学习技术 实现诗歌自动生成
为你写诗 在线尝试 - http://poem.bosonnlp.com/

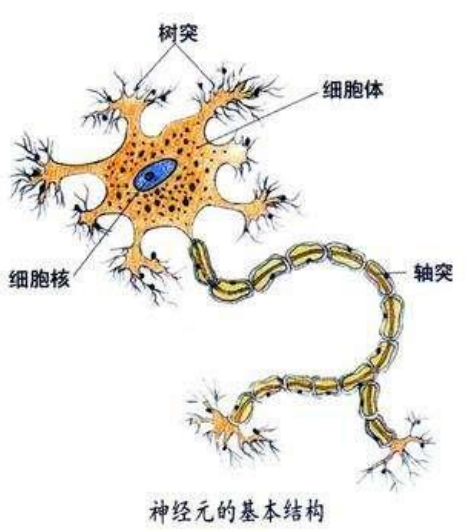
阿里 **鲁班** AI设计 – 1秒中设计8000张海报 自动生成
一天完成4000万张 双11 海报设计
为你设计 演示视频 - https://v.qq.com/iframe/preview.html?vid=t0398cx2tro&amp

艺术领域：
莎士比亚 风格的歌剧、 莫扎特风格的谱曲

卷积神经网络

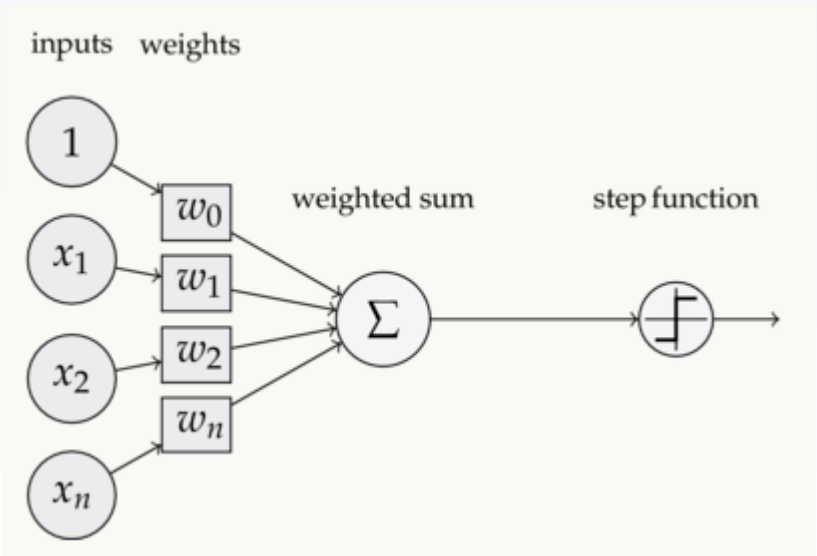
- 感知器
- 多层感知器
- 反向传播算法
- 卷积神经网络

生物学神经元



BNN – 生物神经网络	ANN – 人工神经网络
Soma(细胞体)	节点
Dendrites(树突)	输入
Synapse(突触)	权重连接
Axon(轴突（神经细胞的突起，将信号发送到其他细胞）)	输出

感知器

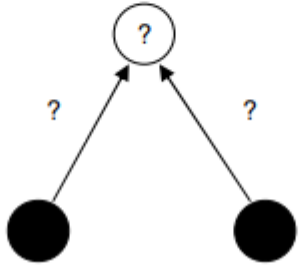

inputs
weights
1
x_1
x_2
x_n
w_0
w_1
w_2
w_n
weighted sum
Σ
step function

单层感知器–逻辑门

NOT	
in	out
0	1
1	0

AND		
in_1	in_2	out
0	0	0
0	1	0
1	0	0
1	1	1

OR		
in_1	in_2	out
0	0	0
0	1	1
1	0	1
1	1	1



单层感知器-逻辑门

在二维平面 存在

$$I_1 \cdot w_1 + I_2 \cdot w_2 - \theta = 0$$

可以变换为如下：

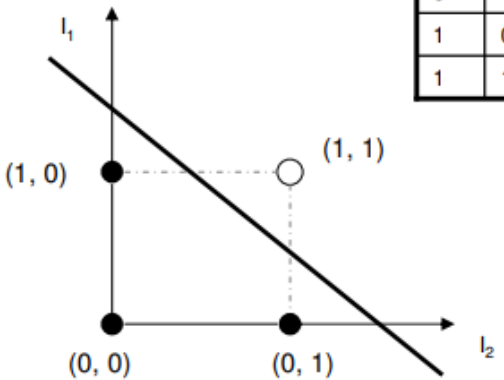
$$I_2 = \frac{\theta}{w_2} - \frac{w_1}{w_2} I_1$$

逻辑门的决策 - 超平面分割

AND

w1=1, w2=1, θ=1.5

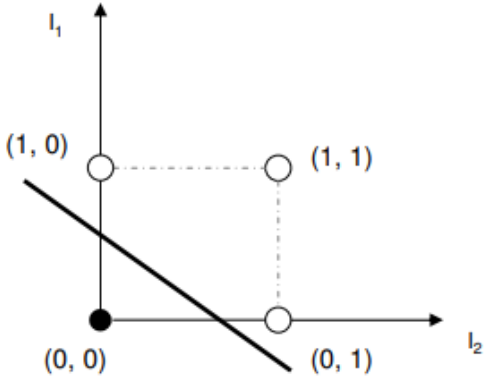
AND		
I_1	I_2	out
0	0	0
0	1	0
1	0	0
1	1	1



OR

w1=1, w2=1, θ=0.5

OR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	1



单层感知器 - 代码实现

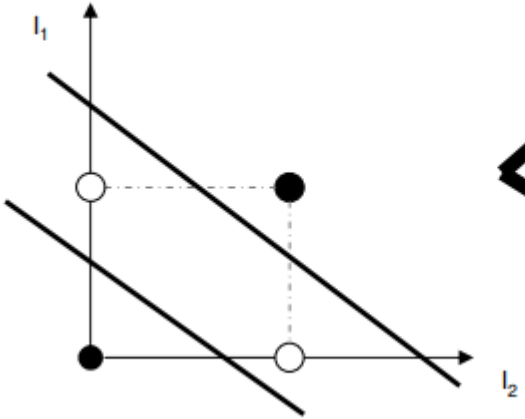
```
x = np.array([
    [0, 0],
    [1, 0],
    [0, 1],
    [1, 1]
])

y = np.array([0, 0, 0, 1])
w = np.random.random(2)
```

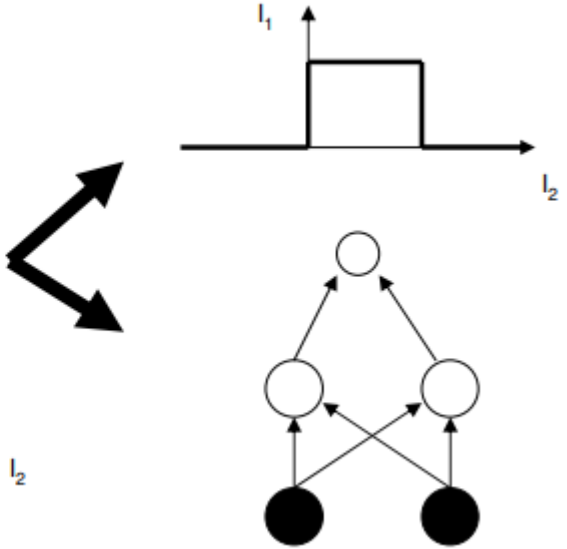
```
# 学习率
lr = 0.11
n = 0
# 输出值
out = 0


def update():
    global x, y, w, lr, n
    n += 1
    out = np.dot(x, w.T)
    dw = lr * (y - out.T).dot(x) / int(x.shape[0])
    w = w + dw


for _ in range(1000):
    update()
    out = np.dot(x, w.T)
    if (out == y.T).all():
        break
```

单层感知器

- 单层感知器的缺陷，无法解决异或门
- 感知器无法解决非线性可分离问题

XOR		
I_1	I_2	out
0	0	0
0	1	1
1	0	1
1	1	0

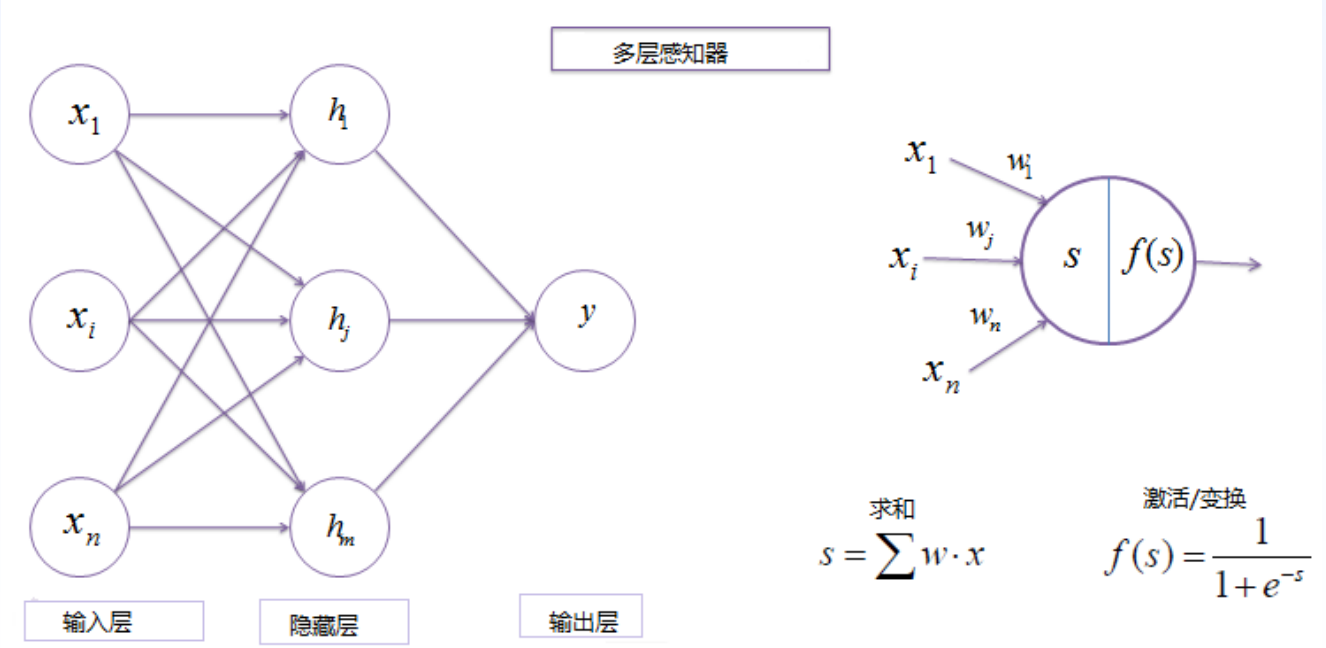




单层感知器

- 单层感知器无法解决复杂任务
- 手工创建网络代价高昂
- 非线性特征也是手工生成
- 需要一种对数据自适应的原生模型
- 前馈传播网络 - MLP(multi layer perceptron)

