第六讲 循环网络

《深度学习》

南开大学 人工智能学院

序列问题

- 卷积网络
  - 处理图像，计算机视觉
  - 卷积网络侧重于对单张图像的局部空间特征建模
- 循环网络
  - 处理序列，自然语言处理
  - 循环网络擅长挖掘数据中的序列特性
- 序列学习：从输入序列中预测输出序列
  - 语音识别、视频理解、机器翻译、股票预测
  - 示例：预测下一个词

    输入序列：南开 大学 很 好 ， 我 爱 南开 大学

    预测序列：大学 很 好 ， 我 爱 南开 大学 #
  - 特点：前面的输入和后面的输出是有关联的

序列问题

- 难点1：需要将长时间跨度上的信息有效关联起来
  - 模型需要理解和记住在序列早期出现的信息，并能将这些信息与后续信息相关联
  - 示例：He swarm across the river to get to the other bank
  - 卷积网络：侧重局部信息建模，无法捕捉到信息之间的长期关联

    循环网络：能够更好地建模长期依赖关系，更擅长处理序列数据
- 难点2：重要信息在序列中出现位置的不确定性
  - 要求模型在全局序列中具有搜索和识别关键信息的能力

    2023年我在天津参加深度学习学术会议

    我参加了天津举行的2023年深度学习学术会议

    我参加了天津举行的深度学习学术会议，时间是2023年

6.1 循环神经网络结构

- 预测 t 时刻的输出需要记住 t 时刻之前的序列信息
- 如何存储记忆？
- 隐状态：功能类似于记忆，**存储了到时刻 t 的历史序列信息**
  - 基于当前输入x_t和前一时刻的隐状态 h_{t-1}来计算时刻 t 处的隐状态

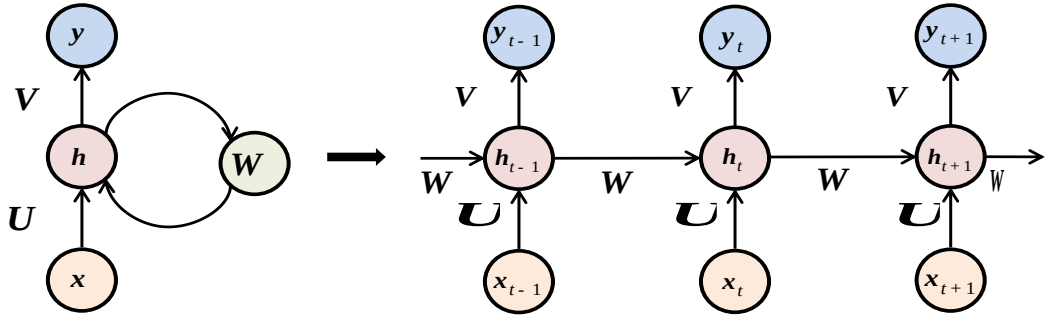
$$h_t = f(x_t, h_{t-1})$$

计算h_t时，不仅要“看着眼前”（ x_t），也要“想着过去”（h_{t-1}）

  - 引入神经网络层中的“线性变换+非线性激活函数”设计f：

$$h_t = \sigma\left(Wh_{t-1} + Ux_t + b_h\right)$$

  - 常用双曲正切函数（tanh）作为激活函数

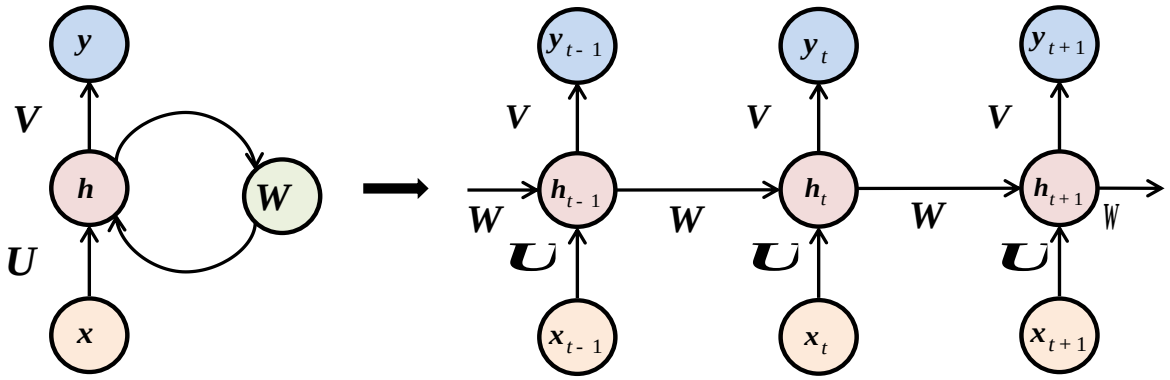


6.1 循环神经网络结构

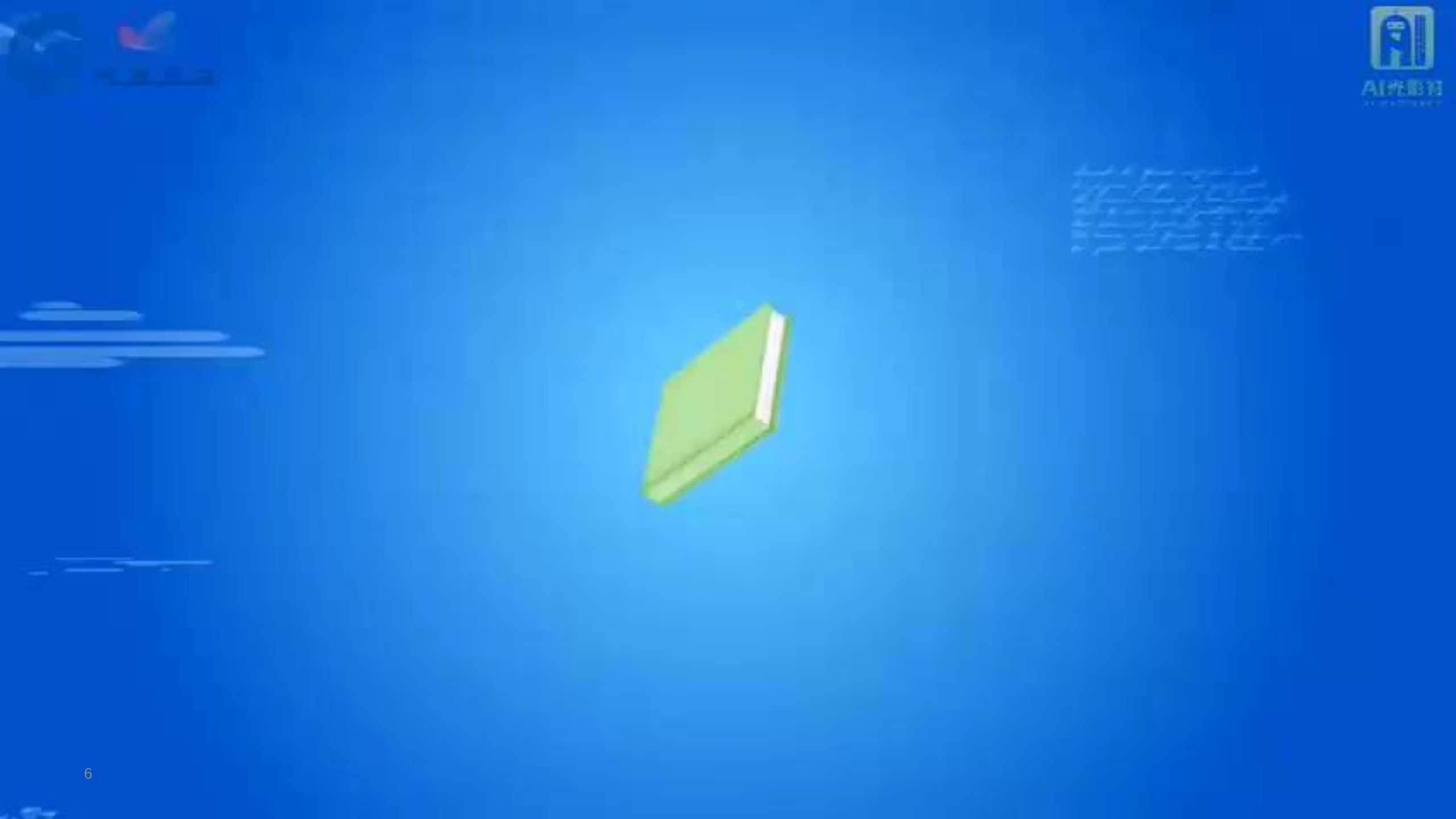
- 输出：时刻 t 处的输出依赖于时刻 t 处隐状态
  - 引入神经网络层中的“线性变换+非线性激活函数”操作：

$$y_t = \sigma(Vh_t + b_q) \quad \text{或} \quad y_t = Vh_t + b_y$$

  - 一般为不使用激活函数的全连接层
  - 通过引入隐状态，输出层只依赖于当前时刻的隐状态，而不需要**显示**依赖之前所有输入 $x_{t-1}, x_{t-2}, \cdots, x_1$



- 循环神经网络（Recurrent Neural Network，RNN）
  - 利用隐藏状态来存储以前所有时刻的信息
- 全连接网络、卷积网络
  - 第 t 层只能接受第 $t-1$ 层的信息



6.1.1 权值共享

- 不同时刻共享相同的隐藏层权重和偏置、输出层权重和偏置

$$h_t = \sigma\left(Wh_{t-1} + Ux_t + b_h\right)$$
$$y_t = Vh_t + b_y$$

  - 权值共享的好处：
    - 在所有时间步学习单一模型，而不需要为每一个时间步学习一个独立的模型，因此参数共享使得模型能够**扩展到不同长度的样本**
    - 参数共享能够减少参数量，从而简化训练难度，提高网络的泛化能力
    - 参数共享能够处理相同信息出现在不同位置的情况

      I went to Beijing in 2009

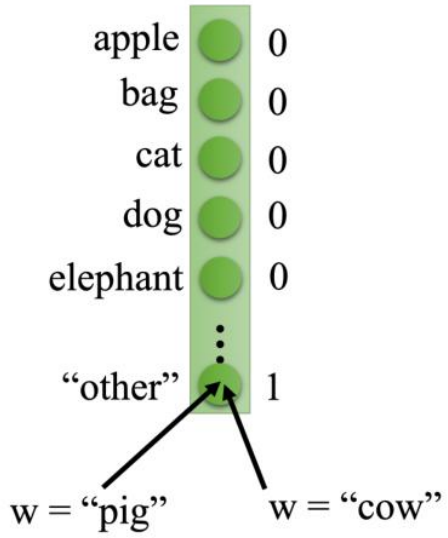
      In 2009, I went to Beijing.

