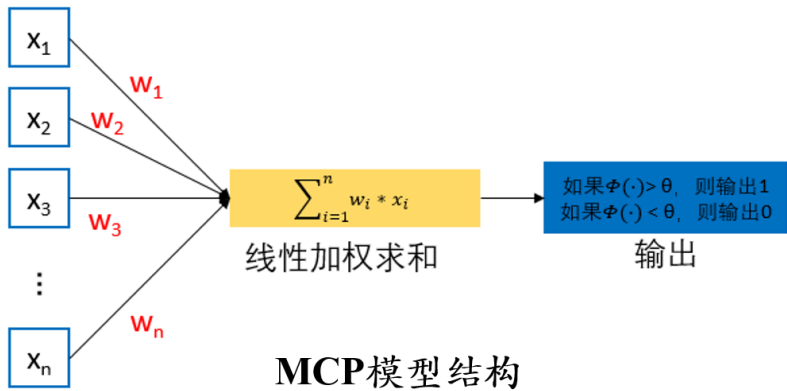


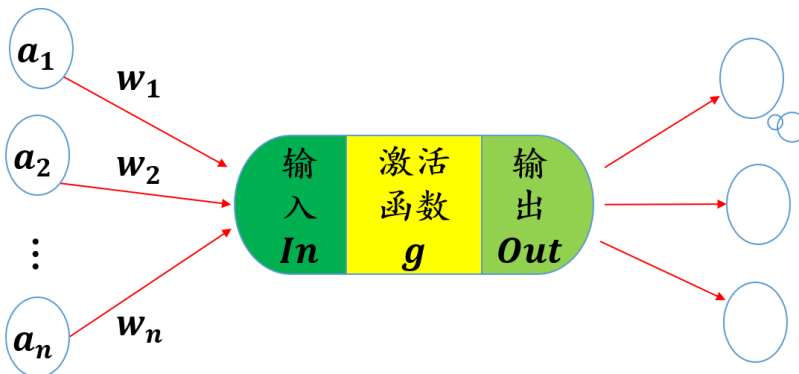
知识点总结（深度学习）

- 前馈神经网络

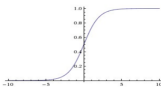
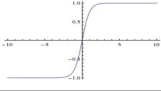
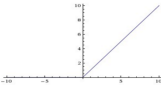
- 基于神经元细胞的结构特性与传递信息方式,MCP模型成为人工神经网络中的最基本结构
- MCP
 - MCP神经元模型对输入数据线性加权求和，然后使用函数 $\phi(\cdot)$ 将加权累加结果映射为0或1，以完成两类分类的任务：



- $y = \Phi(\sum_{i=1}^n w_i x_i)$
- 神经元
 - 神经元是深度学习模型中基本单位



- 对相邻前向神经元输入信息进行加权累加： $In = \sum_{i=1}^n w_i * a_i$
 - 对累加结果进行非线性变换（通过激活函数）： $g(x)$
 - 神经元的输出： $Out = g(In)$
- 常用的激活函数

激活函数名称	函数功能	函数图像	函数求导
Sigmoid	$f(x) = \frac{1}{1 + e^{-x}}$		$f'(x) = f(x)(1 - f(x))$
Tanh	$f(x) = \frac{2}{1 + e^{-2x}} - 1$		$f'(x) = 1 - f(x)^2$
Relu (Rectified Linear Unit)	$f(x) = \begin{cases} 0, & \text{for } x < 0 \\ x, & \text{for } x \geq 0 \end{cases}$		$f'(x) = \begin{cases} 0, & \text{for } x < 0 \\ 1, & \text{for } x \geq 0 \end{cases}$