多层感知器


多层感知器
x_1
x_i
x_n
h_1
h_j
h_n
y
输入层
隐藏层
输出层
x_1
x_i
x_n
w_1
w_j
w_n
s
$f(s)$
求和
$s = \sum w \cdot x$
激活/变换
$f(s) = \frac{1}{1+e^{-s}}$

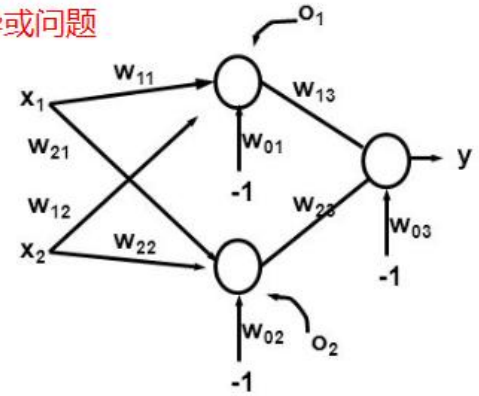
多层感知器解决异或问题

解决异或问题

输入层 2个节点

隐藏层 2个节点

输出层 1个节点



期望输出

x1	x2	o1	o2	y
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	1	0

权重：

w11= w12=1

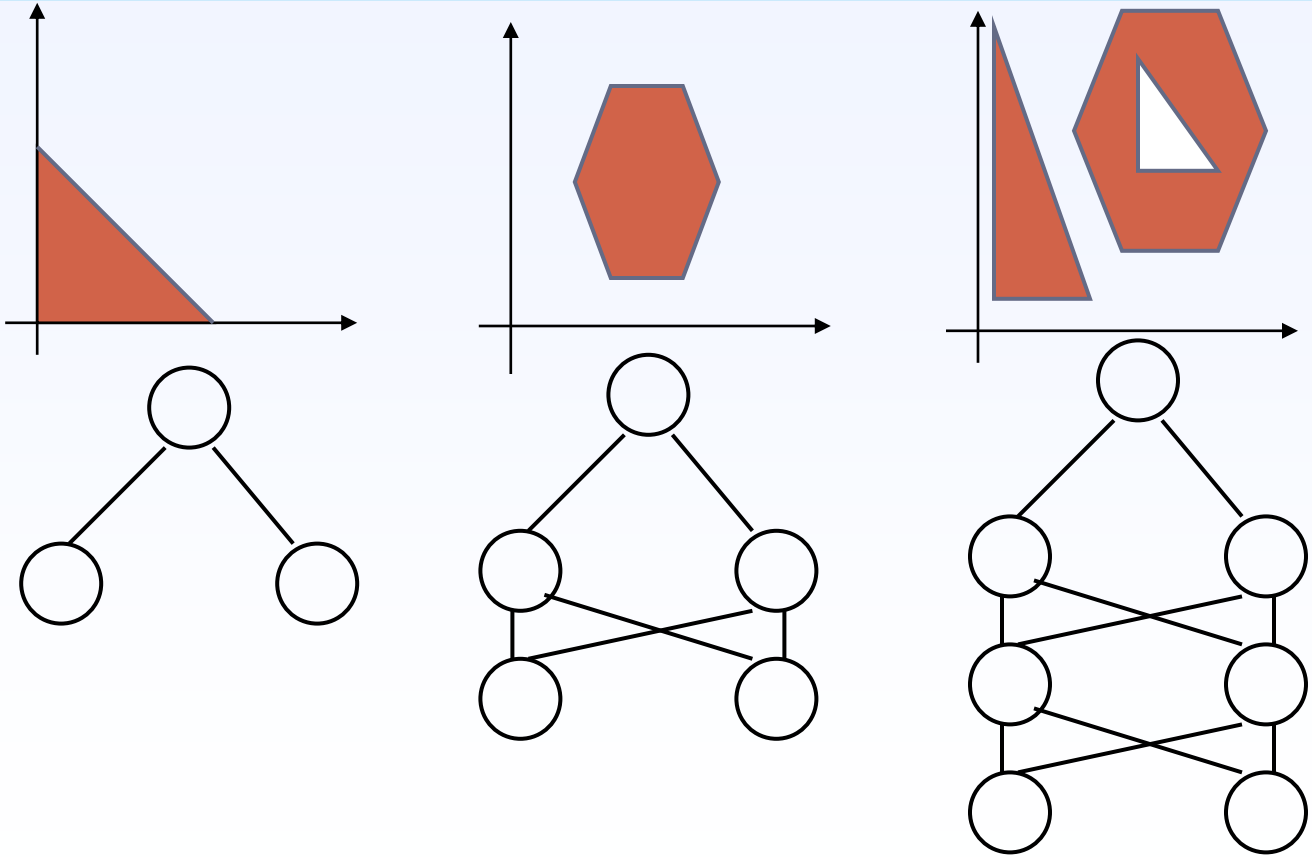
w21=w22 = 1

w01=3/2; w02=1/2; w03=1/2

w13=-1; w23=1

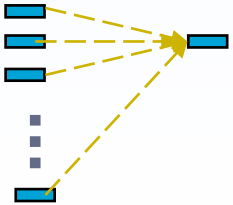
多层感知器

单个/单层感知器，线性分离
二层感知器组合可以分离
三层感知器任何复杂可分



多层感知器网络特征

- 同一层内的节点相互没有连接
- 输入与输出层不直接相连
- 层与层之间的连接方式是全连接
- 经常超过三层
- 输出的节点单元不必与输入节点单元数目相等
- 隐藏层的节点单元数目可以大于或者少于输入/输出层

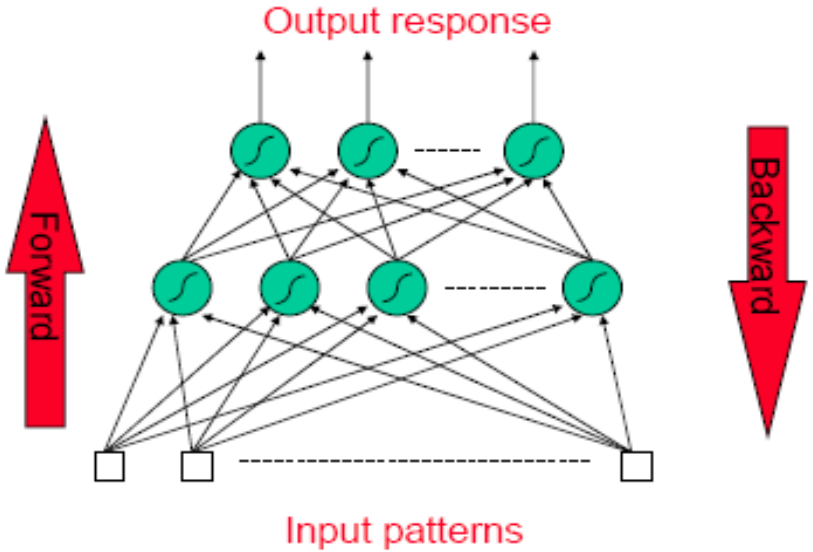


每个节点是一个感知器，通常添加增益b来调整权重w

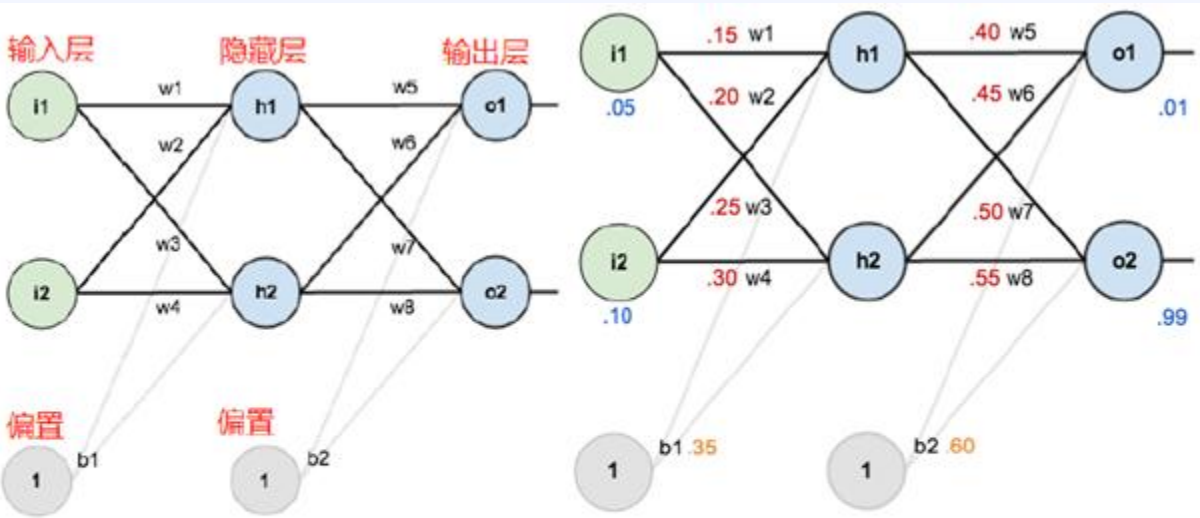
$$y_i = f\left(\sum_{j=1}^{m} w_{ij} x_j + b_i\right)$$

多层感知器学习 – 反向传播算法

- 前馈网络与反馈网络
- BP算法的两个阶段
  - 前向传播阶段
  - 反向传播阶段



反向传播算法 – 前向传播



$$net_{h1} = w_1 i_1 + w_2 i_2 + b_1 = 0.15*0.05 + 0.2*0.1 + 0.35 = 0.3775$$

$$outh_1 = \frac{1}{1+e^{-neth_1}} = \frac{1}{1+e^{-0.3775}} = 0.5932699$$

$$net_{o1} = w_5 outh_1 + w_6 outh_2 + b_2 = 0.4*0.5932699 + 0.45*0.59688 + 0.6 = 1.1059$$

$$outh_2 = \frac{1}{1+e^{-neth_2}} = 0.59688$$

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

$$E_{o1} = \sum \frac{1}{2}(target_{o1} - output_{o1})^2$$

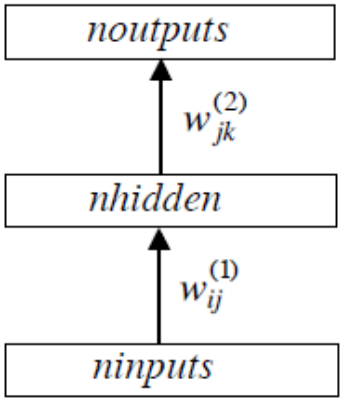
$$= \frac{1}{2}(0.01 - 0.751365)^2 = 0.274811$$

$$E_{o2} = 0.02356$$

$$E_{total} = E_{o1} + E_{o2} = 0.2983$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.1059}} = 0.7513$$