6.1.2 输入输出编码 了解

- 当输入和输出为自然语言句子时的几种编码方式
  - 每个词编码为一个唯一的数字索引
    - 记$|V|$为词表大小，则$[0, |V|-1]$为索引范围
    - t 时刻的输入和输出为**标量**
    - 存在问题：

      对于索引1、2、3，**隐式认为1和2的距离比1和3更近**
  - one-hot coding（独热编码）：将索引映射为0-1**向量**
    - 如果词的索引是整数 i，那么我们将创建一个长度为$|V|$的全0向量，并将第 i 个位置的元素置为1
    - 稀疏向量，只有一个元素为1
    - 不在词表中的词或低频词可以归类到other类
    - 不用编码之间的距离都是一样的，欧式距离均为$\sqrt{2}$

$$\begin{aligned}\text{apple} &= [1,0,0,0,0,\ldots\ldots]\\ \text{bag} &= [0,1,0,0,0,\ldots\ldots]\\ \text{cat} &= [0,0,1,0,0,\ldots\ldots]\\ \text{dog} &= [0,0,0,1,0,\ldots\ldots]\\ \text{elephant} &= [0,0,0,0,1,\ldots\ldots]\end{aligned}$$



(a) 用 other 表示非词表中的单词

6.1.2 输入输出编码

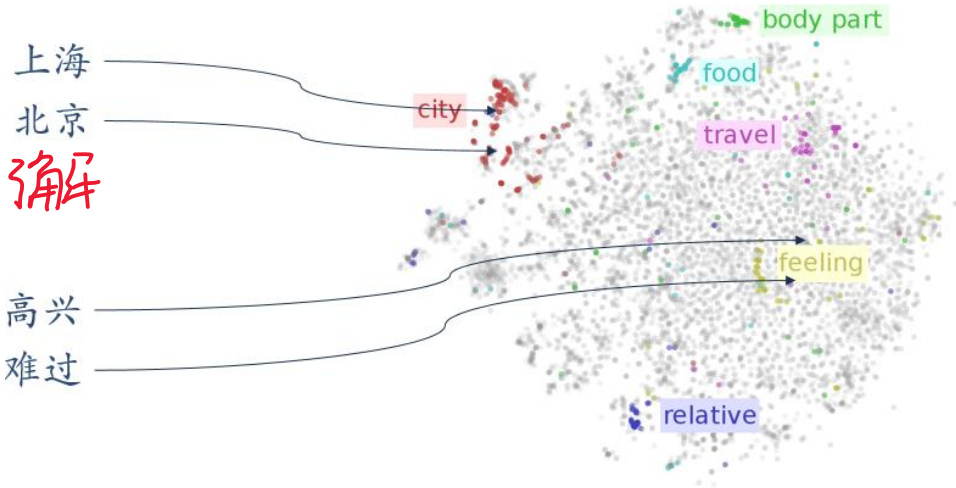
- 当输入和输出为自然语言句子时的几种编码方式
  - 独热编码存在问题：
    - one-hot coding编码下的任意两个向量内积为0，

      表明该编码下的**词彼此之间不含任何相似信息**，

      无论两个词在语义上是近义词还是反义词
    - 在实际应用中，我们更希望具有相似语义的词的编码表示也应该是相似的
  - Word2vec：把一个词映射到一个**低维**稠密向量，从而使得**语义相似的词具有相近的词向量**
    - **词向量把语义上的相似性体现到了词的向量的距离上，从而让词向量获得了语义信息**
    - 通过大规模无监督语料训练可以获得高质量的词向量，

      这些具有语义知识的词向量迁移到其他具体任务上

$$
\begin{aligned}
\text{apple} &= [1,0,0,0,0,\ldots\ldots] \\
\text{bag} &= [0,1,0,0,0,\ldots\ldots] \\
\text{cat} &= [0,0,1,0,0,\ldots\ldots] \\
\text{dog} &= [0,0,0,1,0,\ldots\ldots] \\
\text{elephant} &= [0,0,0,0,1,\ldots\ldots]
\end{aligned}
$$



6.1.2 输入输出编码

- 当输入和输出为自然语言句子时的几种编码方式
  - word2vec不足
    - 本质上是通过训练得到一个词向量的查询表，每个词对应的向量是固定的
    - 无法解决一词多义问题：吃*苹果*与*苹果*手机；*我*（I）*爱我*（my）女儿

6.1.2 输入输出编码

- 当输入和输出为自然语言句子时的几种编码方式
  - 上下文词嵌入
    - 词嵌入向量随着其所在上下文语境的不同而改变
    - 复杂的方式：使用BERT或Transformer编码器的输出作为词嵌入结果
    - 简单的方式：线性词嵌入，对独热编码进行一次线性变换，即Wx，其中 $x \in \mathbb{R}^{|V|}$为独热编码，$W \in \mathbb{R}^{d \times |V|}$为**可学习**的嵌入矩阵

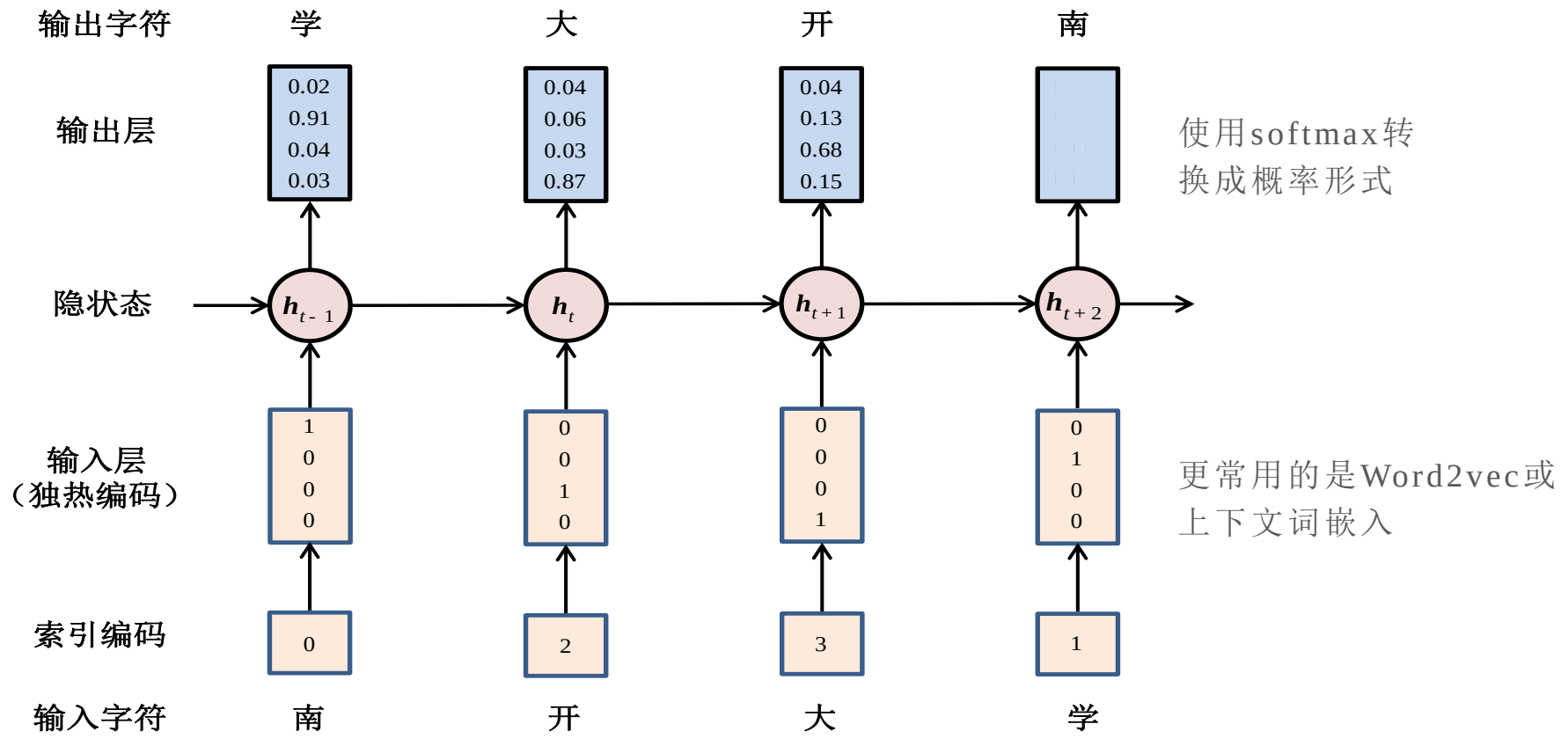
$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} w_{13} \\ w_{23} \\ w_{33} \\ w_{43} \end{bmatrix}$$

