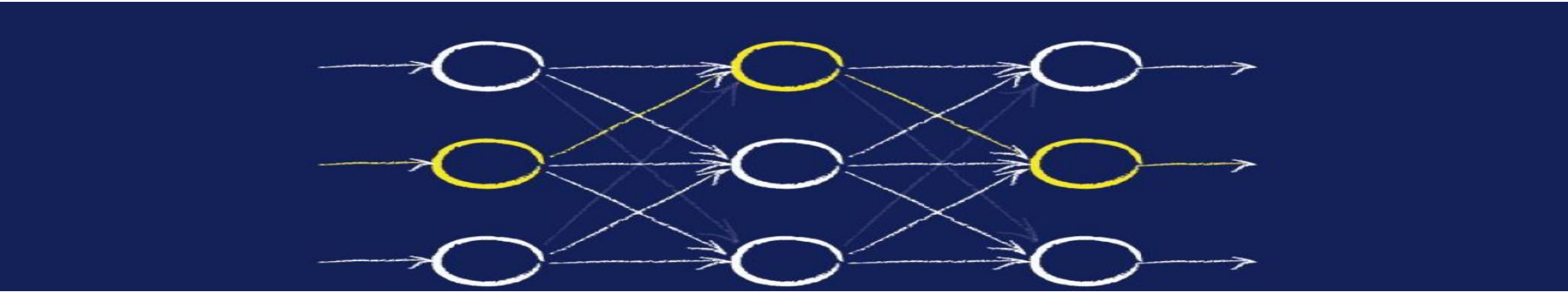
循环神经网络

（Recurrent neural network, RNN）



前馈网络不足

- 连接存在层与层之间，每层的节点之间是无连接的。
- 输入和输出的维数都是固定的，不能任意改变。无法处理变长的序列数据。
- 假设每次输入都是独立的，也就是说每次网络的输出只依赖于当前的输入。

循环神经网络

- 循环神经网络
  - 循环神经网络通过使用带自反馈的神经元，能够处理任意长度的序列。
  - 循环神经网络比前馈神经网络更加符合生物神经网络的结构。
  - 循环神经网络已经被广泛应用在语音识别、语言模型以及自然语言生成等任务上。

循环神经网络

循环神经网络（recurrent neural network, RNN）是一类用于处理序列数据 $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}$ 的神经网络。

例如，考虑这两句话：“I went to Nepal in 2009” 和 “In 2009, I went to Nepal.” 如果我们让一个机器学习模型读取这两个句子，并提取 叙述者去Nepal的年份，无论 “2009 年” 是作为句子的第六个单词还是第二个单词出 现，我们都希望模型能认出 “2009 年” 作为相关资料片段。