反向传播算法 – 前向传播



$$out_k^{(2)} = f^{(2)}\left(\sum_j out_j^{(1)} w_{jk}^{(2)}\right)$$

$$out_j^{(1)} = f^{(1)}\left(\sum_i out_i^{(0)} w_{ij}^{(1)}\right)$$

$$out_i^{(0)} = in_i$$

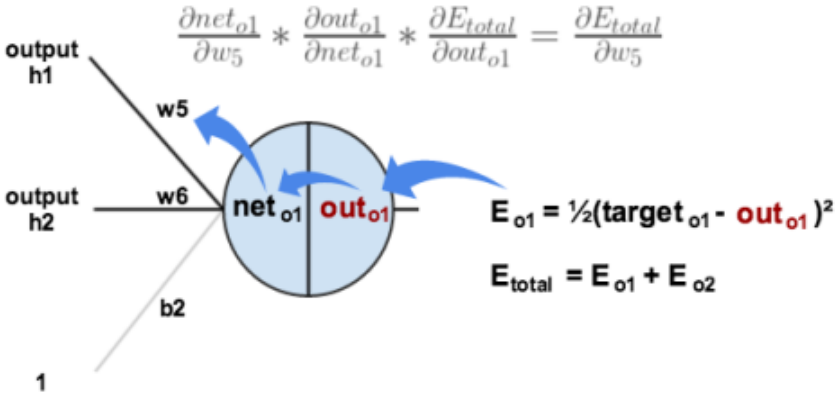
前向传播计算

SSE Loss (sum-squared error)

$$E_{SSE} = \frac{1}{2}\sum_p \sum_j \left(targ_j^p - out_j^{(N)p}\right)^2$$

$$E_{CE} = -\sum_p \sum_j \left[targ_j^p . \log\left(out_j^{(N)p}\right) + (1 - targ_j^p) . \log\left(1 - out_j^{(N)p}\right)\right]$$

反向传播算法 - 反向传播



$$\frac{\partial E_{total}}{\partial w_5} = -(t\arg et_{o1} - out_{o1}) * out_{o1}(1-out_{o1}) * out_{h1} = \delta_{o1} out_{h1}$$

$$w_5^{+} = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

η：表示学习率

输出层：$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} \frac{\partial out_{o1}}{\partial net_{o1}} \frac{\partial net_{o1}}{\partial w_5}$

$$\frac{\partial E_{total}}{\partial out_{o1}} = -(t\arg et_{o1} - out_{o1}) = -(0.01 - 0.751365) = 0.74136507$$

$$\frac{\partial out_{o1}}{\partial net_{o1}} = out_{o1}(1-out_{o1}) = 0.75136507(1-0.75136507) = 0.593269992$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}}$$

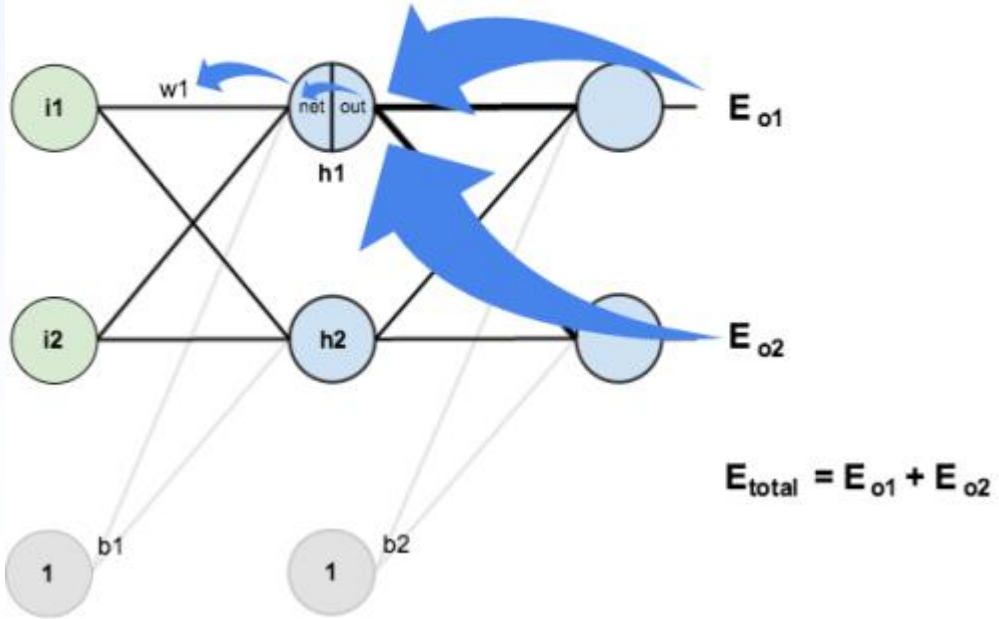
$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} = out_{h1} = 0.5932$$

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.59326992 = 0.082167041$$

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



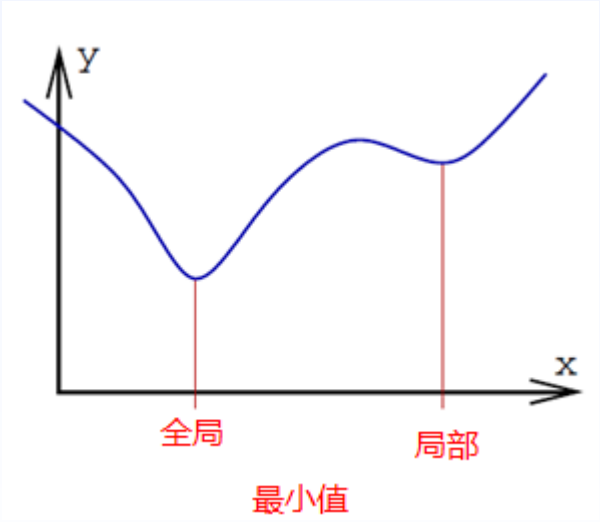
$$\frac{\partial E_{total}}{\partial w_1} = \left(\sum_o \frac{\partial E_{total}}{\partial out_o} * \frac{\partial out_o}{\partial net_o} * \frac{\partial net_o}{\partial out_{h1}}\right) * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

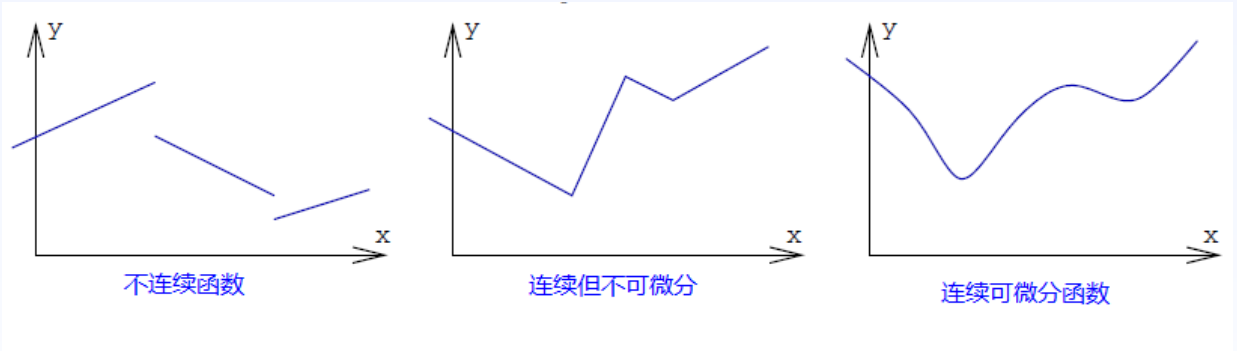
$$\frac{\partial E_{total}}{\partial w_1} = \left(\sum_o \delta_o * w_{ho}\right) * out_{h1}(1 - out_{h1}) * i_1$$

$$\frac{\partial E_{total}}{\partial w_1} = \delta_{h1} i_1$$

$$\Delta w_{kl}^{(m)} = -\eta \frac{\partial E(\{w_{ij}^{(n)}\})}{\partial w_{kl}^{(m)}}$$

梯度下降 - 优化

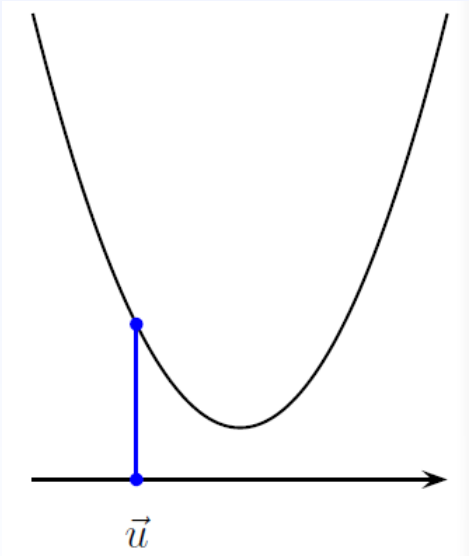




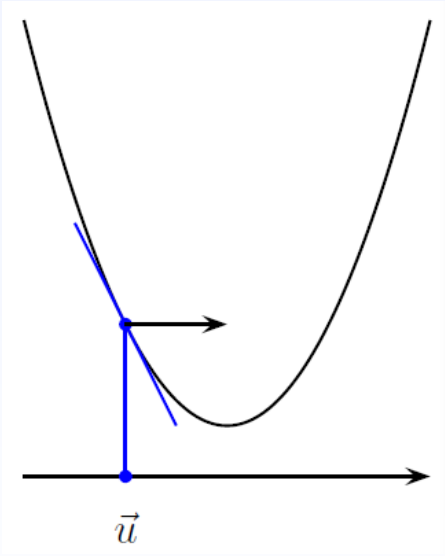
$$\frac{\partial f}{\partial u_i}(\vec{u}^{+}) = 0 \quad \text{for all } i$$