线性变换Wx相当于**列向量抽取**，学习嵌入矩阵就是在上下文语义中学习每个词的词嵌入向量

    - Transformer中的嵌入层：线性词嵌入，并将嵌入矩阵作为模型参数，参加模型的整体训练

6.1.2 输入输出编码

- 在NLP应用中，输入一般使用词嵌入，样本真实输出一般使用one-hot coding编码



6.1.3 损失函数

- 若样本真实输出使用one-hot coding，可以将网络输出通过softmax 转换为概率，表示时刻 t 输出每一个单词的概率

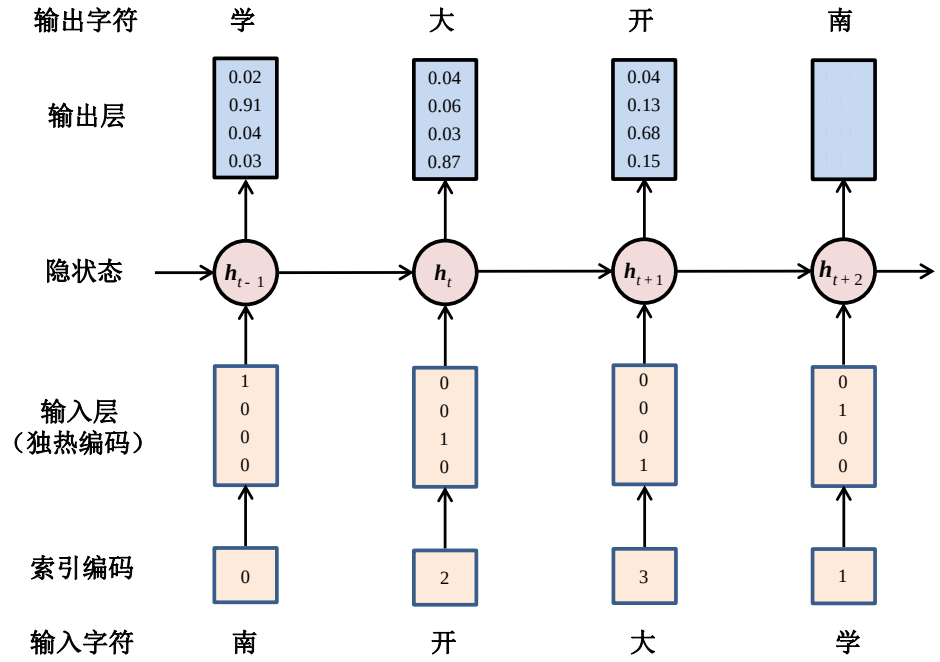
$$\hat{y}_t = \text{softmax}(y_t), \quad \text{where} \quad \hat{y}_{t,j} = \frac{\exp(y_{t,j})}{\sum_{i=1}^{d} \exp(y_{t,i})}, \quad j = 1, 2, \cdots, d$$

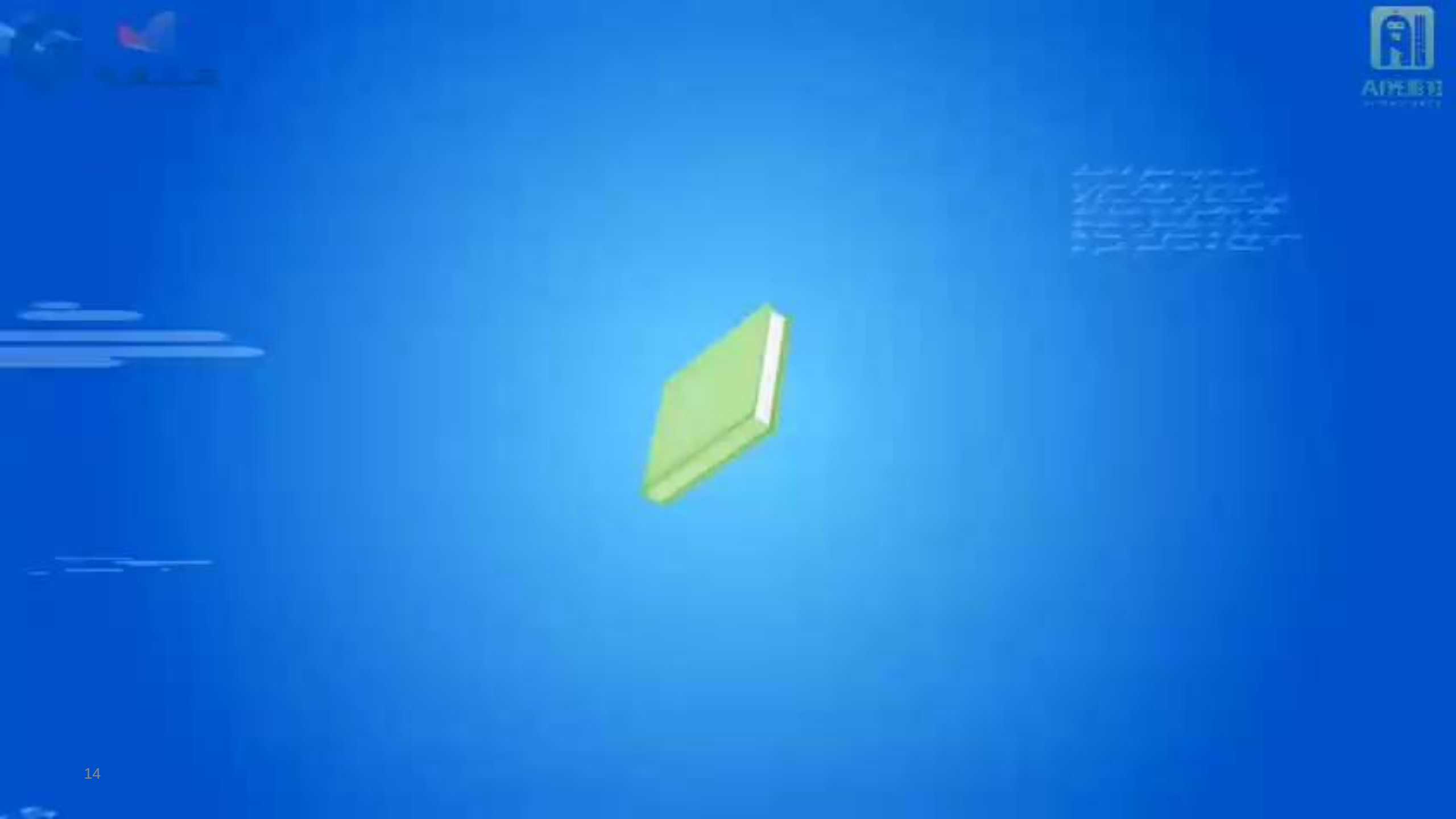
- 使用交叉熵损失函数度量预测输出和真实输出之间的差距：

$$\ell(o_t, \hat{y}_t) = -\sum_{j=1}^{d} o_{t,j} \log \hat{y}_{t,j}$$

- 训练过程的损失函数—所有时刻交叉熵损失的平均：

$$L = \frac{1}{T} \sum_{t=1}^{T} \ell(o_t, \hat{y}_t)$$

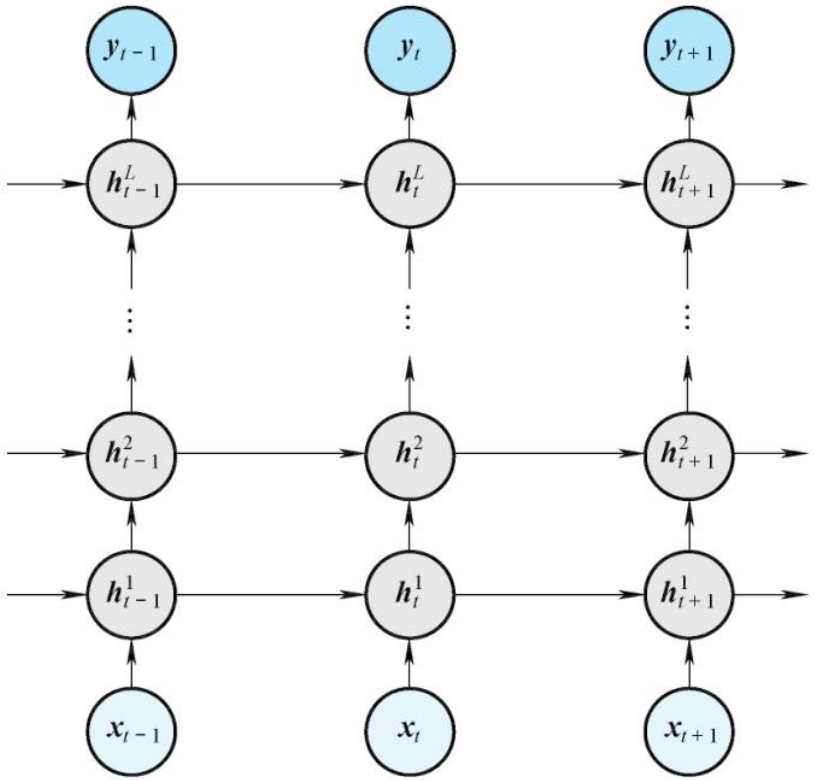




6.1.4 深度循环网络

- 单隐藏层循环网络：只有一个隐藏层
- 多隐藏层深度循环网络：多个隐藏层，每个隐状态都连续地传递到当前层的下一个时间步和下一层的当前时间步

$$
\begin{aligned}
h_t^1 &= \sigma\left(W^1 h_{t-1}^1 + U^1 x_t + b_h^1\right) \\
h_t^2 &= \sigma\left(W^2 h_{t-1}^2 + U^2 h_t^1 + b_h^2\right) \\
&\vdots \\
h_t^L &= \sigma\left(W^L h_{t-1}^L + U^L h_t^{L-1} + b_h^L\right) \\
y_t &= V h_t^L + b_y
\end{aligned}
$$