• 损失函数

- 损失函数又称为代价函数

- 两种最常用损失函数：

- 均方误差损失函数 $MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$

- 交叉熵损失函数

- 用来度量两个概率分布间的差异。假定p和q是数据x的两个概率分布，通过q来表示p的交叉熵可如下计算：

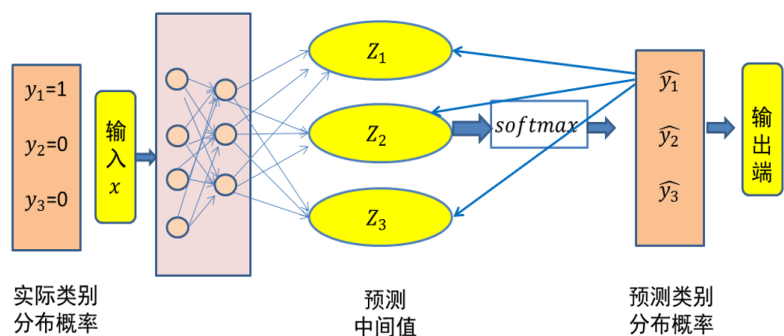
- $H(p, q) = - \sum_x p(x) * \log q(x)$

- 旨在描绘通过概率分布q来表达概率分布p的困难程度。根据公式不难理解，交叉熵越小，两个概率分布p和q越接近。

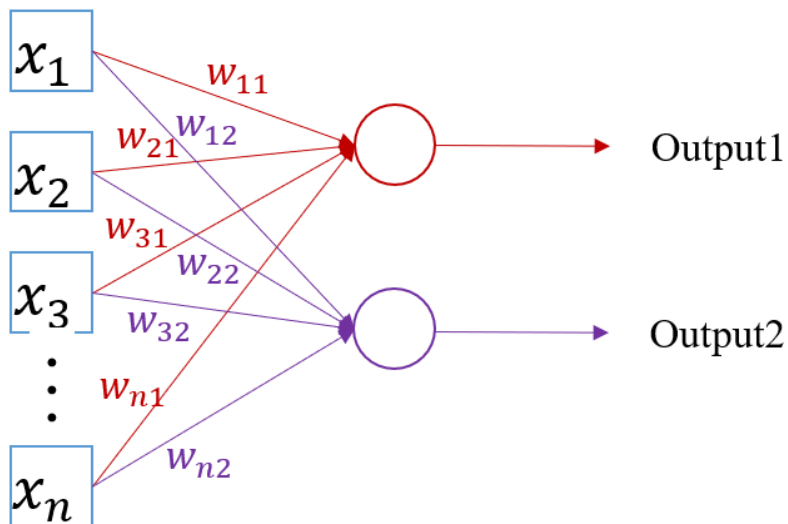
- 假设数据x属于类别1。记x的实际类别分布概率为 $y = (1, 0, 0)$

- \hat{y} 代表模型预测所得类别分布概率

- 那么对x而言，y和 \hat{y} 的交叉熵损失函数定义为 $-y \times \log(\hat{y})$



- 各神经元接受前一级输入并输出到下一级，模型中没有反馈
- 层间“全连接”，即两个相邻层之间的神经元完全成对连接，但层内的神经元不相互连接。
- 感知机网络(Perceptron Networks)是一种特殊的前馈神经网络



- 无隐藏层，只有输入层/输出层
- 无法拟合复杂的数据
- 早期的感知机结构和MCP模型相似，由一个输入层和一个输出层构成，因此也被称为“单层感知机”。感知机的输入层负责接收实数值的输入向量，输出层则能输出1或-1两个值。
- 逻辑异或（XOR）是非线性可分的，因此单层感知机无法模拟
- $w_{ij} (1 \leq i \leq n, 1 \leq j \leq 2)$ 构成了感知机模型参数
- 多层感知机
 - 多层感知机由输入层、输出层和至少一层的隐藏层构成
 - 各个隐藏层中神经元可接收相邻前序隐藏层中所有神经元传递的信息，加工处理后输出给相邻后续隐藏层中所有神经元
 - 各个神经元接受前一级的输入，并输出到下一级，模型中没有反馈
 - 层与层之间通过“全连接”进行链接，即两个相邻层之间的神经元完全成对连接，但层内的神经元不相互连接。
 - $w_{ij} (1 \leq i \leq n, 1 \leq j \leq m), n$ 为神经网络层数、 m 为每层中神经元个数
- 单层和多层感知机：全连接、前馈、无反馈
- 如何优化网络参数？
 - 梯度下降
 - 批量梯度下降、随机梯度下降、小批量梯度下降
 - 梯度下降算法是一种使得损失函数最小化的方法。一元变量所构成函数 φ 在 x 处梯度为：
 - $$\frac{df(x)}{dx} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$
 - 公式

$$f(x + \Delta x) - f(x) = \|\nabla f(x)\| \|\Delta x\| \cos \theta = -\alpha \|\nabla f(x)\|$$