1. 问题介绍

前面介绍过多层全连接网络和卷积神经网络，可以发现这些网络不需要记忆的特性也能处理对应的任务，下面举一个具体的例子引出循环神经网络。

首先我们来看看下面这两句话，如图5.1与图5.2所示。

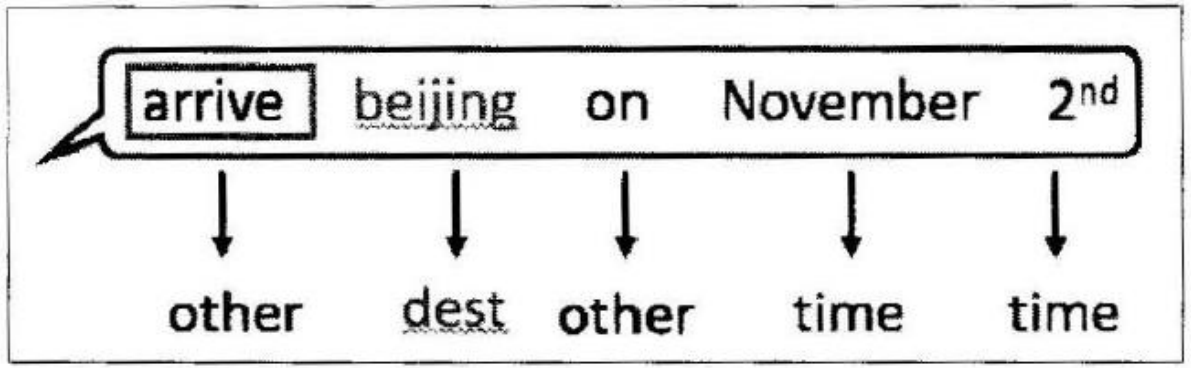


图 5.1　到达北京

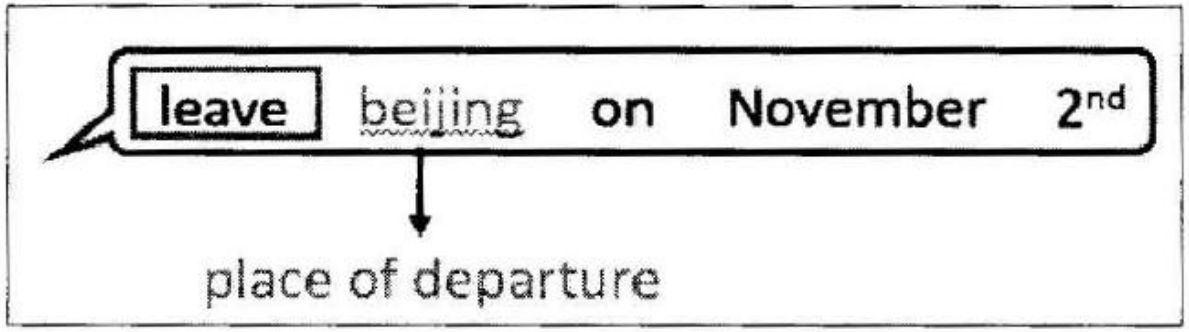
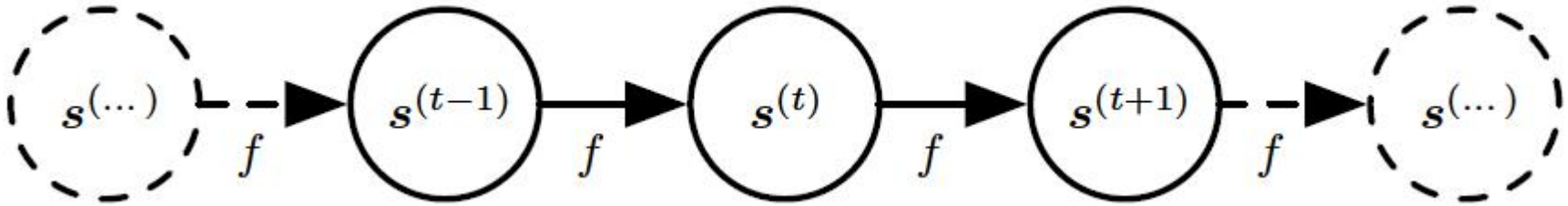


图 5.2　离开北京

如果只输入“beijing”
而没有记忆特性，
网络会输出相同结果；
如果能记忆前面的词，
结果就会不一样。

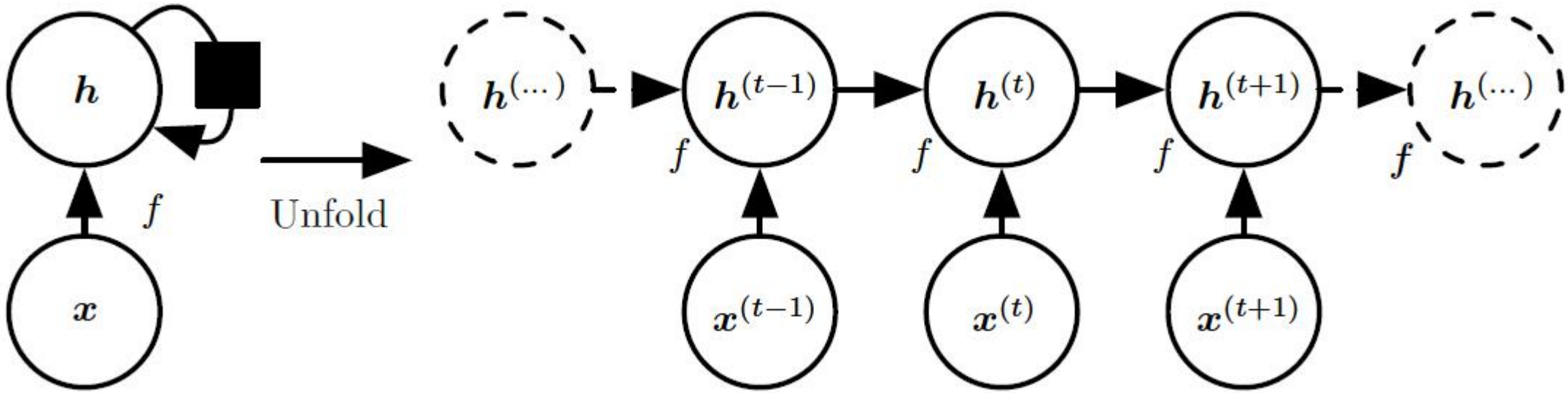
2. 基本结构



$$s^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta),$$

当前状态包含了整个过去序列的信息。

将网络的输出保存在一个记忆单元中，这个记忆单元和下一次的输入一起进入神经网络。
网络输入时会联合记忆单元一起作为输入，网络不仅输出结果，还会保存到记忆单元。



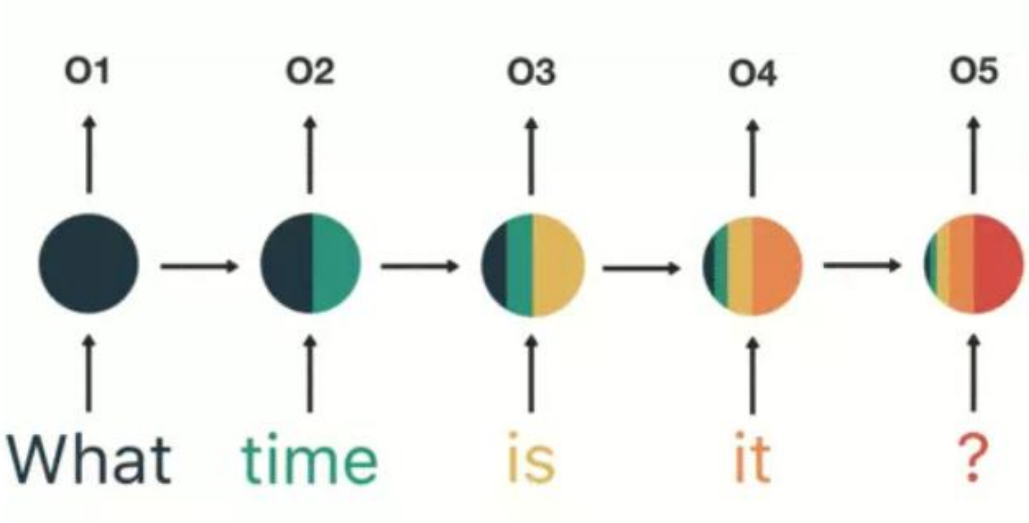
$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta})$$