一个全局最小化$\vec{u}^{*}$是一个点满足

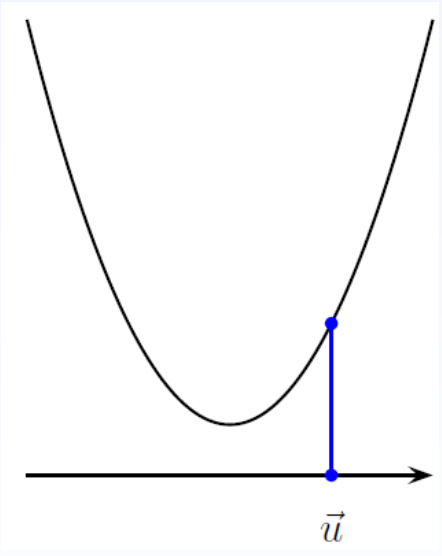
$f(\vec{u}^{*}) \leq f(\vec{u})$



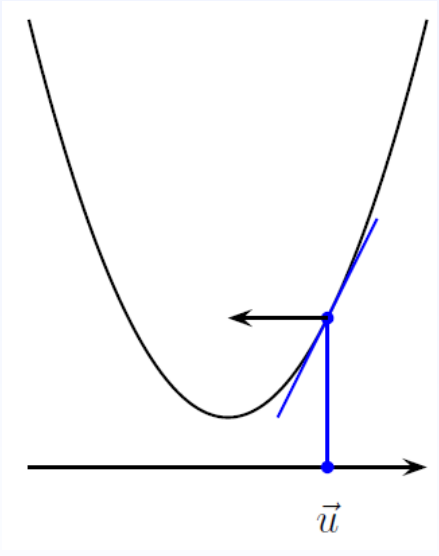
寻找 $\vec{v}$对应成立$f(\vec{v}) < f(\vec{u})$