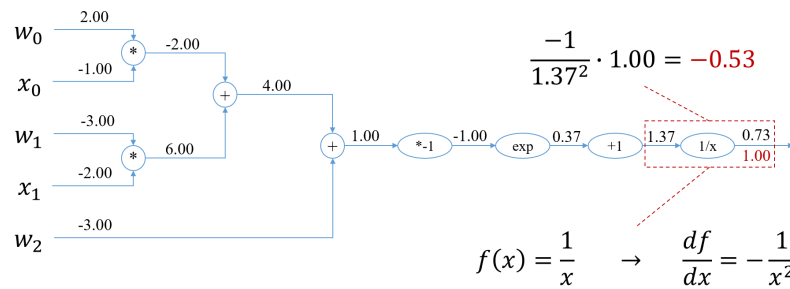
↑ ↑
步长 函数偏导

- 误差反向传播

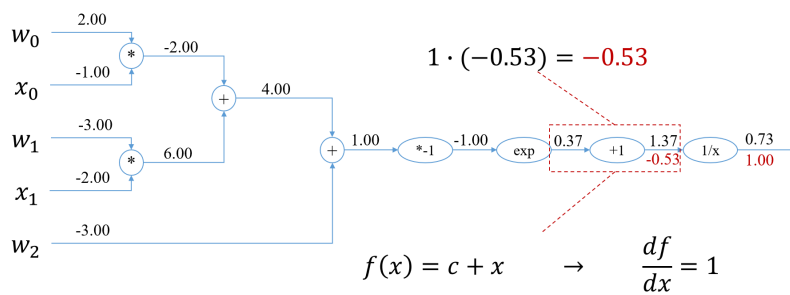
- BP算法将输出层误差反向传播给隐藏层更新参数

- 误差的计算：

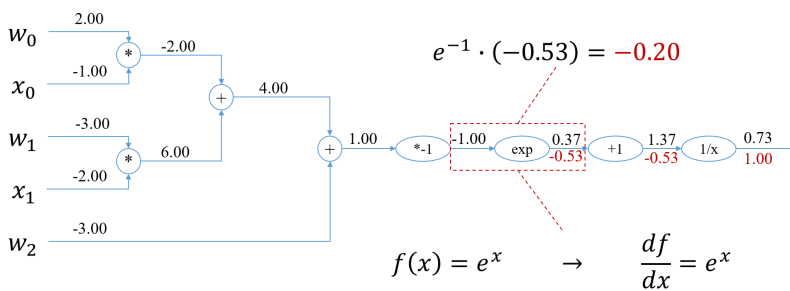
$$f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2 x_2)}}$$



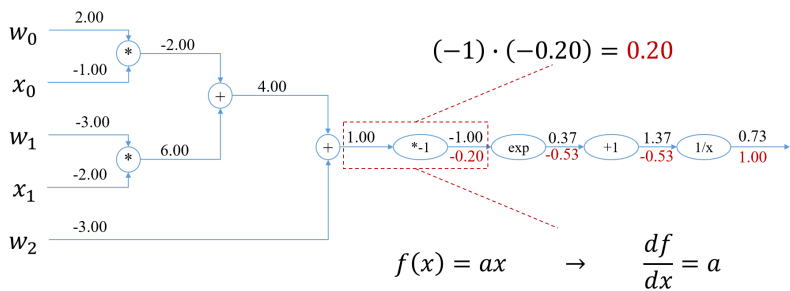
$$f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2 x_2)}}$$



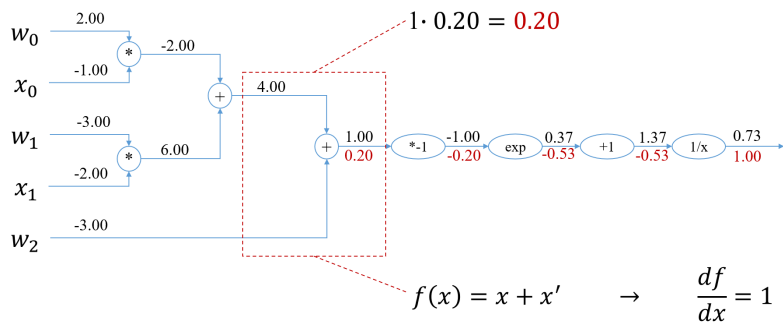
$$f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2 x_2)}}$$



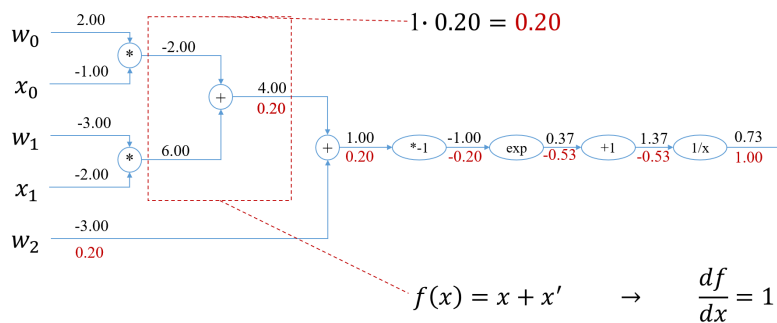
$$f(w, x) = \frac{1}{1 + e^{-(w_0 x_0 + w_1 x_1 + w_2 x_2)}}$$