映射任意长度的序列 $(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$ 到一固定长度的向量 $\boldsymbol{h}^{(t)}$。

可以用一个函数 $g^{(t)}$ 代表经 t 步展开后的循环：

$$\begin{aligned}\boldsymbol{h}^{(t)} &= g^{(t)}(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)}) \\ &= f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)}; \boldsymbol{\theta}).\end{aligned}$$

函数 $g^{(t)}$ 将全部的过去序列 $(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, \ldots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$ 作为输入来生成当前状态，但是展开的循环架构允许我们将 $g^{(t)}$ 分解为函数 f 的重复应用。



输入序列的顺序会改变输出结果

无论序列有多长，都能不断输入网络，最终得到结果。

将序列中的每个数据点依次传入网络即可，如图5.4所示。

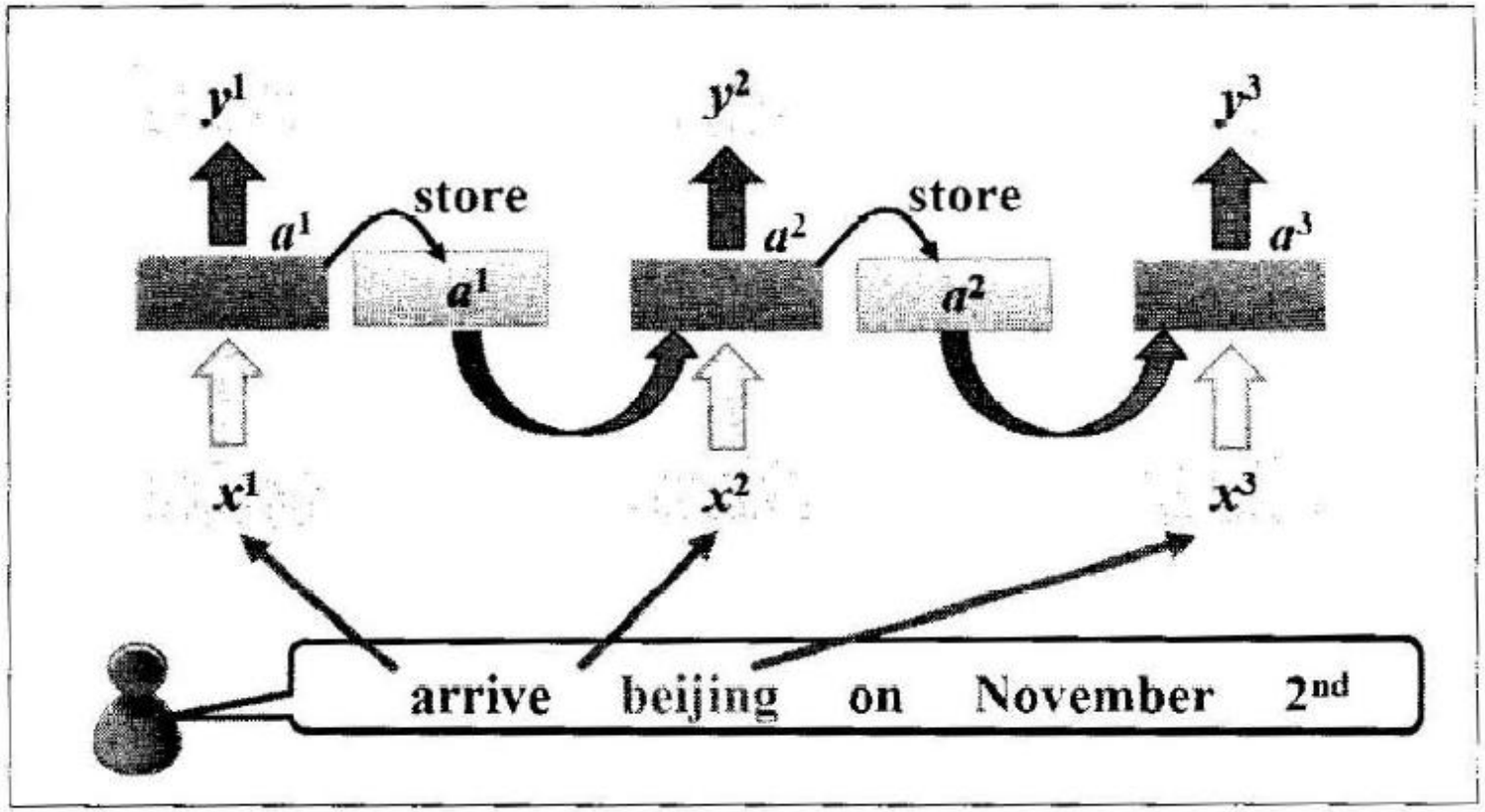
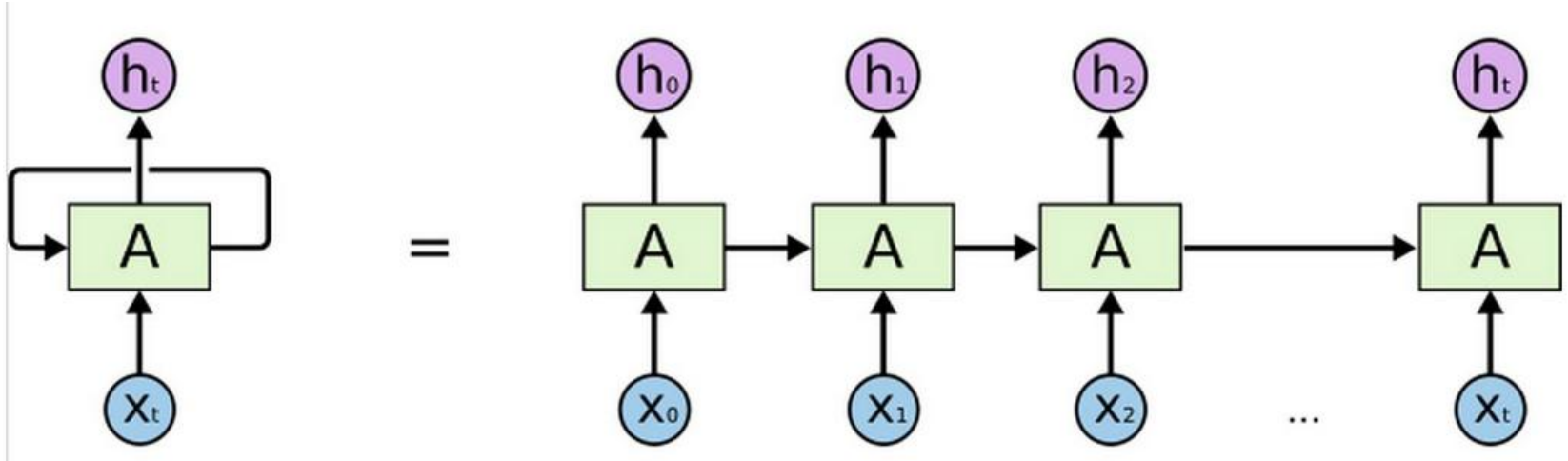
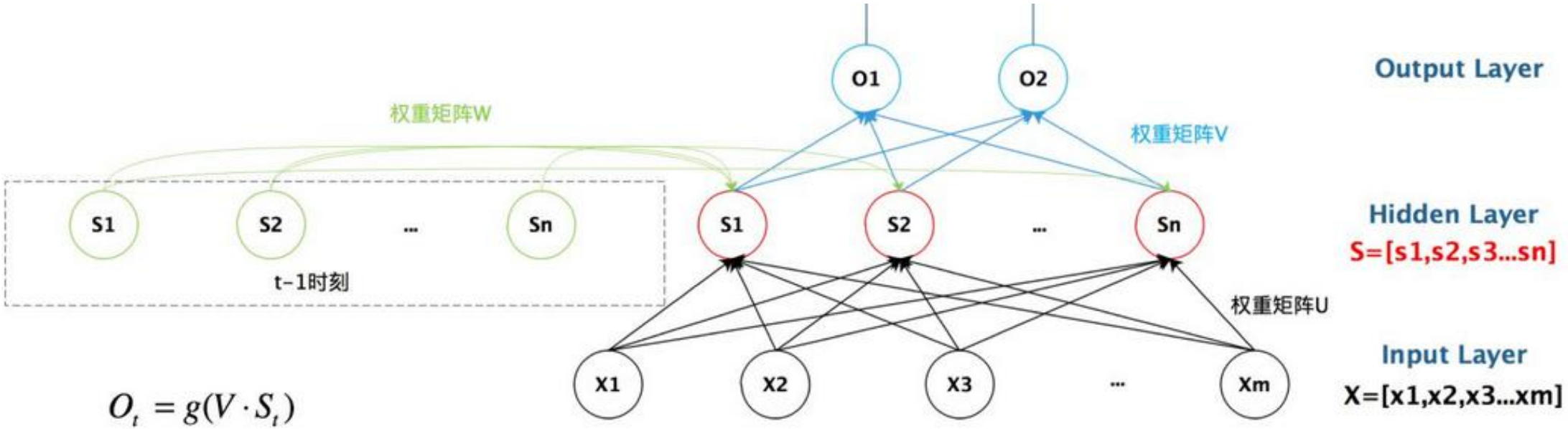


