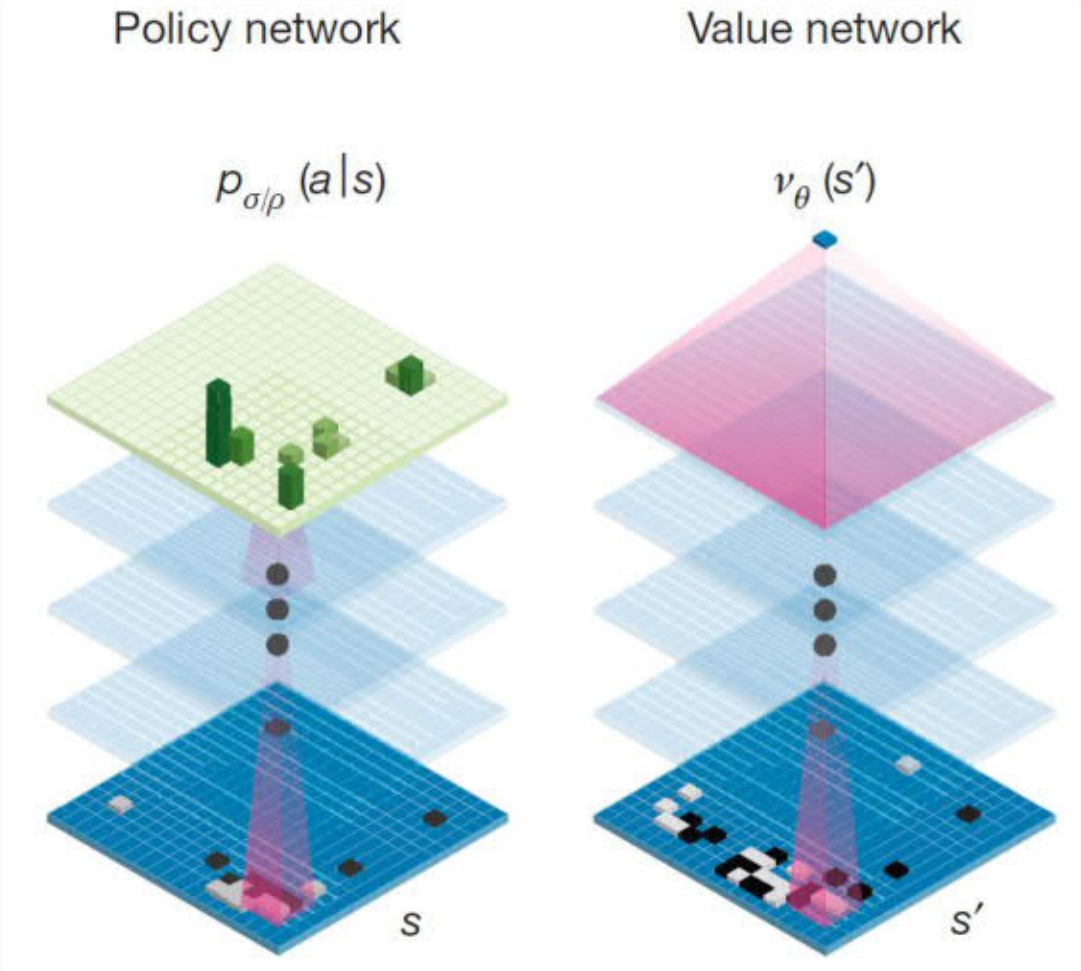
5.1 人工神经网络简介

AlphaGo中的机器学习

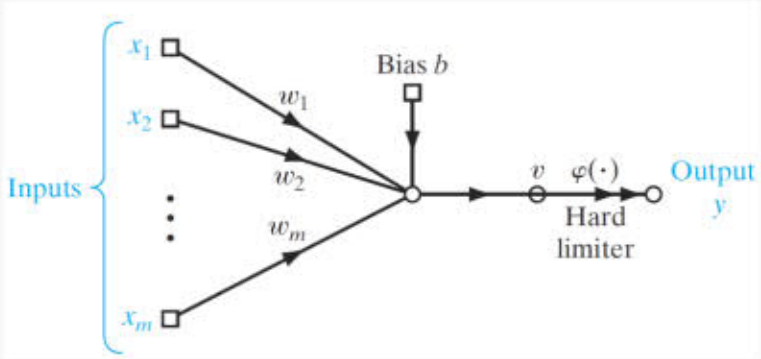
- 策略网络
  - 监督学习
    - 预测下一步人类移动的最佳结果
  - 强化学习
    - 学习去选择下一步移动去最大化获胜率
- 价值网络
  - 在给定当前状态的情况下获胜的期望
- 通过（深度）神经网络实现



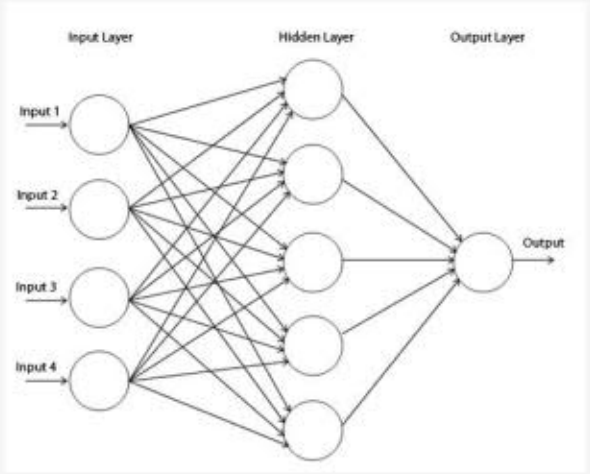
5.1 人工神经网络简介

人工神经网络

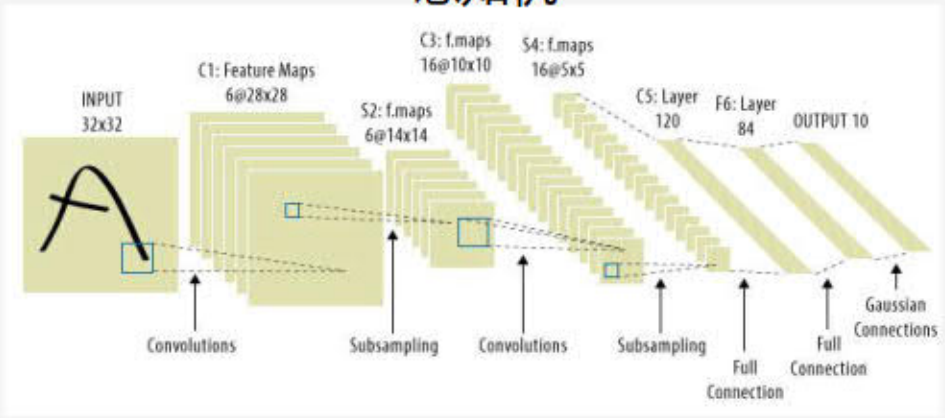
- 神经网络是深度学习的基础



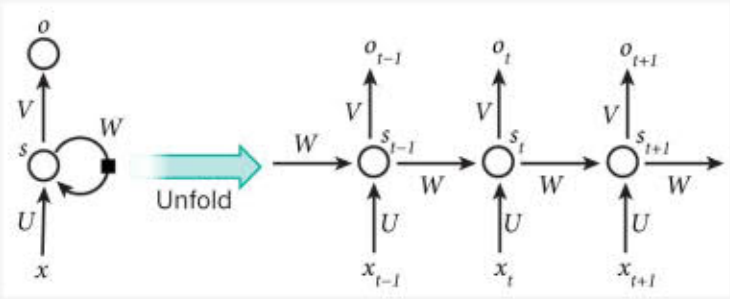
感知机



多层感知机



卷积神经网络



循环神经网络

真正的神经元学习

- 突触通过经验改变尺寸和连接强度
- Hebbian学习：如果两个连接在一起的神经元同时被激发，那么他们之间的连接会变强
- “Neurons that fire together, wire together.”
- 这些都推动了人工神经网络的研究