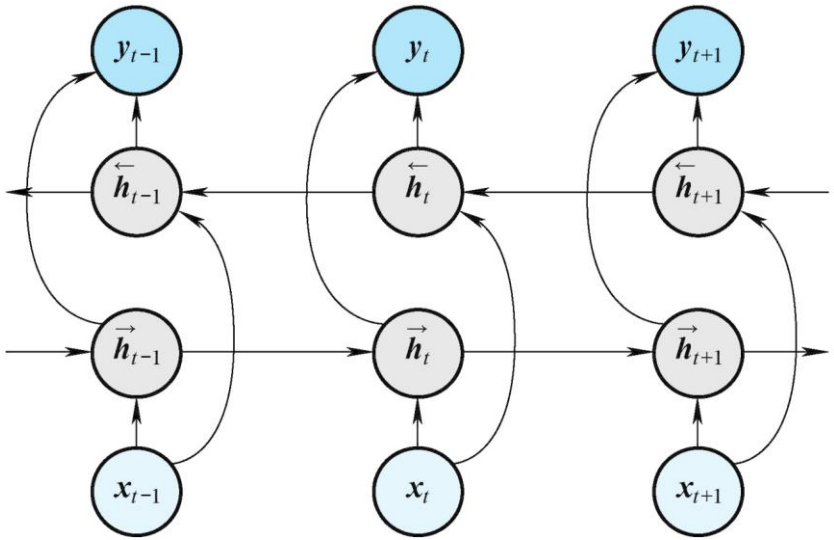
6.1.5 双向循环网络 好3解

- 单向循环网络：主要针对“过去”时间步的状态对“未来”时间步的状态有影响的任务
- 双向循环网络：主要处理同时需要上下文信息的任务
  - 通过从两个方向处理信息，双向循环网络能够获得更全面的上下文视角
  - 填充缺失单词

    我高考考了700分，我想上___学习。

    我高考考了700分，我想上___学习，我喜欢天津这座城市。

    我高考考了700分，我想上___学习，我喜欢天津这座城市，我想学文科。



- 优势
  - 双向网络处理每个词时已经看完了整个句子
  - 单向网络只看了前面部分的句子
- 形式化描述

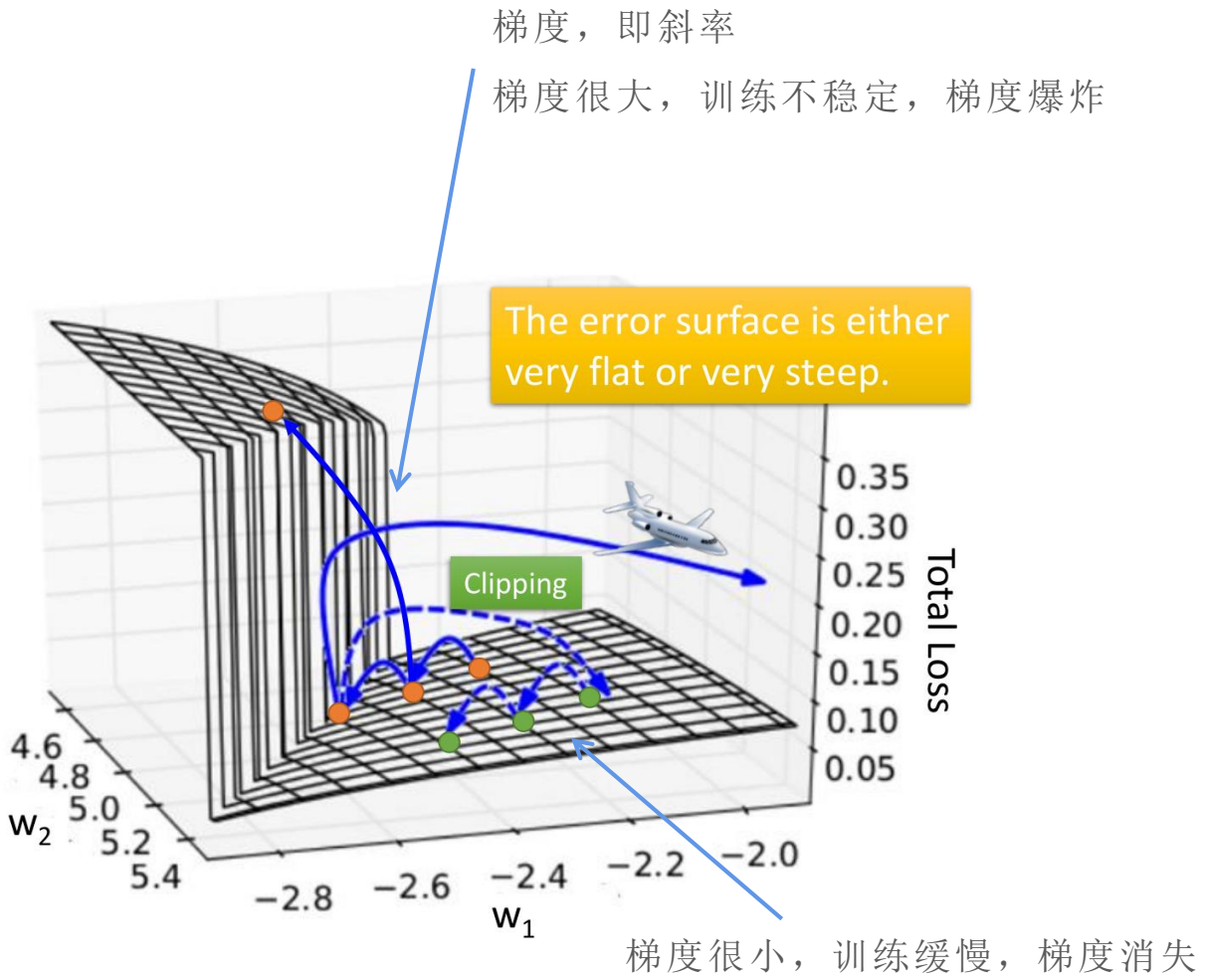
$$\overrightarrow{h}_t = \sigma\left(W^f \overrightarrow{h}_{t-1} + U^f x_t + b_h^f\right)$$

$$\overleftarrow{h}_t = \sigma\left(W^b \overleftarrow{h}_{t+1} + U^b x_t + b_h^b\right)$$

$$y_t = V^f \overrightarrow{h}_t + V^b \overleftarrow{h}_t + b_y$$

6.1.6 梯度消失和爆炸

- 在RNN中，由于权重共享，使用反向传播计算梯度时会连续乘以相同的权重矩阵
  - 如果权重矩阵的特征值小于1，则梯度会逐渐减小，直到几乎消失。
  - 如果权重矩阵的特征值大于1，则梯度会逐渐增大，直到发生爆炸
  - 当RNN的时间步较多或者序列较长时，反向传播过程中梯度会随着层数的增加而指数级地减小或增大，从而**增加模型学习长序列依赖关系的难度，这就是NLP中经典的长距离依赖问题**
  - 梯度爆炸解决方案：梯度截断（见第七讲）



6.1.6 梯度消失和爆炸

- 长距离依赖
  - 当相关信息距离比较近时，RNN容易学习到
    - 示例：“the clouds are in the sky”，最后一个词较容易学
  - 当相关信息距离比较远时，RNN很难学习到
    - 示例：“I grew up in France… I speak fluent French”，学习最后一个词需要依赖前面的France
  - 长距离依赖的难点在于梯度消失或爆炸，由于梯度爆炸或消失问题，RNN实际上只能学习到短周期的依赖关系

6.1.7 RNN PyTorch实现

- PyTorch在torch.nn.RNN函数中实现了对循环神经网络、深度循环网络、双向循环网络的封装
  - 是否使用深度循环网络通过参数num_layers设置
  - 是否使用双向循环网络通过参数bidirectional设置

API: torch.nn.RNN(input_size, hidden_size, num_layers=1, nonlinearity='tanh', bias=True, batch_first=False, dropout=0.0, bidirectional=False, device=None, dtype=None)

参数：input_size：输入数据的特征数量，即每个时间步输入向量的维度；

hidden_size: 隐藏层中神经元的数量，即隐状态的特征维度；

num_layers：隐藏层的层数，默认值为 1，可以设置为大于 1 的常数来实现深度循环网络；

nonlinearity：激活函数类型，可以选 relu 或 tanh，默认为 tanh。

bias：是否使用偏置 $\boldsymbol{b}_h$ 和 $\boldsymbol{b}_y$，默认为使用。

dropout：是否使用 dropout，默认为不使用；

bidirectional：是否使用双向循环网络，默认为不使用；

batch_first、device、dtype：使用默认设置即可，细节参考 PyTorch 文档；

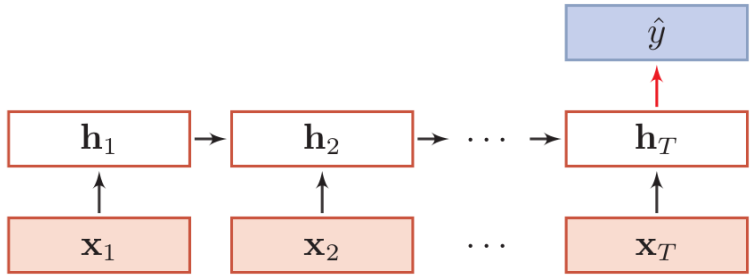
示例：

```
import torch.nn
from torch import nn
rnn = nn.RNN(10, 20, 2)          # 输入维度 10，隐状态维度 20，隐藏层数 2
input = torch.randn(5, 3, 10)    # 初始化输入数据，其形状为 (序列长度, 批量大小, 输入维度)
h0 = torch.randn(2, 3, 20)       # 初始化隐状态，其形状为 (隐藏层数, 批量大小, 隐藏层维度)
output, hn = rnn(input, h0)      # 输出 output 形状为（序列长度, 批量大小, 隐藏层维度）
                                 # 隐状态 hn 形状为 (隐藏层数, 批量大小, 隐藏层维度)
```