斜率为负值，右移

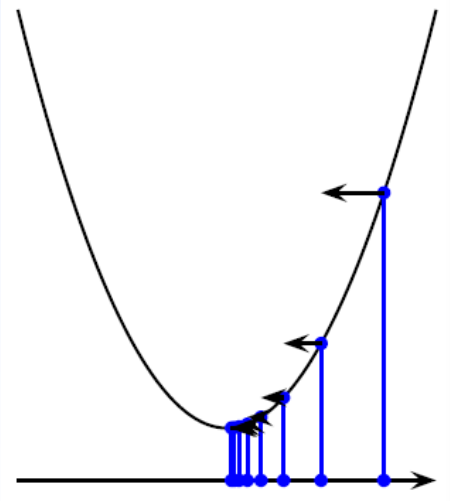


寻找 $\vec{v}$对应成立$f(\vec{v}) < f(\vec{u})$

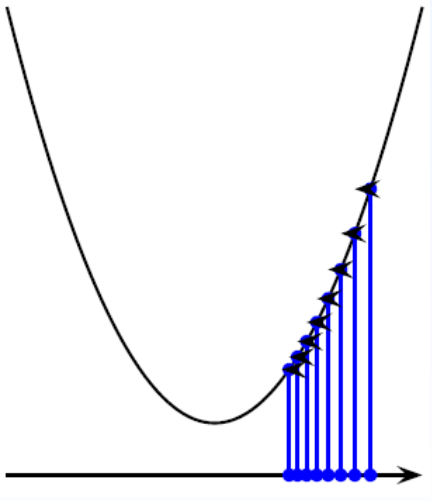


斜率为正值，左移

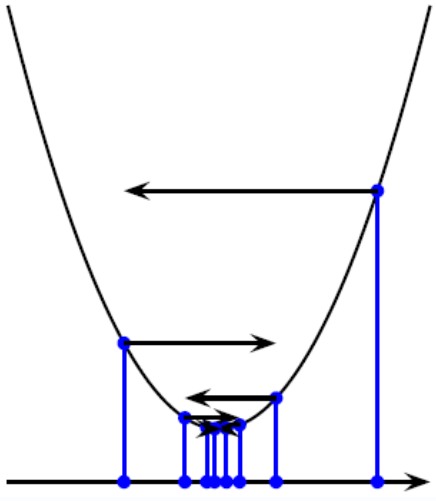
学习率影响



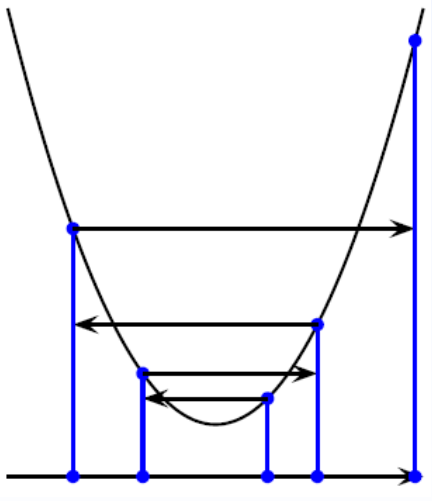
学习率小，正常收敛



学习率过小，收敛很慢



学习率大，快速收敛

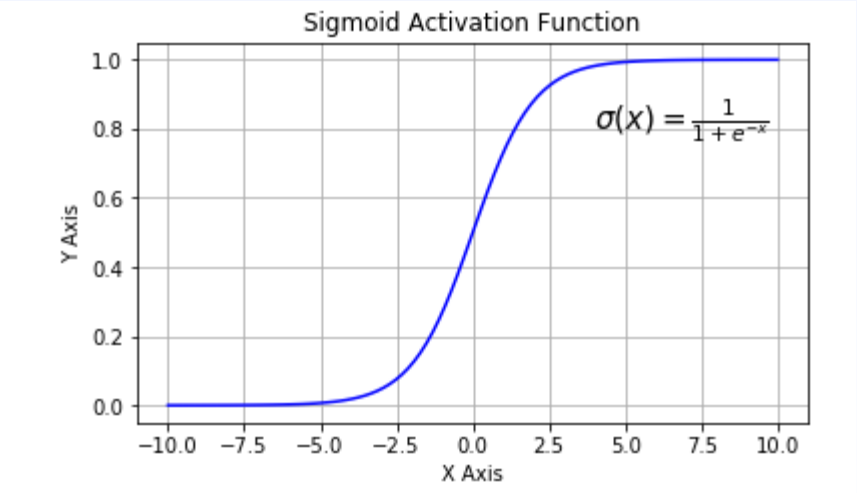


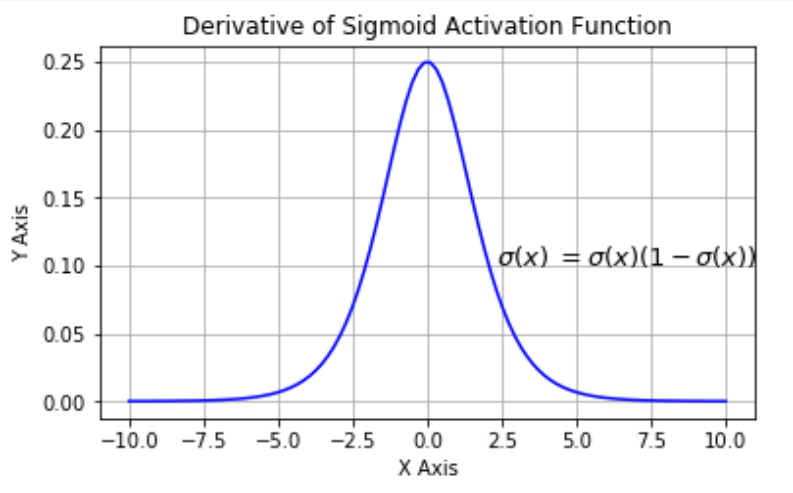
学习率过大，无法收敛

反向传播训练MLP

- 基于epoch – 所有训练样本计算一次更新
- 基于样本数 – 每个样本更新一次

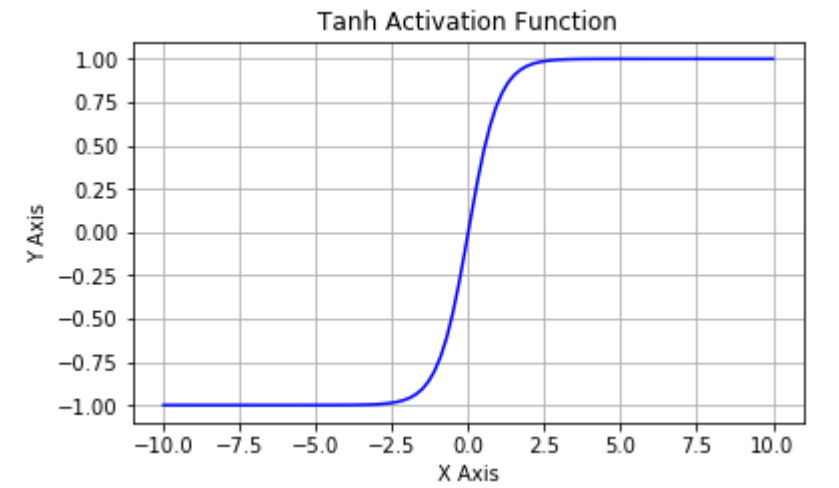
激活函数-sigmoid

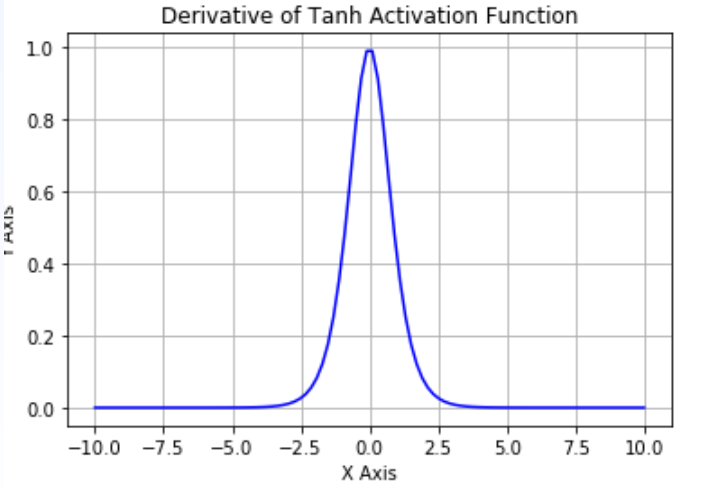




1. 有饱和放电区存在，导致梯度消失
2. 输出结果不是非零中心化输出
3. 指数计算

激活函数-tanh





1. 有饱和放电区存在，导致梯度消失
2. 指数计算

多层感知器的不足

- 全连接层对高维数据导致维度灾难
- 不能添加更多层数，梯度消失/梯度爆炸
- 使用tanh与sigmoid作为激活函数

多层感知器的代码演示

- mnist数据集
- Tensorflow代码实现
- 运行与测试

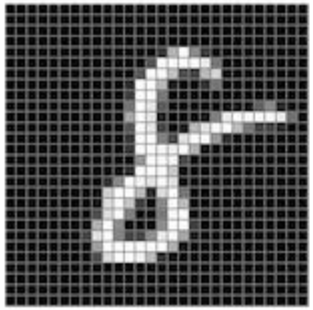
Mnist数据集

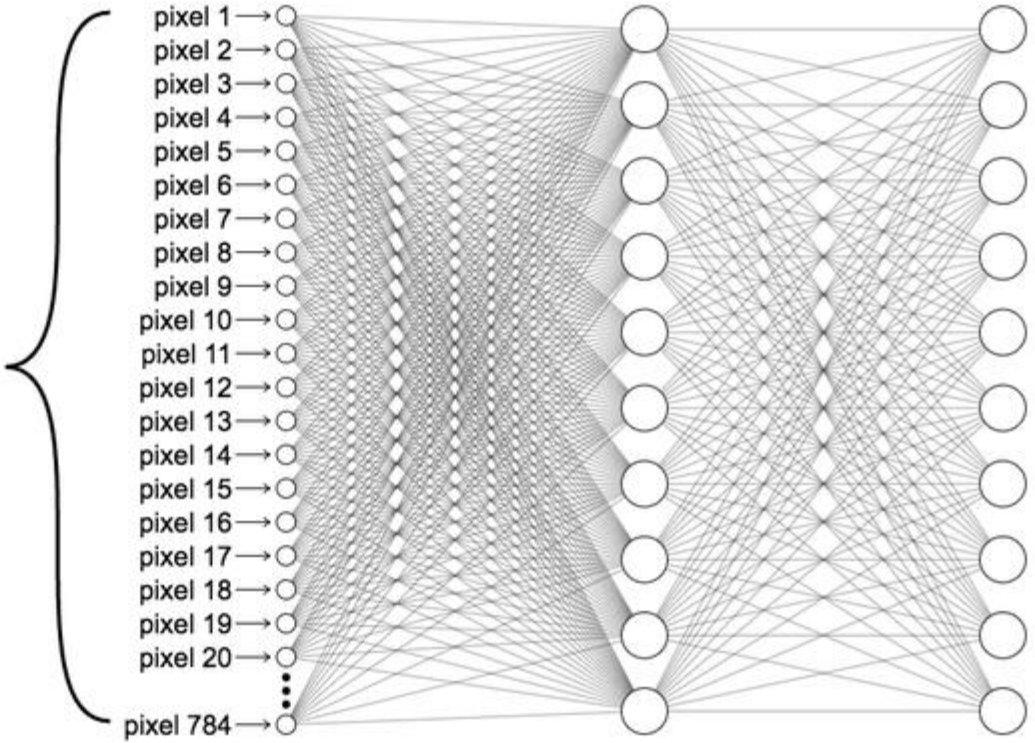
- 数据集下载地址 - http://yann.lecun.com/exdb/mnist/
- 训练数据集6W张图像
- 测试数据集1W张图像
- 数据格式说明28x28大小

Mnist数据集

数据集名称	说明
train-images-idx3-ubyte.gz	训练图像28x28大小，6万张
train-labels-idx1-ubyte.gz	每张图像的数字标记，6万条
t10k-images-idx3-ubyte.gz	测试数据集、1万张图像28x28
t10k-labels-idx1-ubyte.gz	测试数据集标记，表示图像数字

开始移位	类型	值	描述
0000	4字节int类型	0x00000803(2051)	魔数
0004	4字节int类型	60000	图像数目
0008	4字节int类型	28	图像高度
00012	4字节int类型	28	图像宽度





mnist数据集 - 神经网络

MLP网络结构

-输入层784
-隐藏层30
-输出层10

实现0-9手写数字识别

定义网络静态图结构

```
hidden_nodes = 30
x = tf.placeholder(shape=[None, 784], dtype=tf.float32)
y = tf.placeholder(shape=[None, 10], dtype=tf.float32)

W1 = tf.Variable(tf.truncated_normal(shape=[784, hidden_nodes]), dtype=tf.float32)
b1 = tf.Variable(tf.truncated_normal(shape=[1, hidden_nodes]), dtype=tf.float32)

W2 = tf.Variable(tf.truncated_normal(shape=[hidden_nodes, 10]), dtype=tf.float32)
b2 = tf.Variable(tf.truncated_normal(shape=[1, 10]), dtype=tf.float32)

# layer hidden
nn_1 = tf.add(tf.matmul(x, W1), b1)
h1 = tf.nn.sigmoid(nn_1)

# output
nn_2 = tf.add(tf.matmul(h1, W2), b2)
out = tf.nn.sigmoid(nn_2)
```

定义网络静态图结构

```
# loss function
# loss = tf.reduce_sum(tf.square(tf.subtract(y, out)))
loss = tf.nn.sigmoid_cross_entropy_with_logits(logits=nn_2, labels=y)

# BP
step = tf.train.GradientDescentOptimizer(0.05).minimize(loss)
init = tf.global_variables_initializer()

# accuracy
acc_mat = tf.equal(tf.argmax(out, 1), tf.argmax(y, 1))
acc = tf.reduce_sum(tf.cast(acc_mat, tf.float32))
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(10000):
        batch_xs, batch_ys = mnist.train.next_batch(128)
        sess.run(step, feed_dict={x: batch_xs, y: batch_ys})
        if (i+1) % 1000 == 0:
            curr_acc = sess.run(acc, feed_dict={x: mnist.test.images[:1000],
                                                y: mnist.test.labels[:1000]})
            print("current test Accuracy : %f" % (curr_acc))
```

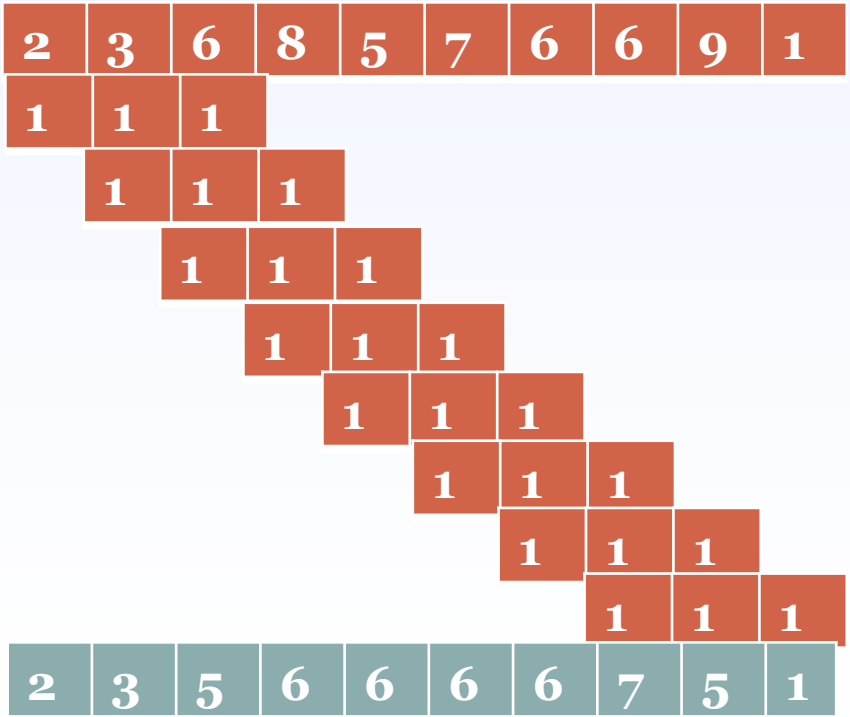
训 练 结 果

```
current test Accuracy : 906.000000
current test Accuracy : 917.000000
current test Accuracy : 921.000000
current test Accuracy : 932.000000
current test Accuracy : 930.000000
current test Accuracy : 924.000000
current test Accuracy : 931.000000
current test Accuracy : 930.000000
current test Accuracy : 942.000000
current test Accuracy : 943.000000
```

卷积神经网络

- 卷积层
- 池化层
- 全连接层
- 激活函数
- 认识LeNet模型

一维离散卷积



二维 离散卷积

二维图像 卷积示例

Image (I)

197	199	195	194	189	190	132	90	112	101
194	194	198	201	189	196	150	85	87	97
194	194	198	195	186	191	109	90	90	124
197	187	195	198	185	186	115	78	81	96
194	190	198	193	187	177	88	86	94	69
194	194	190	190	179	177	93	99	95	100
201	194	191	186	186	181	74	110	82	76
196	194	195	191	183	164	77	119	84	88
192	194	199	192	191	174	89	164	103	129
201	190	187	189	178	168	90	82	88	84

Filter (M)

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

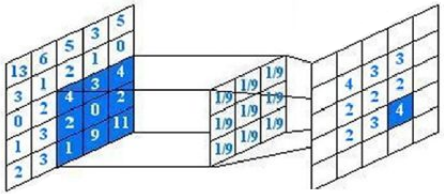
j

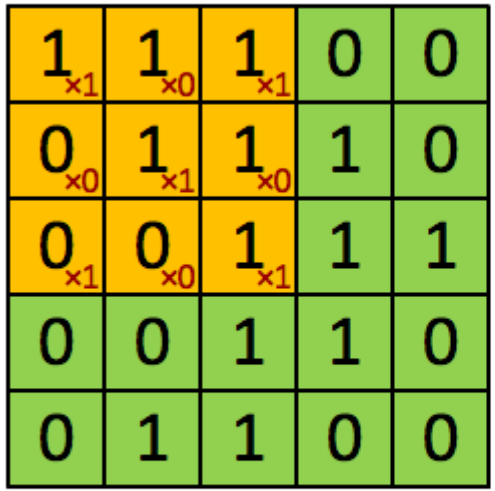
i

=

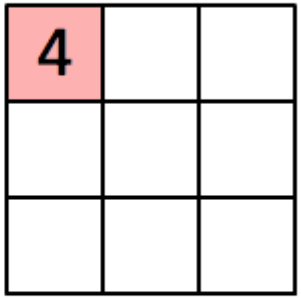
0	0	0	0	0	0	0	0	0	0
0	196	196	194	192	170	137	105	97	0
0	195	196	194	192	167	133	98	92	0
0	194	194	193	189	158	124	92	90	0
0	193	193	191	186	154	122	92	89	0
0	194	192	189	184	149	121	91	90	0
0	194	192	188	182	146	122	93	95	0
0	195	193	190	183	147	128	100	106	0
0	194	192	189	181	146	125	100	105	0
0	0	0	0	0	0	0	0	0	0

$$H(x,y)=\sum_{j=1}^{height}\sum_{i=1}^{width}I(i,j)M(x-i,y-j)$$





Image



Convolved Feature

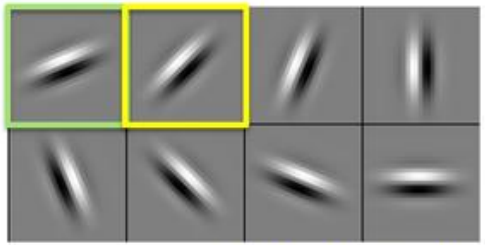
$$M = \begin{pmatrix} m_{00} & m_{01} & m_{02} & m_{03} & m_{04} \\ m_{10} & m_{11} & m_{12} & m_{13} & m_{14} \\ m_{20} & m_{21} & m_{22} & m_{23} & m_{24} \\ m_{30} & m_{31} & m_{32} & m_{33} & m_{34} \\ m_{40} & m_{41} & m_{42} & m_{43} & m_{44} \end{pmatrix}$$

