PyTorch深度学习原理与实现

目录

引言

深度学习发展历程

1. 感知机网络（解决线性可分问题，20世纪40年代）
2. BP神经网络（解决线性不可分问题，20世纪80年代）
3. 深度神经网络（海量图片分类，2010年左右）

常见深度神经网络：CNN、RNN、LSTM、GRU、GAN、DBN、RBM ……

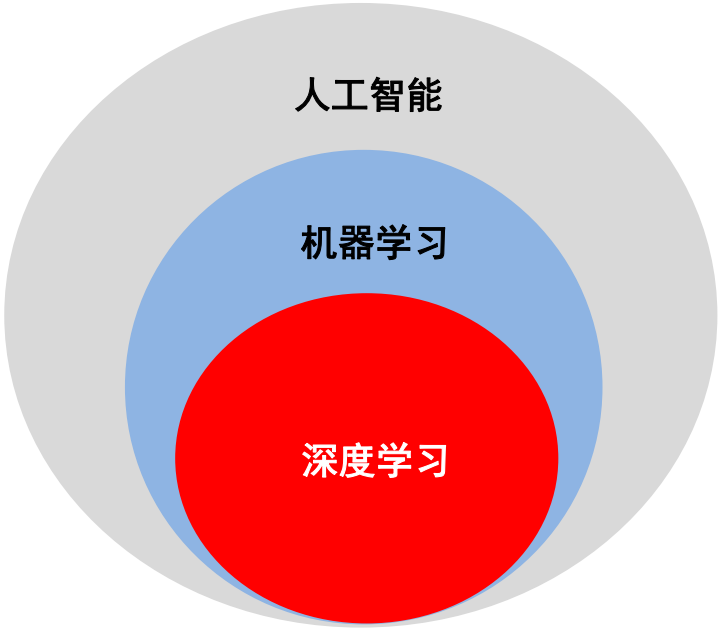
引言

深度应用领域

1. 计算机视觉
2. 语音识别
3. 自然语言处理
4. 人机博弈

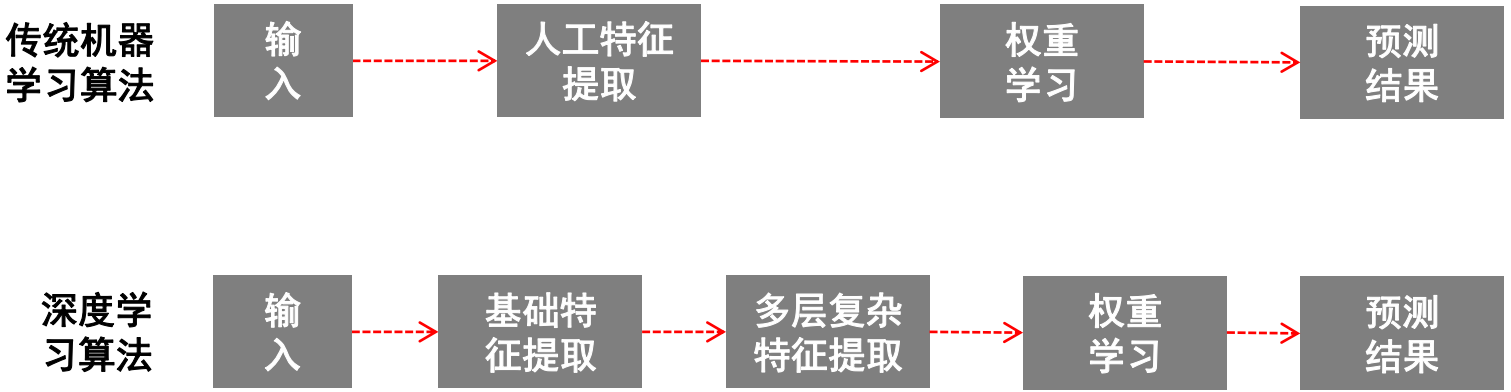
引言

深度学习、机器学习以及人工智能



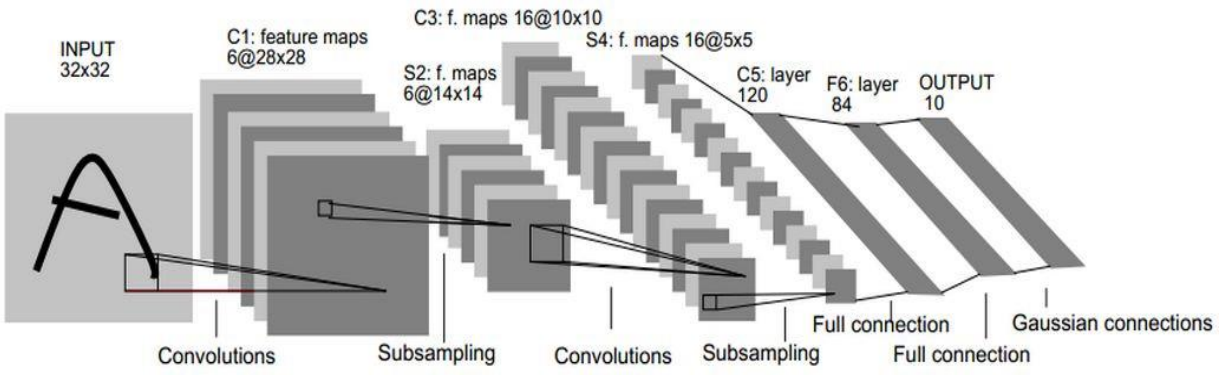
引言

深度学习VS传统机器学习

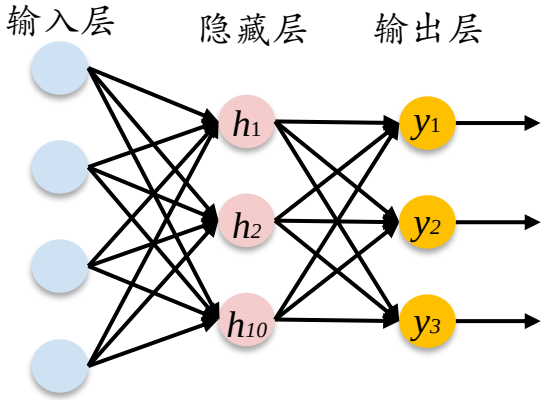


引言

深度神经网络 VS 浅层神经网络



深度神经网络

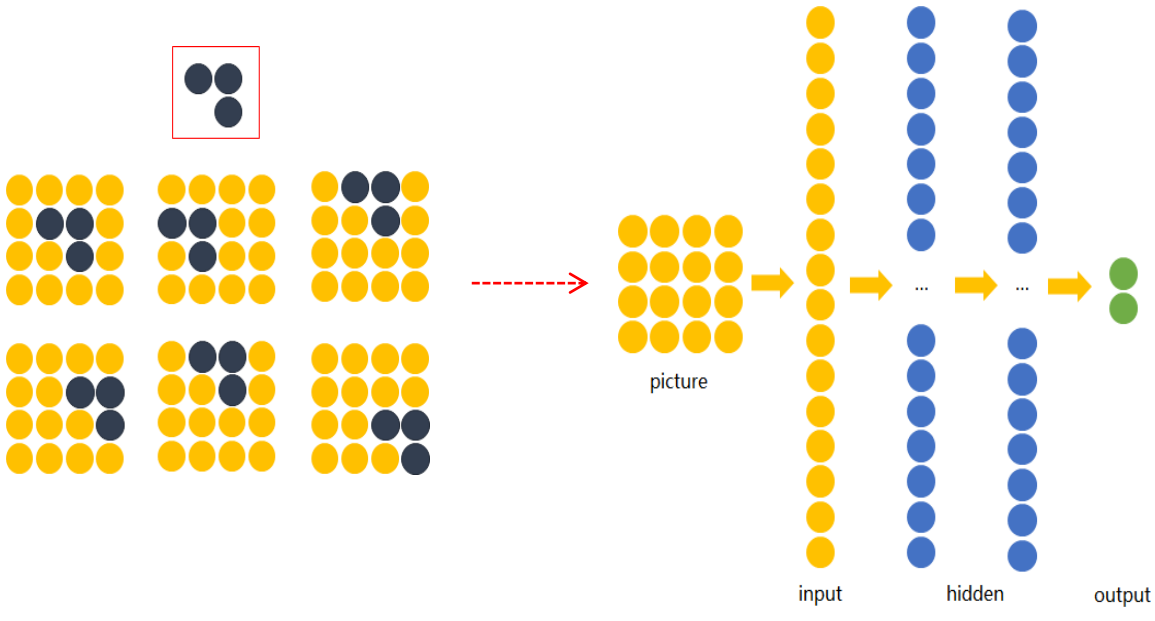


普通网络

目录

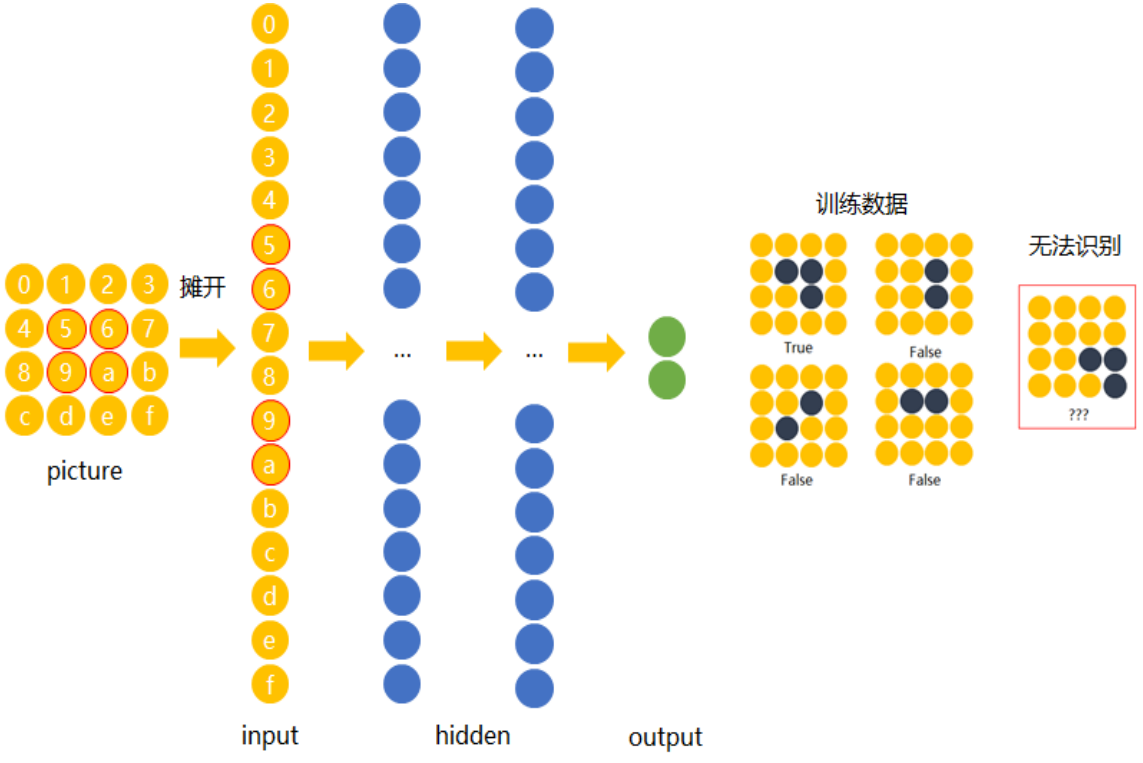
卷积神经网络CNN

用BP神经网络识别图中的“横折”



卷积神经网络CNN

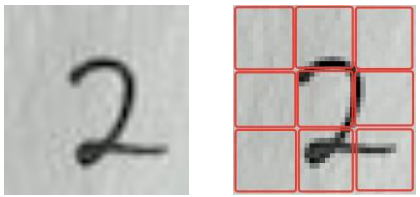
用BP神经网络识别图中的“横折”

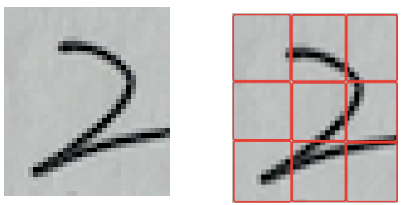


卷积神经网络CNN

用BP神经网络识别图中的“横折”

猜猜这是几：





卷积神经网络CNN

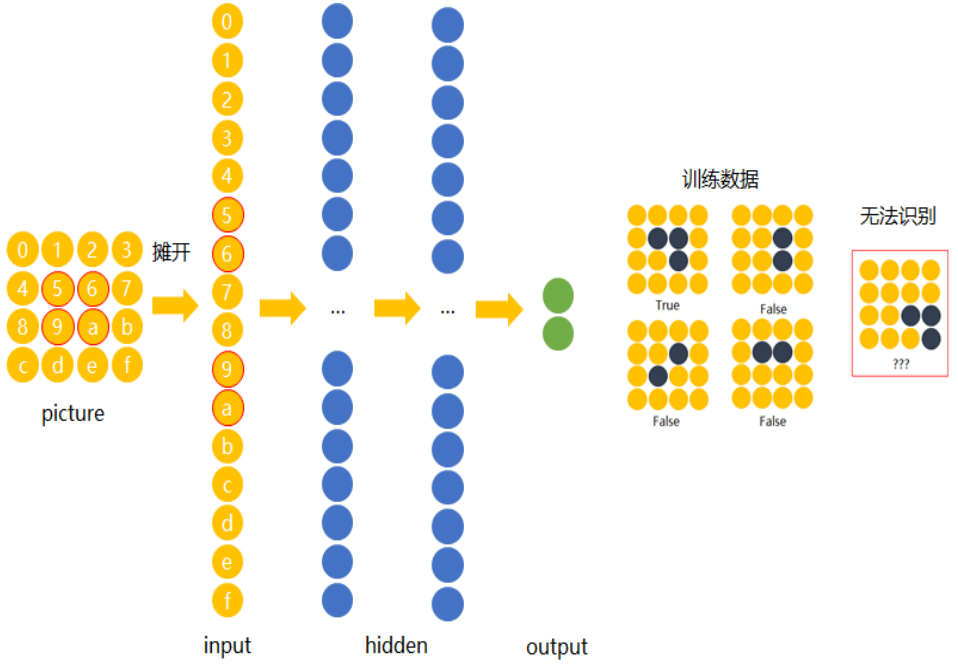
用BP神经网络识别图中的“横折”

BP神经网络缺陷

1. 不能移动
2. 不能变形
3. 运算量大

解决办法

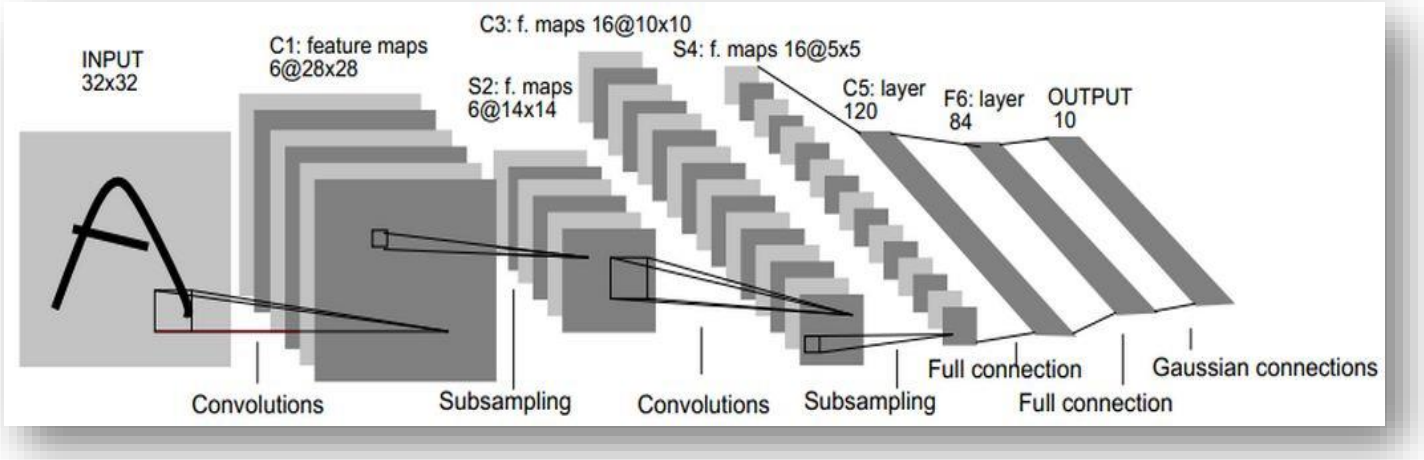
1. 大量物体位于不同位置的数据训练
2. 增加网络的隐藏层个数。
3. 权值共享（不同位置拥有相同权值）