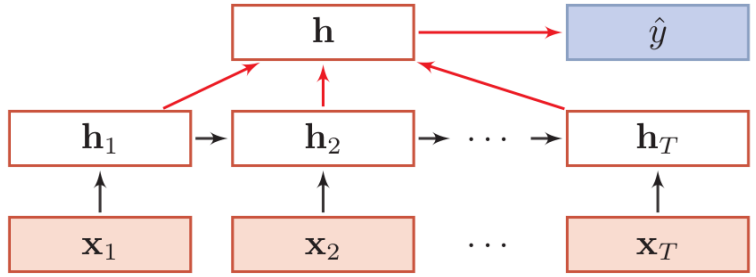
这里output的第三个维度是隐藏层维度，如果希望RNN的输出维度不等于隐藏层维度，需要手动添加一个全连接层进行维度变换（即 $y_t = Vh_t + b_y$，该API并没有计算 $y_t = Vh_t + b_y$，事实上，该API输出的output_t等于h_t）

6.1.7 RNN应用

- 序列到类别

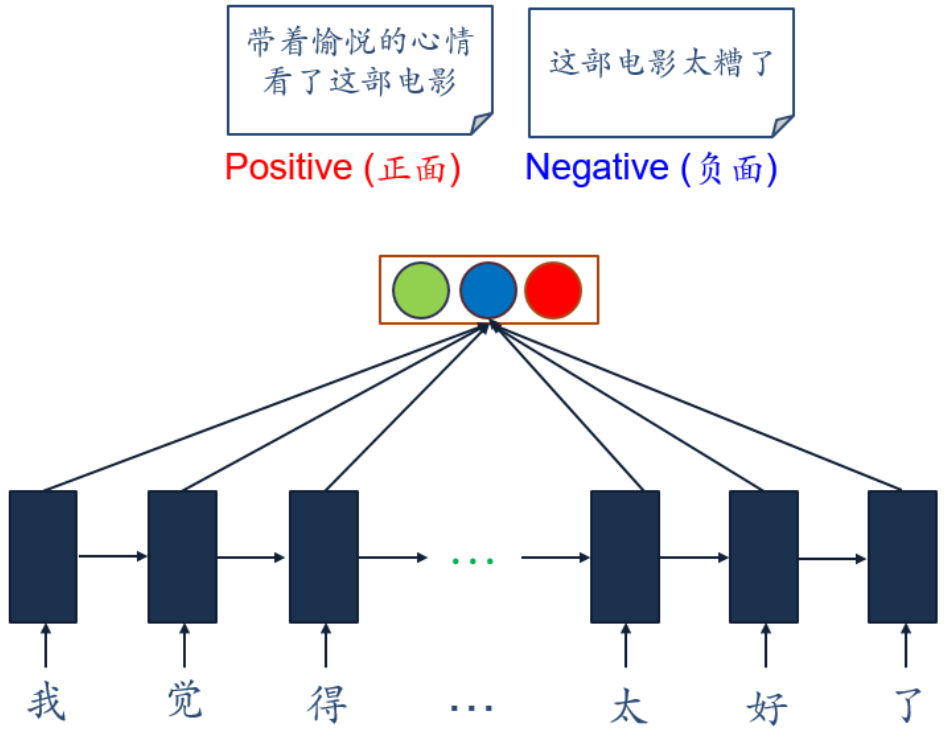


使用最后一个隐状态



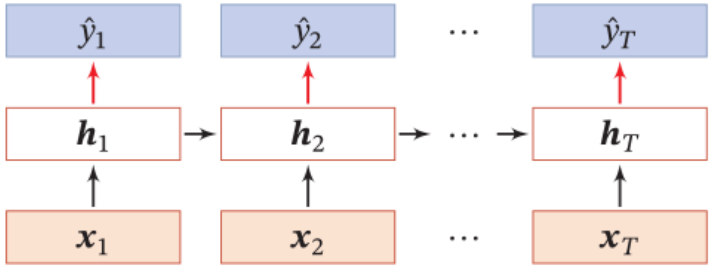
使用所有隐状态的平均

- 情感分类



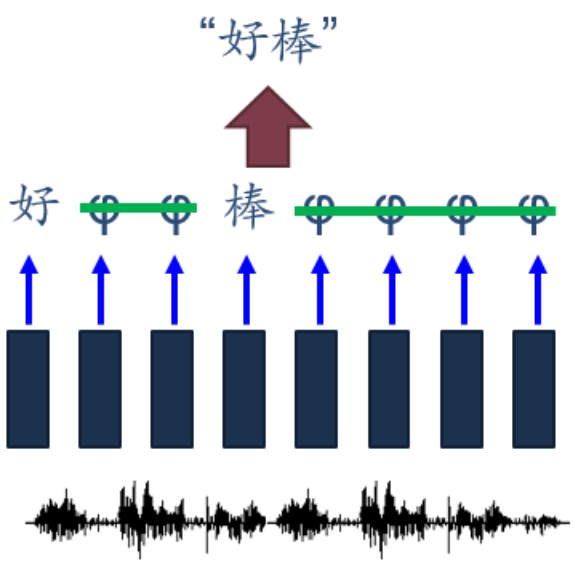
6.1.7 RNN应用

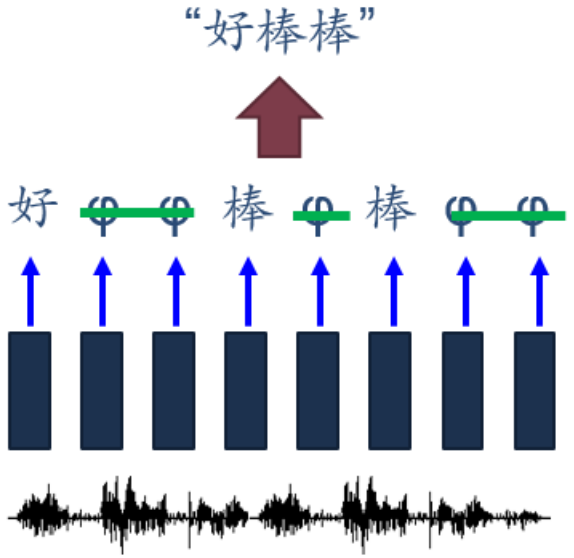
- 同步的序列到序列



对于输入序列中的每一个元素，都必须立即产生一个对应的输出元素

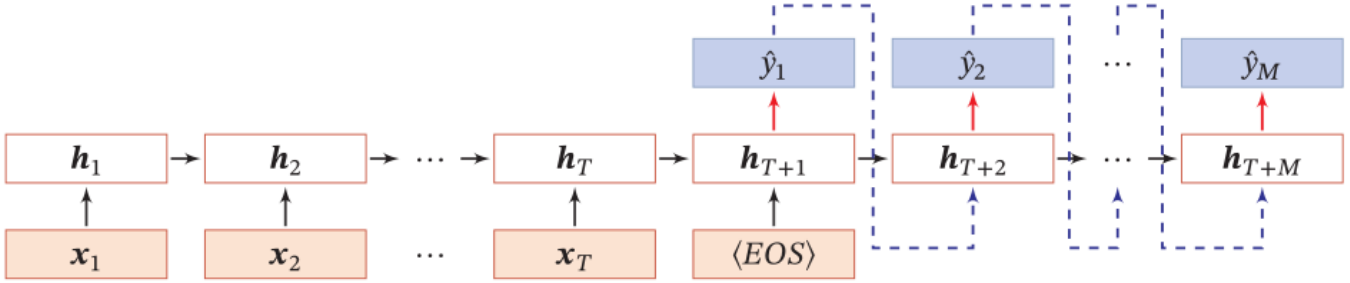
- 实时语音识别





6.1.7 RNN应用

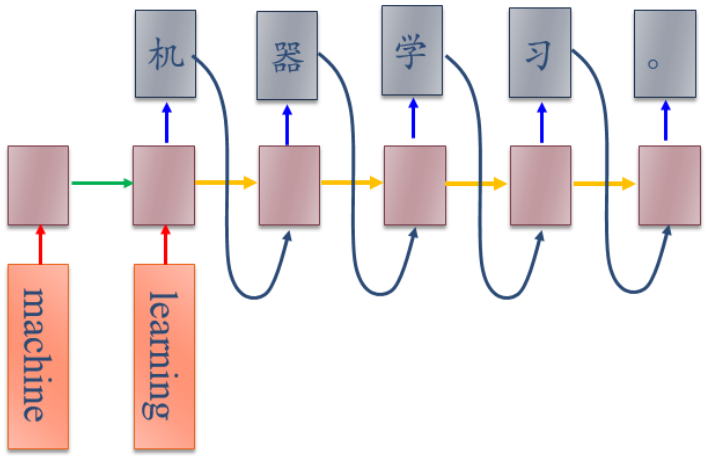
- 异步的序列到序列



模型需要先读取并理解整个输入序列，然后再开始生成输出序列

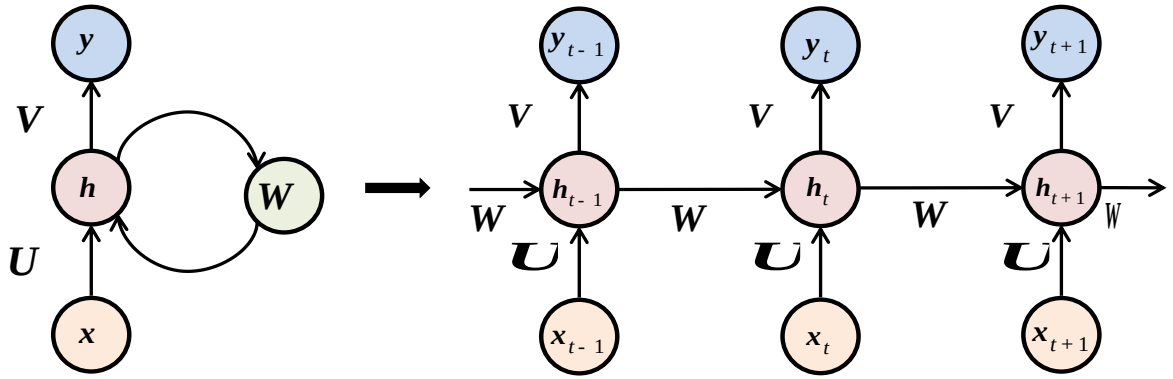
输入和输出之间没有严格的时间步对应关系

- 机器翻译



6.1.8 RNN小结

- 优点
  - 引入了记忆存储单元（隐状态）
- 缺点
  - 长距离依赖问题
  - 记忆容量有限
    - 每个时刻的隐状态是一个长度固定的向量，难以概括前面较长序列的信息
  - 无法并行计算
    - 当前时刻的计算依赖于前一时刻的输出，依次序进行，不同时刻无法并行计算



6.2 门控循环网络

- 循环神经网络当前时刻 t 的输出依赖于当前时刻 t 的隐状态 h_t，h_t 存储了到时刻 t 的历史序列信息
  - 将任意长度的输入 $(x_1, x_2, \cdots, x_t)$ 映射到固定长度的隐状态 h_t 是一个有损映射
    - 水池大小是固定的，注入太多水总会溢出来
  - 希望隐状态能够**有选择性地保留历史序列的重要信息**，尤其是**早期信息**，充分利用有限的隐状态信息空间
- 门控循环单元（Gated Recurrent Unit，GRU）：支持隐状态控制
  - 通过引入重置门、更新门等概念，能够更好地控制隐状态的更新，从而捕捉时间序列中时间步距离较大的依赖关系，解决传统循环神经网络难以捕捉序列长期依赖关系的问题
  - 模型有专门的机制确定应该**何时更新隐状态**，以及应该**何时重置隐状态**，并且这些机制是可学习的
  - 示例：如果第一个词非常重要，模型将学会在第一次观测之后不更新隐状态，跳过不重要的词

6.2.1 重置门

- 重置门：用于计算候选隐藏状态，决定了在结合当前输入和历史信息（存储在隐状态 h_{t-1}）时**“需要使用多少历史信息”**
  - t 时刻的重置门设置为 (0,1) 区间内的向量 r_t
  - 使用重置门之后的候选隐状态更新：

$$\widetilde{h}_t = \tanh\left(U_h x_t + W_h(r_t \cdot h_{t-1}) + b_h\right)$$

其中点乘定义为 $x \cdot y = \begin{bmatrix} x_1 y_1 \\ \vdots \\ x_q y_q \end{bmatrix}$