$$c = \begin{pmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{pmatrix}$$

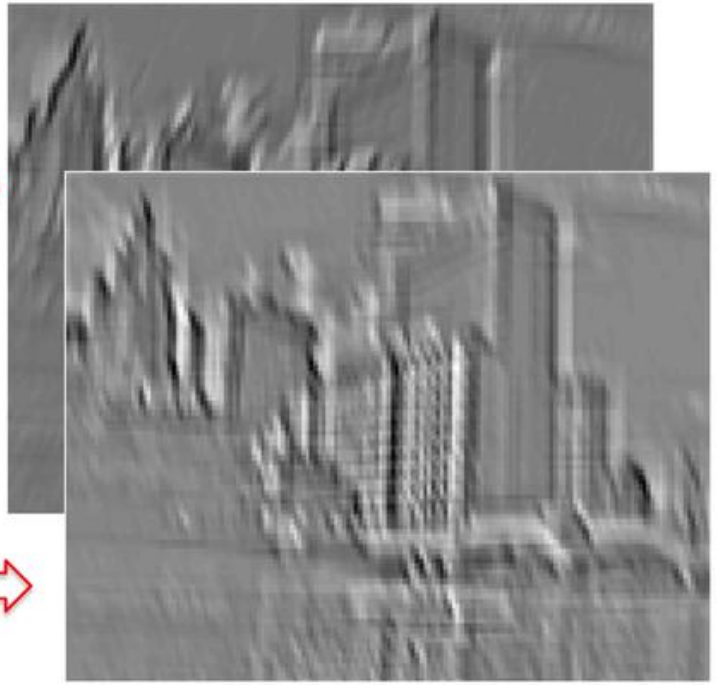
$$\begin{aligned} conv(M,c)[1,1] = & m_{11} * c_{00} + m_{12} * c_{01} + m_{13} * c_{02} + \\ & m_{21} * c_{10} + m_{22} * c_{11} + m_{23} * c_{12} + \\ & m_{31} * c_{20} + m_{32} * c_{21} + m_{33} * c_{22} \end{aligned}$$

多个滤波器-filters

Input image

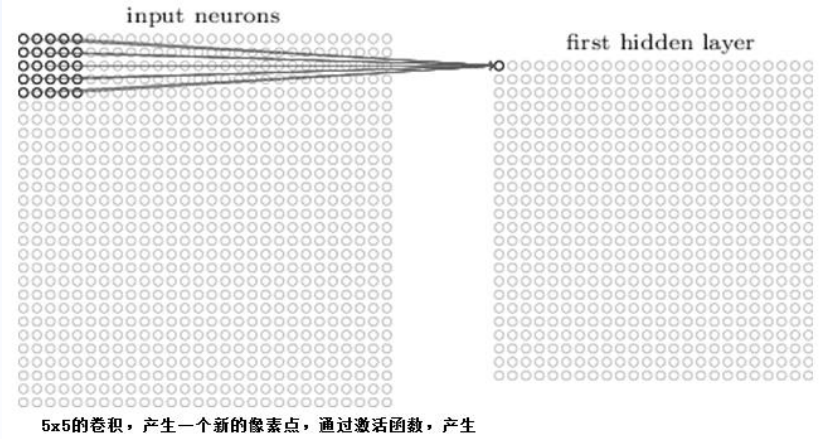


Filter bank (to be learned)



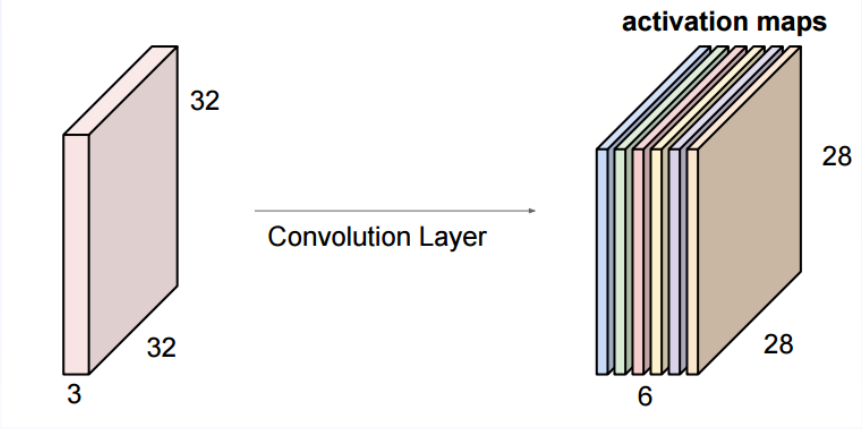
Feature maps

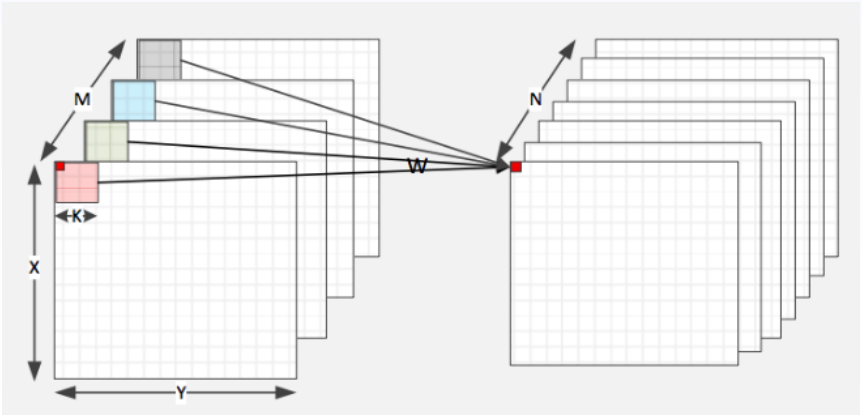
卷积层详解



5x5的卷积，产生一个新的像素点，通过激活函数，产生

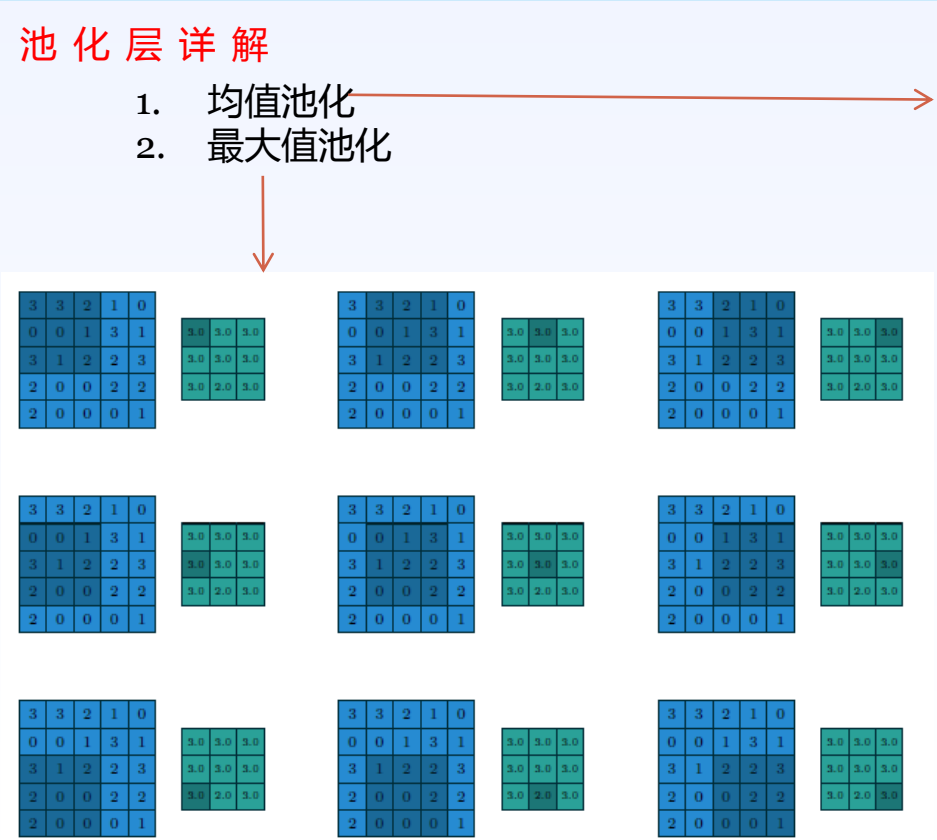
1. 减少了参数总数，降低了计算量
2. 通过filter maps提取特征，保证图像空间特征与结构。