卷积神经网络CNN

卷积神经网络结构[深度学习（DEEP LEARNING）]

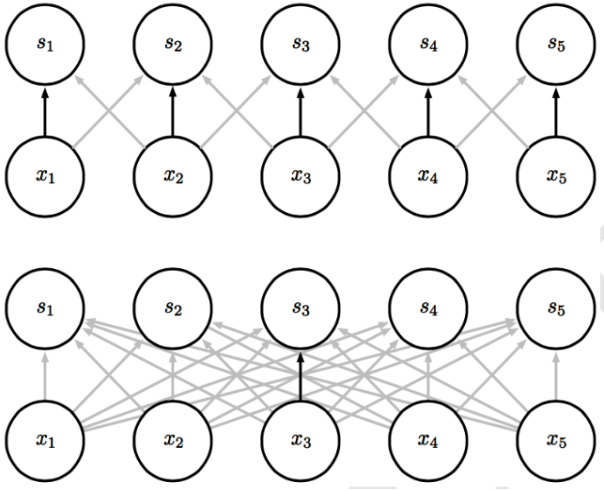
covolutional layer（卷积）、ReLu layer（非线性映射）、pooling layer（池化）、fully connected layer（全连接）、output（输出）的组合，例如下图所示的结构。



卷积神经网络CNN

全连接与局部连接（权值共享）

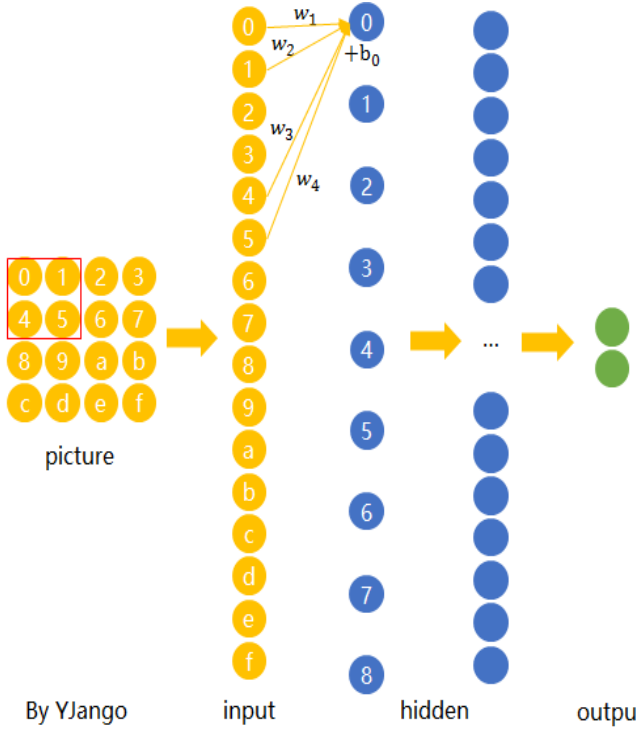
在CNN中，先选择一个局部区域（filter），用这个局部区域去扫描整张图片。 局部区域所圈起来的所有节点会被连接到下一层的一个节点上。



卷积神经网络CNN

全连接与局部连接（权值共享）

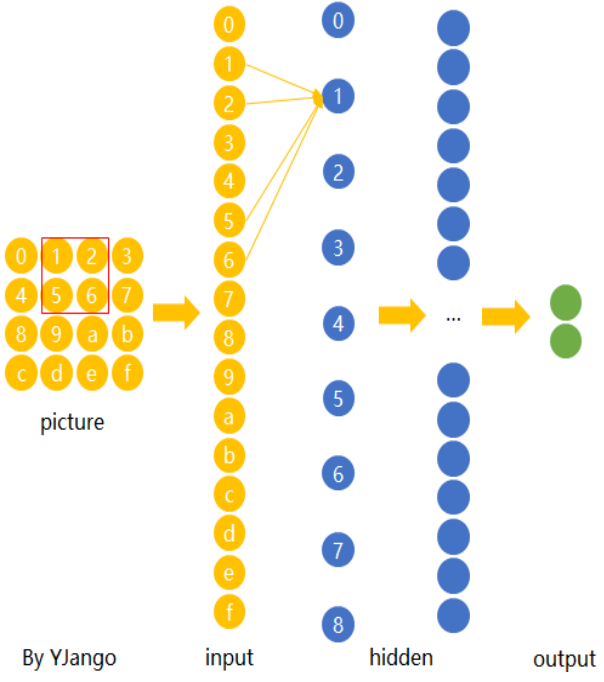
在CNN中，先选择一个局部区域（filter），用这个局部区域去扫描整张图片。 局部区域所圈起来的所有节点会被连接到下一层的一个节点上



卷积神经网络CNN

全连接与局部连接（权值共享）

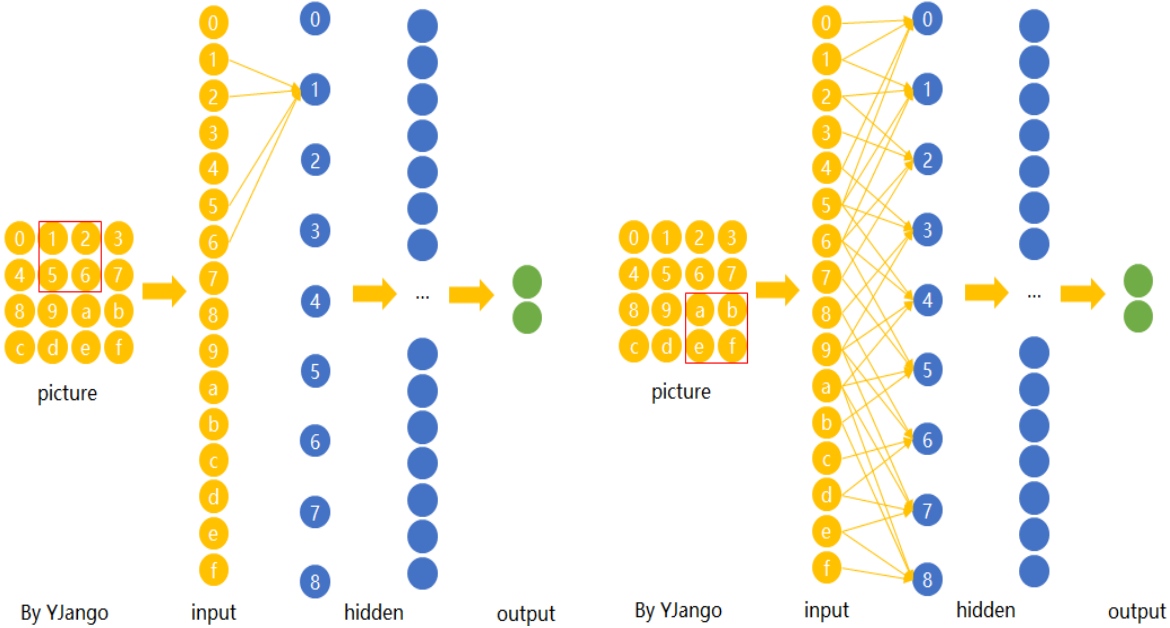
在CNN中，先选择一个局部区域（filter），用这个局部区域去扫描整张图片。 局部区域所圈起来的所有节点会被连接到下一层的一个节点上