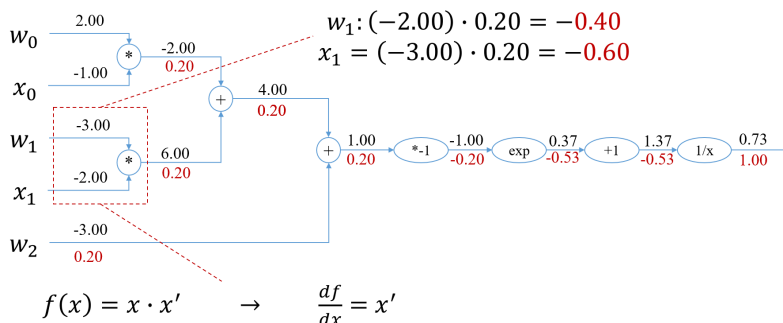
$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2x_2)}}$$



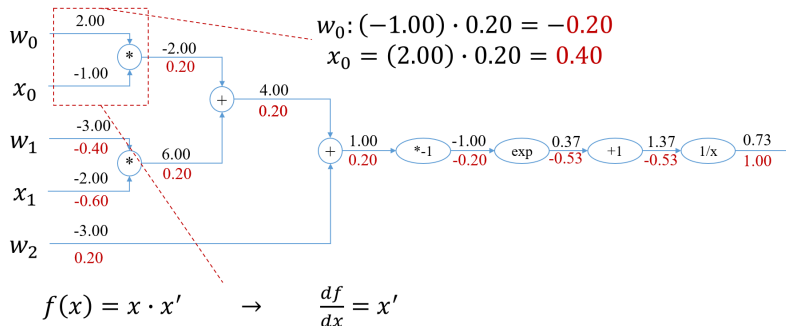
$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2x_2)}}$$



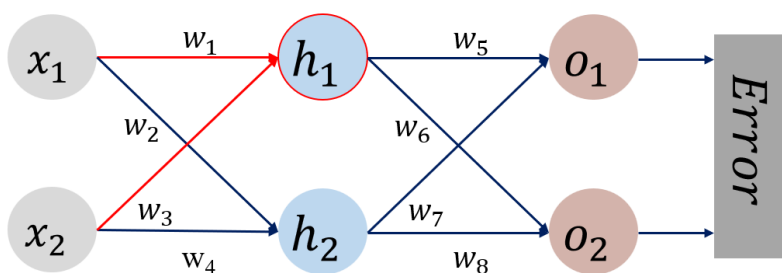
$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2x_2)}}$$



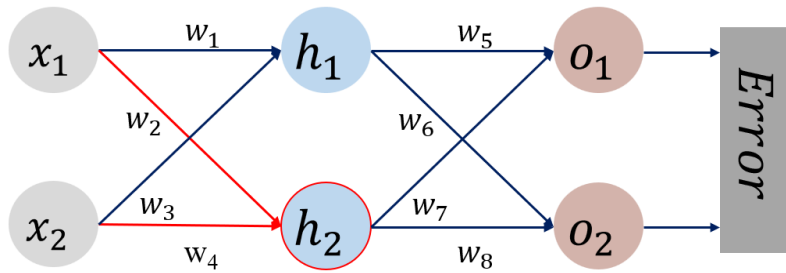
$$f(w, x) = \frac{1}{1 + e^{-(w_0x_0 + w_1x_1 + w_2x_2)}}$$



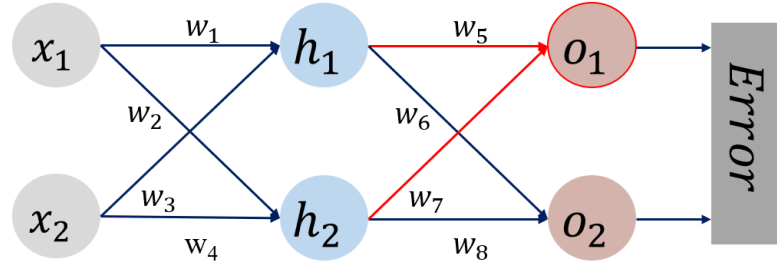
- $In_{h_1} = w_1 * x_1 + w_3 * x_2, \quad h_1 = Out_{h_1} = Sigmoid(In_{h_1})$