  - 重置门物理意义：r_t 中的元素接近1时，保留更多的历史信息

    r_t 中的元素接近0时，遗忘更多的历史信息，此时，当前输入的占比更大，从而有助于捕获序列中的**短期依赖关系**

  - PyTorch中实现的候选隐状态更新方式与上述介绍略有不同

$$\widetilde{h}_t = \tanh\left(U_h x_t + r_t \cdot (W_h h_{t-1}) + b_h\right)$$

6.2.2 更新门

去年考了GRU

- 更新门：用于计算当前隐状态，决定了**在计算h_t时的候选隐状态$\widetilde{h}_t$和前一时刻隐状态h_{t-1}的占比多少**
  - t 时刻的更新门设置为 $(0,1)$ 区间内的向量z_t
  - 使用更新门之后的隐状态更新：

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

  - 更新门物理意义：z_t 中的元素接近1时，更多地使用前一时刻隐状态h_{t-1}，表示更多地记忆历史信息，有助于捕获序列中的**长期依赖关系**

    z_t 中的元素接近0时，更少地使用前一时刻隐状态，更多地使用候选隐状态

6.2.3 学习重置门和更新门

- 使用神经网络中“线性映射 + 非线性变化”模式学习重置门和更新门

$$r_t = \text{sigmoid}\left(U_r x_t + W_r h_{t-1} + b_r\right)$$
$$z_t = \text{sigmoid}\left(U_z x_t + W_z h_{t-1} + b_z\right)$$

  - 权重U, W和偏置b是可训练学习的
  - 使用sigmoid函数将输入值转换到 (0,1) 内

6.2.4 形式化表示汇总

- 门控循环网络形式化表示

去年考点

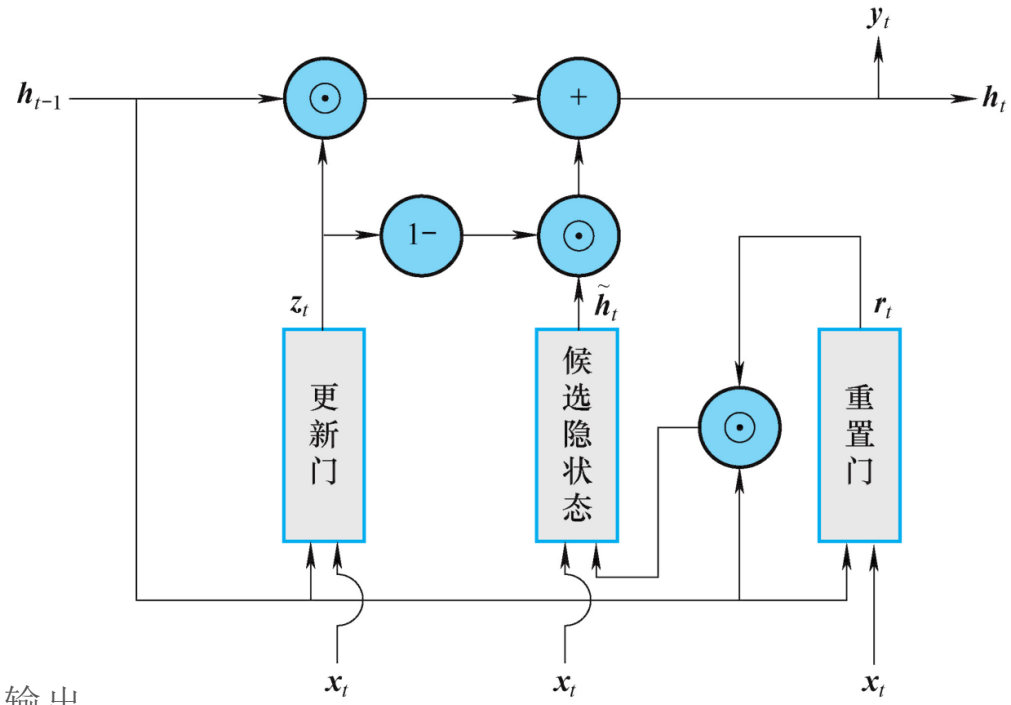
$$r_t = \text{sigmoid}\left(U_r x_t + W_r h_{t-1} + b_r\right) \quad \leftarrow \text{重置门}$$

$$z_t = \text{sigmoid}\left(U_z x_t + W_z h_{t-1} + b_z\right) \quad \leftarrow \text{更新门}$$

$$\widetilde{h}_t = \tanh\left(U_h x_t + W_h(r_t \cdot h_{t-1}) + b_h\right) \quad \leftarrow \text{候选隐状态}$$

$$h_t = z_t \cdot h_{t-1} + (1 - z_t) \cdot \widetilde{h}_t \quad \leftarrow \text{更新隐状态}$$

$$y_t = Vh_t + b_y \quad \leftarrow \text{产生时刻 t 的输出}$$



- 当 $r_t \to 0, z_t \to 0$ 时，时刻 t 的隐状态 h_t 倾向于只接收当前输入 x_t，忽略历史信息，**有助于捕获序列中的短期依赖关系**
- 当 $r_t \to 1, z_t \to 0$ 时，时刻 t 的隐状态 h_t 倾向于同时考虑前一时刻隐状态 h_{t-1} 和当前输入 x_t
- 当 $z_t \to 1$ 时，时刻 t 的隐状态 h_t 倾向于只接收前一时刻隐状态 h_{t-1}，忽略当前输入 x_t，**有助于捕获序列中的长期依赖关系**