卷积神经网络CNN

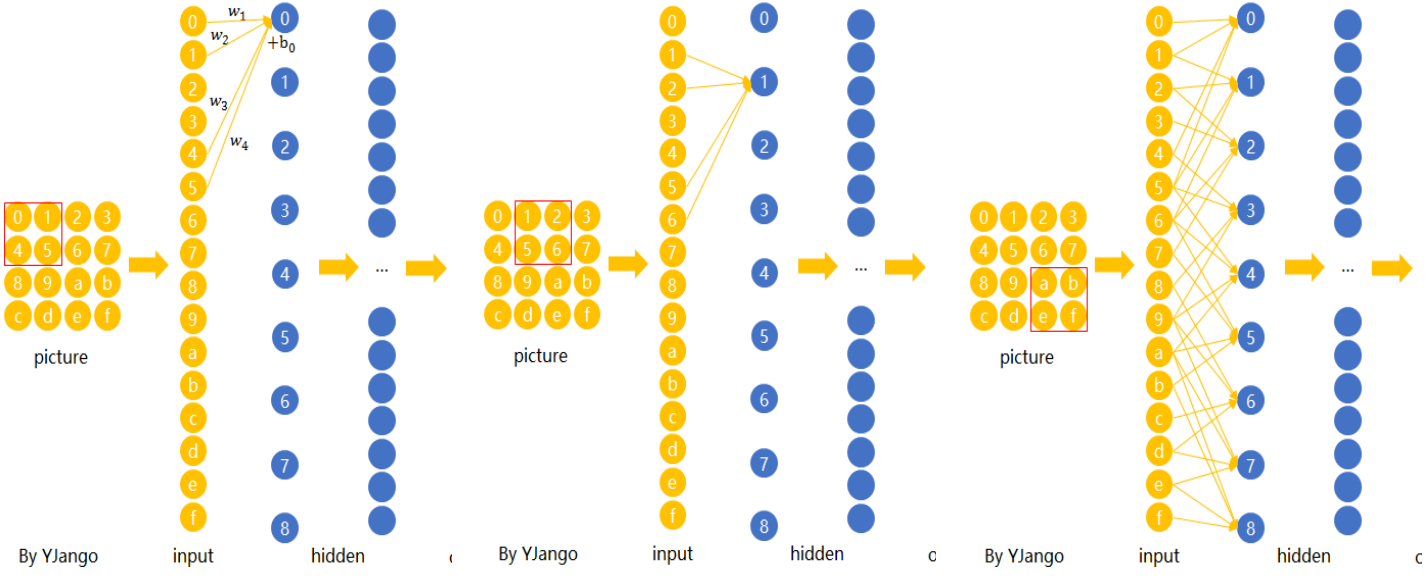
全连接与局部连接（权值共享）

在CNN中，先选择一个局部区域（filter），用这个局部区域去扫描整张图片。 局部区域所圈起来的所有节点会被连接到下一层的一个节点上



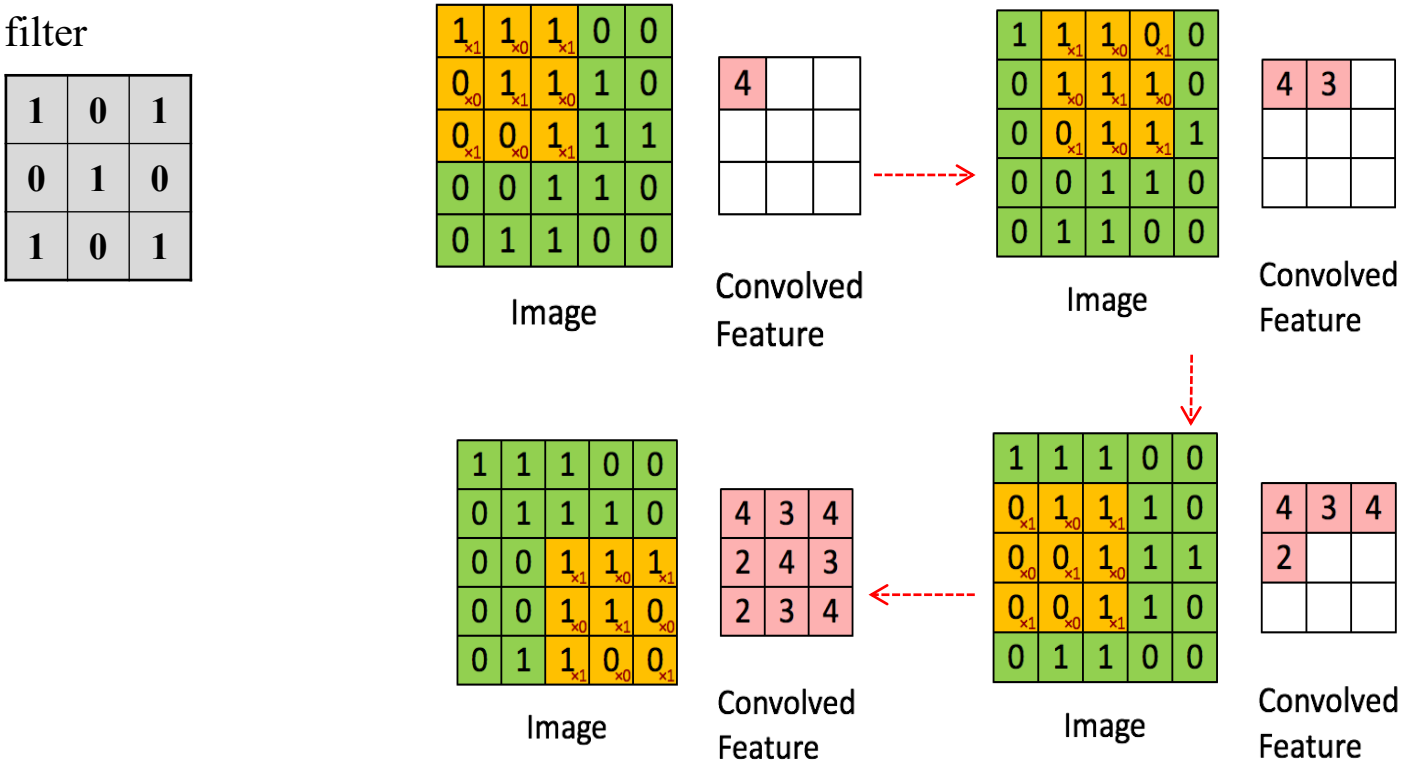
卷积神经网络CNN

全连接与局部连接（权值共享）



卷积神经网络CNN

卷积层－权值共享



卷积层－权值共享

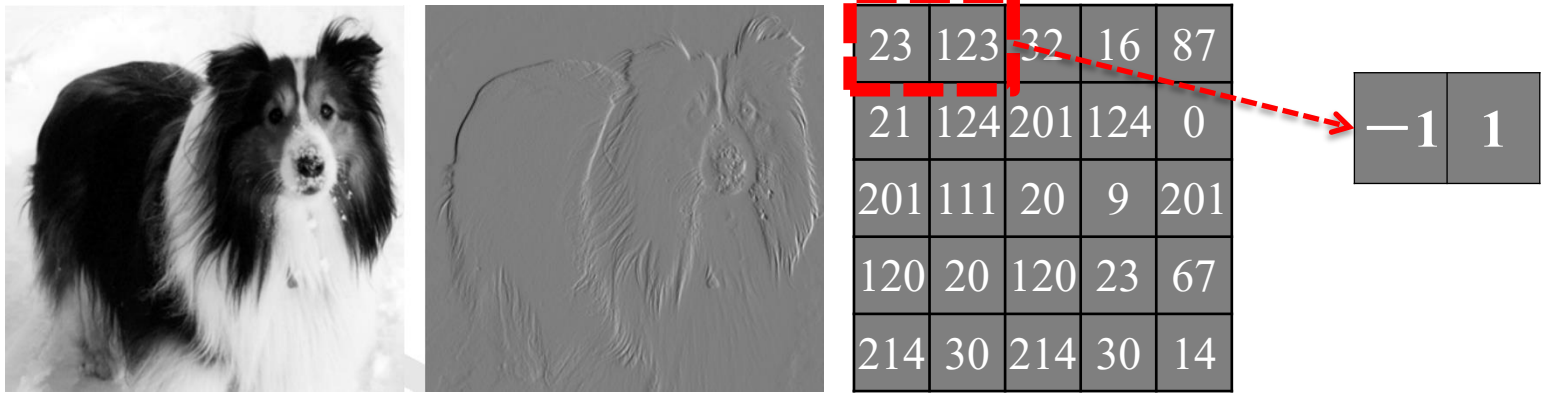
全连接与局部连接（权值共享）

下图（右边像素减左边像素）说明了权值共享是如何提高线性函数在图像边缘检测上的效率的 。

输入：280×320 pix， 输出：280×319 pix， filter：1 × 2

全连接：320 × 280 × 319 × 280 ＝ 80 亿 个权值、160亿次浮点计算

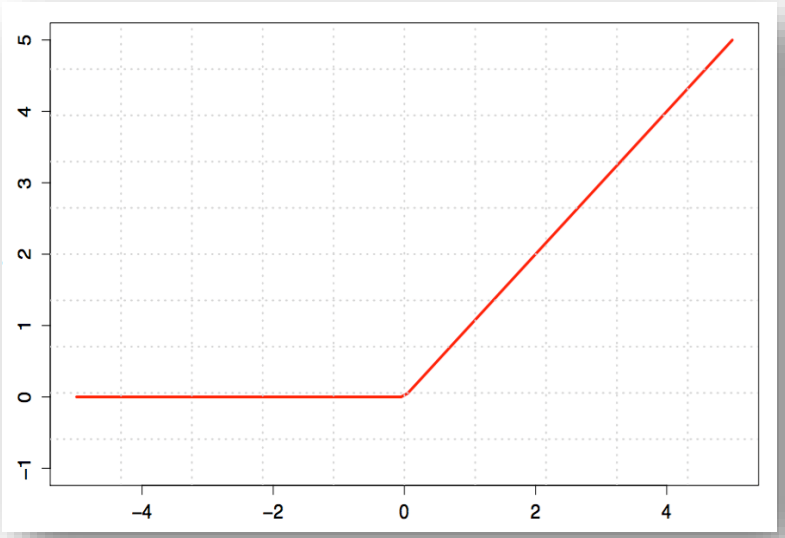
卷积：2个权值、319 × 280 × 3 = 267, 960 次浮点计算



非线性映射ReLU（Rectified Linear Units）

和前馈神经网络一样，经过线性组合和偏移后，会加入非线性增强模型的拟合能力。

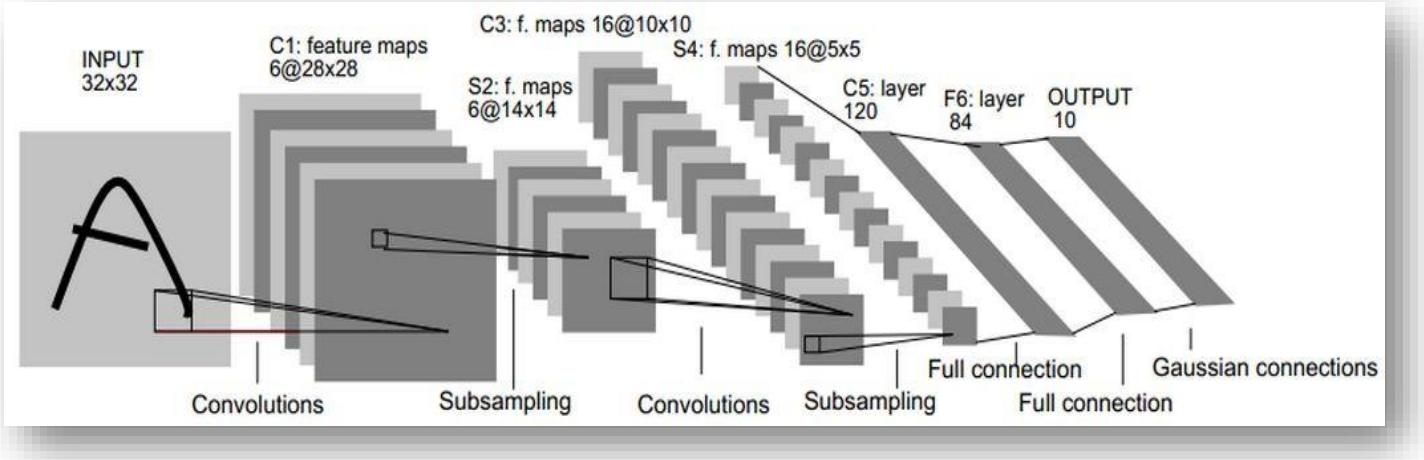
经过线性组合和偏移后，会加入非线性增强模型的拟合能力，将卷积所得的Feature Map经过ReLU变换。



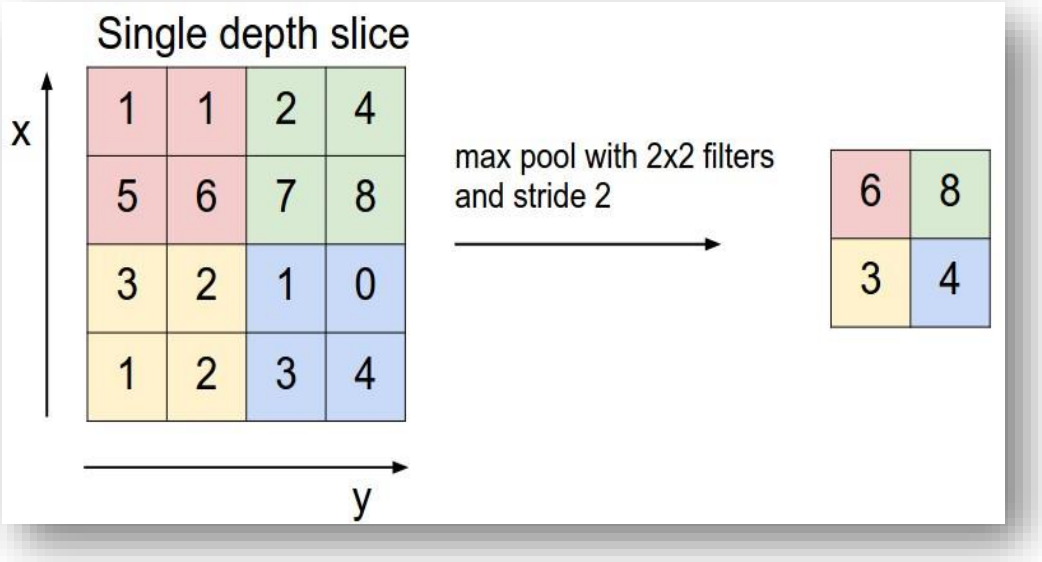
卷积神经网络CNN

卷积神经网络结构[深度学习（DEEP LEARNING）]

covolutional layer（卷积）、ReLu layer（非线性映射）、pooling layer（池化）、fully connected layer（全连接）、output（输出）的组合，例如下图所示的结构。

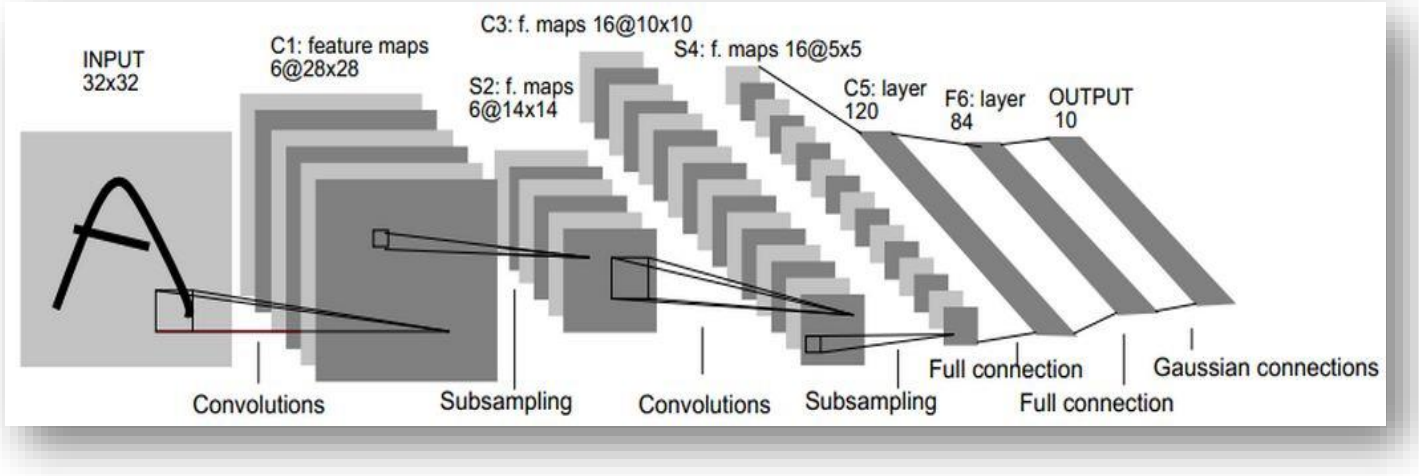


池化（pooling）


Single depth slice
x
y
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
max pool with 2x2 filters
and stride 2
6 8
3 4

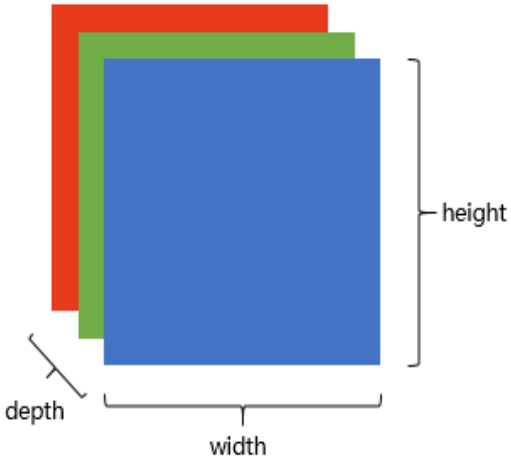
全连接层

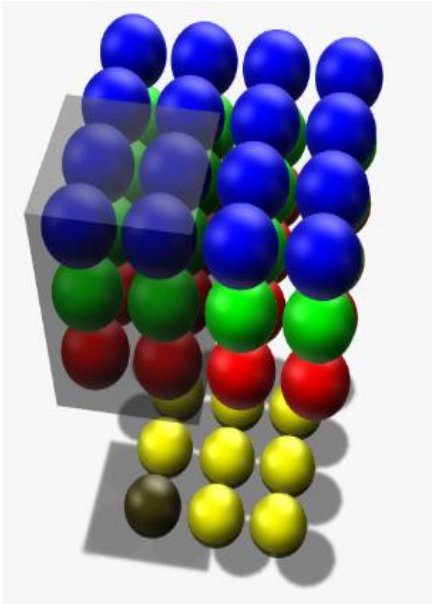
- 当抓取到足以用来识别图片的特征后，接下来的就是如何进行分类。
- 全连接层（也叫前馈层）就可以用来将最后的输出映射到线性可分的空间。
- 卷积网络的最后会将末端得到一个长长的向量，并送入全连接层配合输出层进行分类。



卷积神经网络CNN

高维输入

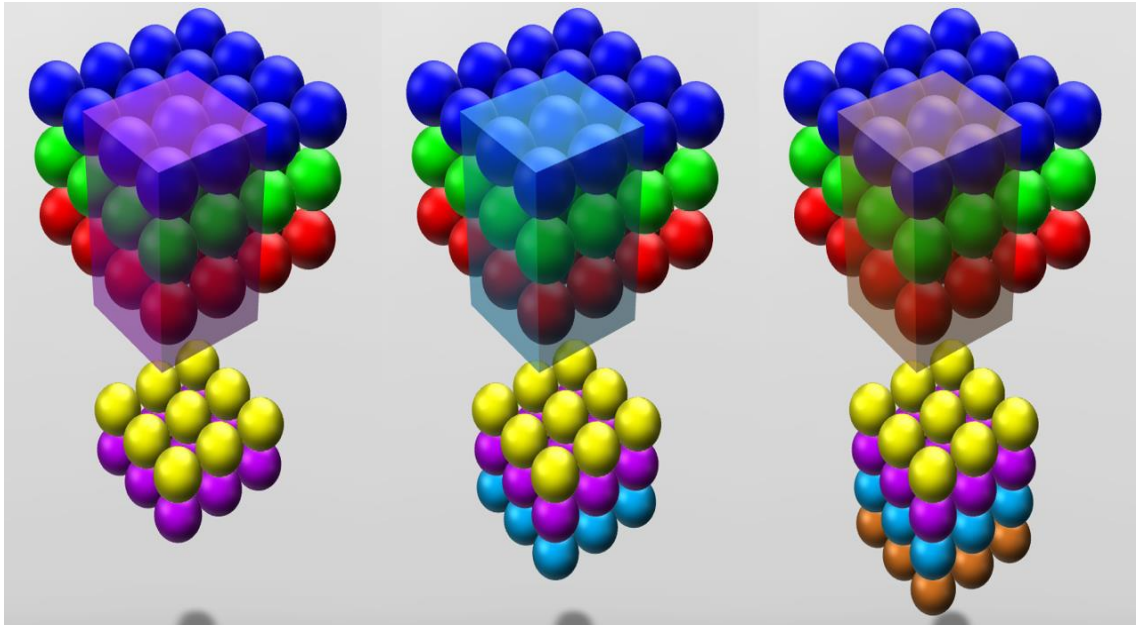




卷积神经网络CNN

高维输入

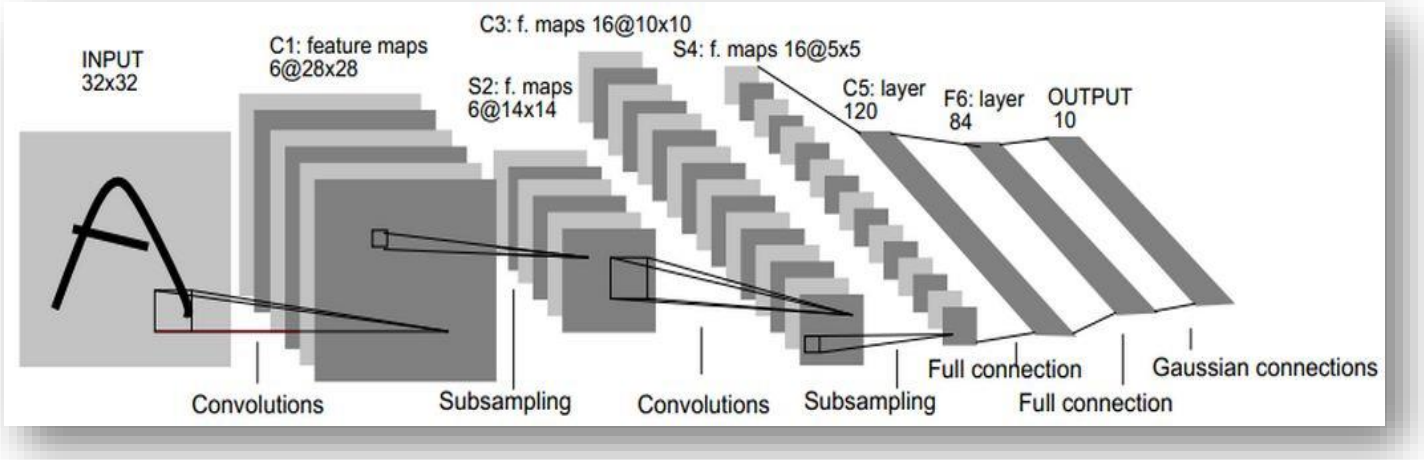
多filters



卷积神经网络CNN

卷积神经网络结构[深度学习（DEEP LEARNING）]

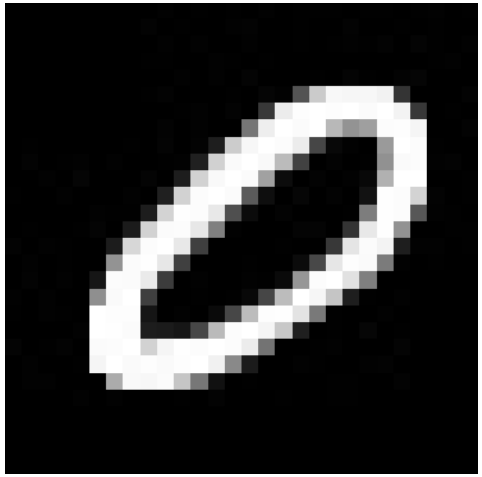
covolutional layer（卷积）、ReLu layer（非线性映射）、pooling layer（池化）、fully connected layer（全连接）、output（输出）的组合，例如下图所示的结构。



卷积神经网络CNN

读入数据

- image = plt.imread('8.jpg') # 读取数据
- plt.imshow(image, cmap='gray')
- plt.show()



卷积神经网络CNN

卷积过程

- CNN采用张量形状（batch_size, color_channels, image_height，image_width）
- image = image.reshape([1, 1, 28, 28])　# 转换照片的维度，使其能正常放入卷积层
- conv2d = torch.nn.Conv2d(in_channels=1, out_channels=32, kernel_size=5)　# 卷积层

卷积神经网络CNN

卷积结果

result_con = conv2d(torch.tensor(image, dtype=torch.float32))　　# 执行卷积操作

池化

- max_pool2d = torch.nn.MaxPool2d(kernel_size=2, stride=2)　# 池化层
- result_pool = max_pool2d(result_con)　# 执行池化操作