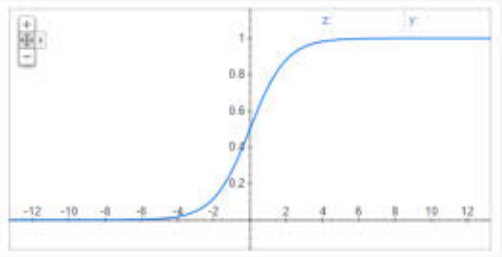
Slides credit: Ray Mooney

5.2 激活函数与损失函数

非线性激活函数

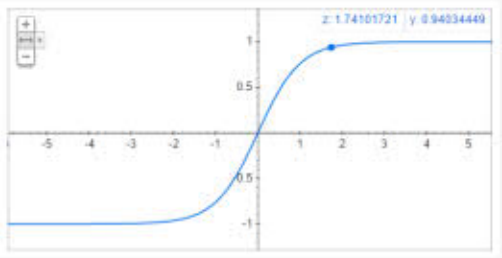
- Sigmoid激活函数

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$



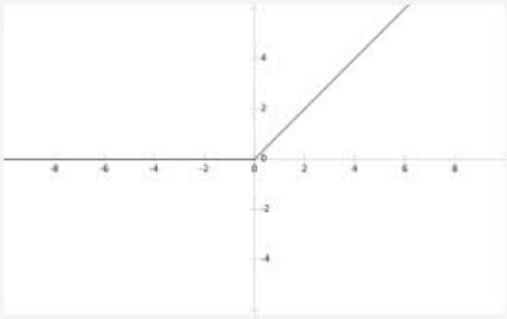
- Tanh激活函数

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



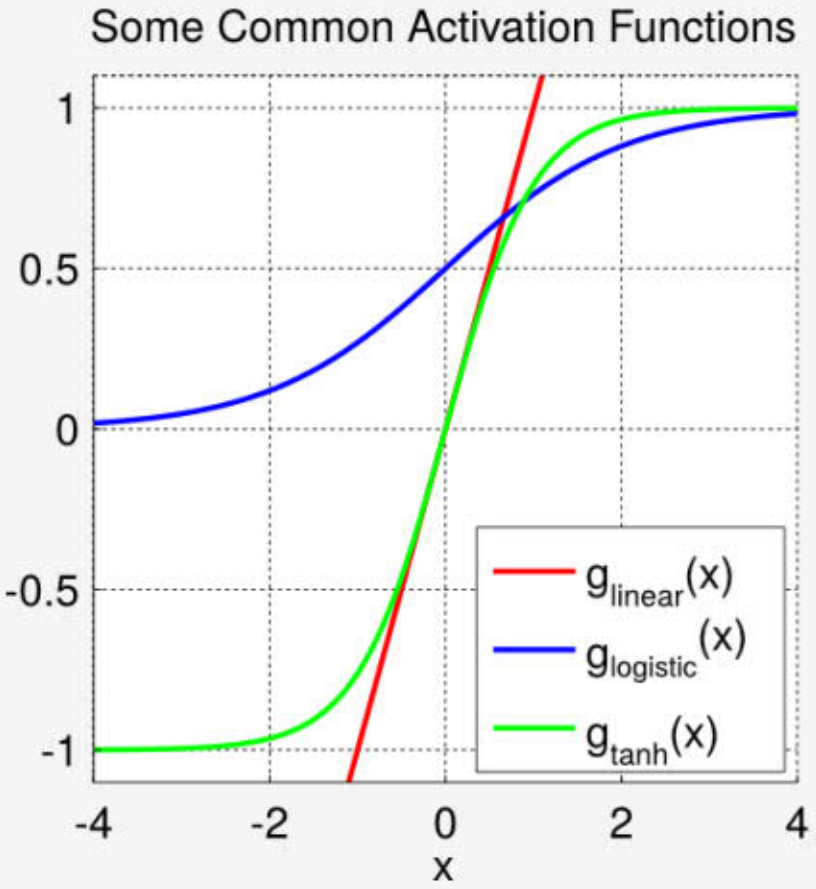
- 线性整流函数（Rectified Linear Unit, ReLU）

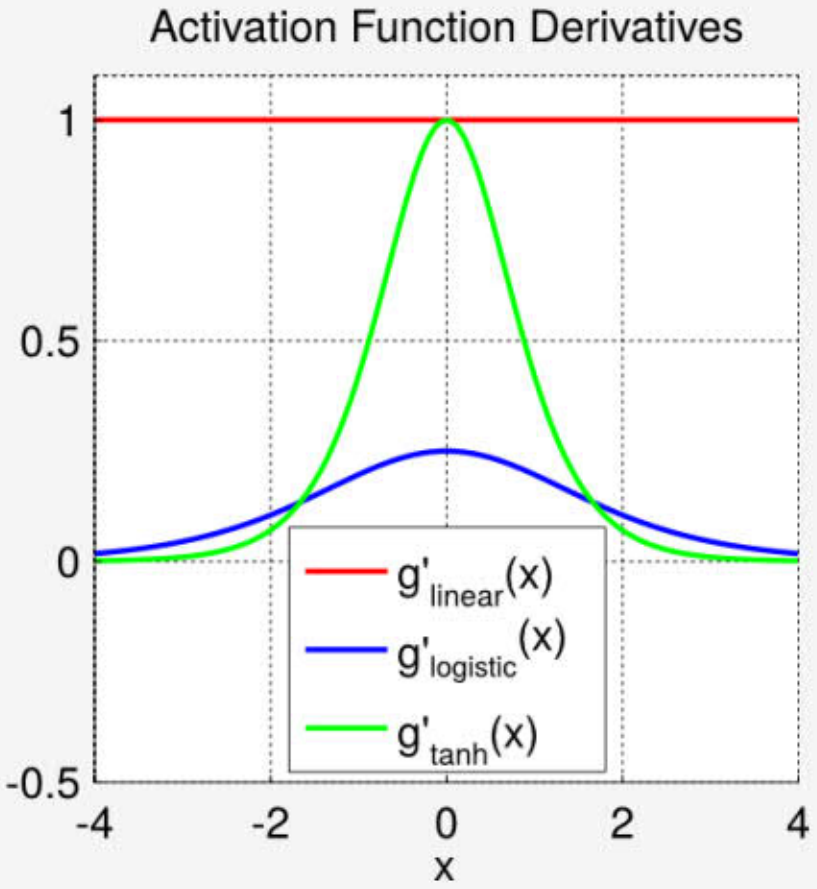
$$ReLU(z) = max(0, z)$$



5.2 激活函数与损失函数

激活函数





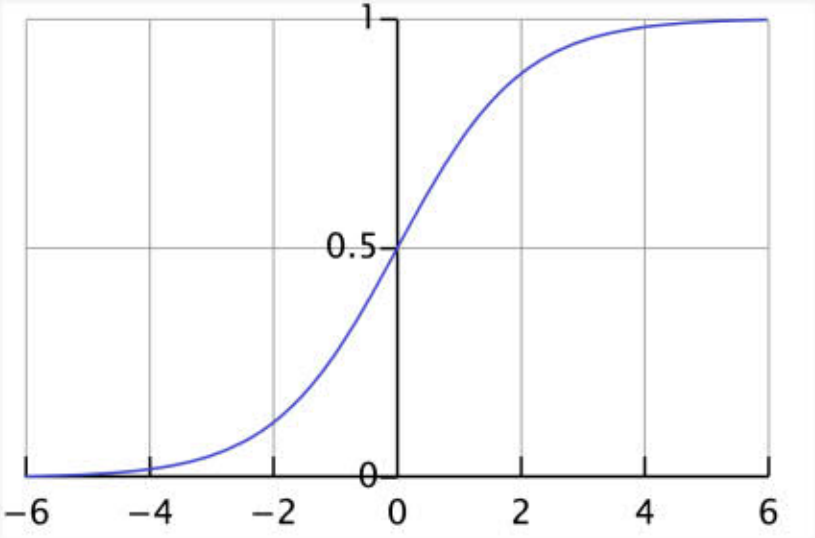
https://dustinstansbury.github.io/theclevermachine/derivation-common-neural-network-activation-functions

5.2 激活函数与损失函数

激活函数

逻辑斯谛Sigmoid函数

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



$$\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1-\sigma(z))$$

- 导数：

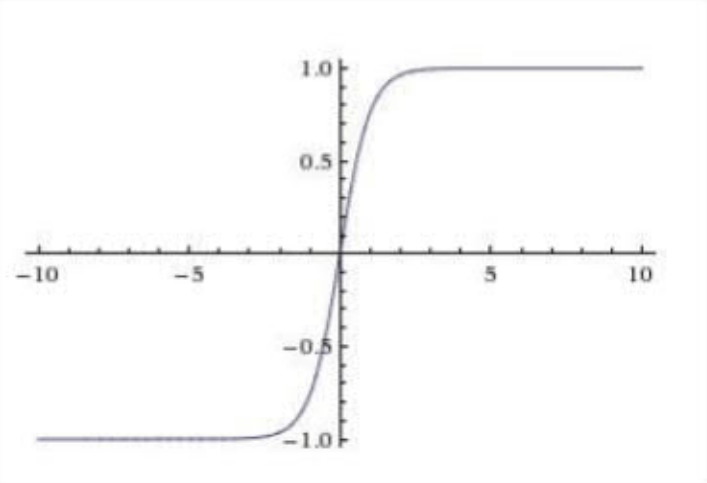
$$f'_{Sigmoid}(x) = f_{Sigmoid}(x)(1-f_{Sigmoid}(x))$$

- 输出范围 [0,1]
- 受生物神经元的启发产生，可以被看做一个人工神经元在当前输入下被“激活”的概率
- 边界值会使梯度消失（为什么？）

激活函数

Tanh函数

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



- 导数：

$$f'_{tanh}(x) = 1 - f_{tanh}(x)^2$$

- 负值较大的输入经过tanh函数会映射为负输出
- 只有接近零的输入会被映射为接近零的输出
- 使网络训练时被“卡住”的可能性降低

https://dustinstansbury.github.io/theclevermachine/derivation-common-neural-network-activation-functions

5.2 激活函数与损失函数

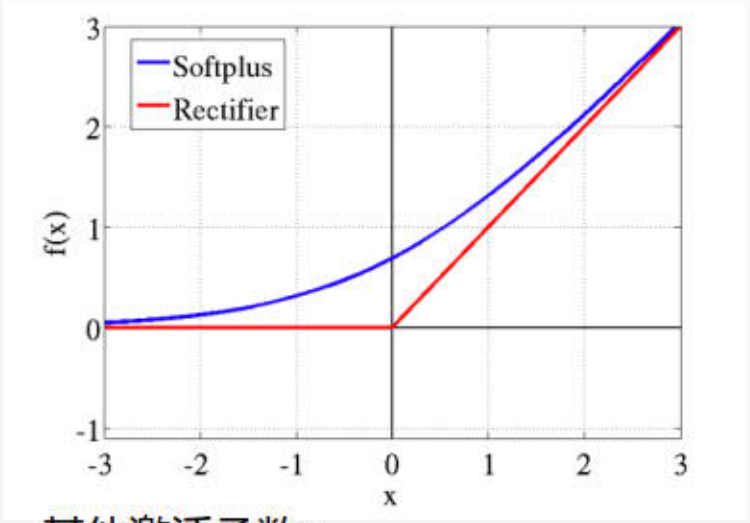
激活函数

ReLU函数(rectified linear unit)

$$f_{ReLU}(x) = max(0, x)$$

ReLU可以被近似为softplus函数

$$f_{Softplus}(x) = log(1 + e^x)$$



其他激活函数：

Leaky ReLU, Exponential LU, Maxout 等

- 导数：

$$f'_{ReLU}(x) = \begin{cases} 1 & if\ x > 0 \\ 0 & if\ x \leq 0 \end{cases}$$

- 另一个版本Noise ReLU：

$$f_{NoisyReLU}(x) = max(0, x + N(0, \delta(x)))$$

- ReLU可以被近似为softplus函数

$$f_{Softplus}(x) = log(1 + e^x)$$

- x增加时ReLU的梯度不会消失
- 可以用来对正值输入进行建模
- 由于无需计算指数函数，所以它的计算速度很快
- 使用它可以不再需要“预训练”过程

http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/40811.pdf

5.2 激活函数与损失函数

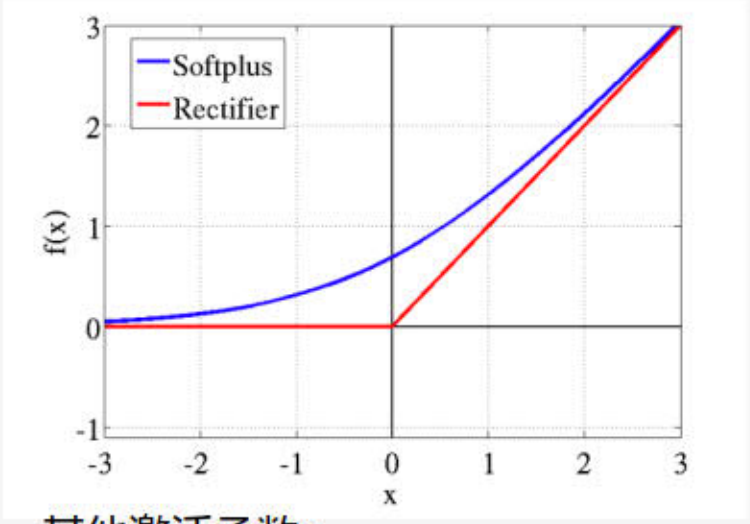
激活函数

ReLU函数(rectified linear unit)