图 5.4　将整个序列传入网络

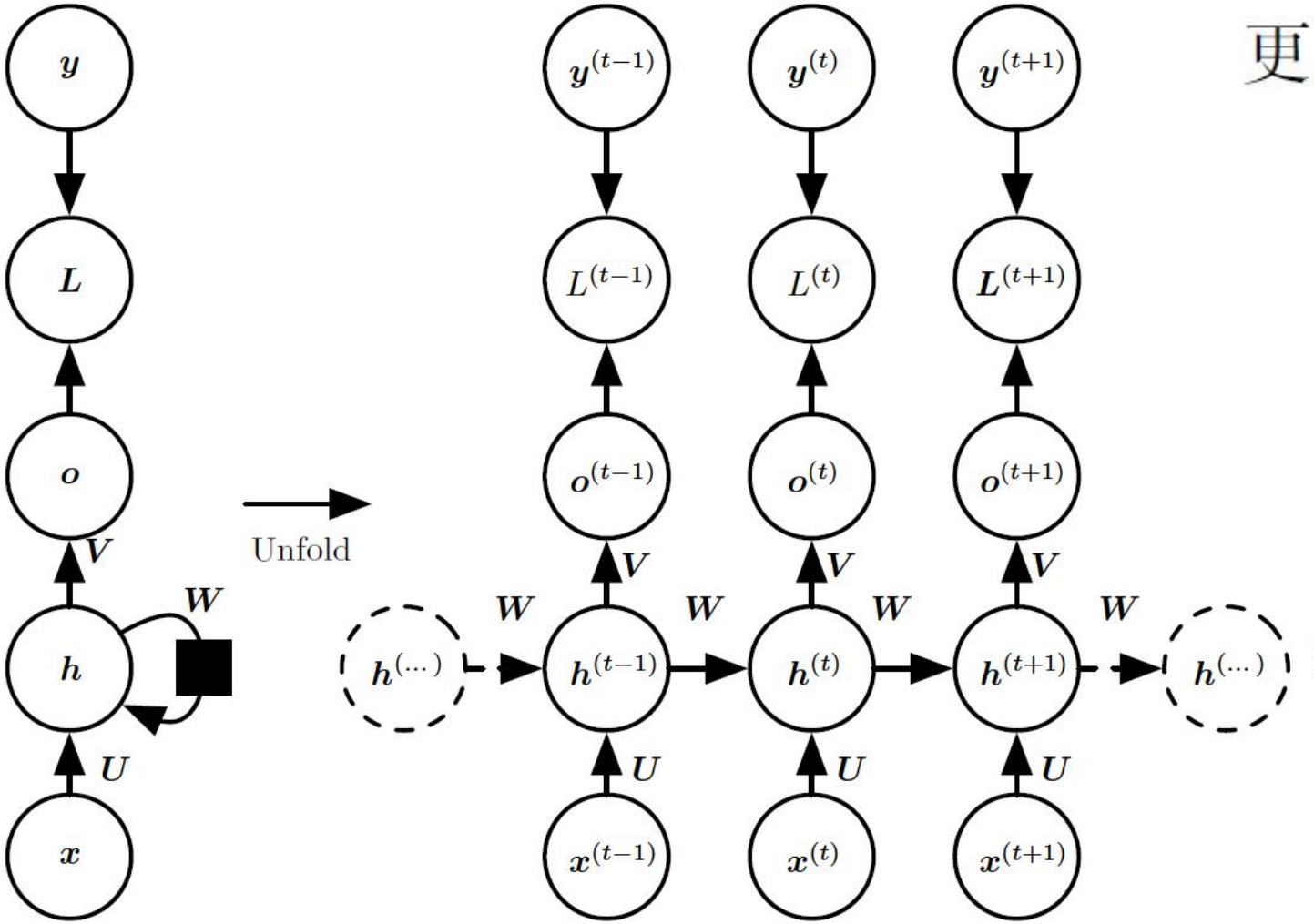
2. 基本结构





$$O_t = g(V \cdot S_t)$$

$$S_t = f(U \cdot X_t + W \cdot S_{t-1})$$



更新方程：

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)},$$

$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)}),$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)},$$

$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}(\boldsymbol{o}^{(t)}),$$

给定的 $\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}$ 后 $\boldsymbol{y}^{(t)}$ 的负对数似然

$$L\big(\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\}, \{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\}\big)$$
$$= \sum_t L^{(t)}$$
$$= -\sum_t \log p_{\mathrm{model}}\big(y^{(t)} \mid \{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\}\big)$$

计算循环网络(将 $\boldsymbol{x}$ 值的输入序列映射到输出值 $\boldsymbol{o}$ 的对应序列) 训练损失的计算图。损失 L 衡量每个 $\boldsymbol{o}$ 与相应的训练目标 $\boldsymbol{y}$ 的距离。RNN输入到隐藏的连接由 权重矩阵 $\boldsymbol{U}$ 参数化，隐藏到隐藏的循环连接由权重矩阵 $\boldsymbol{W}$ 参数化以及隐藏到输出的连接由权重 矩阵 $\boldsymbol{V}$ 参数化。

循环神经网络的梯度

$$\nabla_c L = \sum_t \left(\frac{\partial \boldsymbol{o}^{(t)}}{\partial \boldsymbol{c}}\right)^\top \nabla_{\boldsymbol{o}^{(t)}} L = \sum_t \nabla_{\boldsymbol{o}^{(t)}} L,$$