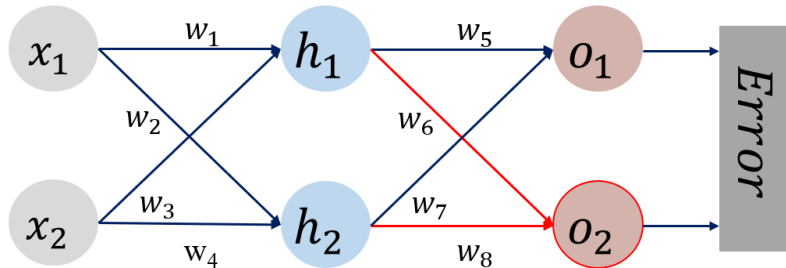
- $In_{h_2} = w_2 * x_1 + w_4 * x_2$ $h_2 = Out_{h_2} = Sigmoid(In_{h_2})$



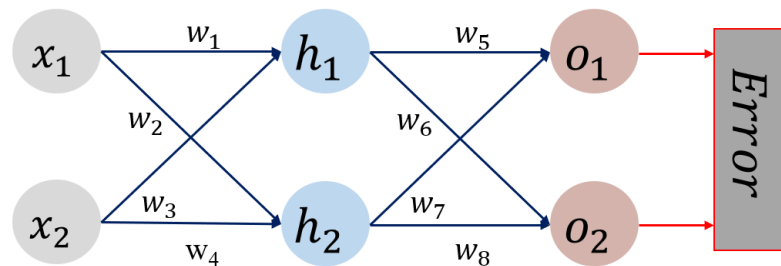
- $In_{o_1} = w_5 * h_1 + w_7 * h_2$ $o_1 = Out_{o_1} = Sigmoid(In_{o_1})$



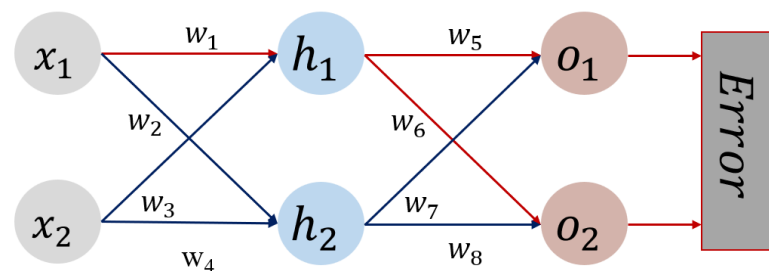
- $In_{o_2} = w_6 * h_1 + w_8 * h_2$ $o_2 = Out_{o_2} = Sigmoid(In_{o_2})$



- $Error = \frac{1}{2} \sum_{i=1}^2 (o_i - y_i)^2$



- 其中 η 被称为学习率, $w'_i = w_i - \eta * \delta_i$

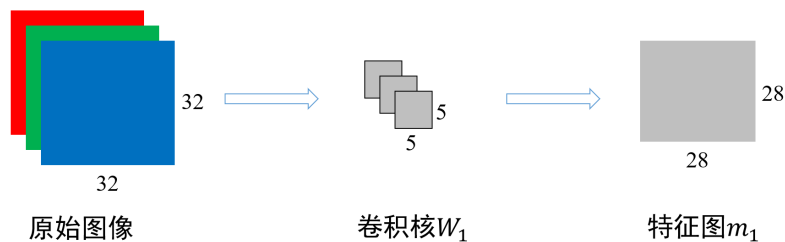


• 卷积神经网络(CNN)

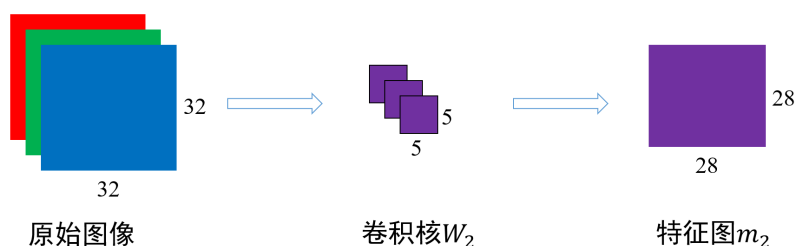
- 前馈神经网络中，输入层的输入直接与第一个隐藏层中所有神经元相互连接。
- 在图像卷积计算中，需要定义一个卷积核
- 卷积核中的权重系数 w_i 是通过数据驱动机制学习得到