$$f_{ReLU}(x) = max(0, x)$$

ReLU可以被近似为softplus函数

$$f_{Softplus}(x) = log(1 + e^x)$$

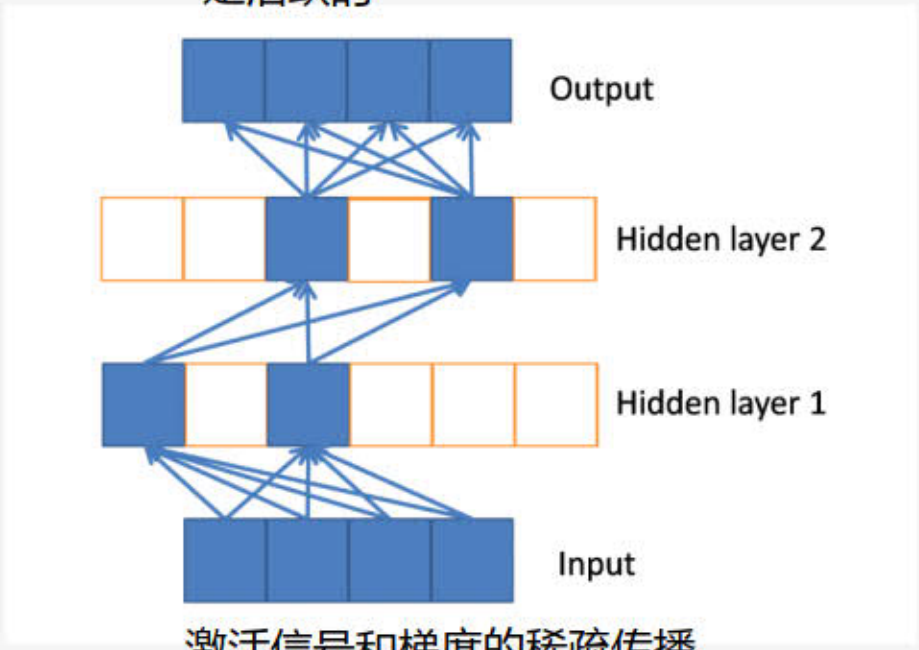


其他激活函数：

Leaky ReLU, Exponential LU, Maxout 等

http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

- 非线性取决于激活或者未激活的神经元构成的传播路径的选择
- 允许稀疏表示：
  - 对于特定的输入，只有一部分神经元是活跃的



激活信号和梯度的稀疏传播

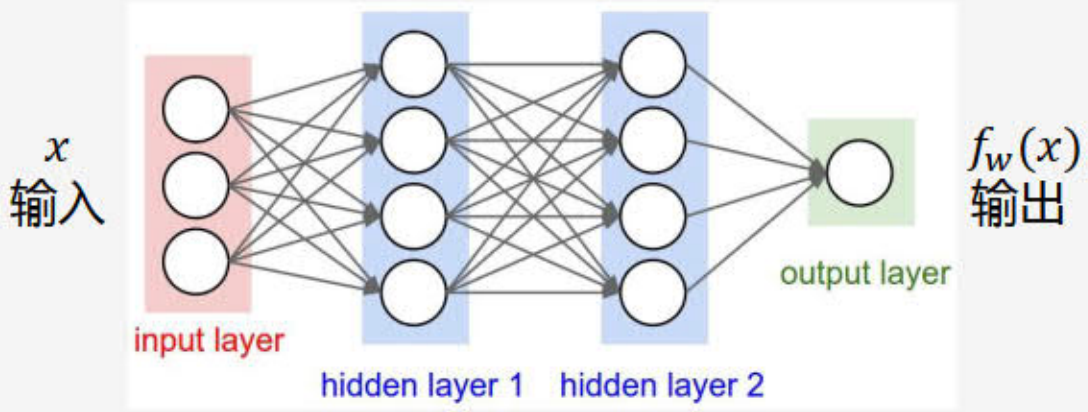
误差/损失函数

- 随机梯度下降
  - 从随机抽取的样本中进行更新（实际上执行批处理更新）：

$$w = w - \eta \frac{\partial \mathcal{L}(w)}{\partial w}$$

- 一个二值输入的平方误差损失：

$$\mathcal{L}(w) = \frac{1}{2}(y - f_w(x))^2$$



https://github.com/shahidmawan/Machine-Learning/blob/master/BP_chapter3_-_bp.pdf

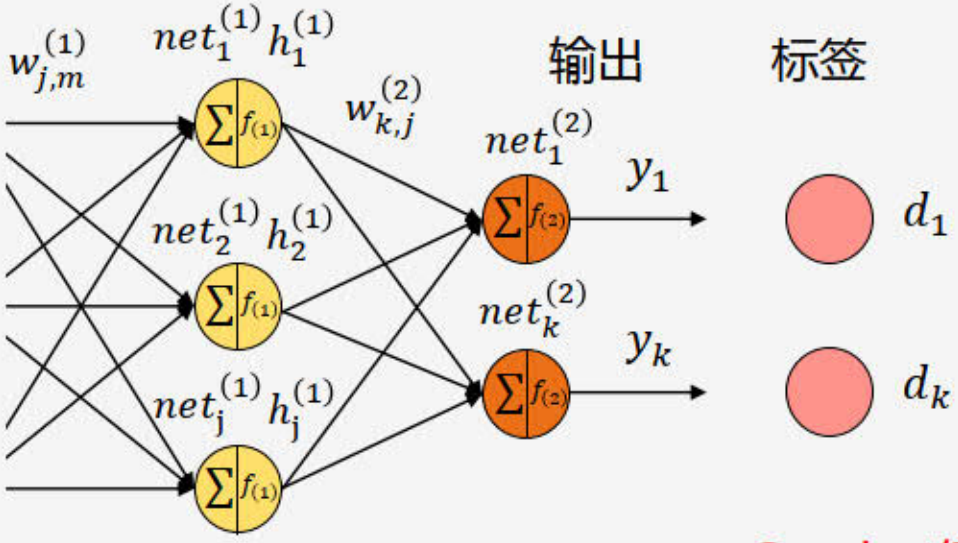
5.2 激活函数与损失函数

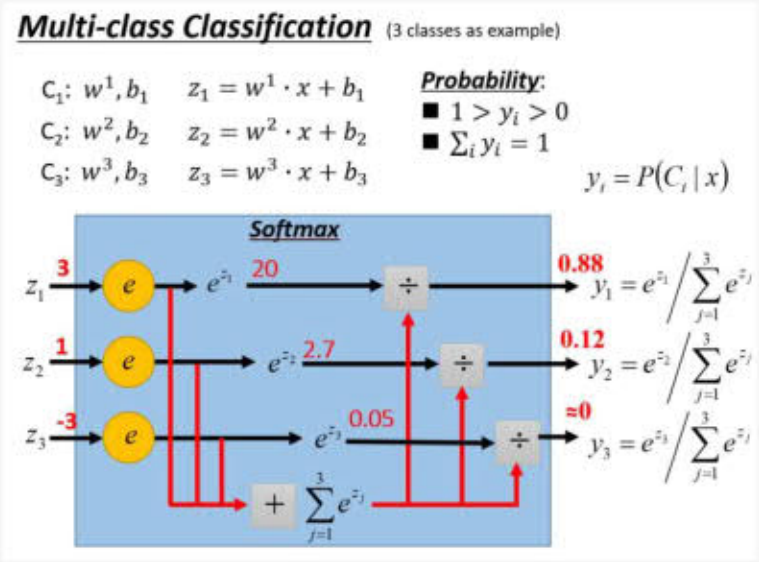
误差/损失函数

多分类问题的Softmax损失（交叉熵损失）

（类标签服从多项分布）

$$\mathcal{L}(w) = -\sum_{k} d_k log\hat{y}_k \qquad 其中 \quad \hat{y}_k = \frac{exp(\sum_j w_{k,j}^{(2)} h_j^{(1)})}{\sum_{k'} exp\left(\sum_j w_{k',j}^{(2)} h_j^{(1)}\right)}$$



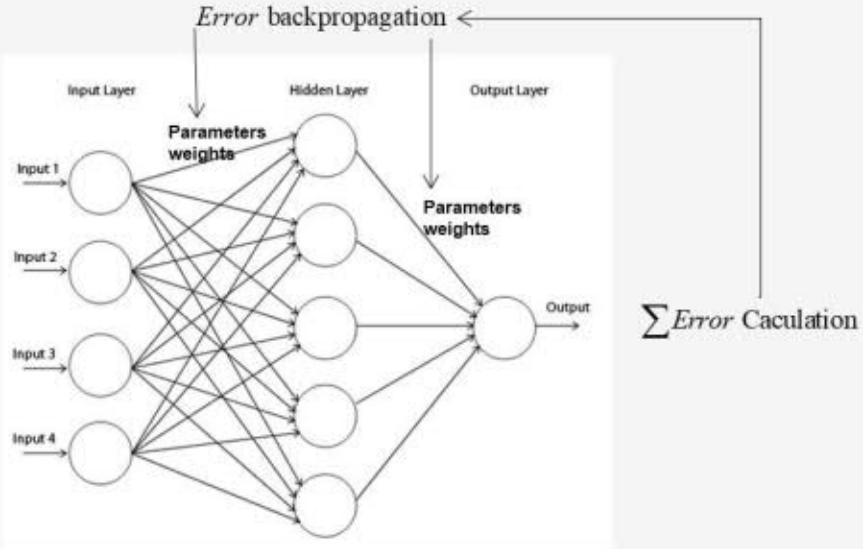


One hot编码的类标签

5.3 反向传播算法介绍

隐含层和反向传播（1986~）

- 多层神经网络的有效算法之一是反向传播算法
- 它在1986年被重新提出，神经网络重新变得流行



注意：反向传播算法最开始在1974年被Werbos[1]发现，之后在1985年左右被Rumelhart, Hinton，Williams[2] 以及Parker[3] 分别独立重新发现

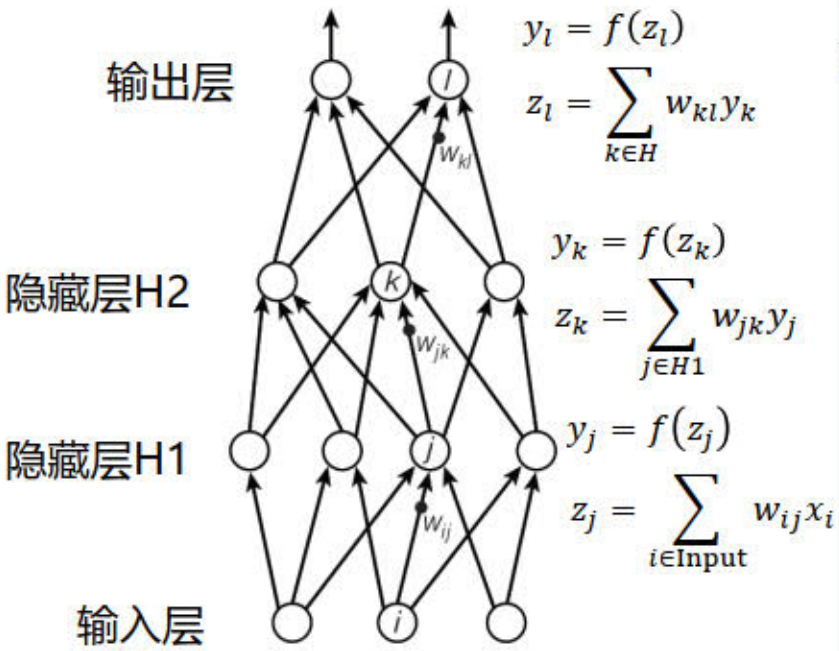
[1] Werbos, Paul John (1975). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. Harvard University.
[2] Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. (8 October 1986). "Learning representations by back-propagating errors". Nature. 323 (6088): 533–536.
[3] Parker, D. B. (1985). "Learning Logic," Technical Report TR-47, Center for Computational Research in Economics and Management Science, Massachusetts Institute of Technology, Cambridge, MA.

5.3 反向传播算法介绍

通过反向传播学习神经网络

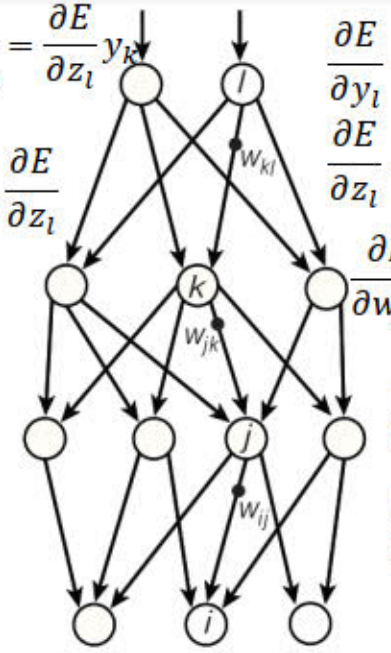
输入数据逐层计算得到输出

将输出与正确答案比较得到误差，然后反向求导数



$$y_l = f(z_l)$$
$$z_l = \sum_{k \in H} w_{kl} y_k$$
$$y_k = f(z_k)$$
$$z_k = \sum_{j \in H1} w_{jk} y_j$$
$$y_j = f(z_j)$$
$$z_j = \sum_{i \in \text{Input}} w_{ij} x_i$$

$$\frac{\partial E}{\partial w_{kl}} = \frac{\partial E}{\partial z_l}\frac{\partial z_l}{\partial w_{kl}} = \frac{\partial E}{\partial z_l} y_k$$
$$\frac{\partial E}{\partial y_k} = \sum_{l \in \text{Out}} w_{kl}\frac{\partial E}{\partial z_l}$$
$$\frac{\partial E}{\partial z_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial z_k}$$