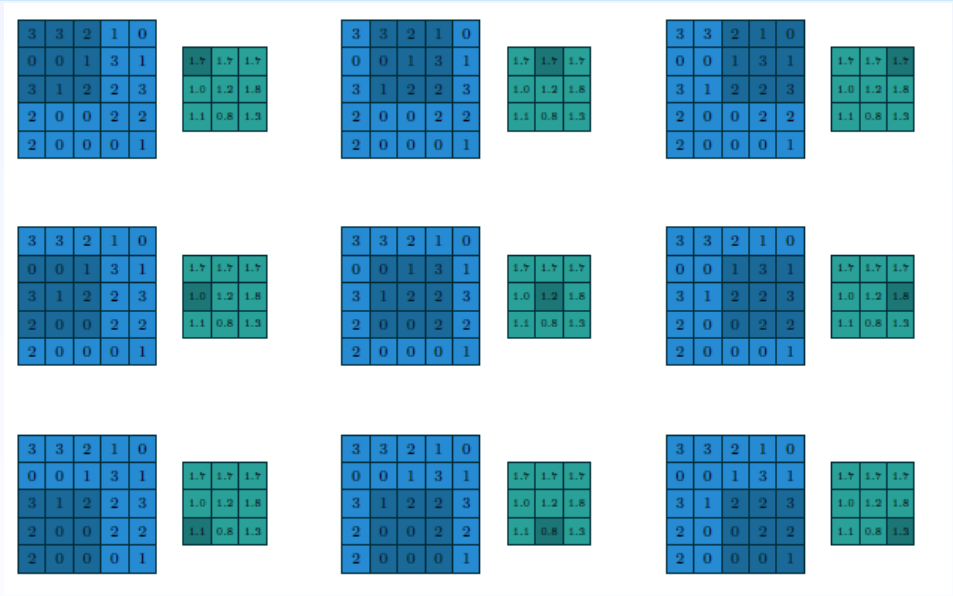




池化层详解

1. 均值池化
2. 最大值池化



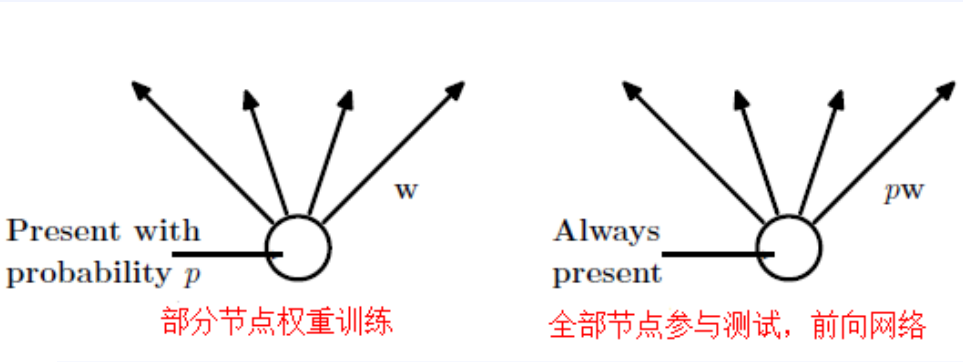


全连接层

- 多层感知器组合
- 能力不够强大???
- 多个隐藏层 与 隐藏层包含足够多的神经元 -》难以训练，过拟合 -》早停法
- 事实证明 增加深度比宽度有效

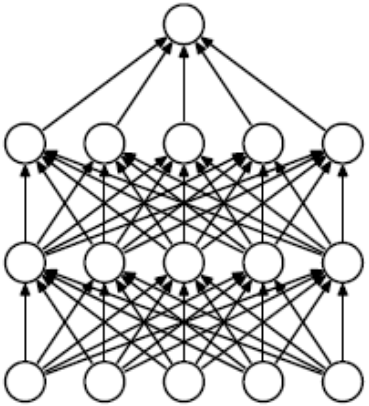
多层感知器与dropout

- 神经网络（多层感知器MLP）
- dropout

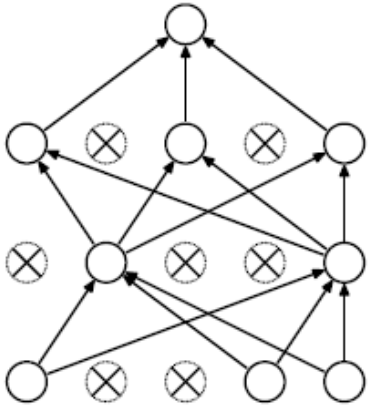


部分节点权重训练

全部节点参与测试，前向网络



标准神经网络



带dropout的神经网络

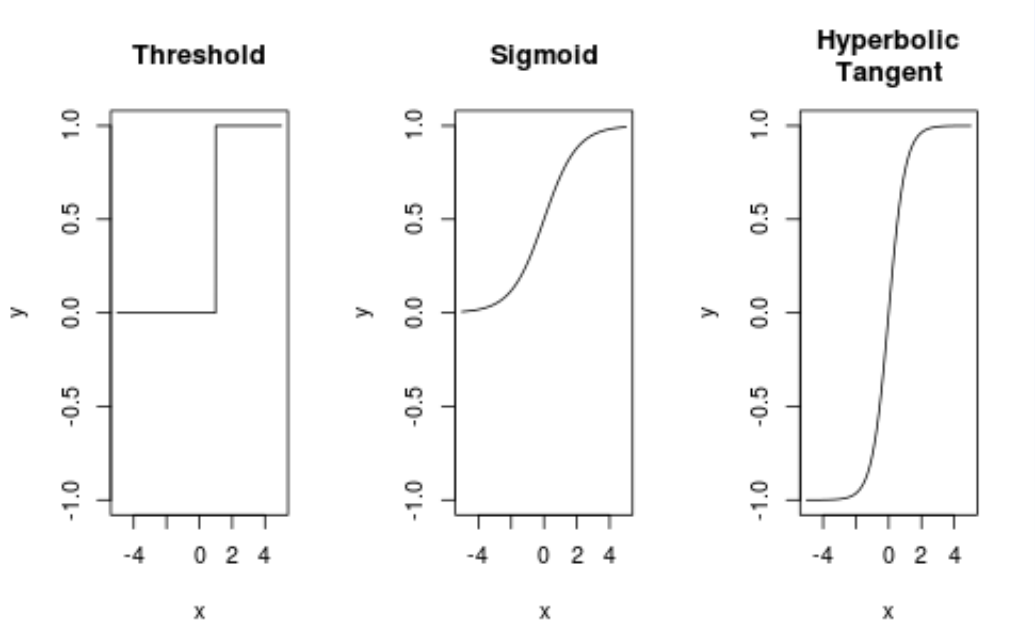
作用：防止过拟合，增加了神经网络的训练层数

基本激活函数

$$threshold(x)=\begin{cases}1 \; if \; x\geq 1\\ 0 \; if \; x<1\end{cases}$$

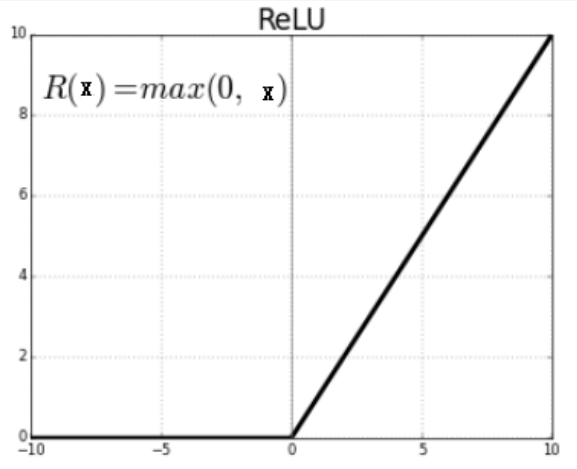
$$Sigmoid(x)=\frac{1}{1+e^{-t}}$$

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



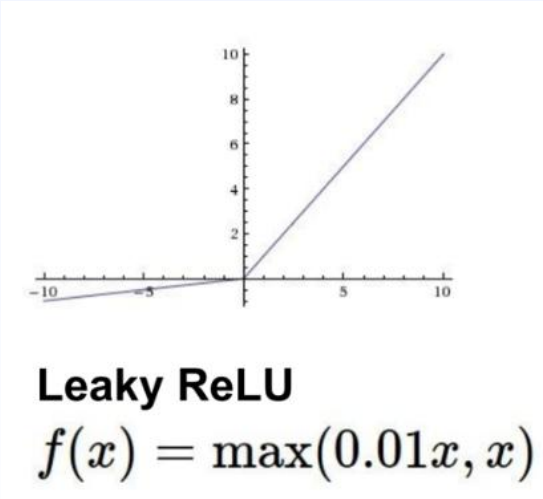
激活函数 - 更多

$\mathrm{Re}\,LU = f(x_j^i) = \max(0, x_j^i)$



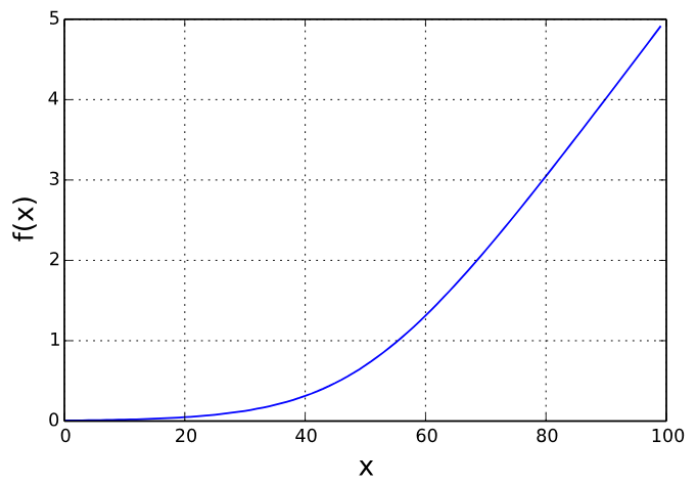
缺点：死亡RELU

Leaky $\mathrm{ReLU} = f(x_j^i) = \max(0.01x_j^i, x_j^i)$

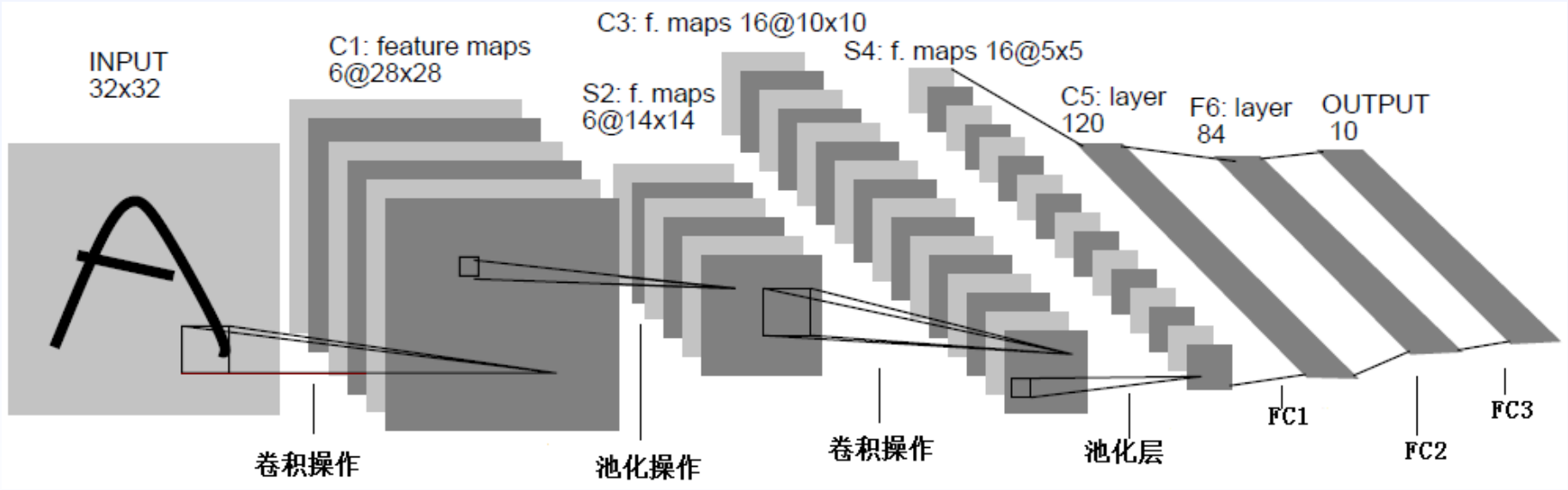


1. 快速收敛
2. 不会死亡
3. 计算简单，相比sigmoid
4. 不会饱和

Smooth $\mathrm{Re}\,LU = f(x_j^i) = \log(1+\exp(x_j^i))$



克服缺点， 不会死亡



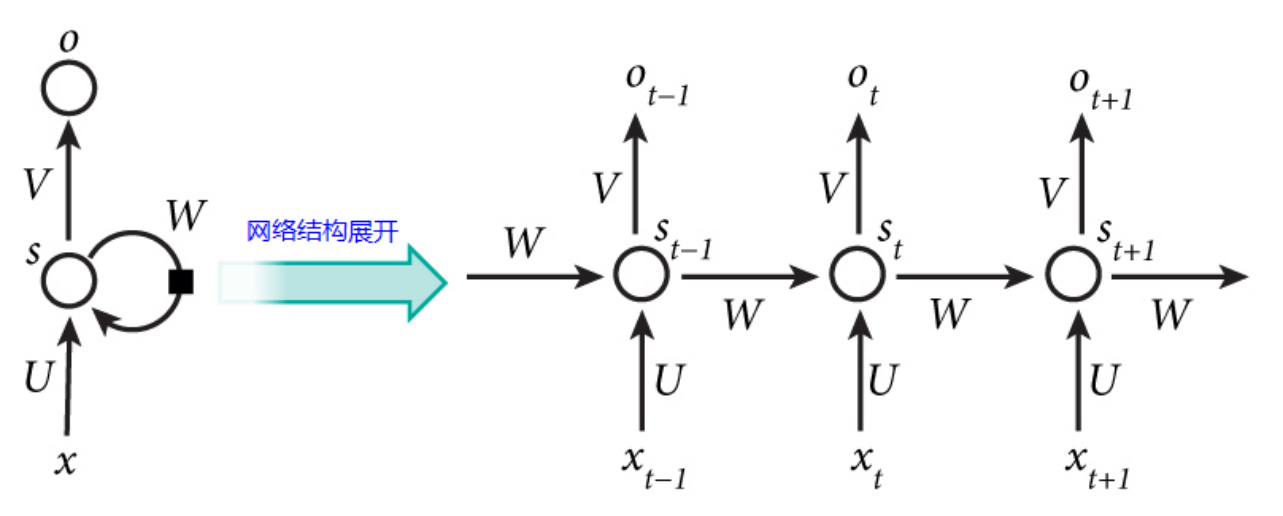
- 输入层(Input Layer)表示输入数据（图像）
- 卷积层(Convolution Layer)通过5x5的卷积核实现特征提取，然后通过2x2大小最大池化，降采样。上图有两个卷积层
- 全连接层(Full connection Layer)，传统神经网络的多层感知器(MLP)。上图有两个全连接层