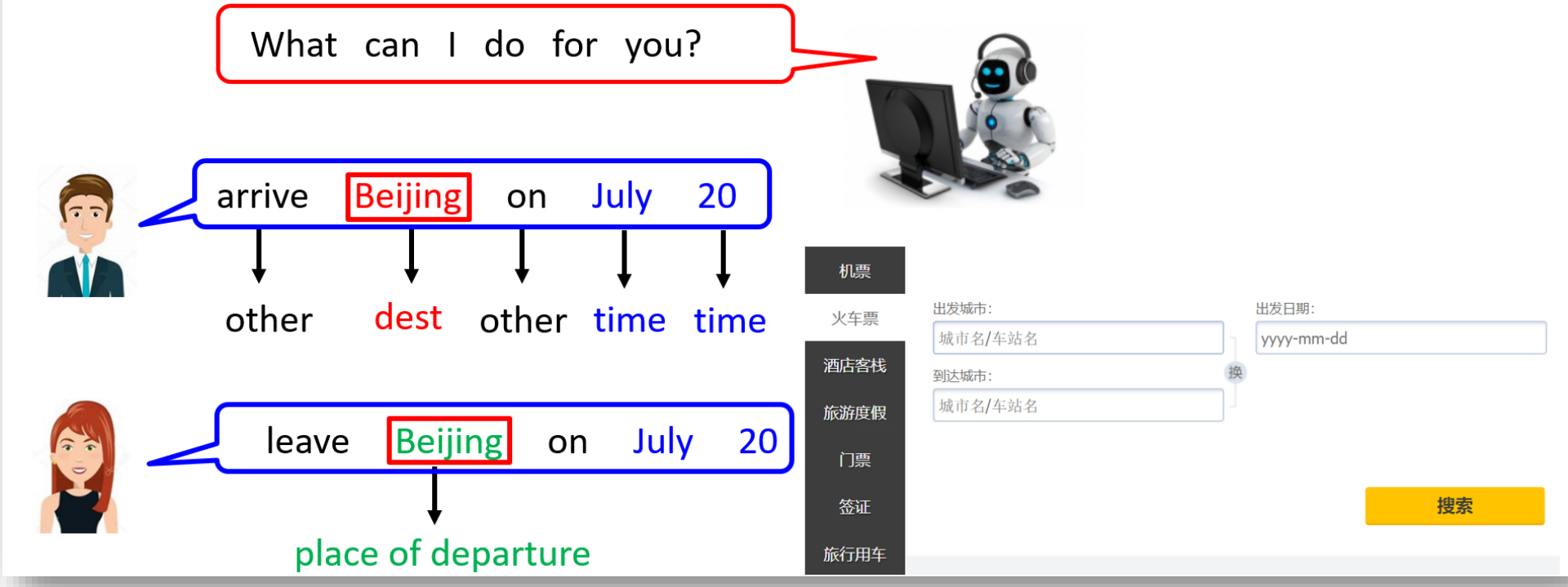
卷积神经网络CNN

池化结果输出

目录

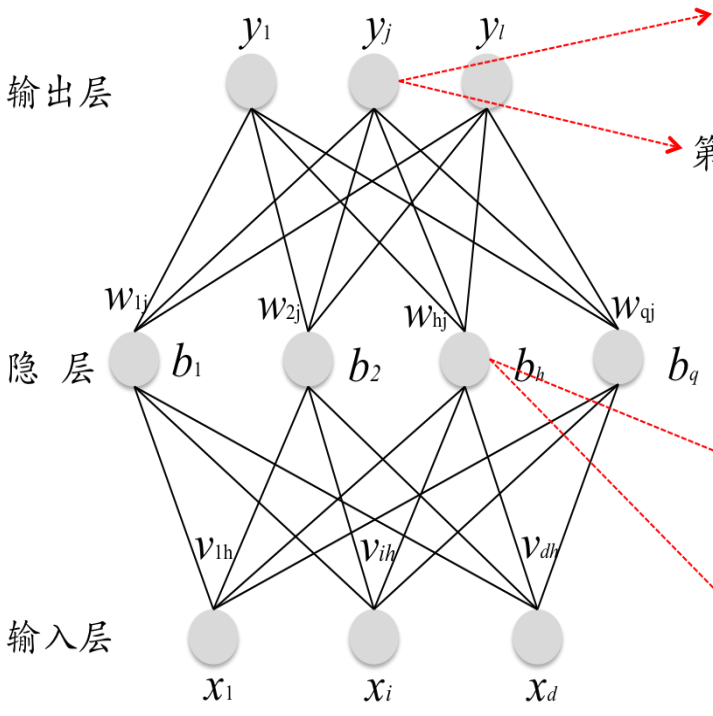
循环神经网络RNN

售票机器人客服如何理解右边两句中“Beijing”？Beijing是目的地还是出发地?



循环神经网络RNN

传统神经网络结构



第j个输出层神经元的输出：

$$\hat{y}_j = f(\beta_j - \theta_j)$$

第j个输出层神经元的输入：

$$\beta_j = \sum_{h=1}^{q} w_{hj} b_h$$

第h个隐层神经元的输出：

$$b_h = f(\alpha_h - \gamma_h)$$

第h个隐层神经元的输入：

$$\alpha_h = \sum_{i=1}^{d} v_{ih} x_i$$

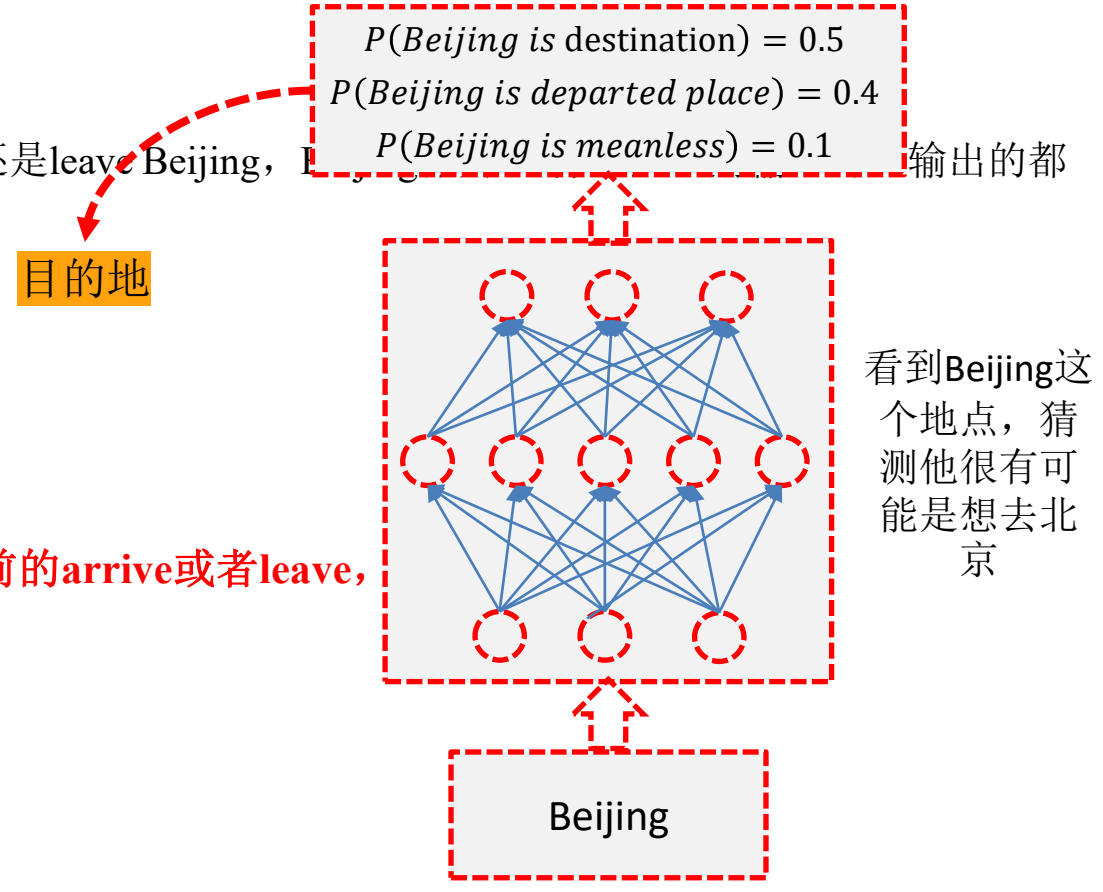
循环神经网络RNN

传统神经网络结构

- 对一般的神经网络，无论是arrive Beijing还是leave Beijing，[illegible]输出的都是Destination

- Input 一样的内容，Output就是一样的内容

- **我们希望神经网络有记忆，记得Beijing前的arrive或者leave，**



循环神经网络RNN

基本概念

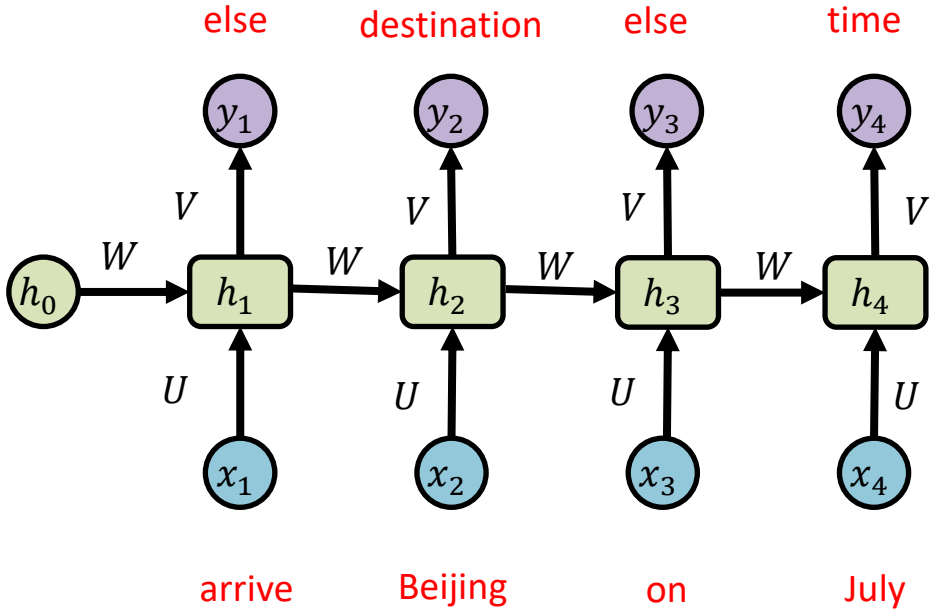
循环神经网络（Recurrent Neural Networks）是一种对序列型数据进行建模的深度模型。

$$y_4 = f(Vh_4 + b) = f(x_1, x_2, x_3, x_4)$$

$x = (x_1 \quad \cdots \quad x_n)$，其中$x_i$是向量。

序列型数据：

- 自然语言处理： x_i视为一个词语
- 语音处理： x_i视为一个每帧的声音信号
- 时间序列：如每日股价



循环神经网络RNN

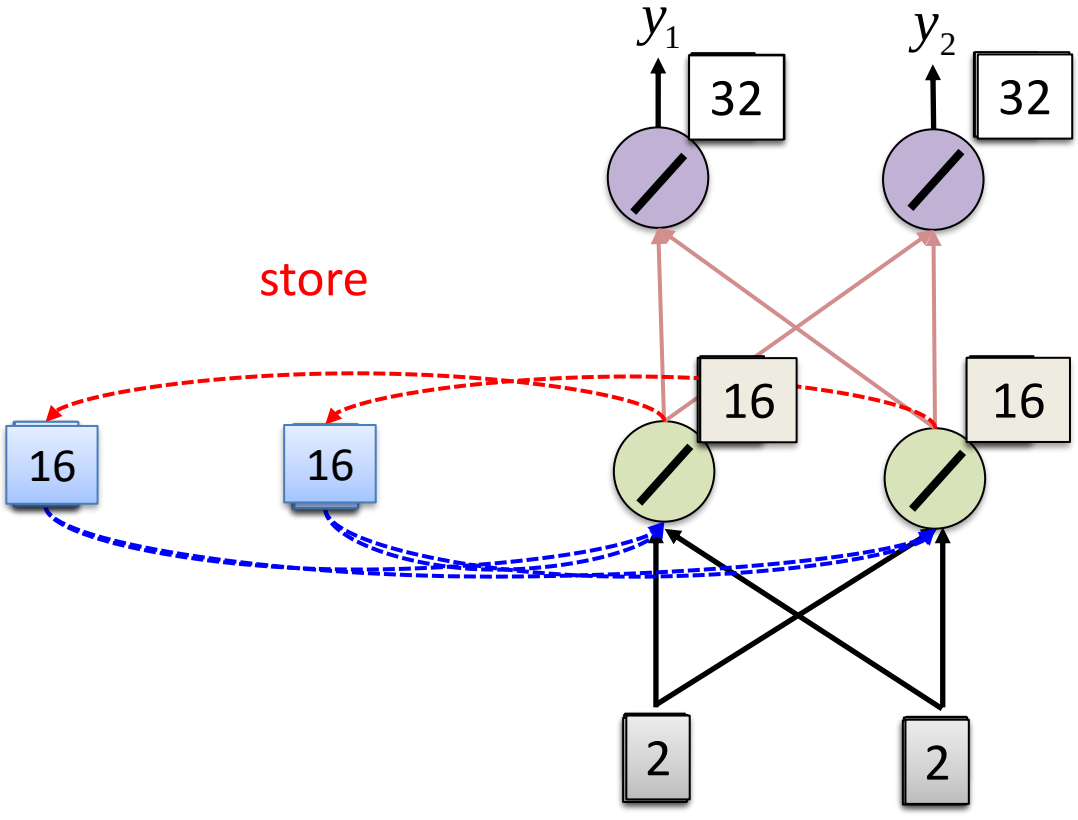
基本概念

- Input sequence: $\begin{bmatrix}1\\1\end{bmatrix}\ \begin{bmatrix}1\\1\end{bmatrix}\ \begin{bmatrix}2\\2\end{bmatrix}\ \cdots\cdots$
- output sequence: $\begin{bmatrix}4\\4\end{bmatrix}\ \begin{bmatrix}12\\12\end{bmatrix}\ \begin{bmatrix}32\\32\end{bmatrix}$

其中：

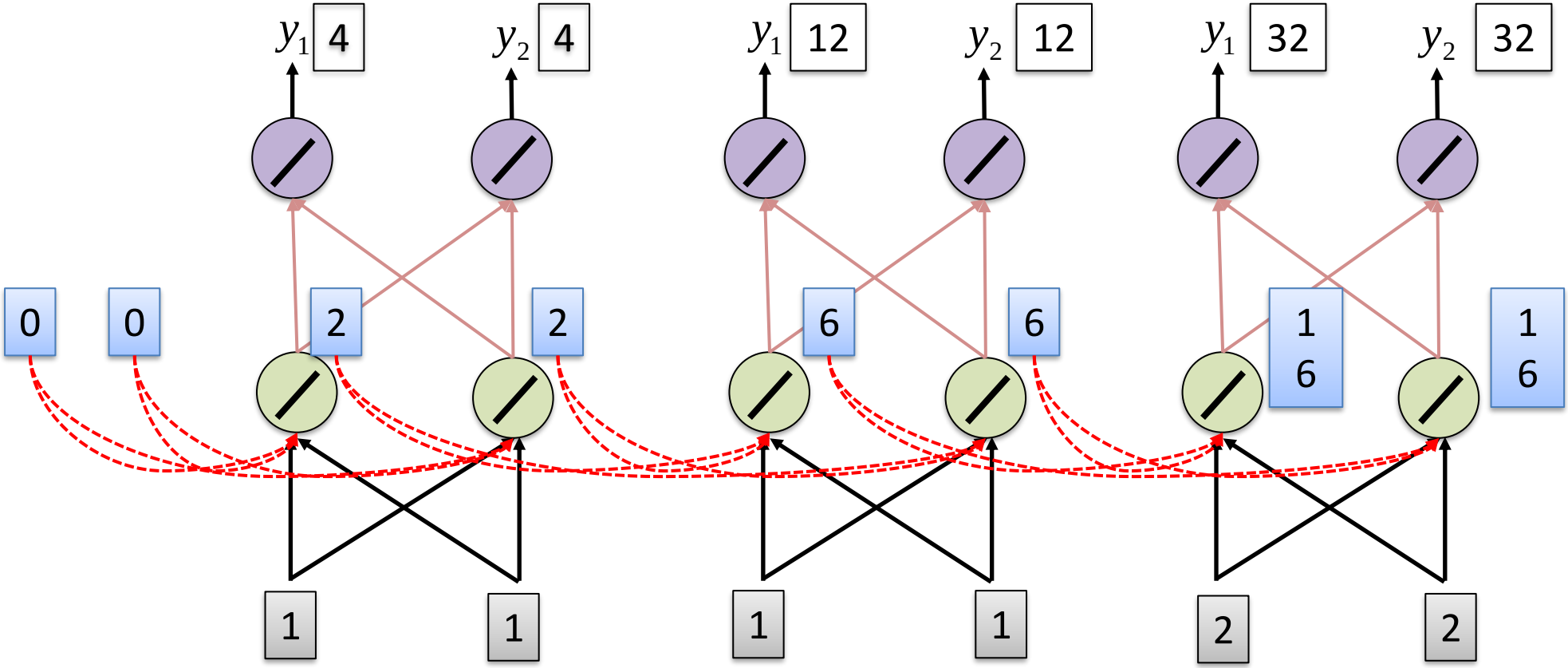
- 所有 weights 都是 “1”, 没有 bias
- 所有激活函数都是 linear
- 隐层设置初始值为[0, 0]

改变 sequence 顺序，改变 output.