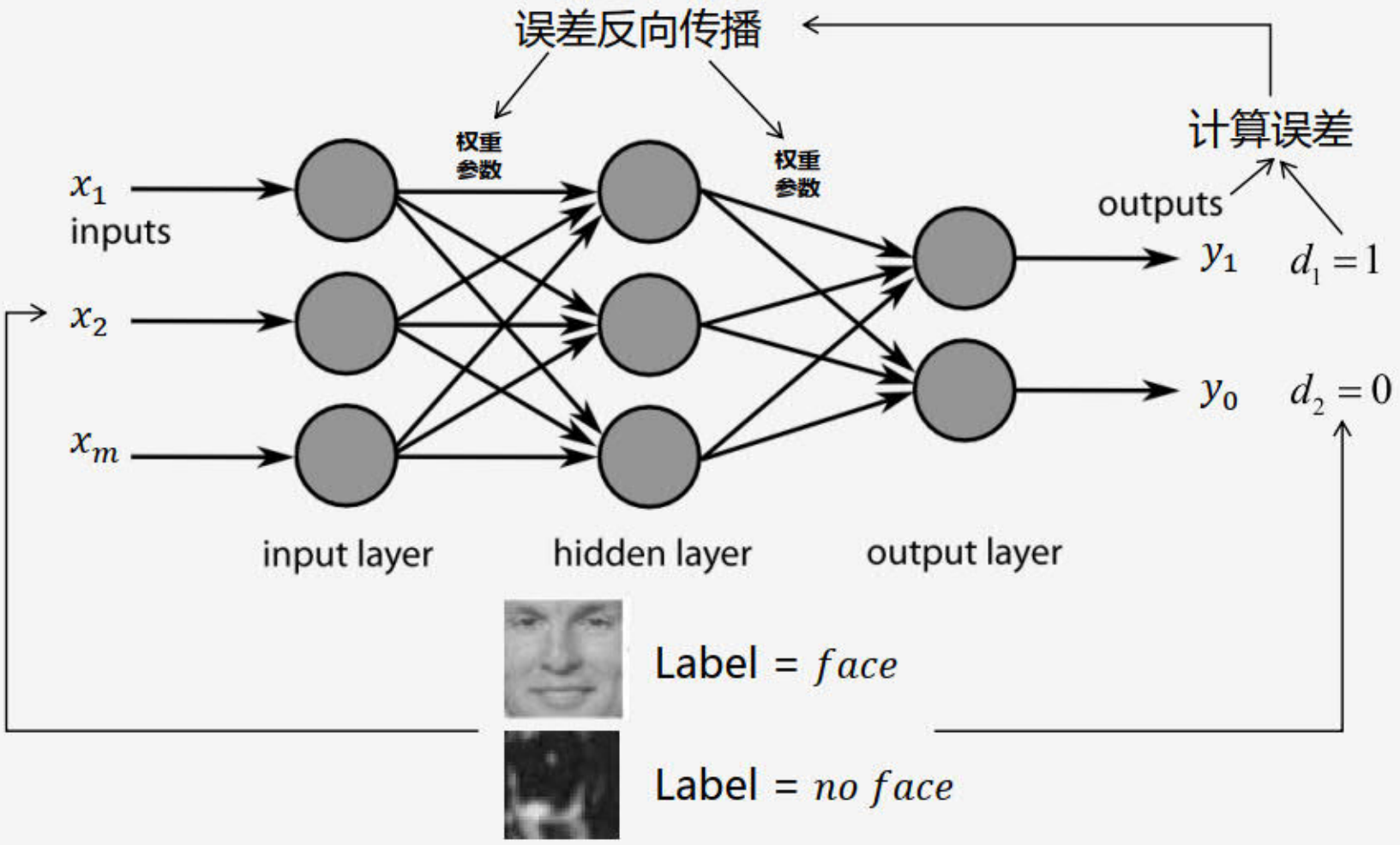


$$\frac{\partial E}{\partial y_l} = y_l - t_l$$
$$\frac{\partial E}{\partial z_l} = \frac{\partial E}{\partial y_l}\frac{\partial y_l}{\partial z_l}$$
$$\frac{\partial E}{\partial w_{jk}} = \frac{\partial E}{\partial z_k}\frac{\partial z_k}{\partial w_{jk}} = \frac{\partial E}{\partial z_k} y_j$$
$$\frac{\partial E}{\partial y_j} = \sum_{k \in H2} w_{jk}\frac{\partial E}{\partial z_k}$$
$$\frac{\partial E}{\partial z_j} = \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_j}$$

[LeCun, Bengio and Hinton. Deep Learning. Nature 2015.]

5.3 反向传播算法介绍

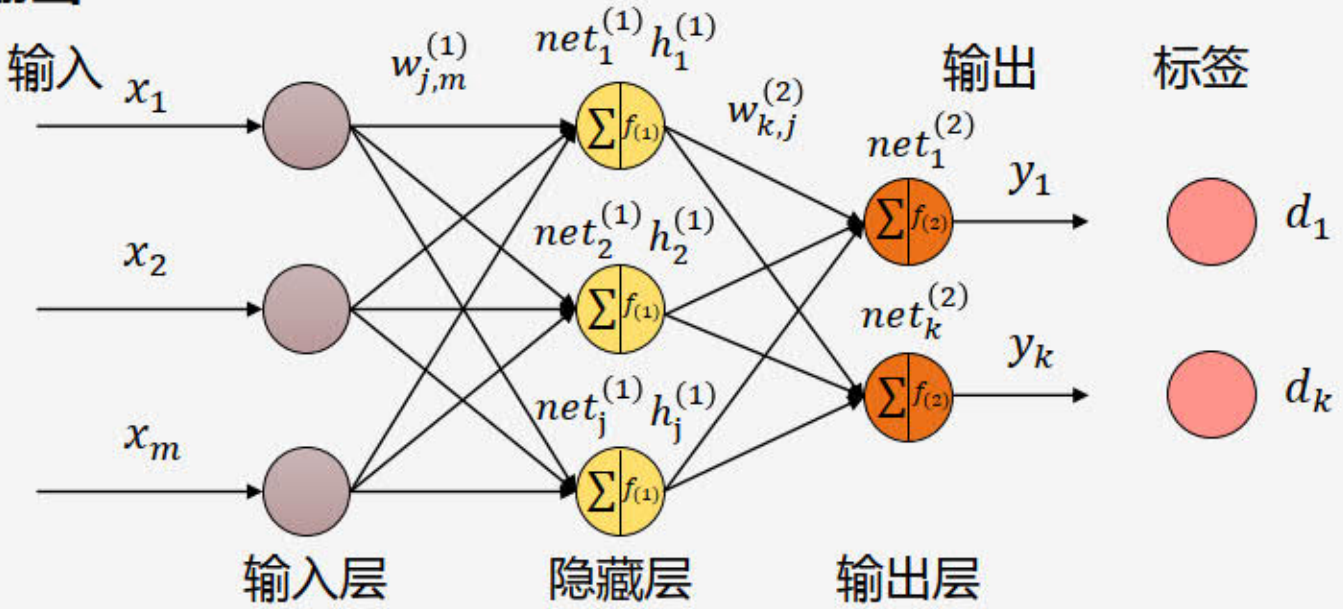
通过反向传播学习神经网络



训练实例

5.3 反向传播算法介绍

预测输出



双层前馈神经网络

前馈预测

$$h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}(\sum_m w_{j,m}^{(1)} x_m) \qquad y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}(\sum_j w_{k,j}^{(2)} h_j^{(1)})$$

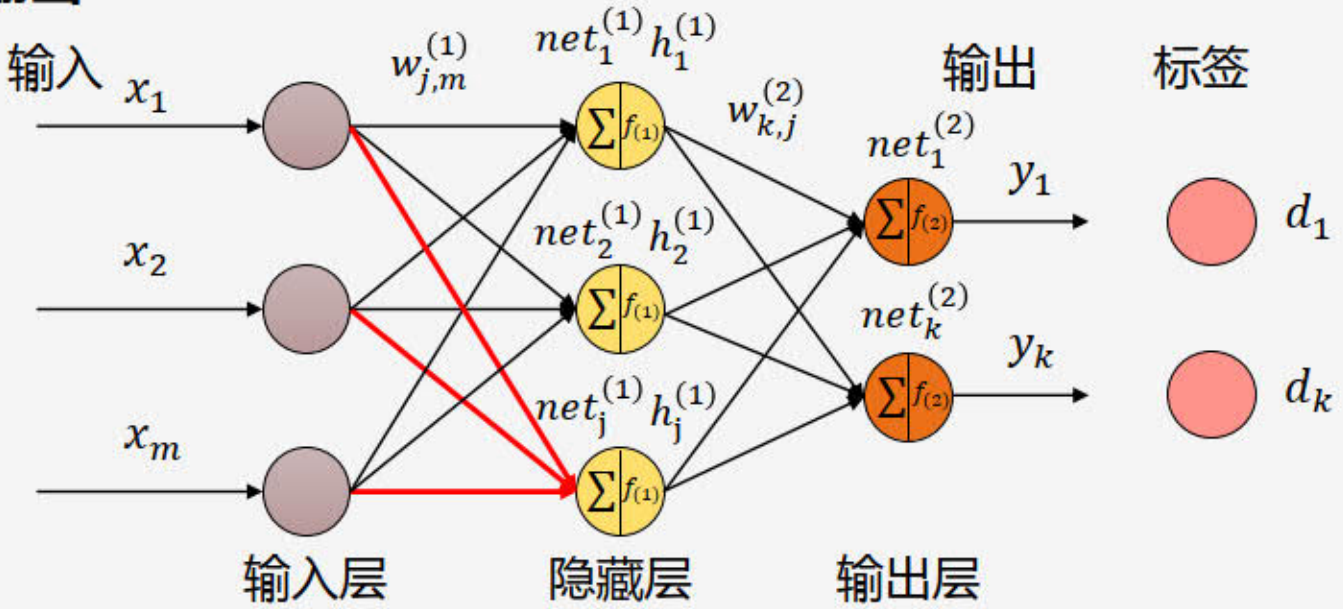
$$x = (x_1, \dots, x_m) \longrightarrow h_j^{(1)} \longrightarrow y_k$$

其中，

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m \qquad net_k^{(2)} = \sum_m w_{k,j}^{(2)} h_j^{(1)}$$

5.3 反向传播算法介绍

预测输出



双层前馈神经网络

前馈预测

$$h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}(\sum_m w_{j,m}^{(1)} x_m) \qquad y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}(\sum_j w_{k,j}^{(2)} h_j^{(1)})$$

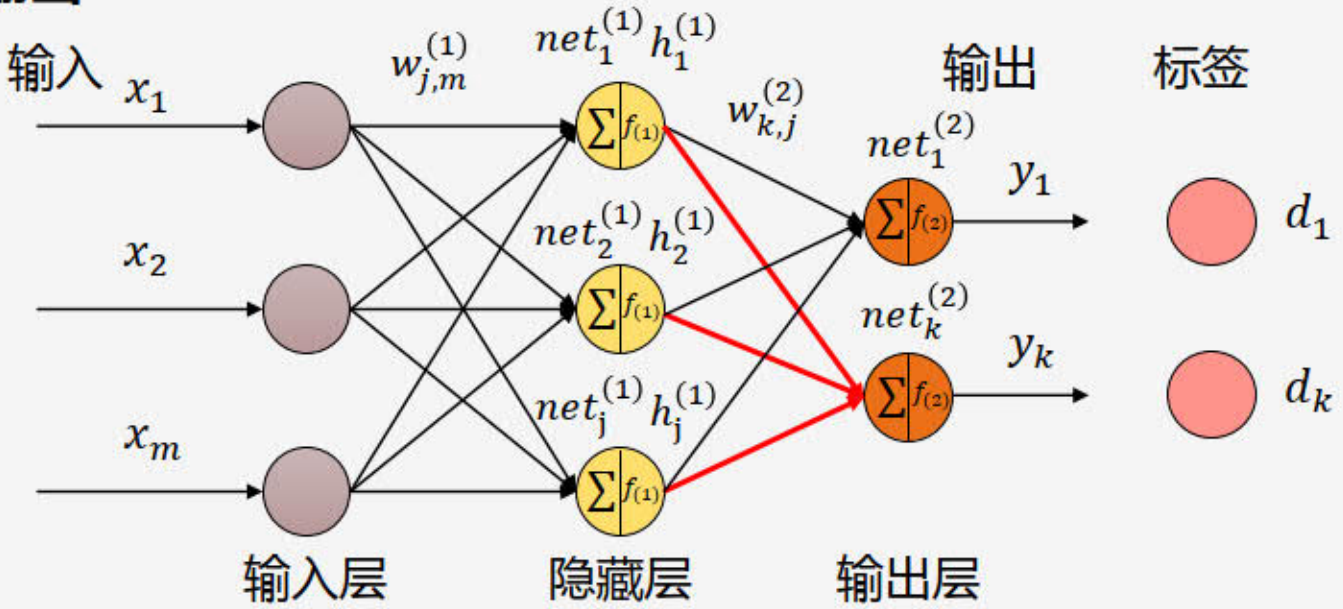
$$x = (x_1, \dots, x_m) \longrightarrow h_j^{(1)} \longrightarrow y_k$$