- 在对原始图像做卷积操作后，可使用ReLU激活函数对卷积后结果进行处理,神经科学家发现,人的视觉神经细胞对不同的视觉模式具有特征选择性

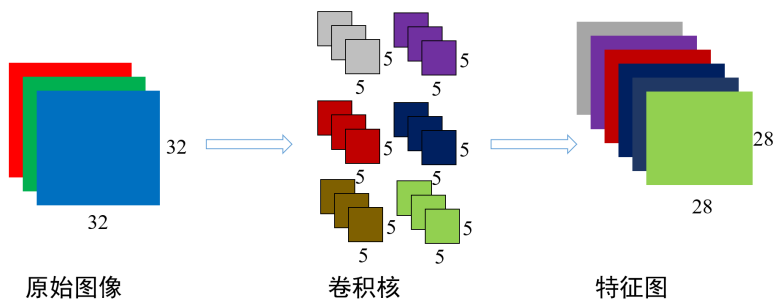
有一张 $32*32*3$ (RGB)的图像，使用 $5*5*3$ 的卷积核 W_1 ，步长为1对其进行卷积操作。卷积核 W_1 在原始图像上从左到右、从上到下进行计算，改变 $5*5$ 子块区域中的中心像素点值，得到 $28*28$ 的特征图 m_1



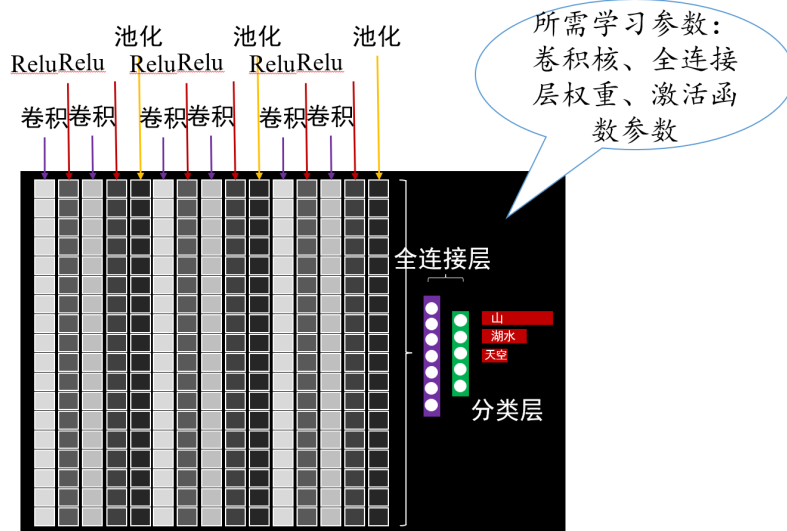
使用另一个 $5*5*3$ 的卷积核 W_2 与原始图像做卷积操作，得到特征图 m_2



使用6个 $5*5*3$ 的卷积核与原始图像做卷积操作，则得到6个 $28*28$ 的特征图
注意:6个 $5*5*3$ 的卷积核均是数据驱动学习得到，其刻画了不同的视觉模式



- 池化(pooling)操作
 - 常用的池化操作有：
 - 最大池化
 - 平均池化
- 对于输入的海量标注数据，通过多次迭代训练，卷积神经网络在若干次卷积操作、接着对卷积所得结果进行激活函数操作和池化操作下，最后通过全连接层来学习得到输入数据的特征表达，即分布式向量表达(distributed vector representation)。
- 卷积神经网络基本架构



• 如何计算卷积结果的分辨率

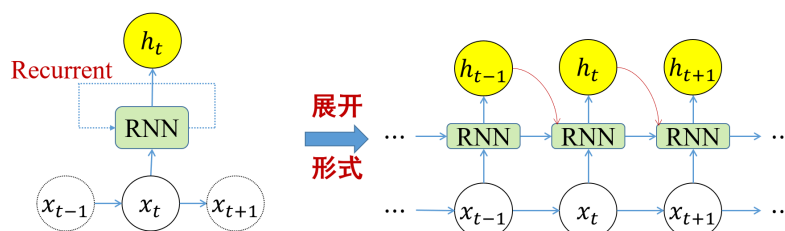
- 图像大小 $\omega \times \omega$
- 卷积核大小为 $F \times F$
- Padding为 P
- 步长为 S
- 结果: $(\omega - F + 2P)/S + 1$

• 神经网络正则化

- $L1$ 范数: 数学表示为 $\|W\|_1 = \sum_{i=1}^N |w_i|$, 指模型参数 W 中各个元素的绝对值之和。 $L1$ 范数也被称为“稀疏规则算子” (Lasso regularization)。
- $L2$ 范数: 数学表示为 $\|W\|_2 = \sqrt{\sum_{i=1}^N w_i^2}$, 指模型参数 $L2$ 中各个元素平方和的开方。

• 循环神经网络(RNN)