循环神经网络RNN

基本概念



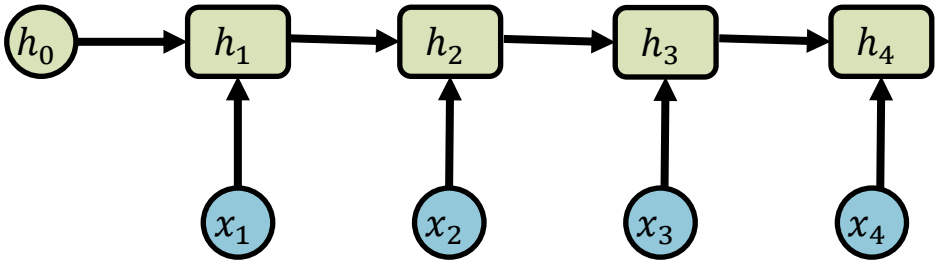
循环神经网络RNN

隐状态（Hidden State）h

- 记忆储存： h可以对序列的数据提取特征，然后再转为输出

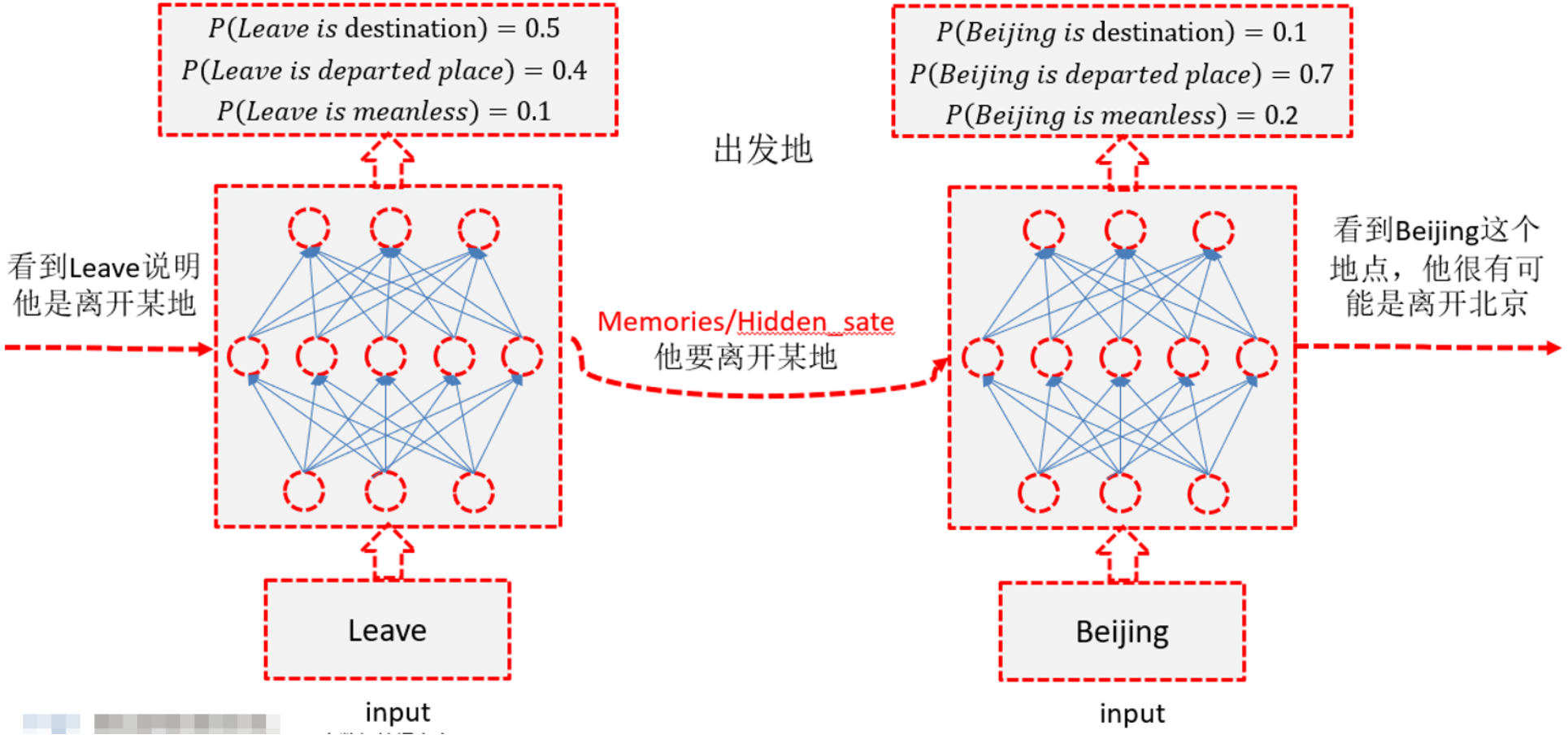
$$h_1 = f(Ux_1 + Wh_0 + b)$$

$$h_2 = f(Ux_2 + Wh_1 + b)$$



- U、W：权值矩阵；b：偏置项； f：激活函数，在经典RNN中，一般使用tanh作为激活函数。
- 一个箭头表示对相应的向量做一次类似于$f(Wx + b)$的变换。
- 在计算时，**每一步使用的参数 U、W、b 都是一样的**，即**每个步骤的参数都是共享的**。

循环神经网络RNN

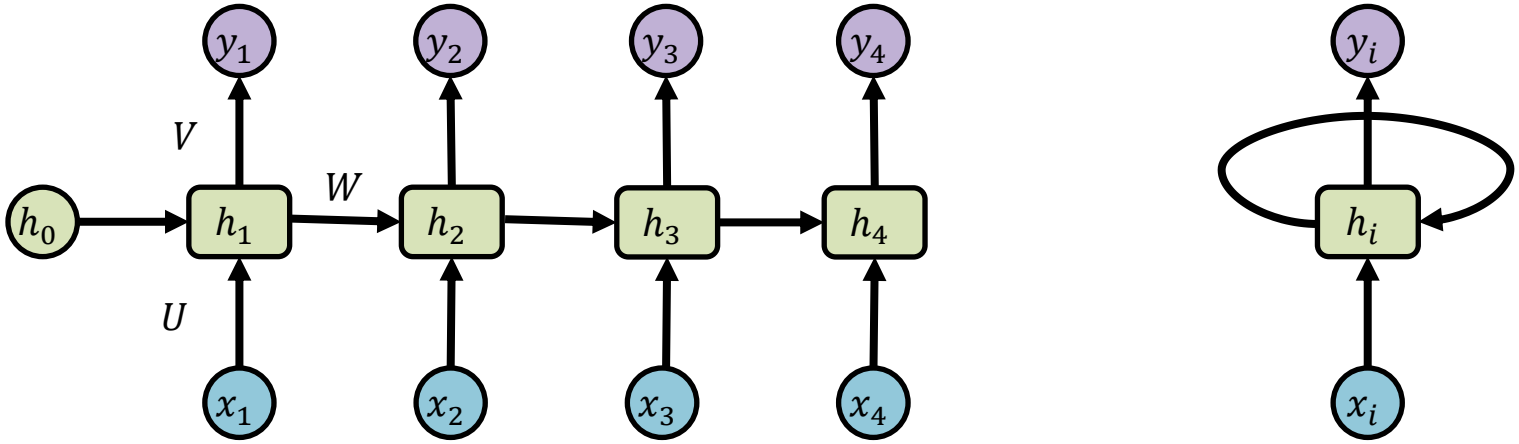

P(Leave is destination) = 0.5
P(Leave is departed place) = 0.4
P(Leave is meanless) = 0.1
出发地
P(Beijing is destination) = 0.1
P(Beijing is departed place) = 0.7
P(Beijing is meanless) = 0.2
看到Leave说明他是离开某地
Memories/Hidden_sate
他要离开某地
看到Beijing这个地点，他很有可能是离开北京
Leave
Beijing
input
input

循环神经网络RNN

输出状态

最原始的RNN结构输入x和输出y的序列必须是等长的。

$$y_1 = SoftMax(Vh_1 + c)$$



循环神经网络RNN

随时间反向传播（BPTT）算法

真实值： y_t

$$z_t \stackrel{\text{def}}{=} Vh_t$$

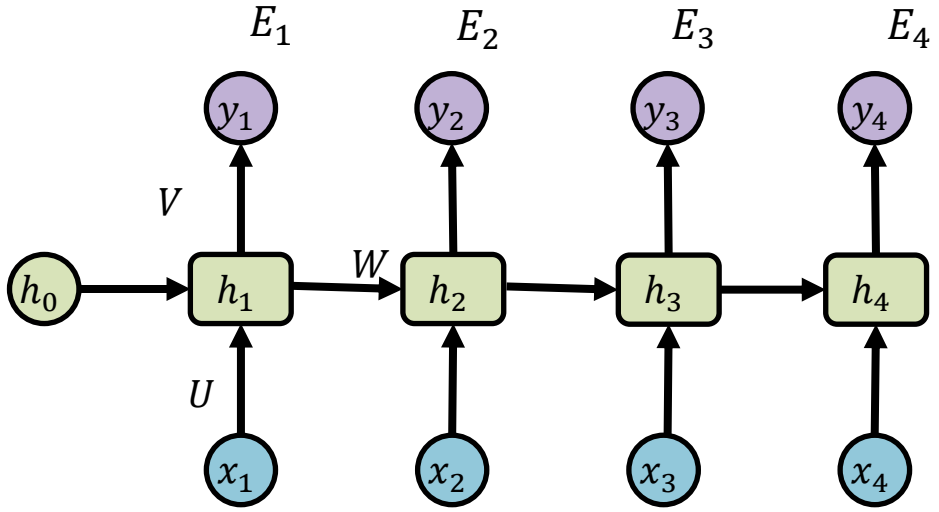
预测： $\hat{y}_k = softmax(Vh_t) = softmax(z_t)$

隐层： $h_t = tanh(Ux_t + Wh_{t-1})$

损失函数： $E_t(y_t - \hat{y}_k) = -y_t \ln(\hat{y}_k)$

$$E(y - \hat{y}) = -\sum_t^T y_t \ln(\hat{y}_k)$$

则： $\frac{\partial E}{\partial W} = \sum_t^T \frac{\partial E_t}{\partial W}$



循环神经网络RNN

随时间反向传播（BPTT）算法

$$z_t \overset{\text{def}}{=} Vh_t$$

预测： $\hat{y}_k = softmax(Vh_t) = softmax(z_t)$

隐层： $h_t = tanh(Ux_t + Wh_{t-1})$

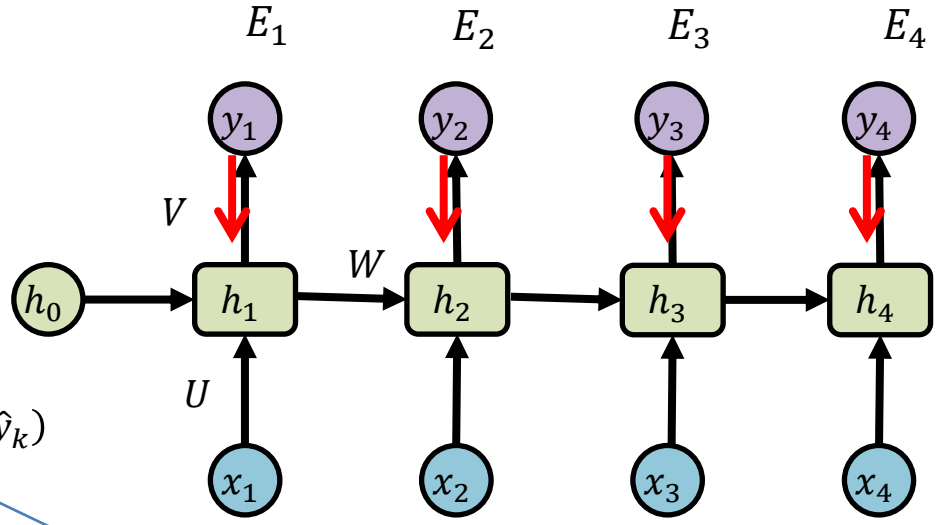
损失函数： $E_t(y_t - \hat{y}_k) = -y_t \ln(\hat{y}_k)$

$$E(y - \hat{y}) = -\sum_t^T y_t \ln(\hat{y}_k)$$

则： $\frac{\partial E}{\partial W} = \sum_t^T \frac{\partial E_t}{\partial W}$

$\frac{\partial E_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3}\frac{\partial z_3}{\partial V} = (y_t - \hat{y}_k) \otimes h_3$，其中 $\frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial z_3} = (y_t - \hat{y}_k)$

$\otimes$ 是两个向量的外积



https://blog.csdn.net/qian99/article/details/78046329

循环神经网络RNN

随时间反向传播（BPTT）算法

预测：$\hat{y}_k = softmax(z_t)$，$z_t \stackrel{\text{def}}{=} Vh_t$

隐层：$h_t = tanh(s_t)$，$s_t \stackrel{\text{def}}{=} Ux_t + Wh_{t-1}$

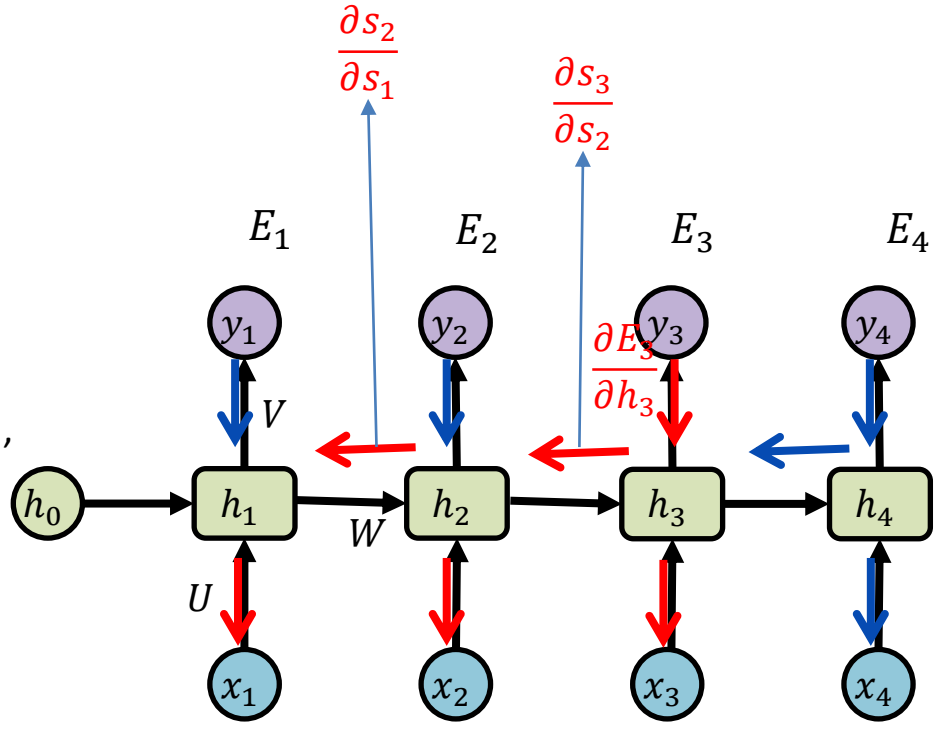
损失函数：$E_t(y_t - \hat{y}_k) = -y_t \log(\hat{y}_k)$

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial h_3}\frac{\partial h_3}{\partial W}$$

，其中$h_3 = tanh(Ux_3 + Wh_2), h_2 = tanh(Ux_2 + Wh_1),$

$h_1 = tanh(Ux_1 + Wh_0)$，$\delta_k \stackrel{\text{def}}{=} \frac{\partial E_t}{\partial s_k}$

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial s_3}\frac{\partial s_3}{\partial W} = \sum_{k=0}^{3}\frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial h_3}\frac{\partial h_3}{\partial h_k}\frac{\partial h_k}{\partial W}$$