其中，

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m \qquad net_k^{(2)} = \sum_m w_{k,j}^{(2)} h_j^{(1)}$$

5.3 反向传播算法介绍

预测输出



双层前馈神经网络

前馈预测

$$h_j^{(1)} = f_{(1)}(net_j^{(1)}) = f_{(1)}(\sum_m w_{j,m}^{(1)} x_m) \qquad y_k = f_{(2)}(net_k^{(2)}) = f_{(2)}(\sum_j w_{k,j}^{(2)} h_j^{(1)})$$

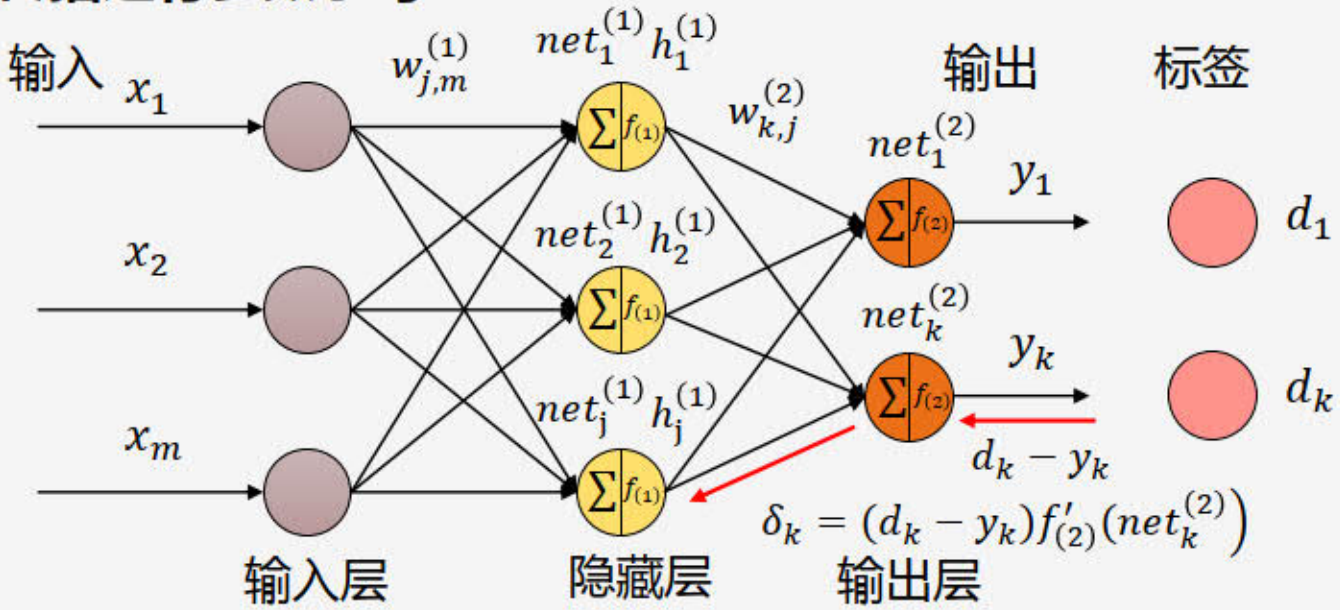
$$x = (x_1, \dots, x_m) \longrightarrow h_j^{(1)} \longrightarrow y_k$$

其中，

$$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m \qquad net_k^{(2)} = \sum_m w_{k,j}^{(2)} h_j^{(1)}$$

5.3 反向传播算法介绍

反向传播进行参数学习



$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$ 双层前馈神经网络 $net_k^{(2)} = \sum_m w_{k,j}^{(2)} h_j^{(1)}$

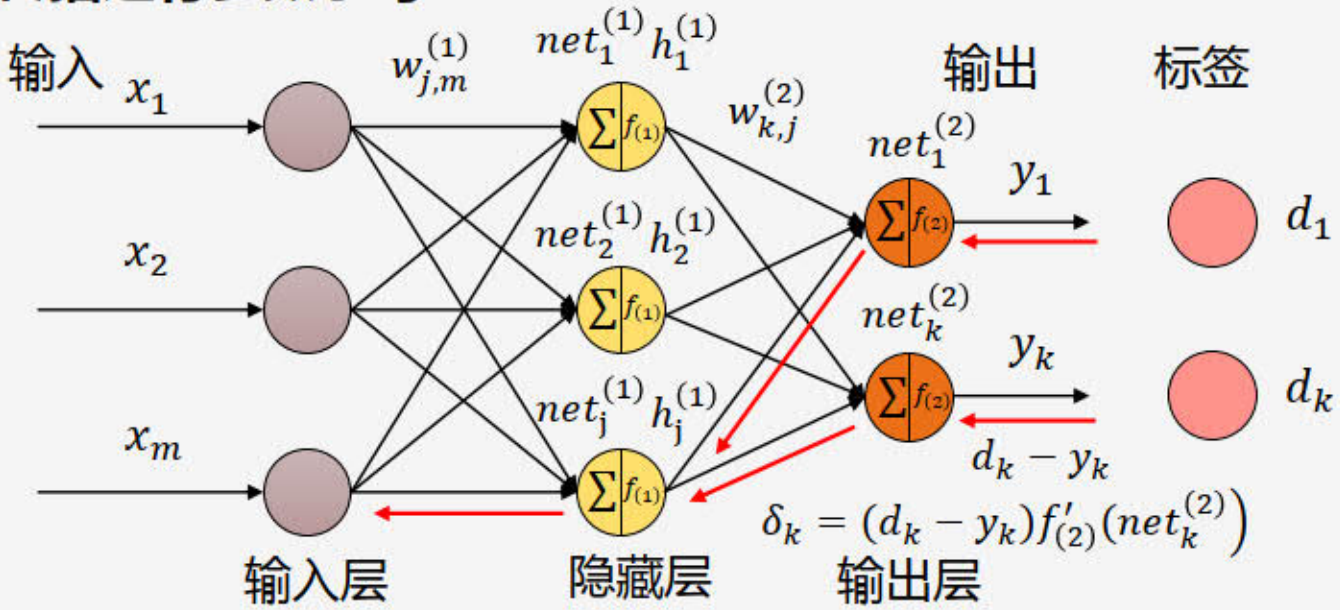
$$w_{k,j}^{(2)} = w_{k,j}^{(2)} + \Delta w_{k,j}^{(2)}$$

$$\Delta w_{k,j}^{(2)} = \eta Error_k \, Output_j = \eta \delta_k h_j^{(1)} \leftarrow E(W) = \frac{1}{2}\sum_k (y_k - d_k)^2$$

$$\Delta w_{k,j}^{(2)} = -\eta \frac{\partial E(W)}{\partial w_{k,j}^{(2)}} = -\eta (y_k - d_k) \frac{\partial y_k}{\partial net_k^{(2)}} \frac{\partial net_k^{(2)}}{\partial w_{k,j}^{(2)}} = \eta (d_k - y_k) f'_{(2)}(net_k^{(2)}) h_j^{(1)} = \eta \delta_k h_j^{(1)}$$

5.3 反向传播算法介绍

反向传播进行参数学习



$net_j^{(1)} = \sum_m w_{j,m}^{(1)} x_m$ 双层前馈神经网络 $net_k^{(2)} = \sum_m w_{k,j}^{(2)} h_j^{(1)}$

$$w_{j,m}^{(1)} = w_{j,m}^{(1)} + \Delta w_{j,m}^{(1)}$$

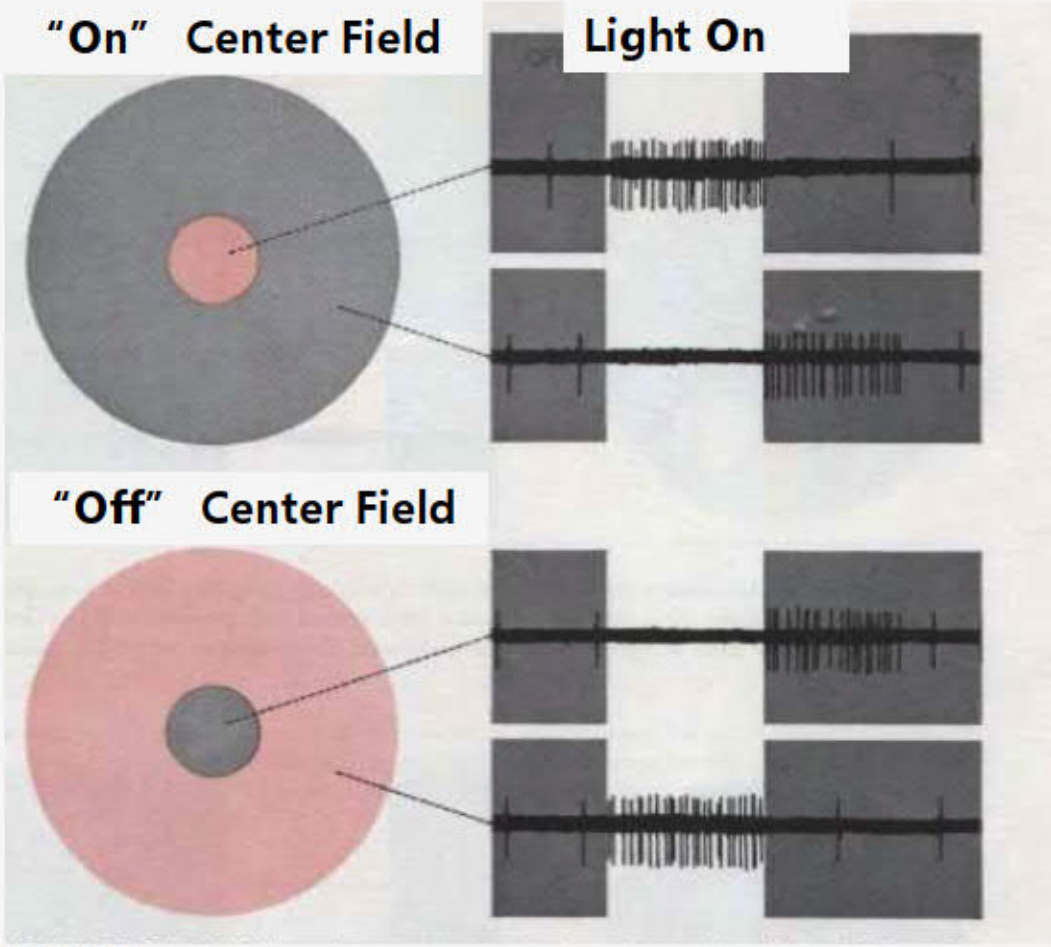
$\Delta w_{j,m}^{(1)} = \eta Error_j \ Out_m = \eta \delta_j x_m$ ← $E(W) = \frac{1}{2}\sum_k (y_k - d_k)^2$

$$\Delta w_{j,m}^{(1)} = -\eta \frac{\partial E(W)}{\partial w_{j,m}^{(1)}} = -\eta \frac{\partial E(W)}{\partial h_j^{(1)}} \frac{\partial h_j^{(1)}}{\partial w_{j,m}^{(1)}} = \eta \sum_k (d_k - y_k) f'_{(2)}(net_k^{(2)}) w_{k,j}^{(2)} x_m f'_{(1)}(net_j^{(1)}) = \eta \delta_j x_m$$