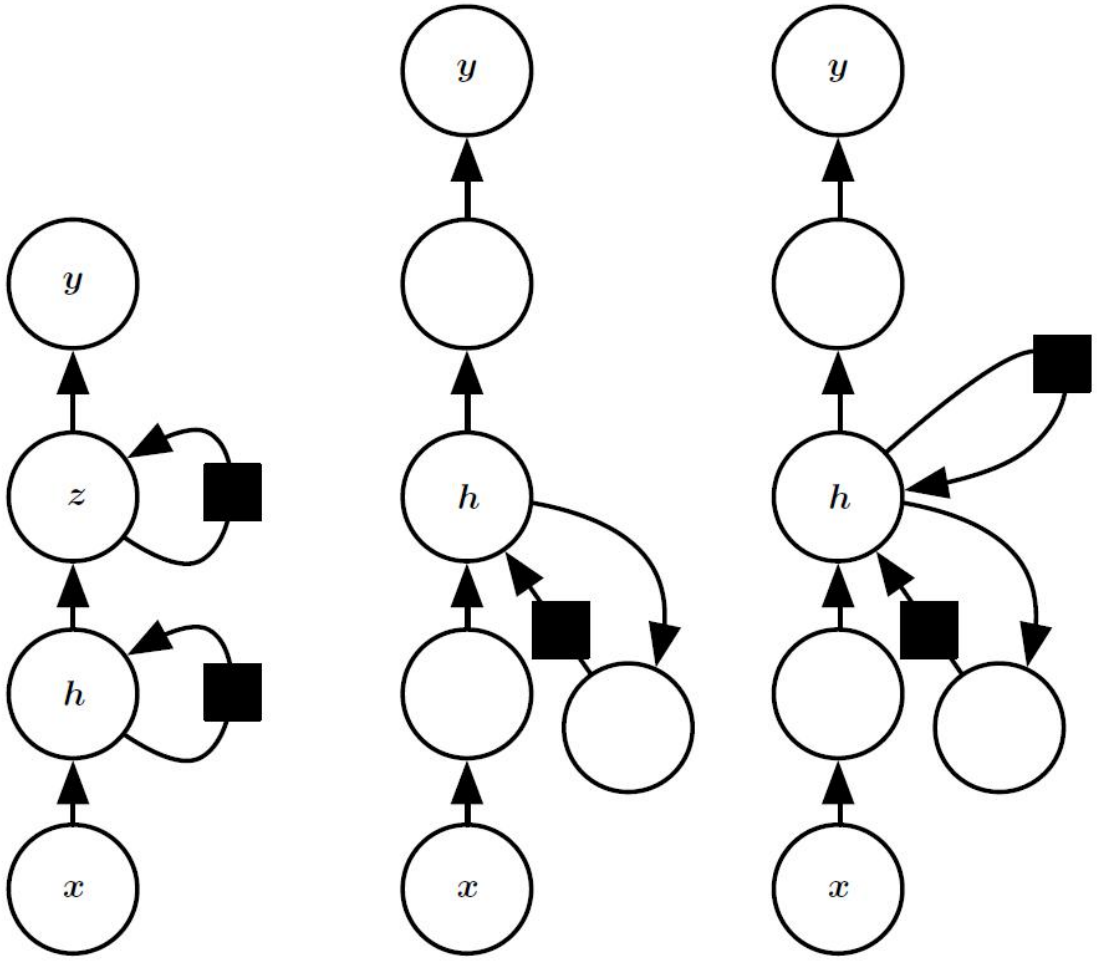
$$\nabla_b L = \sum_t \left(\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{b}^{(t)}}\right)^\top \nabla_{\boldsymbol{h}^{(t)}} L = \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) \nabla_{\boldsymbol{h}^{(t)}} L,$$

$$\nabla_V L = \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) \nabla_V o_i^{(t)} = \sum_t (\nabla_{\boldsymbol{o}^{(t)}} L) \boldsymbol{h}^{(t)\top},$$

$$\begin{aligned}\nabla_W L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{W^{(t)}} h_i^{(t)} \\ &= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L) \boldsymbol{h}^{(t-1)\top},\end{aligned}$$

$$\begin{aligned}\nabla_U L &= \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) \nabla_{U^{(t)}} h_i^{(t)} \\ &= \sum_t \operatorname{diag}\left(1 - \left(\boldsymbol{h}^{(t)}\right)^2\right) (\nabla_{\boldsymbol{h}^{(t)}} L) \boldsymbol{x}^{(t)\top},\end{aligned}$$

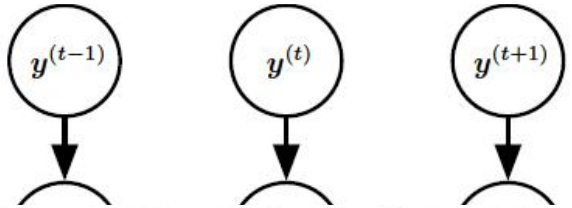
深度循环网络



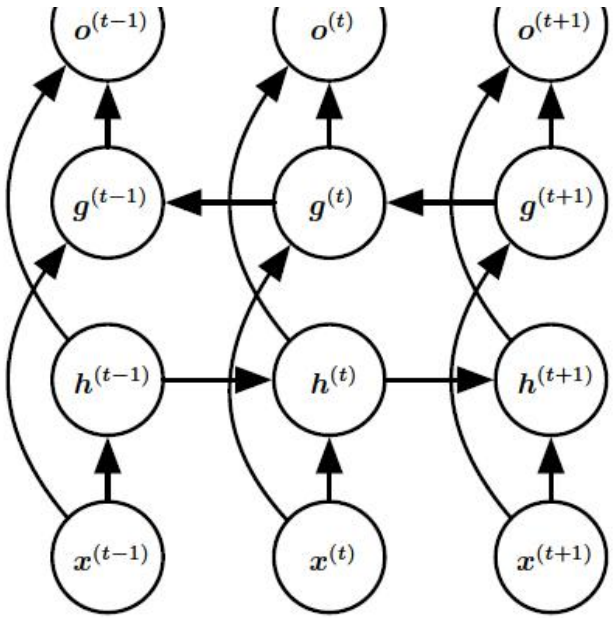
(a) 隐藏循环状态可 以被分解为具有层次的组。(b) 可以向输入到隐藏 隐藏到隐藏以及隐藏到输出的部分引入更深的 计算 。 (c) 可以引入跳跃连接来缓解路径延长 的效应。