- 循环神经网络 (Recurrent Neural Network, RNN) 是一类处理序列数据 (如文本句子、视频帧等) 时所采用的网络结构。
- 循环神经网络的本质是希望模拟人所具有的记忆能力, 在学习过程中记住部分已经出现的信息, 并利用所记住的信息影响后续结点输出。
- 循环神经网络在处理数据过程中构成了一个循环体



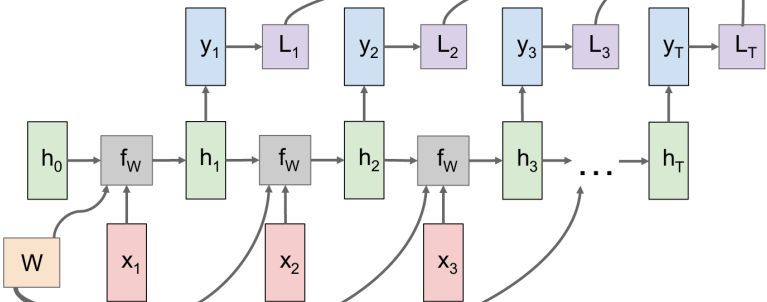
$$h_t = \Phi(U \times x_t + W \times h_{t-1}) = \Phi(U \times x_t + W \times \Phi(U \times x_{t-1} + W \times h_{t-2}))$$

$$= \Phi \left(U \times \underset{t \text{ 时刻输入}}{x_t} + W \times \Phi \left(U \times \underset{t-1 \text{ 时刻输入}}{x_{t-1}} + W \times \Phi \left(U \times \underset{t-2 \text{ 时刻输入}}{x_{t-2}} + \dots \right) \right) \right)$$