卷积神经网络：感受野

感受野

- 视网膜中的神经元只响应视野的受限区域中的光刺激
- 两个视网膜神经节细胞感受野的动物实验
  - 视野是视网膜的圆形区域
  - 当中心点亮周围变暗时，细胞（上部）响应
  - 当中心变暗并且周围照亮时，细胞（下部）响应
  - 当中心和周围都被照亮时，两个单元都给出响应和关闭响应，但都没有响应像只有中心或周围被照亮时那样强烈

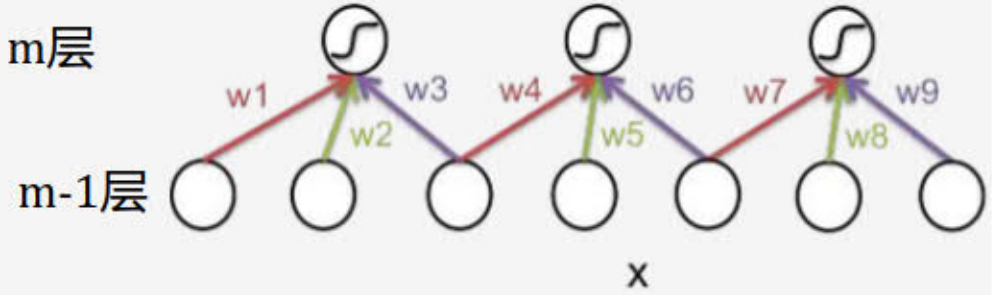


Hubel D.H. : The Visual Cortex of the Brain Sci Amer 209:54-62, 1963

卷积神经网络（Convolutional Neural Network）

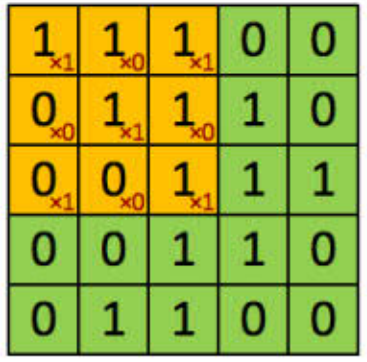
- 通过局部相关性稀疏连接
  - 过滤器：m层中隐含单元的输入来自m-1层中具有空间连接的感受野的单元的子集
- 共享权重
  - 每个过滤器都在整个视野中复制。 这些复制的单元共享相同的权重并形成特征映射。

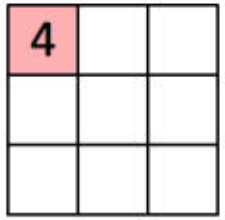
一维情况



相同颜色的边权重相同

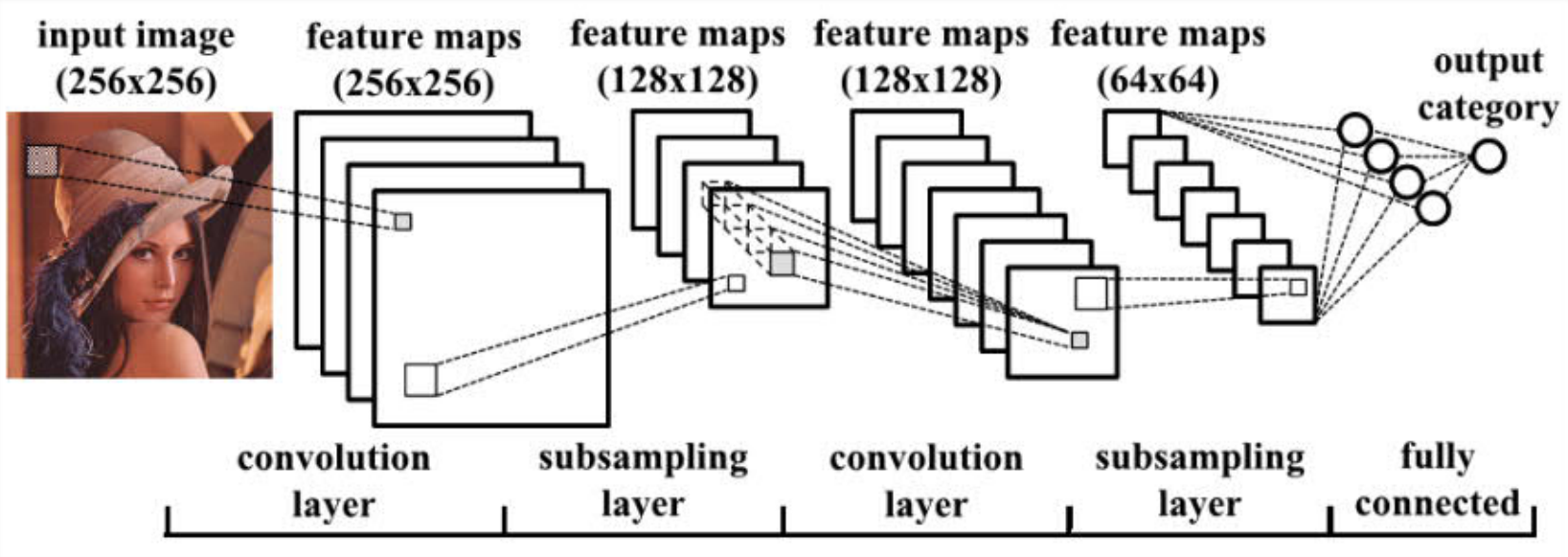
二维情况(下标是权重)





https://deeplearningtutorials.readthedocs.io/en/latest/lenet.html

卷积神经网络

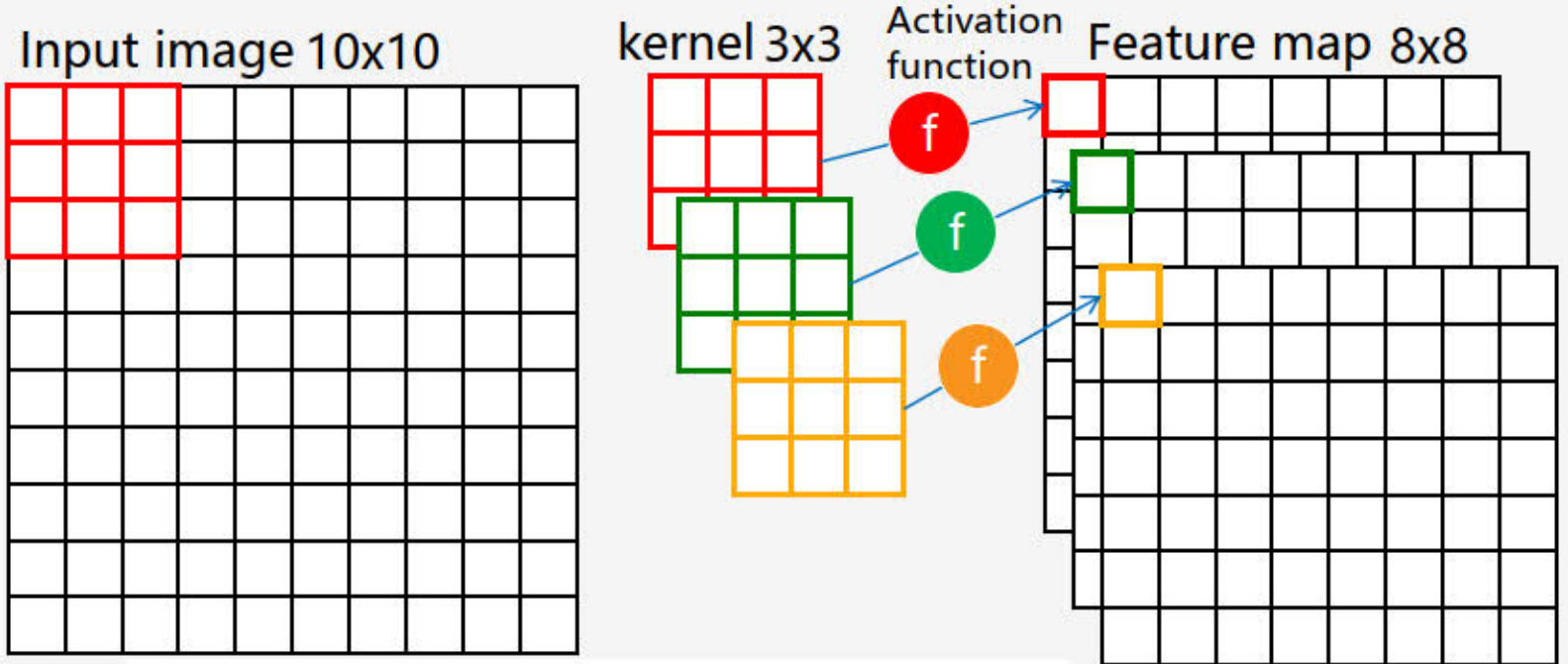


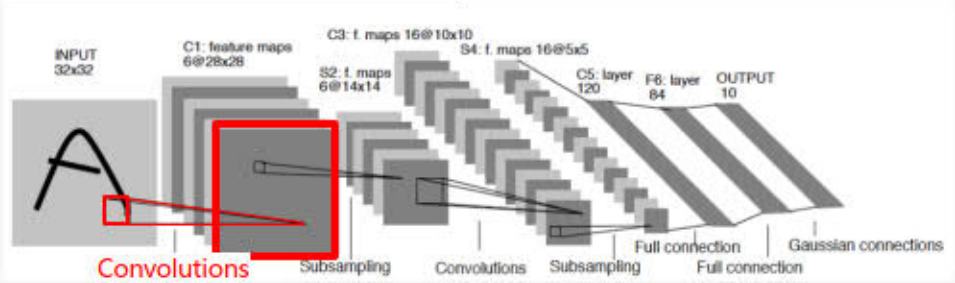
Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11) 1998