循环神经网络RNN

随时间反向传播（BPTT）算法：更新权重

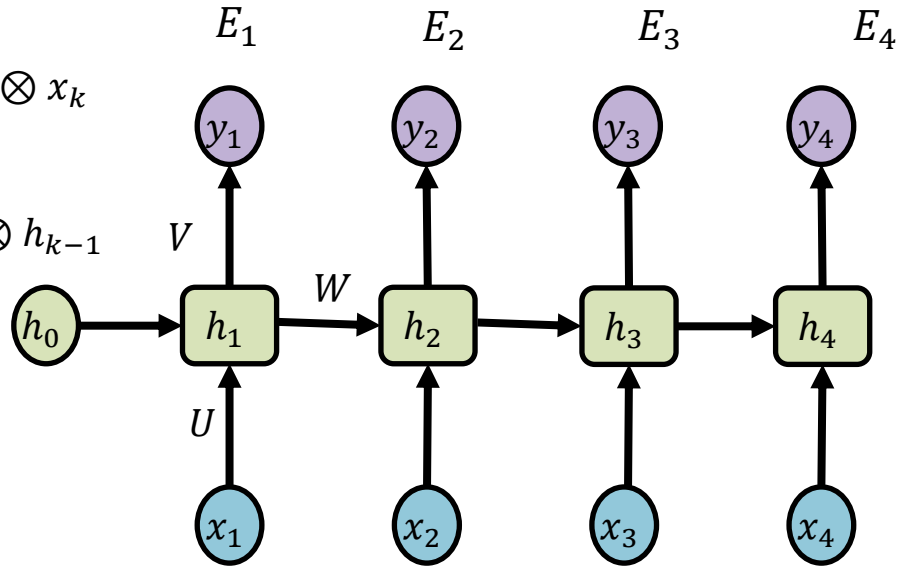
$$W = W - \eta \frac{\partial E}{\partial W} = W - \eta \sum_t^T \frac{\partial E_t}{\partial W} = W - \eta \sum_t^T (\hat{y}_t - y_t) \otimes h_t$$

$$U = U - \eta \frac{\partial E}{\partial U} = U - \eta \sum_t^T \frac{\partial E_t}{\partial U} = U - \eta \sum_t^T \sum_k^t \delta_k \otimes x_k$$

$$V = V - \eta \frac{\partial E}{\partial V} = V - \eta \sum_t^T \frac{\partial E_t}{\partial W} = V - \eta \sum_t^T \sum_k^t \delta_k \otimes h_{k-1}$$

其中，$\delta_t = \frac{\partial E_t}{\partial s_t} = V^T(\hat{y}_t - y_t) \odot (1 - h_t \odot h_t)$

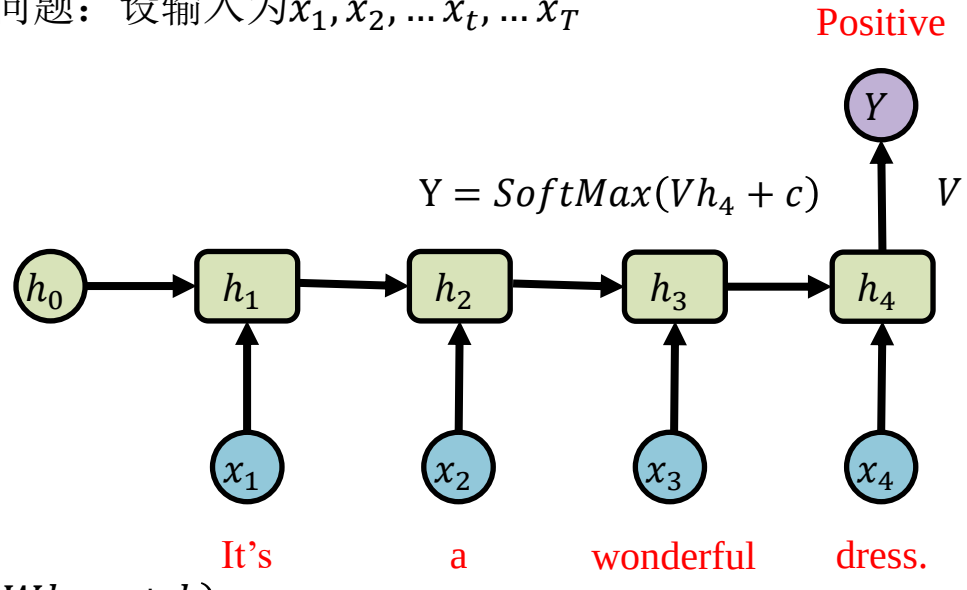
$$\delta_k = \frac{\partial E_t}{\partial s_k} = (W^T \delta_{k+1}) \odot (1 - h_k \odot h_k)$$



RNN变体结构

N VS 1 RNN结构

- 通常用于处理序列分类问题：设输入为$x_1, x_2, \ldots x_t, \ldots x_T$
  - 文本分类
  - 句子情感倾向
  - 视频类别



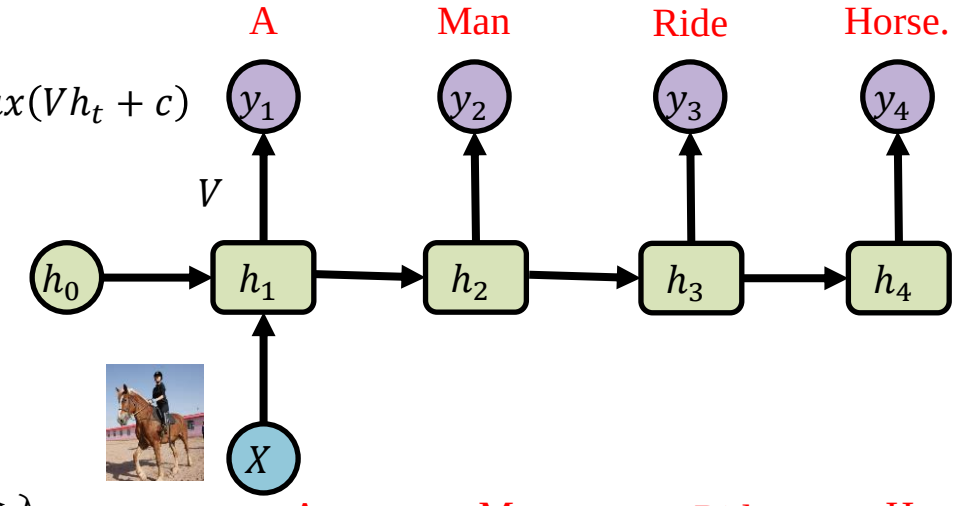
- 隐状态： $h_t = f(Ux_t + Wh_{t-1} + b)$
- 输出：$Y = SoftMax(Vh_T + c)$

RNN变体结构

1 VS N RNN结构

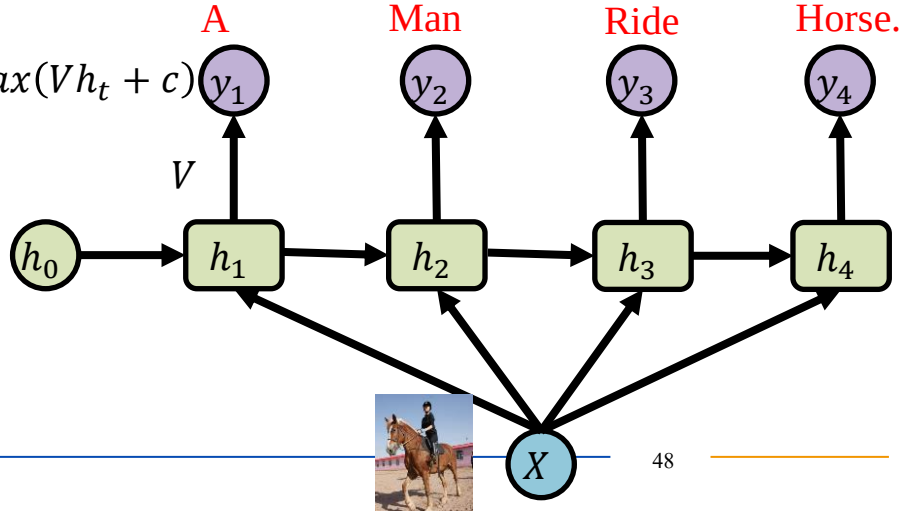
$$y_t = SoftMax(Vh_t + c)$$

- 设输入为X，输出为$y_1, y_2, \dots y_t, \dots y_T$
  - 图像生成文字
  - 从类别中生成音乐或语音
- 只在序列开始进行输入计算。
- 还有一种结构是把输入信息 x 作为每个阶段的输入。



$$y_t = SoftMax(Vh_t + c)$$

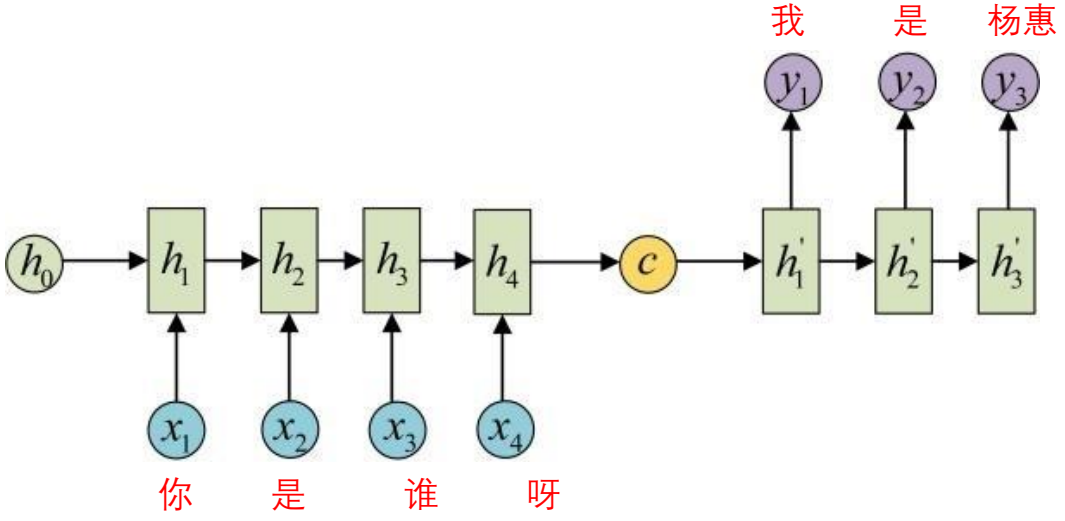
- 隐状态： $h_t = f(UX + Wh_{t-1} + b)$
- 输出：Y $= SoftMax(Vh_T + c)$



RNN变体结构

N vs M

- 这种结构又叫Encoder-Decoder模型，也可以称之为Seq2Seq模型。
- 可以考虑为两个RNN网络构成
- 原始的N vs N RNN要求序列等长，然而我们遇到的大部分问题序列都是不等长的，如**机器翻译**中，源语言和目标语言的句子往往并没有相同的长度。



RNN变体结构

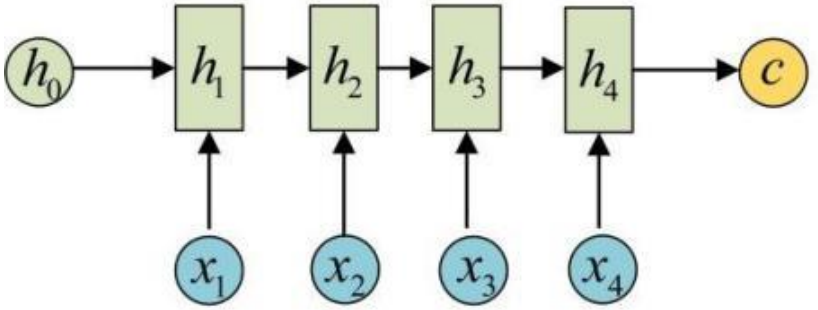
N vs M

- **为此，Encoder-Decoder结构先将输入数据编码成一个上下文向量c：**
- 得到c有多种方式，最简单的方法就是把Encoder的最后
- 还可以对最后的隐状态做一个变换得到c。
- 也可以对所有的隐状态做变换。

$$（1）\ c = h_4$$

$$（2）\ c = q(h_4)$$

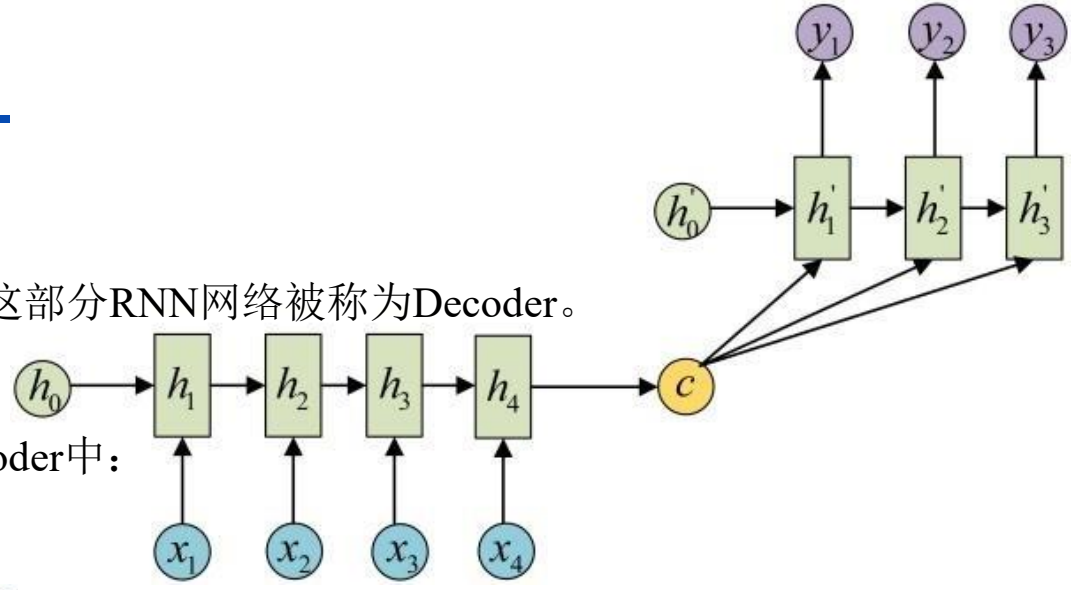
$$（3）\ c = q(h_1, h_2, h_3, h_4)$$



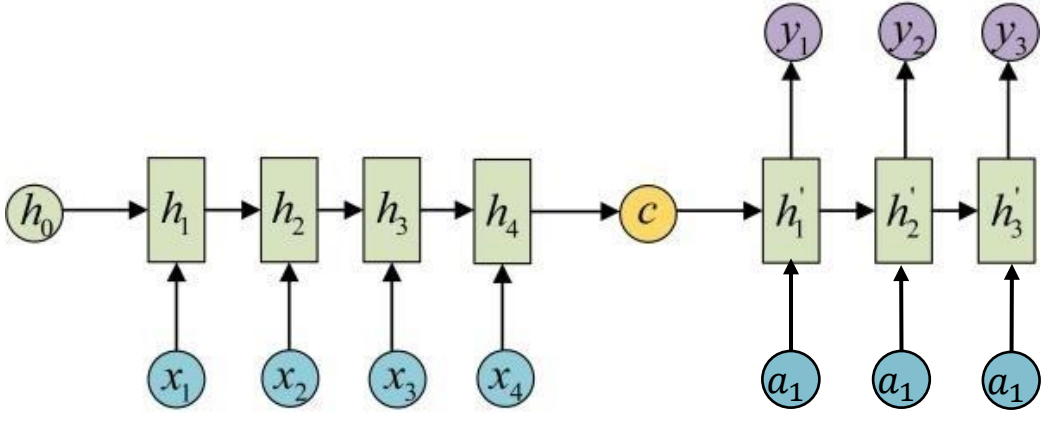
RNN变体结构

N vs M

拿到 c 之后，就用另一个RNN网络对其进行解码，这部分RNN网络被称为Decoder。



- 一种做法是将 c 当做之前初始状态 h_0 输入到Decoder中：



还有一种做法是将c当做每一步的输入

RNN变体结构

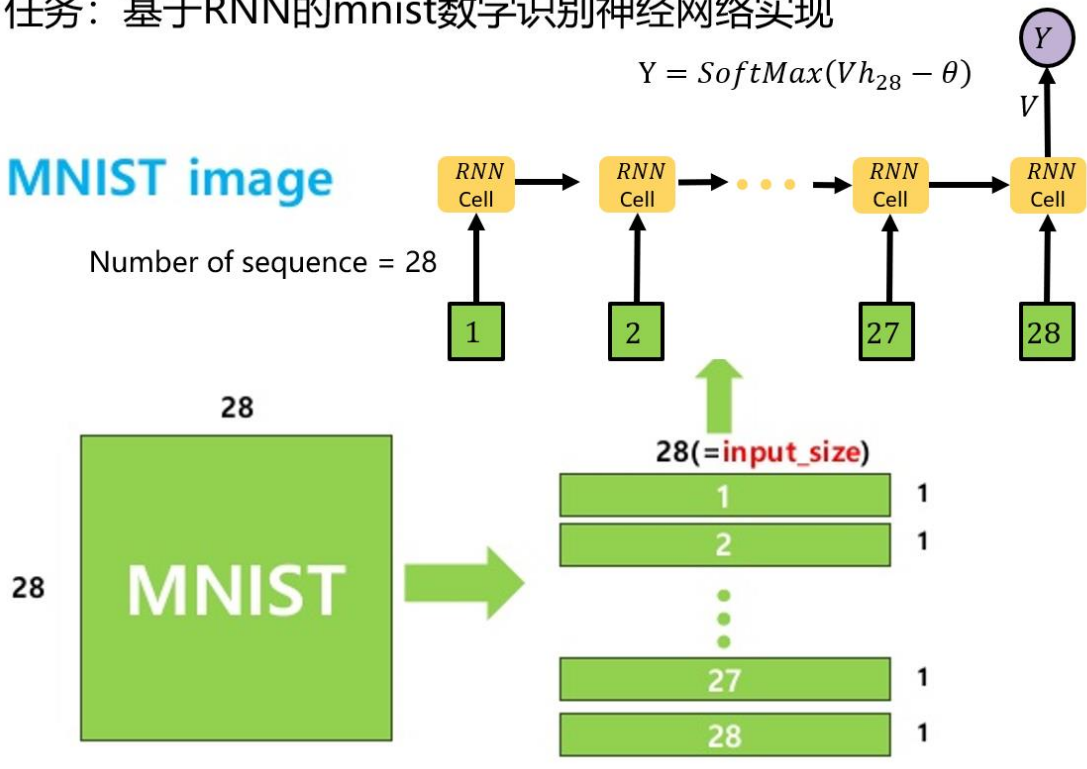
N vs M

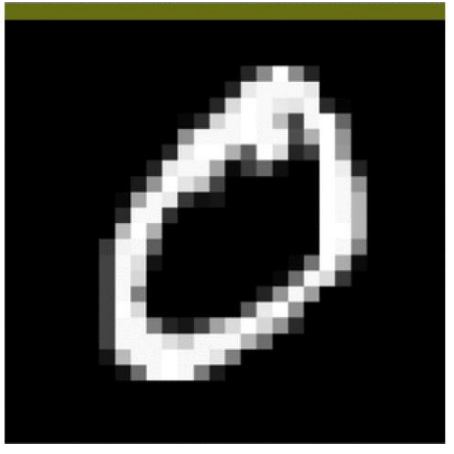
由于这种Encoder-Decoder结构不限制输入和输出的序列长度，因此应用的范围非常广泛，比如：