双向RNN

双向 RNN 结合时间上从序列起点开始移动的 RNN 和另一个时间上 从序列末尾开始移动的 RNN。这 允许输出单元 $\boldsymbol{o}^{(t)}$ 能够计算同时依赖于过去 和未来。



使用双向循环神经网络，网络会先从序列的正方向读取数据，再从反方向读取数据，最后将网络输出的两种结果合在一起形成网络的最终输出结果。



3. 存在的问题

- 记忆最大的问题在于它有遗忘性。
- 如果一项任务需要依赖近期信息来预测结果，RNN往往具有较好的表现，但对于长时依赖问题，RNN就不在那么尽如人意。

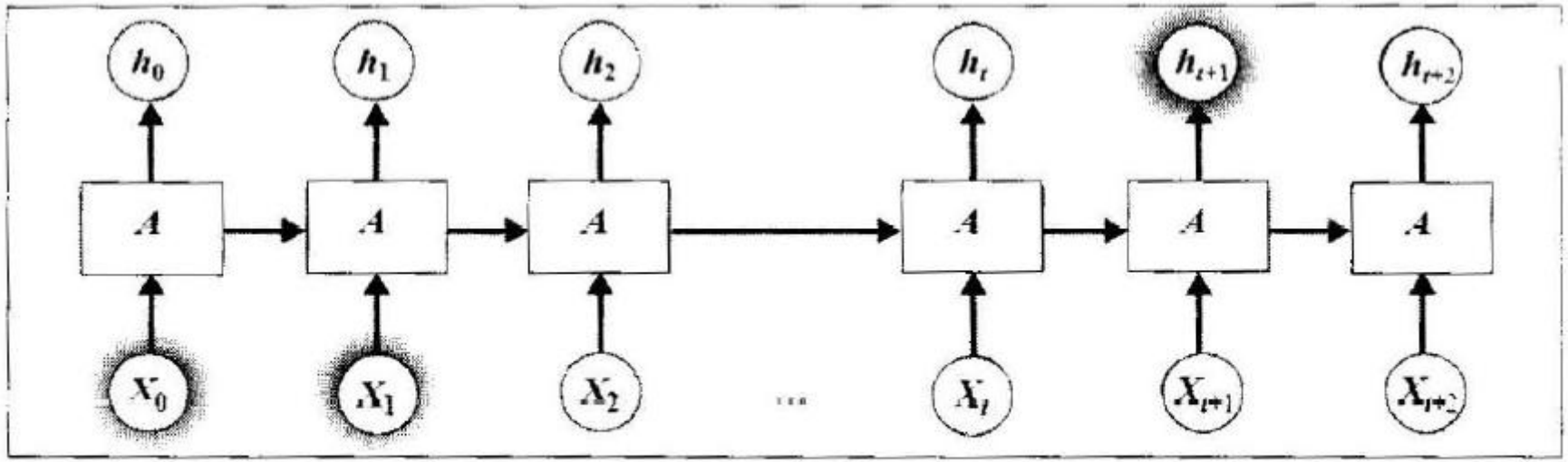


图 5.9　循环神经网络记忆长时间信息

梯度消失/爆炸

梯度

$$\frac{\partial \mathcal{L}}{\partial U} = \sum_{t=1}^{T}\sum_{k=1}^{t}\left(\prod_{i=k}^{t-1}\mathbf{diag}(f'(\mathbf{z}_i))U^{\mathrm{T}}\right)\frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_t}\mathbf{h}_k^{\mathrm{T}}$$

我们定义$\gamma = \|\mathbf{diag}(f'(\mathbf{z}_i))U^{\mathrm{T}}\|$，则在上面公式中的括号里面为$\gamma^{t-k}$。如果$\gamma > 1$，当$t-k \to \infty$时，$\gamma^{t-k} \to \infty$，会造成系统不稳定，也就是所谓的**梯度爆炸问题**（Gradient Exploding Problem）；相反，如果$\gamma < 1$，当$t-k \to \infty$时，$\gamma^{t-k} \to 0$，会出现和深度前馈神经网络类似的**梯度消失问题**（Gradient Vanishing Problem）。