5.4 卷积神经网络

卷积层

- 范例：具有3x3过滤器的10x10输入图像产生8x8输出图像
- 三个不同的过滤器(权重不同)产生3张8 × 8的图像





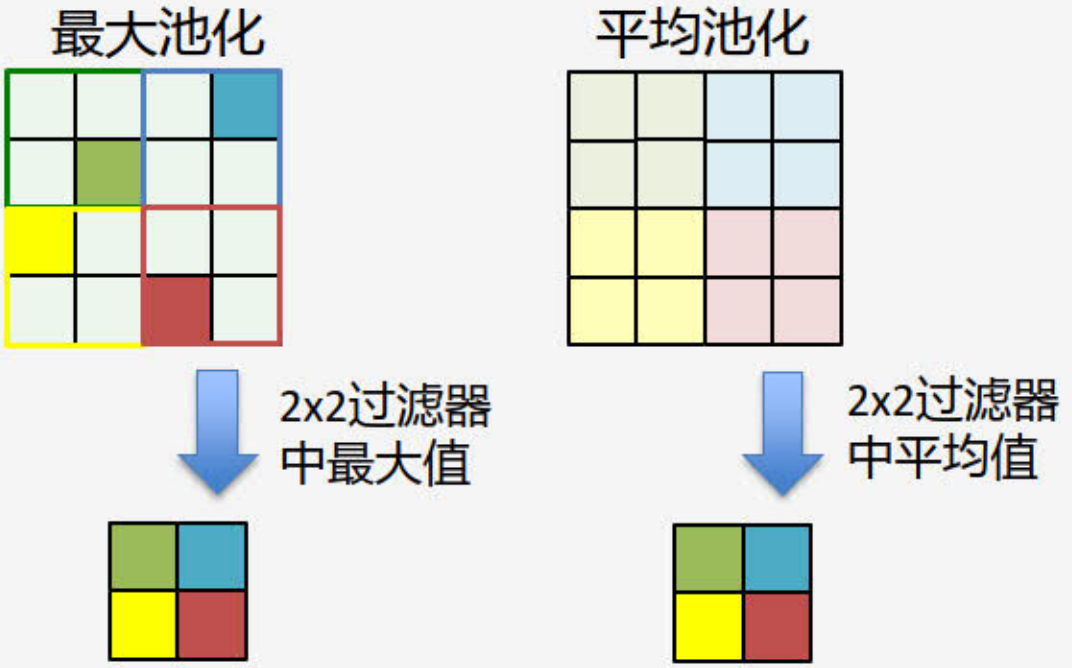
下采样池化层

池化

- 将输入图像划分为一组非重叠矩形，并且对于每个这样的子区域，输出最大值或平均值

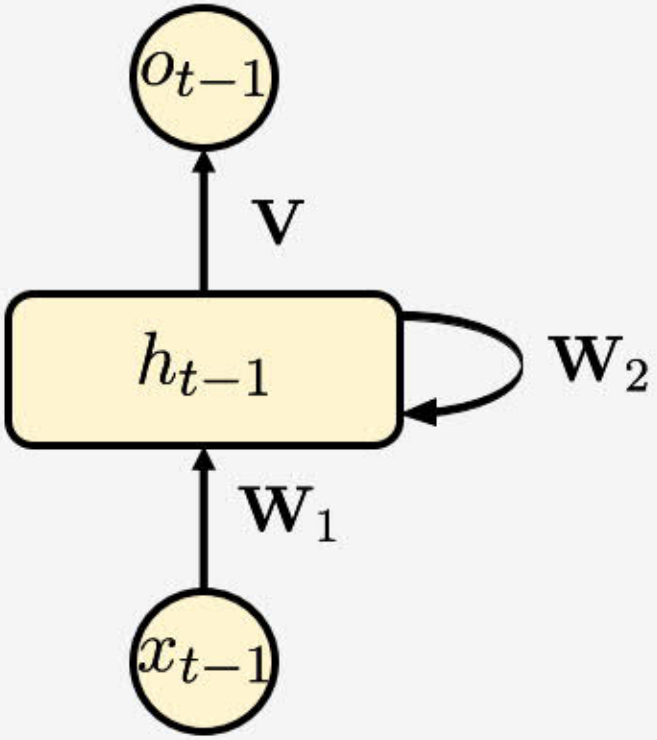
最大池化

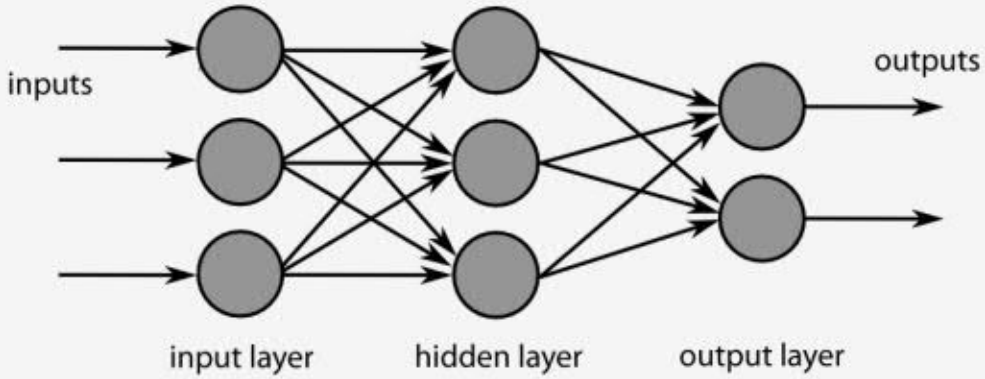
- 优点
  - 减少计算
  - 是一种获取给定兴趣区域响应最强烈的节点的方法
- 缺点
  - 可能会导致准确的空间信息丢失



循环神经网络（Recurrent Neural Network）

- 从前馈神经网络到循环神经网络
  - 前馈神经网络一个有向无环图，无自连边
  - 循环神经网络有一条自连的边，将隐层信息随着时间步往前传



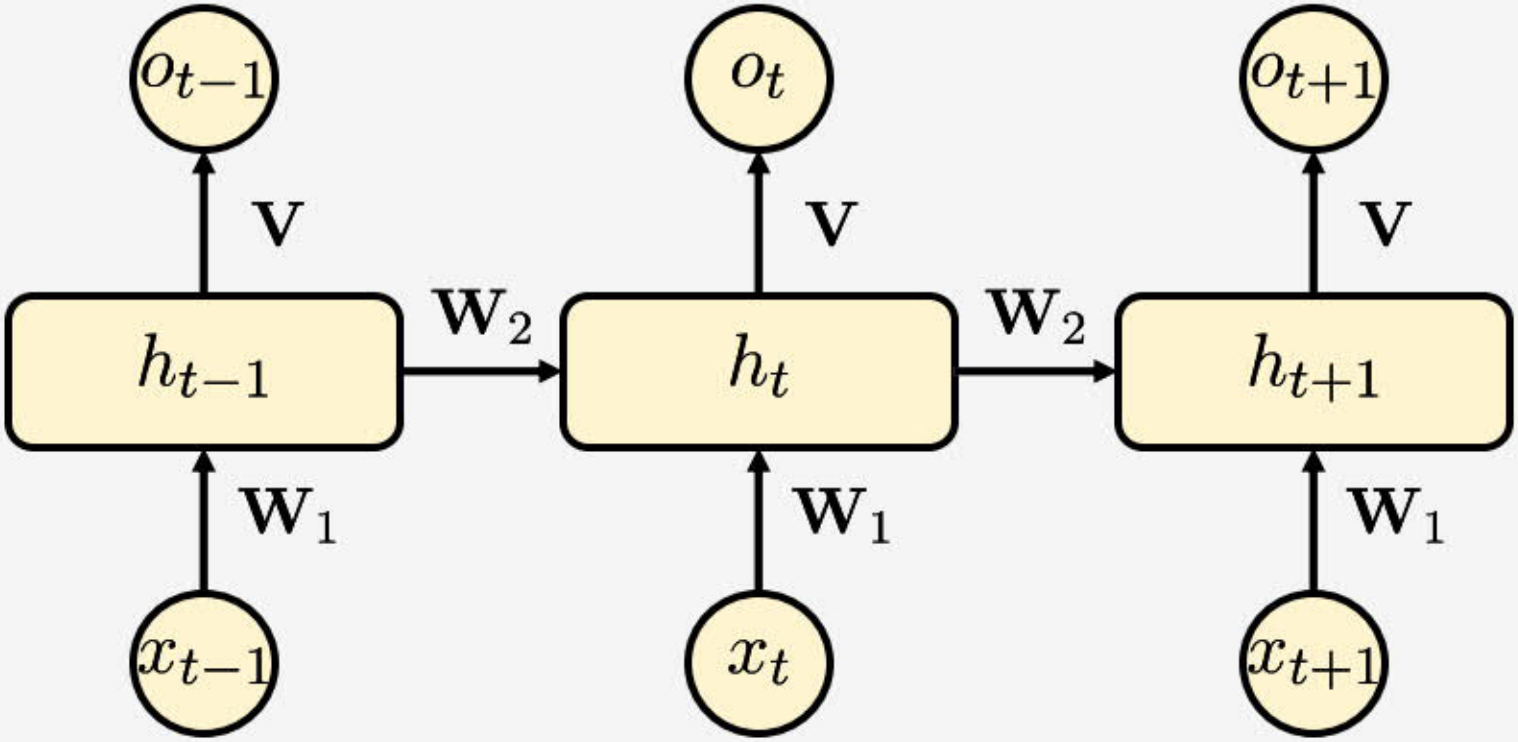


前馈神经网络

https://dennybritz.com/posts/wildml/recurrent-neural-networks-tutorial-part-1/

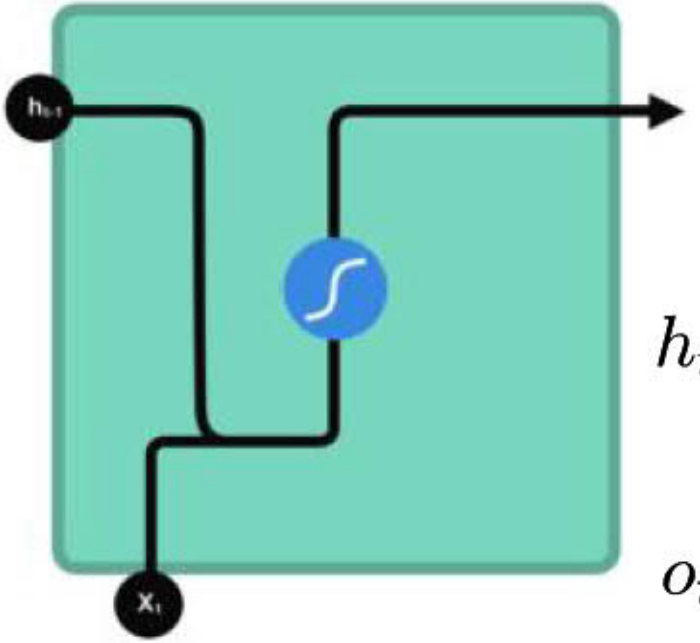
循环神经网络（Recurrent Neural Network）

- 从前馈神经网络到循环神经网络
  - 前馈神经网络一个有向无环图，无自连边
  - 循环神经网络有一条自连的边，将隐层信息随着时间步往前传



https://dennybritz.com/posts/wildml/recurrent-neural-networks-tutorial-part-1/

RNN工作原理图示



Tanh function

new hidden state

previous hidden state

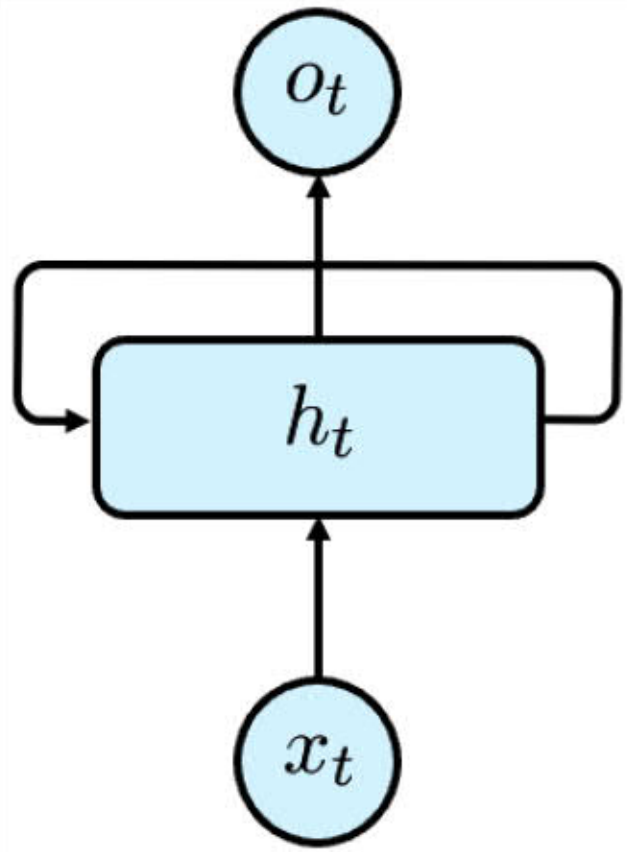
input

concatenation

$$h_t = \tanh(\mathbf{W}_1 x_t + \mathbf{W}_2 h_{t-1})$$
$$= \tanh(\mathbf{W}[x_t, h_{t-1}])$$
$$o_t = \sigma(\mathbf{V} h_t)$$