- 机器翻译，Encoder-Decoder的最经典应用，事实上这一结构就是在机器翻译领域最先提出的。
- 文本摘要，输入是一段文本序列，输出是这段文本序列的摘要序列。
- 阅读理解，将输入的文章和问题分别编码，再对其进行解码得到问题的答案。
- 语音识别，输入是语音信号序列，输出是文字序列。

循环神经网络RNN

任务：基于RNN的mnist数字识别神经网络实现

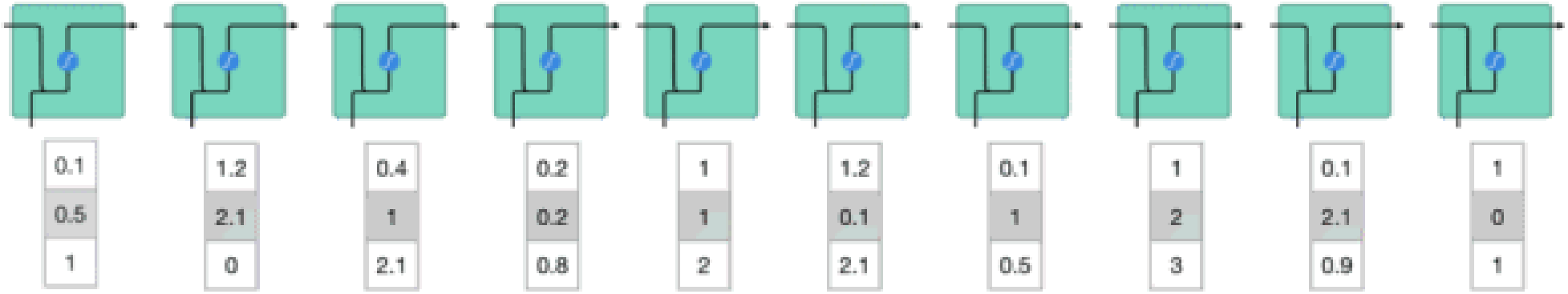




循环神经网络RNN

RNN回顾

RNN 工作动态：第一个单词被转换成机器可读的向量。然后，RNN 逐个处理向量序列。



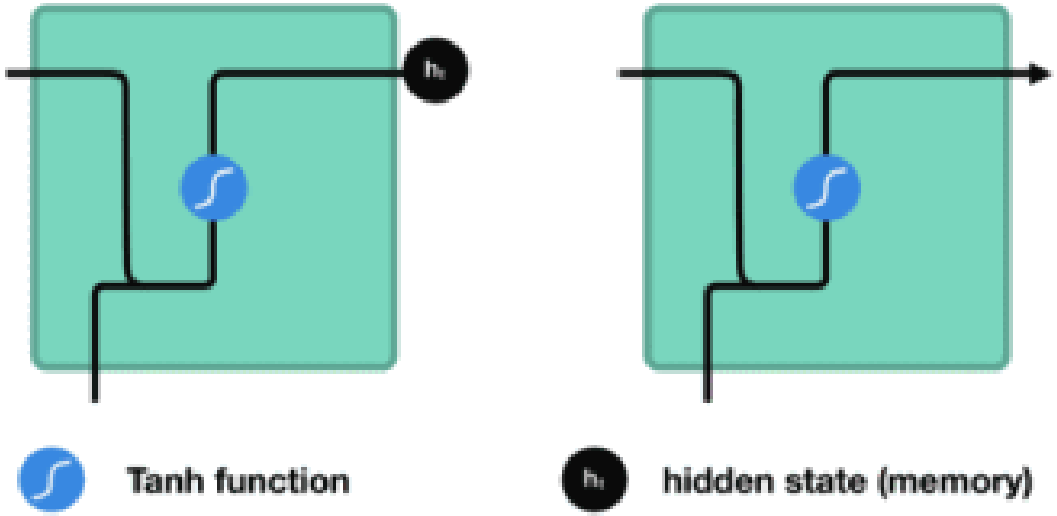
arrive Beijing

逐个处理序列

循环神经网络RNN

RNN回顾

在处理过程中，它将之前的隐状态传递给序列的下一个步骤。隐状态作为神经网络的记忆，保存着网络先前观察到的数据信息。

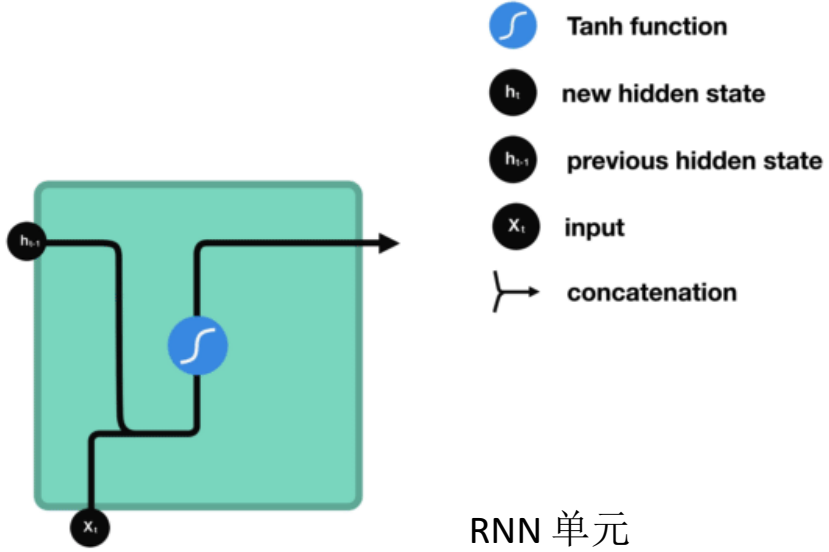


传递隐藏状态到下一时间步

循环神经网络RNN

RNN回顾

计算隐状态：首先，输入和之前的隐状态组合成一个向量。这个向量现在有当前输入和先前输入的信息。向量通过 tanh 激活，输出是新的隐状态，或神经网络的记忆。



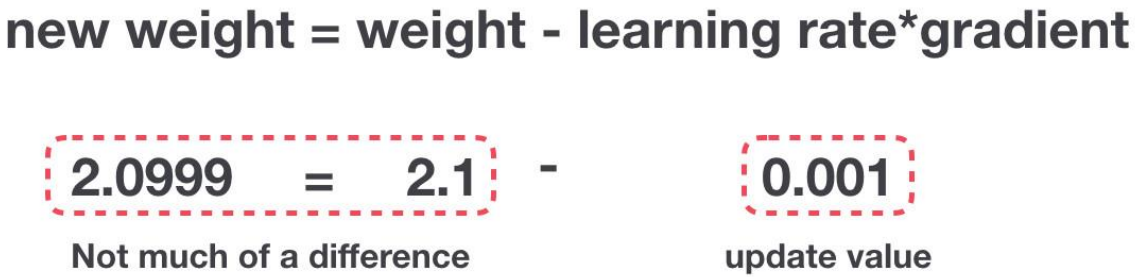
RNN 单元

目录

长短时记忆网络LSTM

梯度爆炸和梯度消失，参考介绍：https://www.cnblogs.com/mengnan/p/9480804.html

- 在 RNN 中，因为通常前期的层会因为梯度消失而停止学习，RNN 会忘记它在更长的序列中看到的东西，从而只拥有短期记忆。



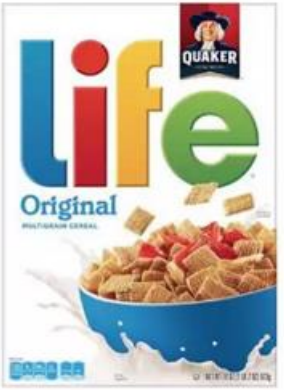
- 为了解决长序列（长时间记忆），提出LSTM（此外还有GRU）。

长短时记忆网络LSTM

记住重要的信息

Amazing! This box of cereal gave me a perfectly balanced breakfast, as all things should be. I only ate half of it but will definitely be buying again!

原文



A Box of Cereal
$3.99

Amazing! This box of cereal gave me a perfectly balanced breakfast, as all things should be. I only ate half of it but will definitely be buying again!

你可能记住的

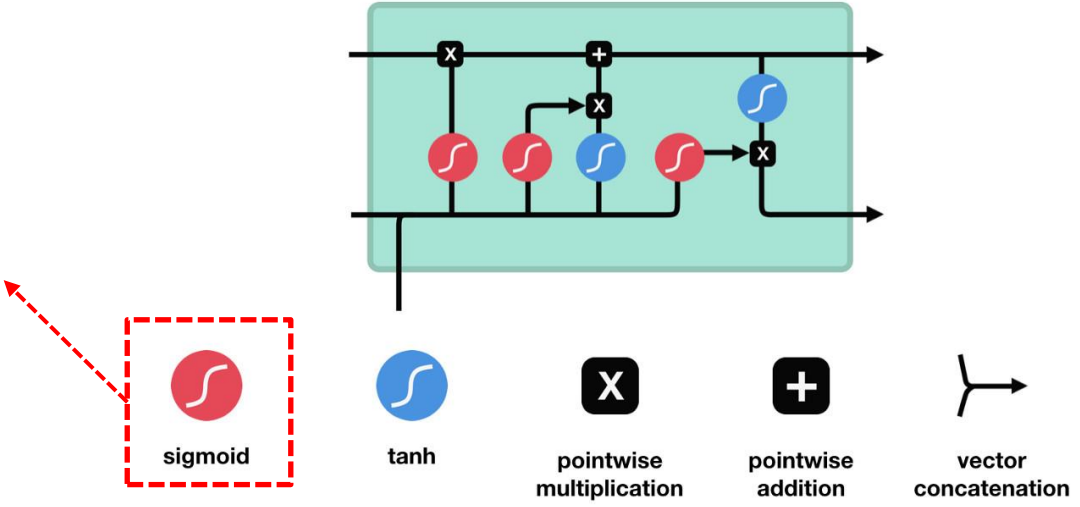
- 这就是 LSTM 或 GRU 的作用，它可以学会只保留相关的信息以进行预测。

长短时记忆网络LSTM

长短时记忆LSTM(Long Short-Term Memory)网络

LSTM 具有与循环神经网络相似的控制流，它在前向传播时处理传递信息的数据，两者区别在于单元内的处理过程不同，它有三个门：忘记门、输入门、输出门。在训练过程中，门可以学习到哪些信息是需要保存或遗忘的。

门控机制
主要由 sigmoid
激活函数构成

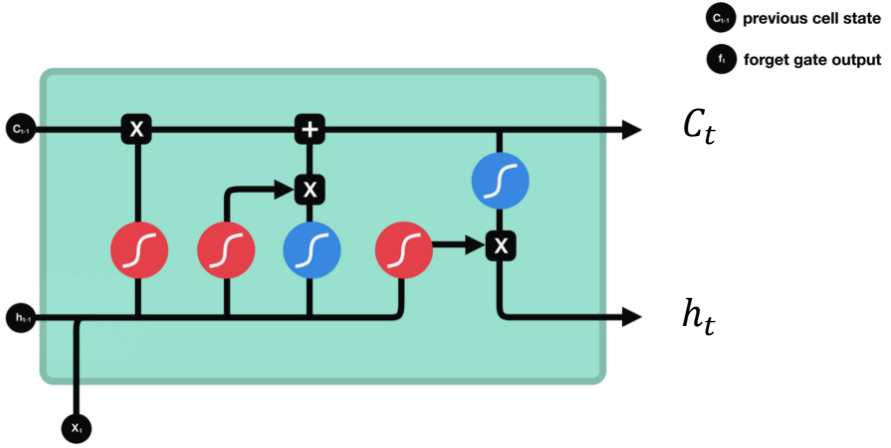


长短时记忆网络LSTM

遗忘门（forget gate）：遗忘或保存

决定哪些信息应该被丢弃或保存。

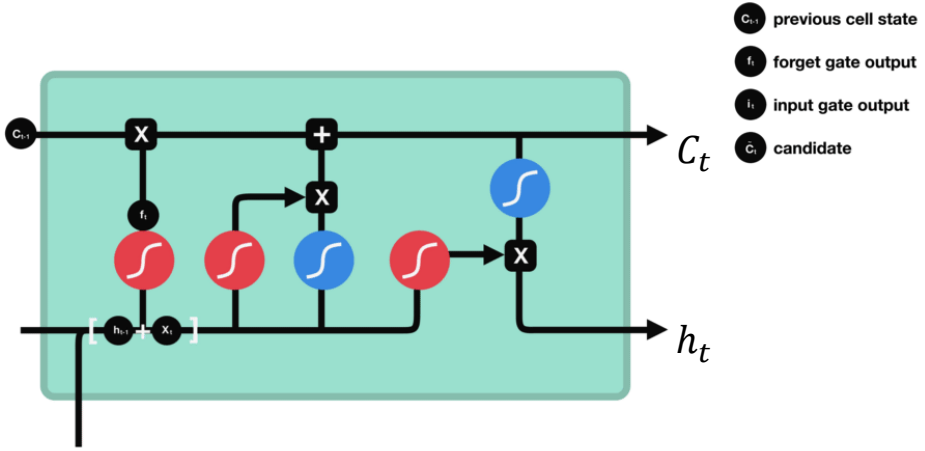
在遗忘门中，来自先前隐状态的信息和来自当前输入的信息传递到 sigmoid 函数，并将值压缩到 0 和 1 之间。越接近 0 意味着丢弃，越接近 1 意味着保留。