• RNN隐藏状态更新

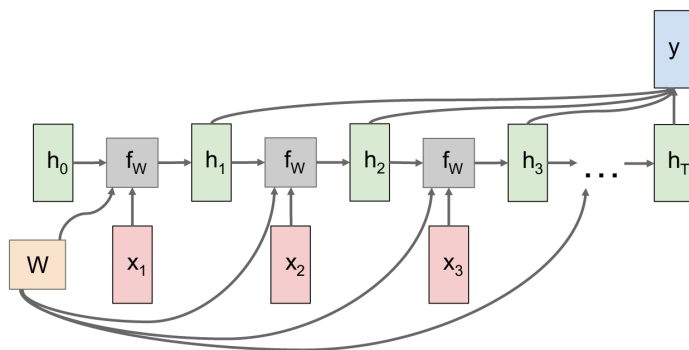
某一时刻的
输入向量

- ### RNN: 计算图: Many to Many



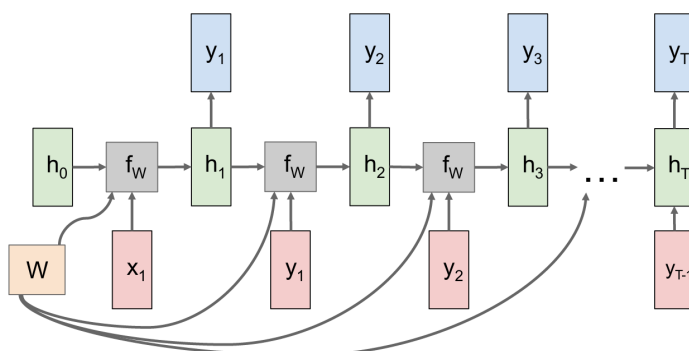
- many-one

RNN: 计算图: Many to One



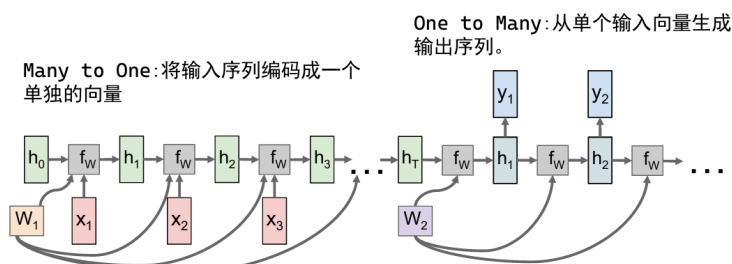
- one-many

RNN: 计算图: One to Many



- Many to One + One to Many

序列到序列: Many to One + One to Many



Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

- 处理变长序列

- 输入序列和输出序列可以有不同长度
- 不需要预先知道输出序列的确切长度

- 随时间反向传播

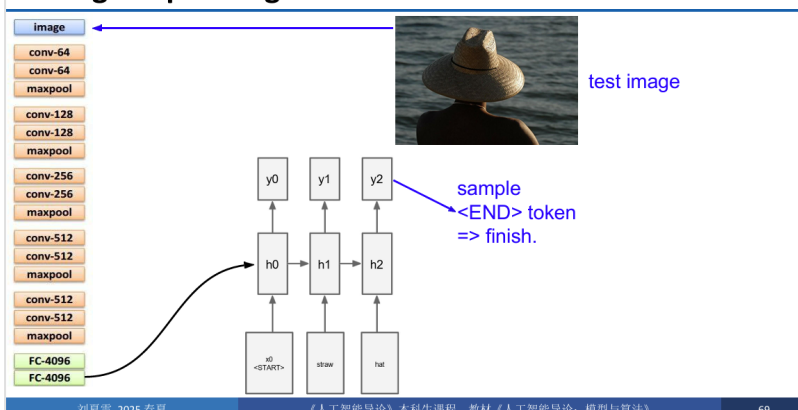
- 前向传播整个序列来计算损失，然后反向传播整个序列来计算梯度。

- 截断的反向传播算法

- 通过序列的若干片段进行前向和反向传播，而不是整个序列。
- 在时间上无限地向前传递隐藏状态，但只反向传播有限的步数。

- 应用举例：视觉信息（CNN）和语言生成（RNN）结合，生成对图像内容的自然语言描述。

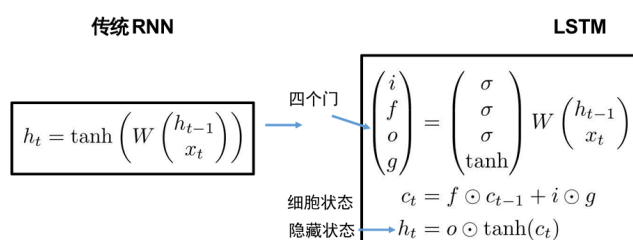
Image Captioning



• 多层RNNs

- Long Short Term Memory (LSTM): LSTM是为了解决传统RNN的梯度消失和梯度爆炸问题而设计的。

Long Short Term Memory (LSTM)



• 门控机制 - 四个门

- # 输入门 (i): 控制新信息的输入 $i_t = \text{sigmoid}(W_i @ [h_{t-1}, x_t] + b_i)$
- # 遗忘门 (f): 控制旧信息的遗忘 $f_t = \text{sigmoid}(W_f @ [h_{t-1}, x_t] + b_f)$
- # 输出门 (o): 控制信息的输出 $o_t = \text{sigmoid}(W_o @ [h_{t-1}, x_t] + b_o)$
- # 候选值 (g): 新的候选信息 $g_t = \tanh(W_g @ [h_{t-1}, x_t] + b_g)$
- g: 门控, 决定向单元状态写入多少信息
- i: 输入门, 控制是否向单元写入信息
- f: 遗忘门, 是否清除细胞单元状态
- o: 输出门, 决定当前输出输入到隐藏层的数量

• 细胞状态 - 解决梯度消失

- # 细胞状态更新: 加法而非乘法 $c_t = f_t \odot c_{t-1} + i_t \odot g_t$ # 保留旧信息 添加新信息
- # 隐藏状态输出 $h_t = o_t \odot \tanh(c_t)$

- | | | |
|--------|--------|------------|
| • 特征传统 | RNN | LSTM |
| • 状态更新 | 完全替换 | 选择性更新 |
| • 梯度传播 | 连续相乘衰减 | 加法连接, 梯度稳定 |

- 信息控制 无控制机制 四个门精确控制
- 长期记忆 很差 优秀
- 参数数量 少 多 (4倍)
- LSTM不能保证完全避免梯度消失或爆炸，但它为模型学习长距离依赖关系提供了更简便的途径。
- LSTM能使得梯度连续流动的方法：
 - # LSTM细胞状态更新 $c_t = f_t \odot c_{t-1} + i_t \odot g_t$ # \uparrow 加法连接，不是乘法！
 - # 反向传播时 $\frac{\partial c_{t-1}}{\partial c_t} = f_t$ # 直接等于遗忘门的值 # 传统RNN的梯度计算 $dh_{prev} = dh_t * \tanh'(c_t) * W_{hh}$ # 经过激活函数导数
 - # 多个时间步的梯度 $\frac{\partial c_0}{\partial c_T} = \prod_{t=1}^T f_t$ # 只依赖于遗忘门，没有激活函数导数
 - RNN 中的梯度反向传播可能会出现爆炸或消失。梯度爆炸可以通过梯度裁剪来控制，梯度消失则通过加性交互（如 LSTM）来缓解。
 -