- 输出层(Output Layer)

卷积神经网络-代码实现与演示

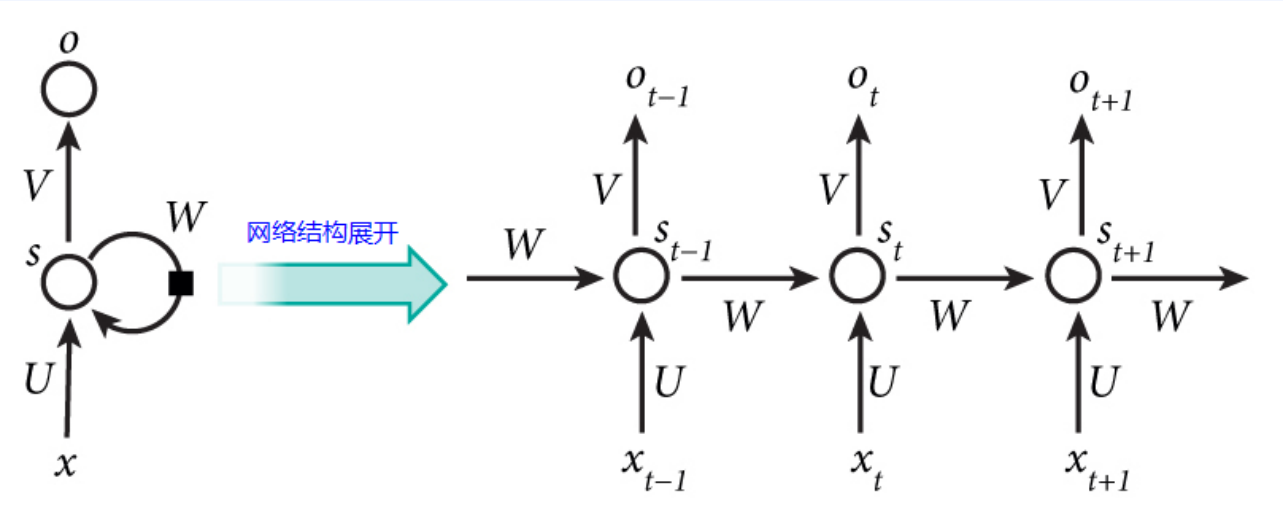
- CNN网络构建
- Mnist数据
- 训练与测试

循环神经网络-RNN

- CNN当前节点的输入以树结构形式仅包含上一层节点信息
- RNN当前节点的输入包含之前所有节点信息
- RNN可以解决序列预测问题



循环神经网络-基本原理



x_t表示t时刻的输入

s_t表示t时刻的隐藏状态，是RNN的记忆存储$s_t = f(Ux_t + W_{s_{t-1}})$

$o_t$$t$时刻的序列预测$o_t = soft\max(V_{s_t})$

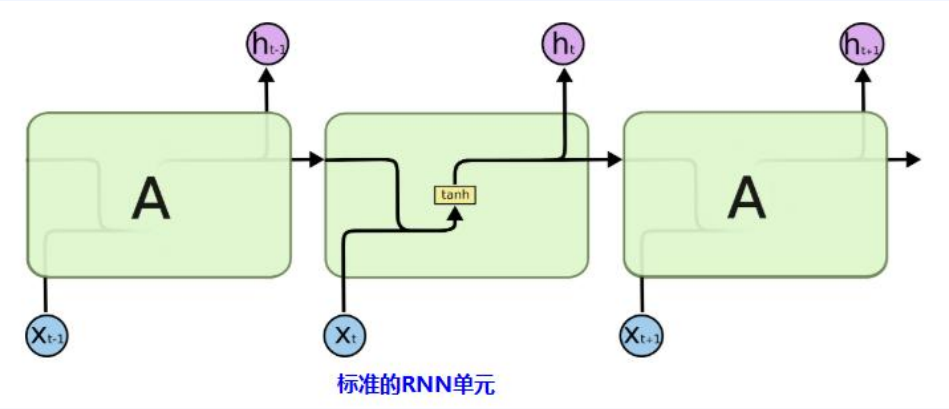
循环神经网络RNN-应用领域

- 语言模型与文本生成
- 机器翻译
- 语音识别
- CNN+RNN图像描述

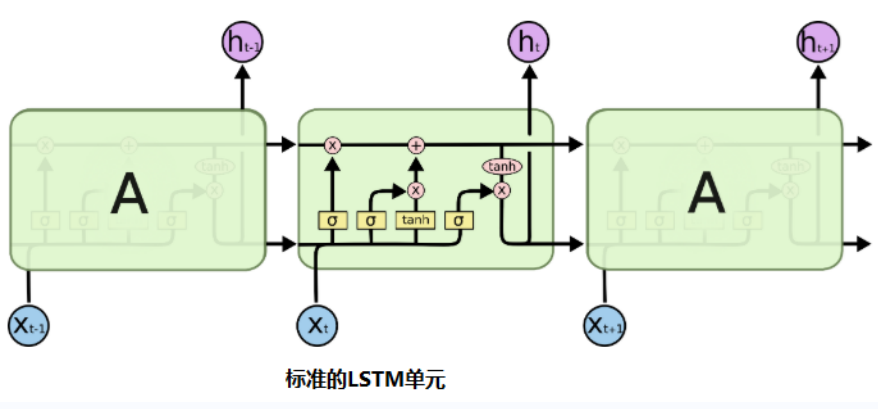
循环神经网络RNN训练方法

- BPTT（Backpropagation Through Time）
- 对过长的序列训练会很困难
- 容易导致梯度消失与梯度爆炸问题
- RNN模型扩展 - 双向RNN/深度RNN

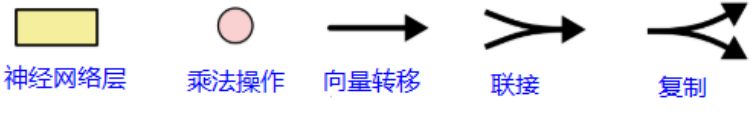
循环神经网络-经典模型LSTM（Long Short Term Memory networks）



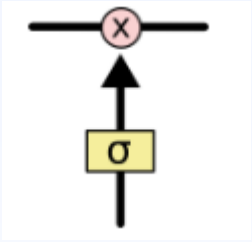
标准的RNN单元



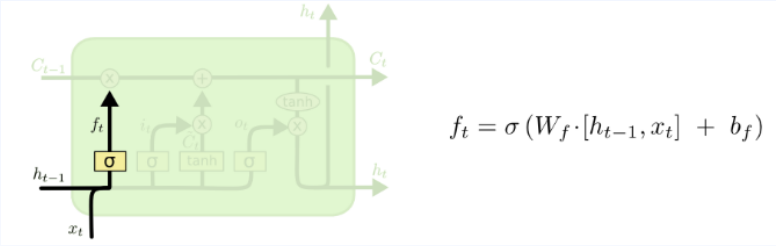
标准的LSTM单元



循环神经网络-经典模型LSTM（Long Short Term Memory networks）



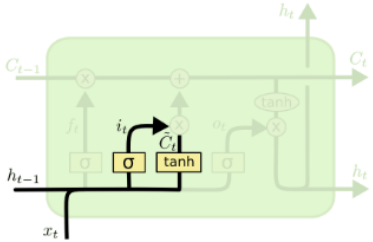
1 – 表示通过
0 – 表示不通过



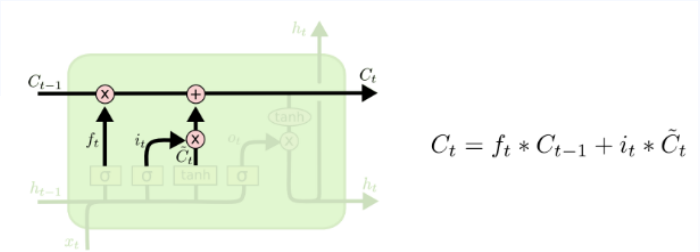
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

遗忘门

输入门

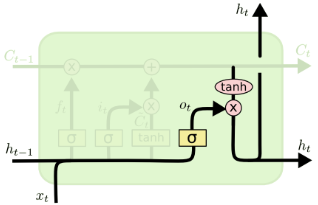


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$



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

传送门



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

输出门

课程问题咨询微信

Thank You!