长短时记忆网络LSTM

输入门（input gate）：更新单元状态

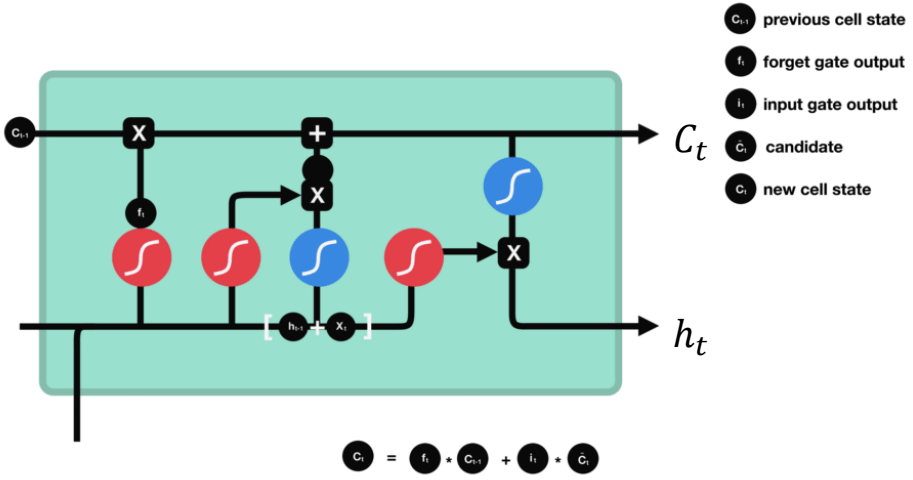
首先，我们将前面的隐状态和当前输入传递给一个 sigmoid 函数，它通过将值转换为 0 到 1 来决定将更新哪些值。0 表示不重要，1 表示重要。还可以将隐状态和当前输入传递给 tanh 函数，使值变为-1 到 1 之间的值，以帮助调节神经网络。然后将 tanh 输出与 sigmoid 输出相乘，sigmoid 输出将决定保留 tanh 输出的重要信息。



长短时记忆网络LSTM

单元状态

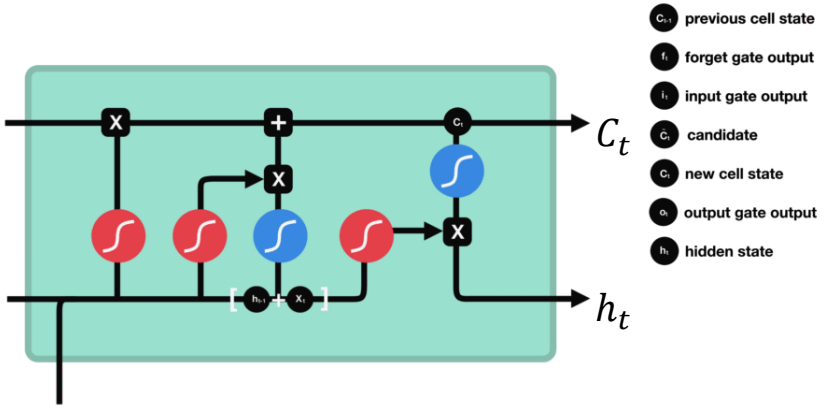
- 首先，单元状态逐点乘以遗忘向量，如果它与接近 0 的值相乘，就有可能在单元状态中得到低值。
- 然后，从输入门读取上一步输出，并逐点相加，将单元状态更新为神经网络认为相关的新值，这就得到了新的单元状态。



长短时记忆网络LSTM

输出门（output gate）：决定下一个隐藏状态

- 记住，隐藏状态包含先前输入的信息，隐藏状态也用于预测。
- 首先，我们将前面的隐状态和当前输入传递给一个 sigmoid 函数。
- 然后我们将新修改的单元状态传递给 tanh 函数。我们将 tanh 输出与 sigmoid 输出相乘，以确定隐状态应该包含的信息。新的单元状态和新的隐藏状态随后被转移到下一步中。

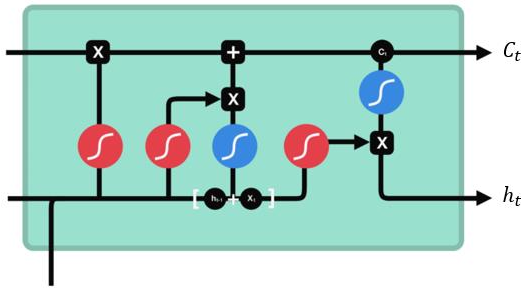


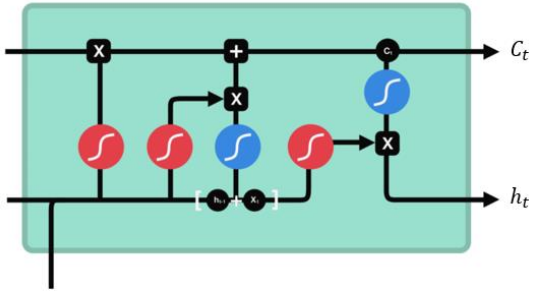
长短时记忆网络LSTM

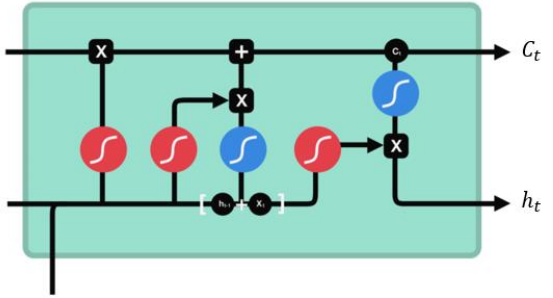
传输示例

- Input sequence: $\begin{bmatrix}1\\1\end{bmatrix}$ $\begin{bmatrix}2\\2\end{bmatrix}$ $\begin{bmatrix}3\\3\end{bmatrix}$ …… output sequence: $\begin{bmatrix}0.37\\0.37\end{bmatrix}$ $\begin{bmatrix}0.76\\0.76\end{bmatrix}$ $\begin{bmatrix}0.54\\0.54\end{bmatrix}$

- 假设hidden_size = 4，$W_i, W_f, W_c, W_o \in \mathbb{R}^{6\times4}$，偏置项为0。

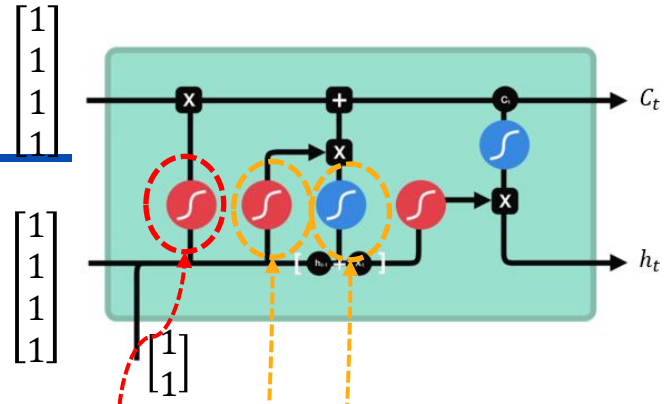






长短时记忆网络LSTM

假设hidden_size = 4，$W_i, W_f, W_c, W_o \in \mathbb{R}^{6\times4}$，偏置项为0。



- forget gate: $f_t = \sigma(W_f[h_{t-1}, x_t])$

$$sigmoid\begin{bmatrix} -1.086 & 0.9976 & 0.283 & -1.506 & -0.5786 & 1.651 \\ -2.426 & -0.429 & 1.266 & -0.8667 & -0.6787 & -0.0947 \\ 1.491 & -0.6387 & -0.444 & -0.4343 & 2.205 & 2.188 \\ 1.004 & 0.3862 & 0.7373 & 1.491 & -0.936 & 1.176 \end{bmatrix} \times \begin{bmatrix} 1. \\ 1. \\ 1. \\ 1. \\ 1. \\ 1. \end{bmatrix} = \begin{bmatrix} 0.44064978 \\ 0.03807978 \\ 0.98746603 \\ 0.97933431 \end{bmatrix}$$

- input gate: $i_t = \sigma(W_i[h_{t-1}, x_t])$

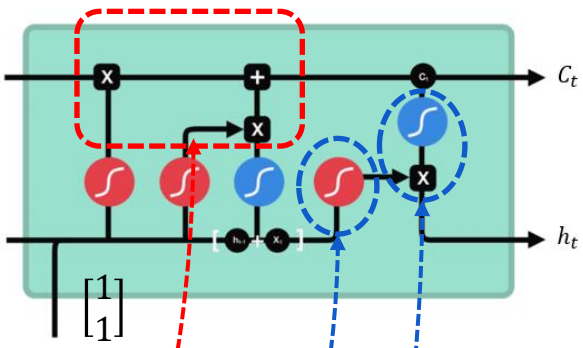
$$sigmoid\begin{bmatrix} -1.254 & -0.6377 & 0.907 & -1.429 & -0.14 & -0.862 \\ -0.2556 & -2.799 & -1.771 & -0.6997 & 0.9272 & -0.1736 \\ 0.002846 & 0.688 & -0.8794 & 0.2837 & -0.805 & -1.728 \\ -0.3909 & 0.5737 & 0.3386 & -0.01183 & 2.393 & 0.4128 \end{bmatrix} \times \begin{bmatrix} 1. \\ 1. \\ 1. \\ 1. \\ 1. \\ 1. \end{bmatrix} = \begin{bmatrix} 0.03183251 \\ 0.00839264 \\ 0.08035127 \\ 0.96494223 \end{bmatrix}$$

- $\tilde{C}_t = tanh(W_c[h_{t-1}, x_t])$

$$tanh\begin{bmatrix} 0.9785 & 2.238 & -1.294 & -1.039 & 1.744 & -0.798 \\ 0.02968 & 1.069 & 0.8906 & 1.755 & 1.496 & 1.069 \\ -0.773 & 0.795 & 0.3142 & -1.326 & 1.417 & 0.807 \\ 0.0455 & -0.233 & -1.198 & 0.1996 & 0.4685 & -0.831 \end{bmatrix} \times \begin{bmatrix} 1. \\ 1. \\ 1. \\ 1. \\ 1. \\ 1. \end{bmatrix} = \begin{bmatrix} 0.94983372 \\ 0.99999339 \\ 0.84377336 \\ -0.91357679 \end{bmatrix}$$

长短时记忆网络LSTM

假设hidden_size = 4，$W_i, W_f, W_c, W_o \in \mathbb{R}^{6\times4}$，偏置项为0。



$$\begin{bmatrix}1\\1\\1\\1\end{bmatrix} \quad \begin{bmatrix}1\\1\end{bmatrix}$$

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

$$\begin{bmatrix}0.44064978\\0.03807978\\0.98746603\\0.97933431\end{bmatrix} \times \begin{bmatrix}1.\\1.\\1.\\1.\end{bmatrix} + \begin{bmatrix}0.03183251\\0.00839264\\0.08035127\\0.96494223\end{bmatrix} \times \begin{bmatrix}0.94983372\\0.99999339\\0.84377336\\-0.91357679\end{bmatrix} = \begin{bmatrix}0.47088537\\0.04647236\\1.05526428\\0.09778549\end{bmatrix}$$

- output gate: $O_t = \sigma(W_o[h_{t-1}, x_t])$

$$sigmoid \begin{bmatrix}1.162 & -1.098 & -2.123 & 1.04 & -0.4033 & -0.126\\-0.8374 & -1.606 & 1.255 & -0.689 & 1.661 & 0.807\\-0.3147 & -1.086 & -0.7324 & -1.213 & 2.088 & 0.1644\\1.15 & -1.268 & 0.181 & 1.178 & -0.335 & 1.031\end{bmatrix} \times \begin{bmatrix}1.\\1.\\1.\\1.\\1.\\1.\end{bmatrix} = \begin{bmatrix}0.17539679\\0.64344133\\0.25093573\\0.87411754\end{bmatrix}$$

- $h_t = O_t * tanh(C_t)$

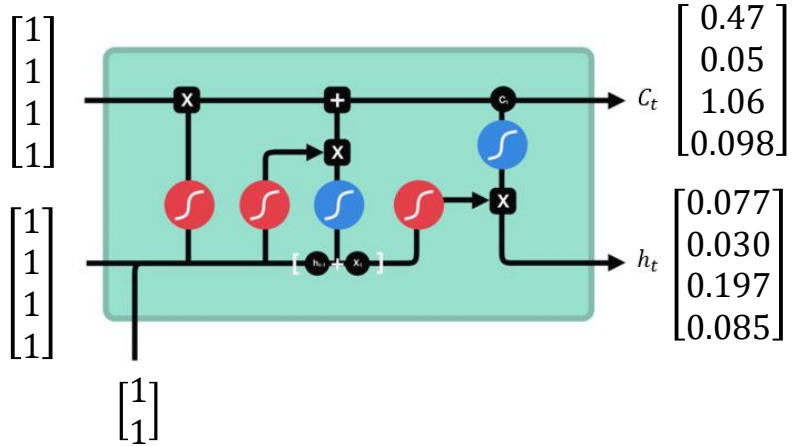
$$\begin{bmatrix}0.17539679\\0.64344133\\0.25093573\\0.87411754\end{bmatrix} \times tanh \begin{bmatrix}0.47088537\\0.04647236\\1.05526428\\0.09778549\end{bmatrix} = \begin{bmatrix}0.07698418\\0.02988073\\0.19669461\\0.08520461\end{bmatrix}$$

长短时记忆网络LSTM

假设hidden_size = 4，$W_i, W_f, W_c, W_o \in \mathbb{R}^{6\times4}$，偏置项为0。

output: $y_t = tanh(Vh_t)$

$$tanh\left(\begin{bmatrix}1.,\ 1.,\ 1.,\ 1.\\ 1.,\ 1.,\ 1.,\ 1.\end{bmatrix} \times \begin{bmatrix}0.07698418\\ 0.02988073\\ 0.19669461\\ 0.08520461\end{bmatrix}\right) = \begin{bmatrix}0.3702943\\ 0.3702943\end{bmatrix}$$



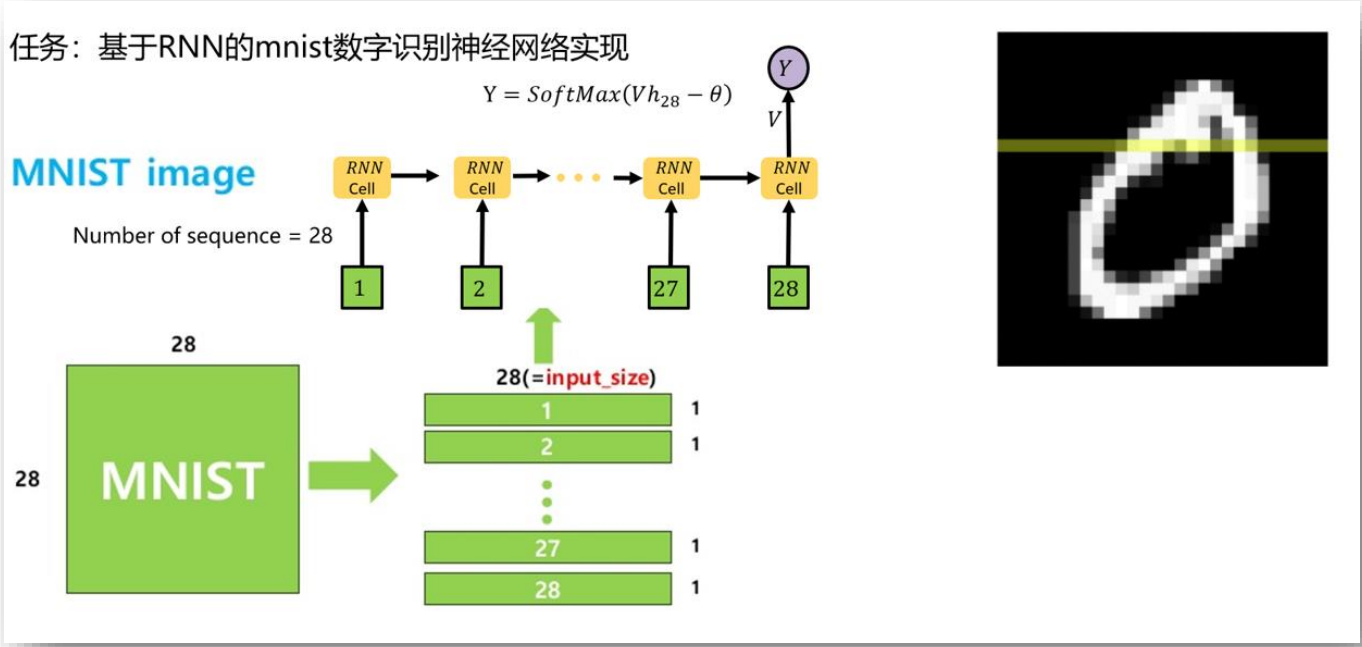
$$\begin{bmatrix}1\\1\\1\\1\end{bmatrix}\quad\begin{bmatrix}1\\1\\1\\1\end{bmatrix}\quad\begin{bmatrix}1\\1\end{bmatrix}$$

$$C_t = \begin{bmatrix}0.47\\0.05\\1.06\\0.098\end{bmatrix}\qquad h_t = \begin{bmatrix}0.077\\0.030\\0.197\\0.085\end{bmatrix}$$

利用RNN&LSTM实现手写数字识别

任务实现

1. 加载数据
2. 数据加工
3. 构建模型（搭建网络）
4. 模型配置
5. 模型训练
6. 性能验证



Thank you!