https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

RNN的工作方式

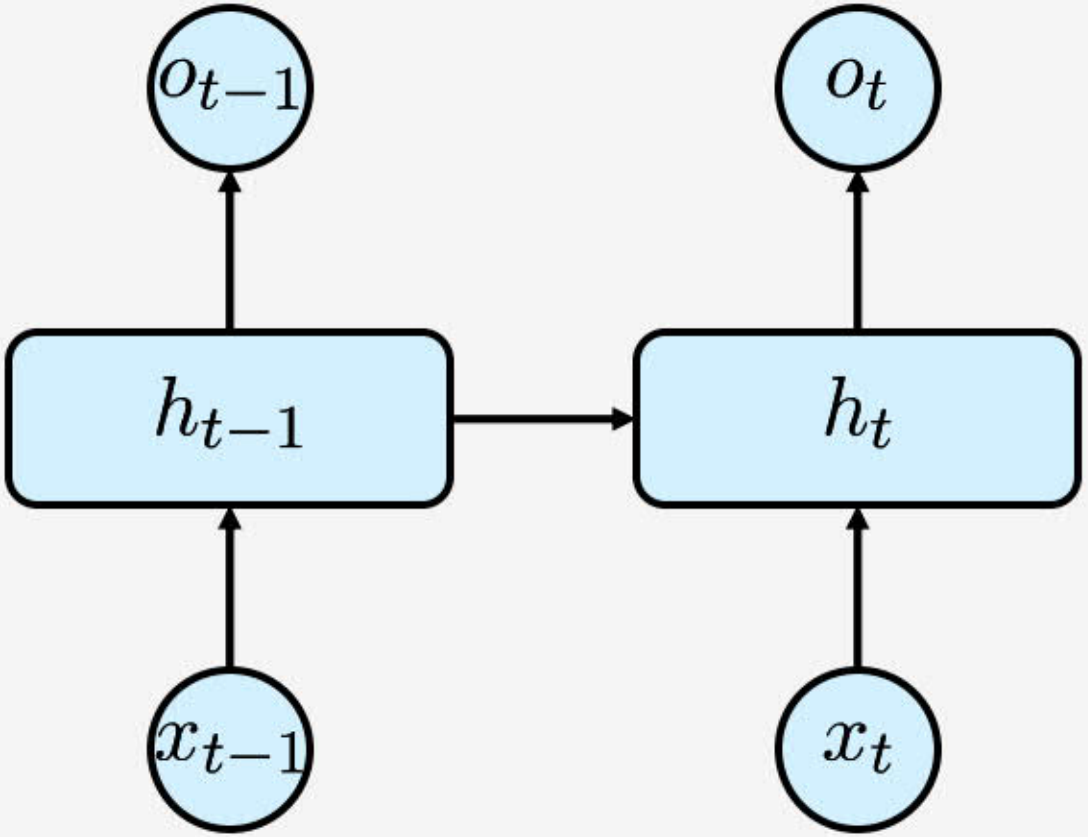


$$o_t = \sigma(\mathbf{V}h_t)$$

$$\begin{aligned} h_t &= \tanh(\mathbf{W}_1 x_t + \mathbf{W}_2 h_{t-1}) \\ &= \tanh(\mathbf{W}[x_t, h_{t-1}]) \end{aligned}$$

把 x_t, h_{t-1} 接起来

RNN的工作方式

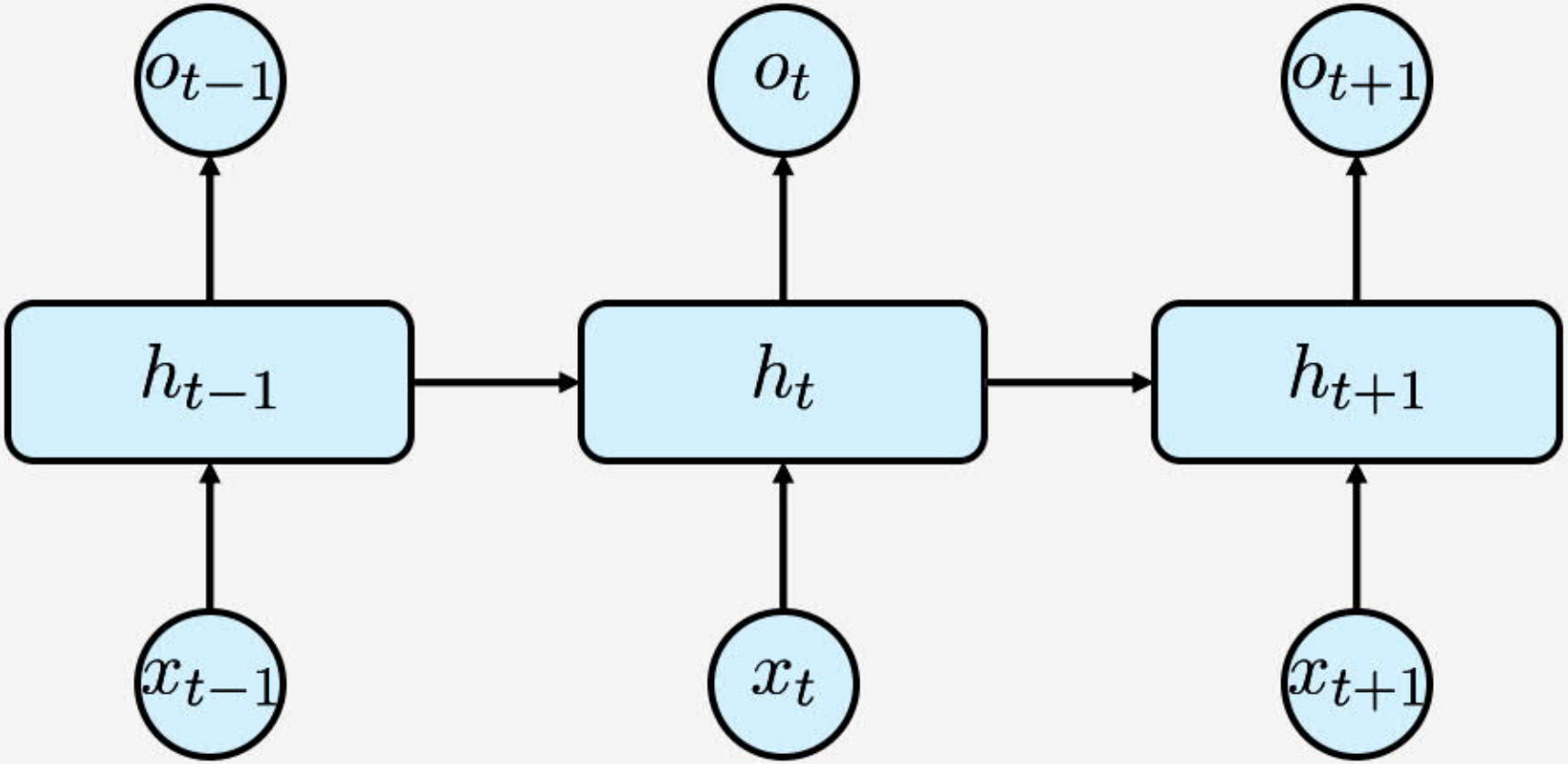


$$o_t = \sigma(\mathbf{V}h_t)$$

$$h_t = \tanh(\mathbf{W}[x_t, h_{t-1}])$$

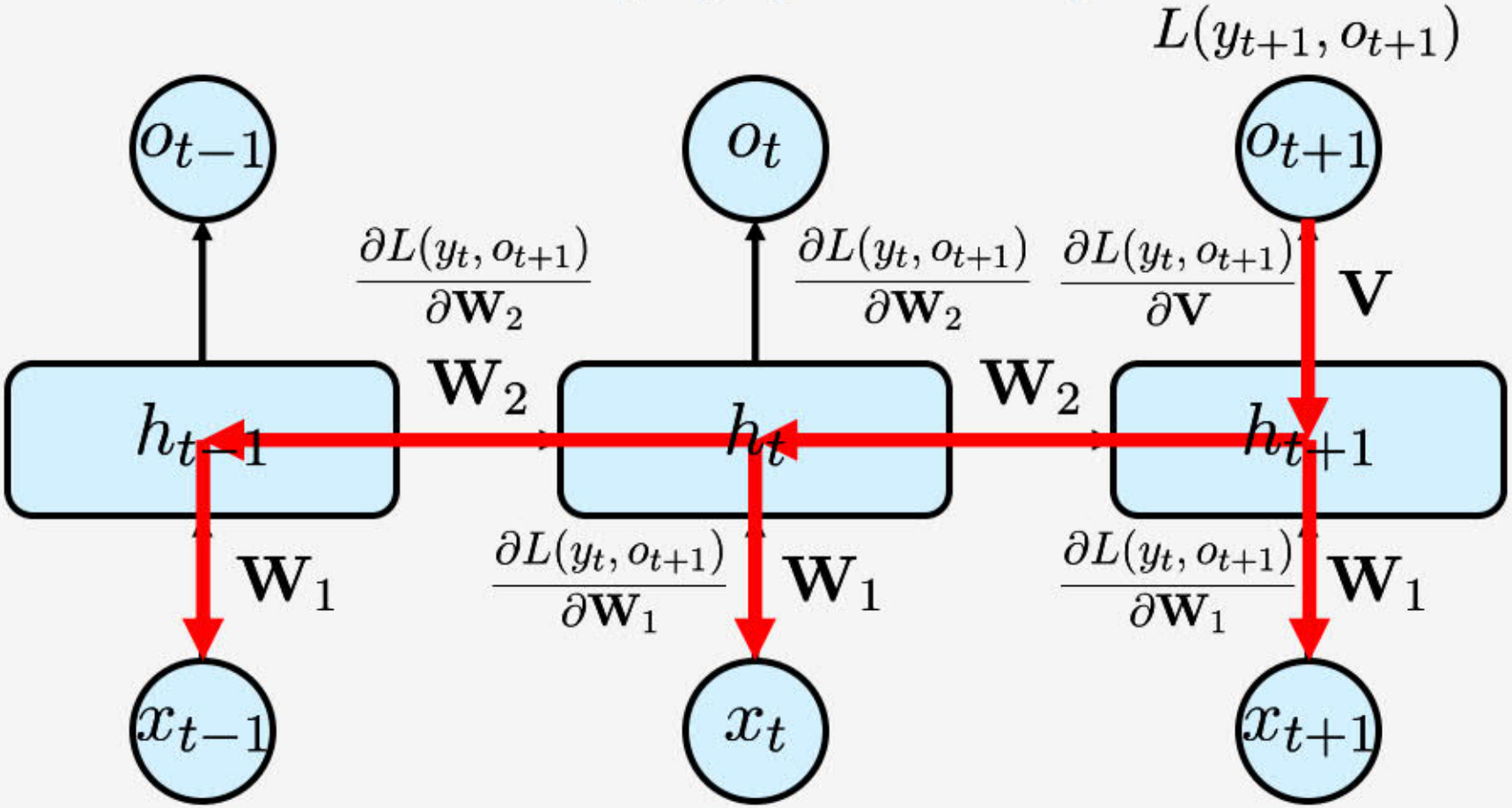
5.5 循环神经网络

RNN的工作方式



5.5 循环神经网络

RNN的反向传播学习（Backpropagation through time)



在上图例子中，BPTT让t+1时间步的损失梯度更新了：

- V一次，W_2两次，W_1三次

神经网络总结

- 通用近似：双层神经网络可以逼近任何函数
- 到目前为止，反向传播是多层神经网络最重要的训练方案
- 深度学习，即用大数据训练的深度神经网络，效果非常好
  - MLP（DNN）、CNN、RNN、LSTM
  - 感兴趣的同学可以关注Attention和Transformer
- 结合其他机器学习模型的神经网络取得了进一步成功