6.2.5 减缓梯度消失或爆炸

- 直观解释1
  - 循环神经网络产生梯度消失或爆炸的原因是隐状态间的依赖关系，导致反向传播时产生一个递归关系式
  - 门控循环网络通过重置门和更新门可以**截断**隐状态的递归关系，从而减缓梯度消失或爆炸
    - 当$r_t \to 0, z_t \to 0$ 时，时刻 t 的隐状态h_t倾向于只接收当前输入x_t，与 h_{t-1}的递归被截断
- 直观解释2
  - 循环神经网络采用的是$h_t = f(x_t, h_{t-1})$的前向传播形式
  - 门控循环神经网络采用的是$h_t = z_t \cdot h_{t-1} + (1 - z_t) \cdot f(x_t, h_{t-1})$的前向传播形式，**引入类似ResNet的旁支路径**，从而使用与ResNet类似的机制减缓梯度消失问题
    - ResNet: $h_t = \sigma(W_t h_{t-1} + b_t) + h_{t-1}$

6.2.6 PyTorch实现

- PyTorch在torch.nn.GRU函数中实现了对GRU的封装

API: torch.nn.GRU(input_size, hidden_size, num_layers=1, bias=True, batch_first=False, dropout=0.0, bidirectional=False, device=None, dtype=None)

参数：input_size：输入数据的特征数量，即每个时间步输入向量的维度；
hidden_size: 隐藏层中神经元的数量，即隐状态的特征维度；
num_layers：隐藏层的层数，默认值为 1，可以设置为大于 1 的常数来实现深度 GRU；
bias：是否使用偏置，默认为使用。
dropout：是否使用 dropout，默认为不使用；
bidirectional：是否使用双向 GRU，默认为不使用；
batch_first、device、dtype：使用默认设置即可，细节参考 PyTorch 文档；

示例：
```
import torch.nn
from torch import nn
rnn = nn.GRU(10, 20, 2)          # 输入维度 10，隐状态维度 20，隐藏层数 2
input = torch.randn(5, 3, 10)    # 初始化输入数据，其形状为 (序列长度, 批量大小, 输入维度)
h0 = torch.randn(2, 3, 20)       # 初始化隐状态，其形状为 (隐藏层数, 批量大小, 隐藏层维度)
output, hn = rnn(input, h0)      # 输出 output 形状为（序列长度, 批量大小, 隐藏层维度）
                                 # 隐状态 hn 形状为 (隐藏层数, 批量大小, 隐藏层维度)
```

6.3 长短期记忆网络

- 长短期记忆网络（long short-term memory, LSTM）采用更复杂的门控形式
  - 引入遗忘门f_t、输入门i_t、输出门o_t

$$
\begin{aligned}
i_t &= \text{sigmoid}\left(U_i x_t + W_i h_{t-1} + b_i\right) \\
f_t &= \text{sigmoid}\left(U_f x_t + W_f h_{t-1} + b_f\right) \\
o_t &= \text{sigmoid}\left(U_o x_t + W_o h_{t-1} + b_o\right)
\end{aligned}
$$

  - 引入记忆单元c_t及新记忆单元 $\widetilde{c}_t$

$$
\begin{aligned}
\widetilde{c}_t &= \tanh\left(U_c x_t + W_c h_{t-1} + b_c\right) \\
c_t &= f_t \cdot c_{t-1} + i_t \cdot \widetilde{c}_t
\end{aligned}
$$

  - 隐状态由记忆单元和输出门决定

$$
h_t = o_t \cdot \tanh(c_t)
$$

  - 输出由隐状态决定

$$
y_t = V h_t + b_y
$$

- 记忆信息c_t同时考虑了遗忘门与上一时间步记忆信息的交互，以及输入门和当前时间步新记忆信息的交互
  - 在第一项中，遗忘门控制了过去信息被遗忘的程度
  - 第二项中，输入门控制了当前时间步新记忆信息有多少被后续时间步利用。
- 输出门控制了记忆信息有多少被写入隐状态

6.3 长短期记忆网络

- LSTM中的计算示意图

i_t 决定多少当前信息被使用

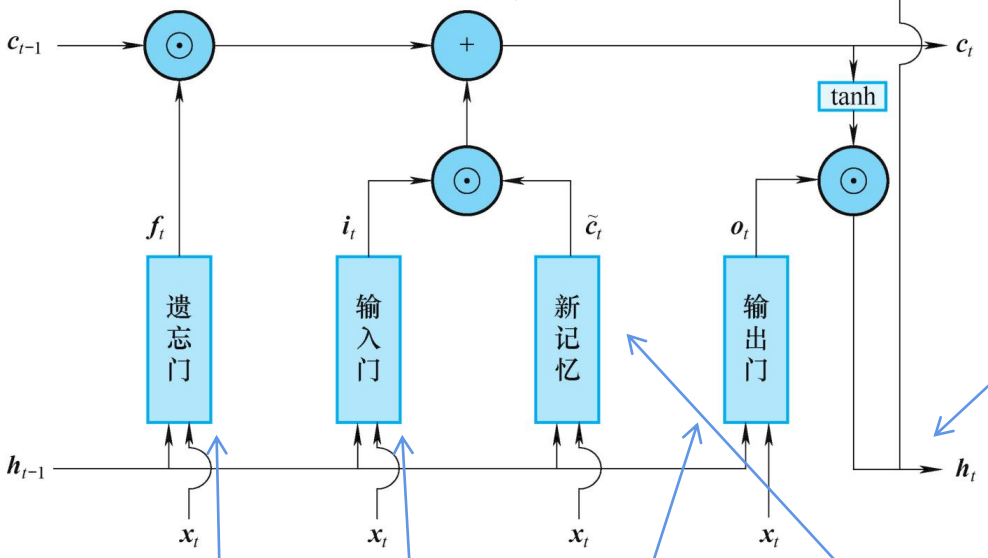
f_t 决定前一时刻的信息有多少需要被遗忘，多少需要被保留

$$c_t = f_t \cdot c_{t-1} + i_t \cdot \tilde{c}_t$$

$$y_t = Vh_t + b_y$$

信息高速公路：遗忘门的引入使得信息更快速地前向传播，以及误差更方便地反向传播（类似于ResNet中的旁支）

门控：对信息进行筛查，有选择性地让哪些信息通过



$$h_t = o_t \cdot \tanh(c_t)$$

o_t 控制记忆单元对隐藏层状态的影响程度

$$i_t = \text{sigmoid}\left(U_i x_t + W_i h_{t-1} + b_i\right)$$
$$f_t = \text{sigmoid}\left(U_f x_t + W_f h_{t-1} + b_f\right)$$
$$o_t = \text{sigmoid}\left(U_o x_t + W_o h_{t-1} + b_o\right)$$

$$\tilde{c}_t = \tanh\left(U_c x_t + W_c h_{t-1} + b_c\right)$$

6.3 长短期记忆网络

- 同样可以减缓梯度消失和梯度爆炸问题，从而可以处理更大长度的输入序列，解决**长距离依赖问题**
  - 长距离依赖的难点在于梯度消失或爆炸
  - LSTM可以**减缓梯度消失或爆炸**，从而**解决长距离依赖问题**

6.3 长短期记忆网络

- PyTorch在torch.nn.LSTM函数中实现了对LSTM的封装

API: torch.nn.LSTM(input_size, hidden_size, num_layers=1, bias=True, batch_first=False, dropout=0.0, bidirectional=False, proj_size=0, device=None, dtype=None)

参数：input_size：输入数据的特征数量，即每个时间步输入向量的维度；
hidden_size: 隐藏层中神经元的数量，即隐状态的特征维度；
num_layers：隐藏层的层数，默认值为 1，可以设置为大于 1 的常数来实现深度 LSTM；
bias：是否使用偏置，默认为使用。
dropout：是否使用 dropout，默认为不使用；
bidirectional：是否使用双向 LSTM，默认为不使用；
batch_first、proj_size 、device、dtype：使用默认设置即可，细节参考 PyTorch 文档；

示例：
```
import torch.nn
from torch import nn
rnn = nn.LSTM(10, 20, 2)          # 输入维度 10，隐状态维度 20，隐藏层数 2
input = torch.randn(5, 3, 10)     # 初始化输入数据，其形状为 (序列长度, 批量大小, 输入维度)
h0 = torch.randn(2, 3, 20)        # 初始化隐状态，其形状为 (隐藏层数, 批量大小, 隐藏层维度)
c0 = torch.randn(2, 3, 20)        # 初始化记忆单元，其形状为 (隐藏层数, 批量大小, 隐藏层维度)
output, hn, cn = rnn(input, h0)   # 输出 output 形状为（序列长度, 批量大小, 隐藏层维度）
                                  # 隐状态 hn 形状为 (隐藏层数, 批量大小, 隐藏层维度)
                                  # 记忆单元 cn 形状为 (隐藏层数, 批量大小, 隐藏层维度)
```

6.4 编码器—解码器框架

- 同步序列到序列问题：输入和输出等长，每个时刻一个输入、一个输出
- 异步序列到序列问题：将输入序列映射到不一定等长的输出序列
  - 机器翻译
  - 通常使用编码器—解码器框架

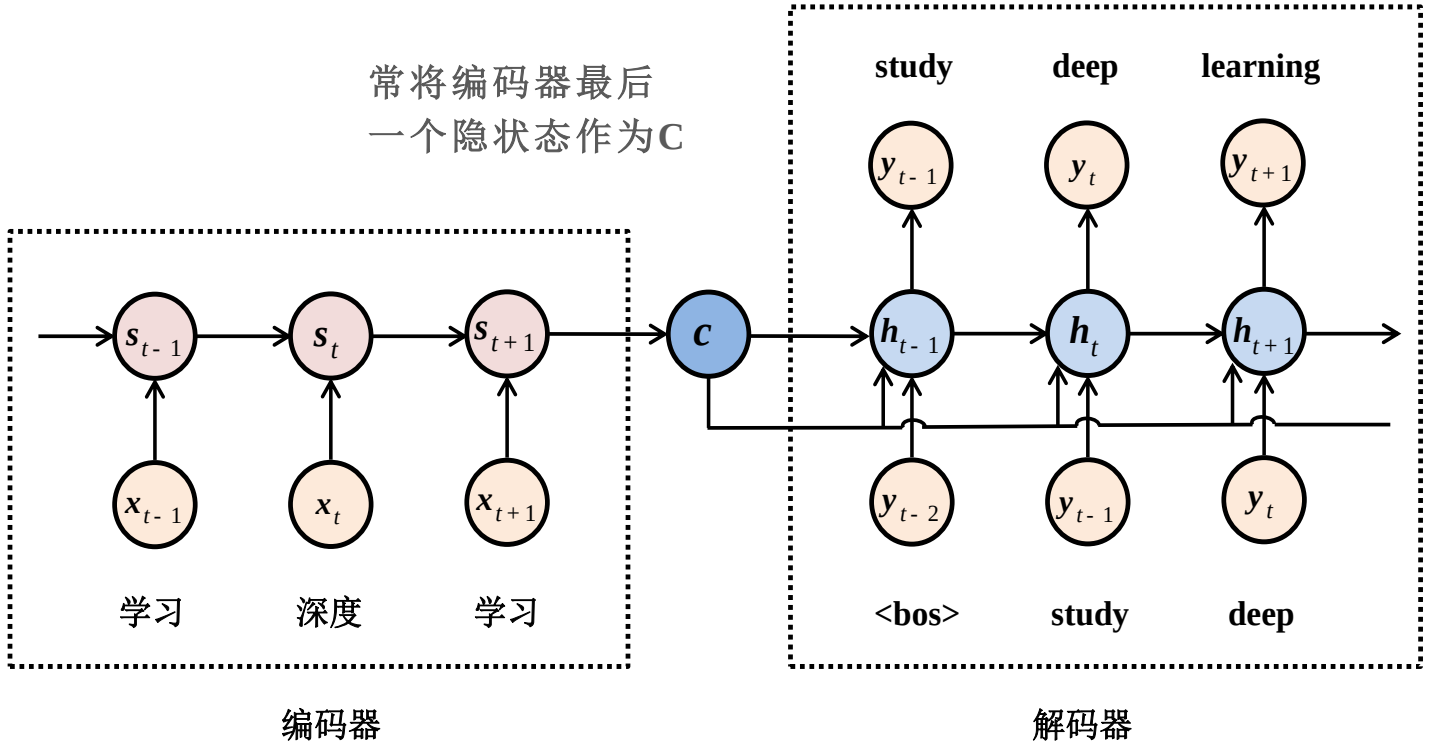
6.4 编码器—解码器框架 ☆ 计算

- 编码器：接受一个长度可变的序列作为输入，并对输入序列进行编码和特征提取，形成特征向量，记为 C，常被称为上下文向量（ context ）
  - 理解输入序列并将结果保存在载体C中，C可视为输入序列的“浓缩”表示
- 解码器：对形成的特征向量进行解码，将C映射到长度可变的序列
  - 从C中提取合适信息，产生预期目标
- 示例：给定英文句子“I went to the game yesterday”，该句子被编码器编码为一个有固定长度的语义向量 C，C 被假定包含了这个句子的全部语义信息；然后，解码器利用 C 来生成中文目标句子“我昨天去参加比赛”中的每一个词

6.4 编码器—解码器框架

早期的编码器和解码器
使用RNN/GRU/LSTM
现在使用Transformer

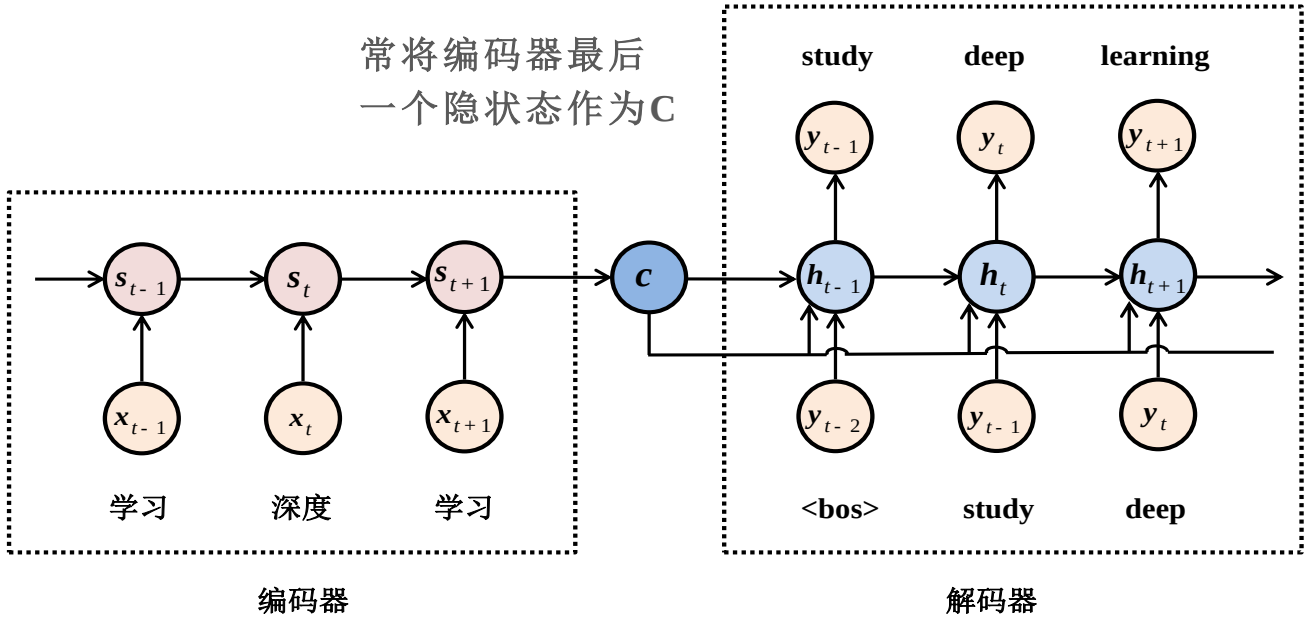
常将编码器最后
一个隐状态作为C



C作为解码器隐藏层0时刻的初始输入
以及解码器隐状态转换的输入$h_t = f(h_{t-1}, y_{t-1}, C)$

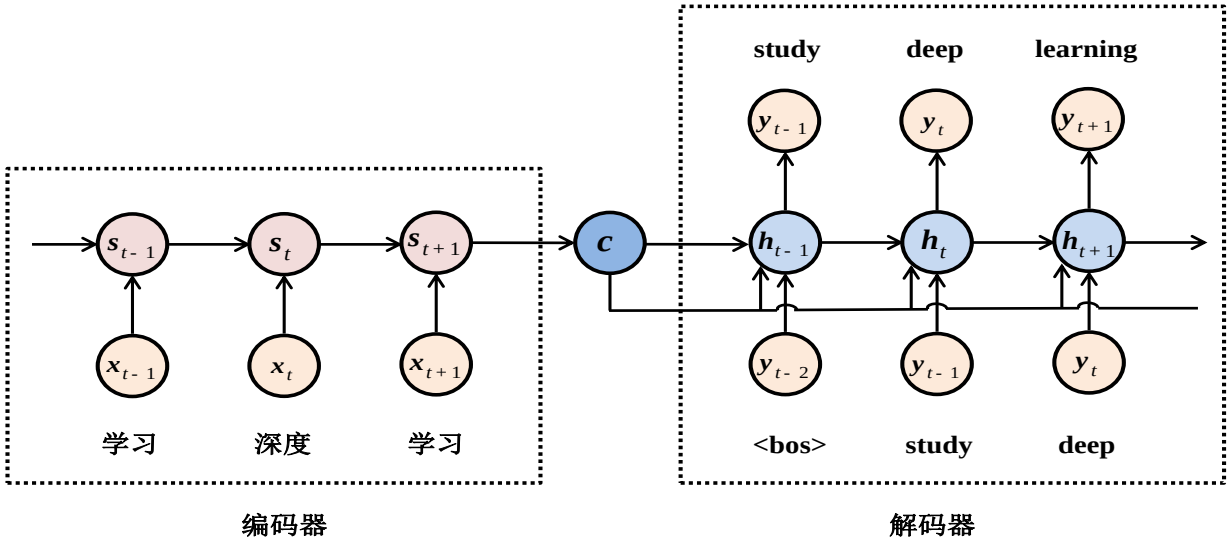
6.4 编码器—解码器框架

- 编码器—解码器框架的不足
  - 编码器RNN输出的 C 的维度太小而难以适当地概括一个长序列
    - 常将RNN最后一层隐状态输出作为C
    - 源句子中越在前面的单词，对形成 C 的影响越小；如果源句子过长，那么源句子前面单词的语义信息几乎没有被编码到 C 中，因而解码器就很难使用前面单词的语义



6.4 编码器—解码器框架

- 编码器—解码器框架的不足
  - 并非所有输入词元都对解码某个词元有用，但在每个解码步骤中仍使用相同编码的上下文特征C
    - 当解码器生成目标句子中的每一个词时，都使用同一个C；这意味着不管生成哪个词，都按照同一信息量使用源句子中的每个词，这显然不合理



- 解决方案：
  - 注意力机制：解码不同词元时，把注意力集中在更相关的输入词元上

总结

- 介绍了基本的循环神经网络，包括深度循环网络和双向循环网络
- 介绍了门控循环单元、LSTM
- 介绍了编码器—解码器框架