由于梯度爆炸或消失问题，实际上只能学习到短周期的依赖关系。这就是所谓的长期依赖问题。

长期依赖问题

- 循环神经网络在时间维度上非常深！
  - 梯度消失或梯度爆炸

- 改进方法
  - 循环边改为线性依赖关系

  - 增加非线性

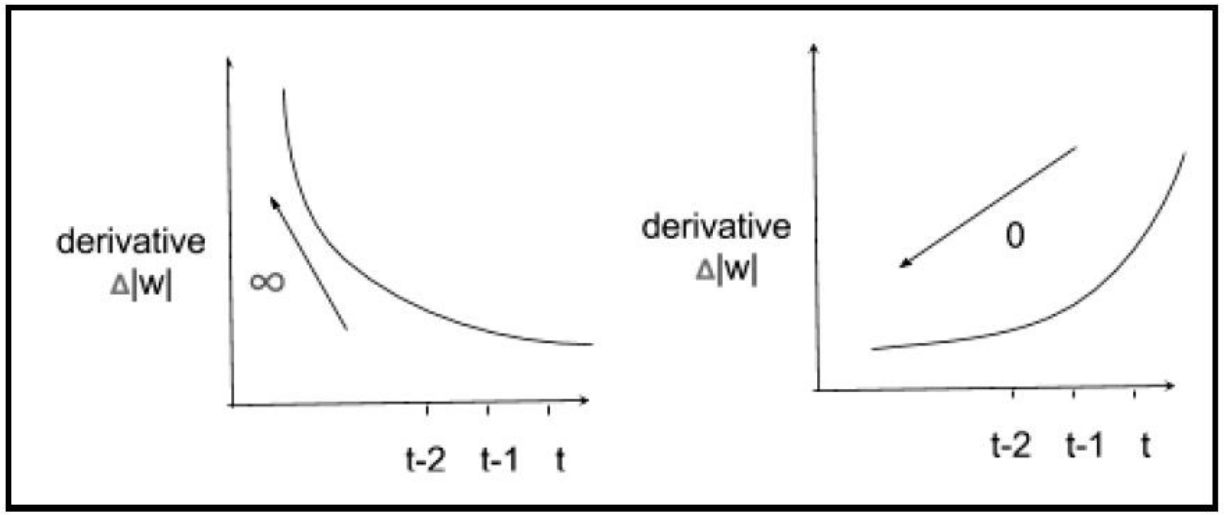
$$\mathbf{h}_t = \mathbf{h}_{t-1} + \mathbf{W}\mathbf{g}(\mathbf{x}_t)$$

$$\mathbf{c}_t = \mathbf{c}_{t-1} + \mathbf{W}\mathbf{g}(\mathbf{x}_t)$$

$$\mathbf{h}_t = \tanh(\mathbf{c}_t).$$

长期依赖的挑战

经过许多阶段传播后的梯度倾向于消失（大部分情况）或爆炸（很少，但对优化过程影响很大）。



门控RNN（gated RNN）

✓ 长短期记忆（long short-term memory，LSTM）

✓ 门控循环单元（gated recurrent unit，GRU）

Understanding LSTM Networks:http://colah.github.io/posts/2015-08-Understanding-LSTMs/

➢ LSTM

LSTM 是 Long Short Term Memory Networks 的缩写，按字面翻译就是长的短时记忆网络，从字面意思知道它解决的仍然是短时记忆的问题，只不过这种短时记忆比较长，能在一定程度上解决长时依赖的问题。LSTM 的网络结构是 1997 年由 Hochreiter 和 Schmidhuber 提出的，随后这种网络结构变得非常流行，有很多人跟进相关的工作解决了很多实际的问题，现在 LSTM 仍然被广泛地使用。

循环神经网络的结构都是链式循环的网络结构，LSTM 的网络结构大体上也是这样的结构，不过 LSTM 在网络的内部有着更加复杂的结构，所以它能够处理长时依赖的问题。

LSTM Networks

LSTM 的抽象网络结构示意图如图5.10所示。从图 5.10 中可以看出 LSTM 由三个门来控制，这三个门分别是输入门、遗忘门和输出门。顾名思义，输入门控制着网络的输入，遗忘门控制着记忆单元，输出门控制着网络的输出。这其中最重要的就是遗忘门，遗忘门的作用是决定之前的哪些记忆将被保留，哪些记忆将被去掉，正是由于遗忘门的作用，使得 LSTM 具有了长时记忆的功能，对于给定的任务，遗忘门能够自己学习保留多少以前的记忆，这使得不再需要人为干扰，网络就能够自主学习。

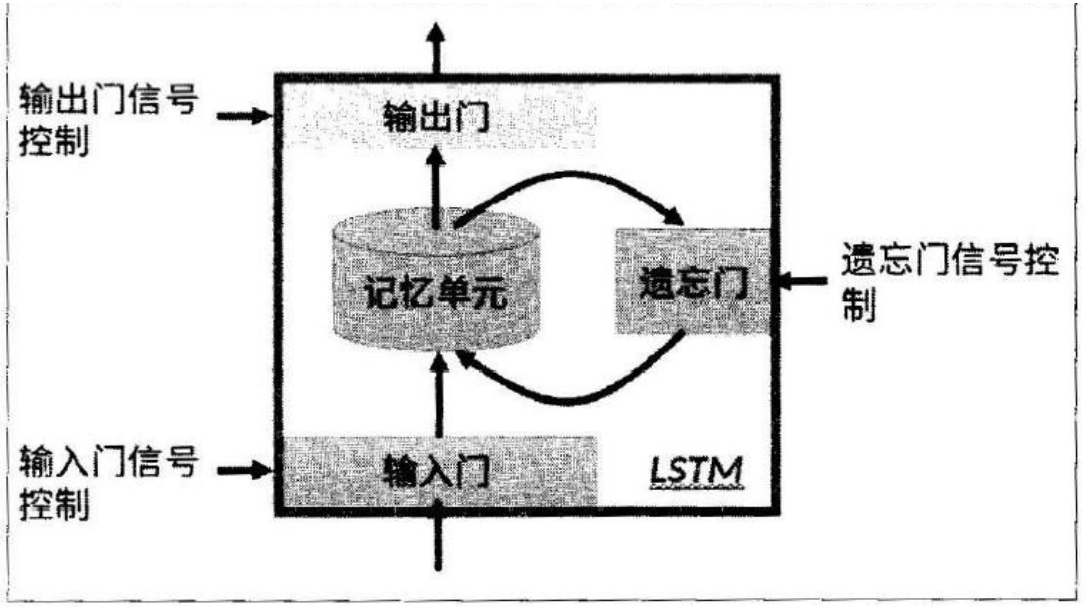
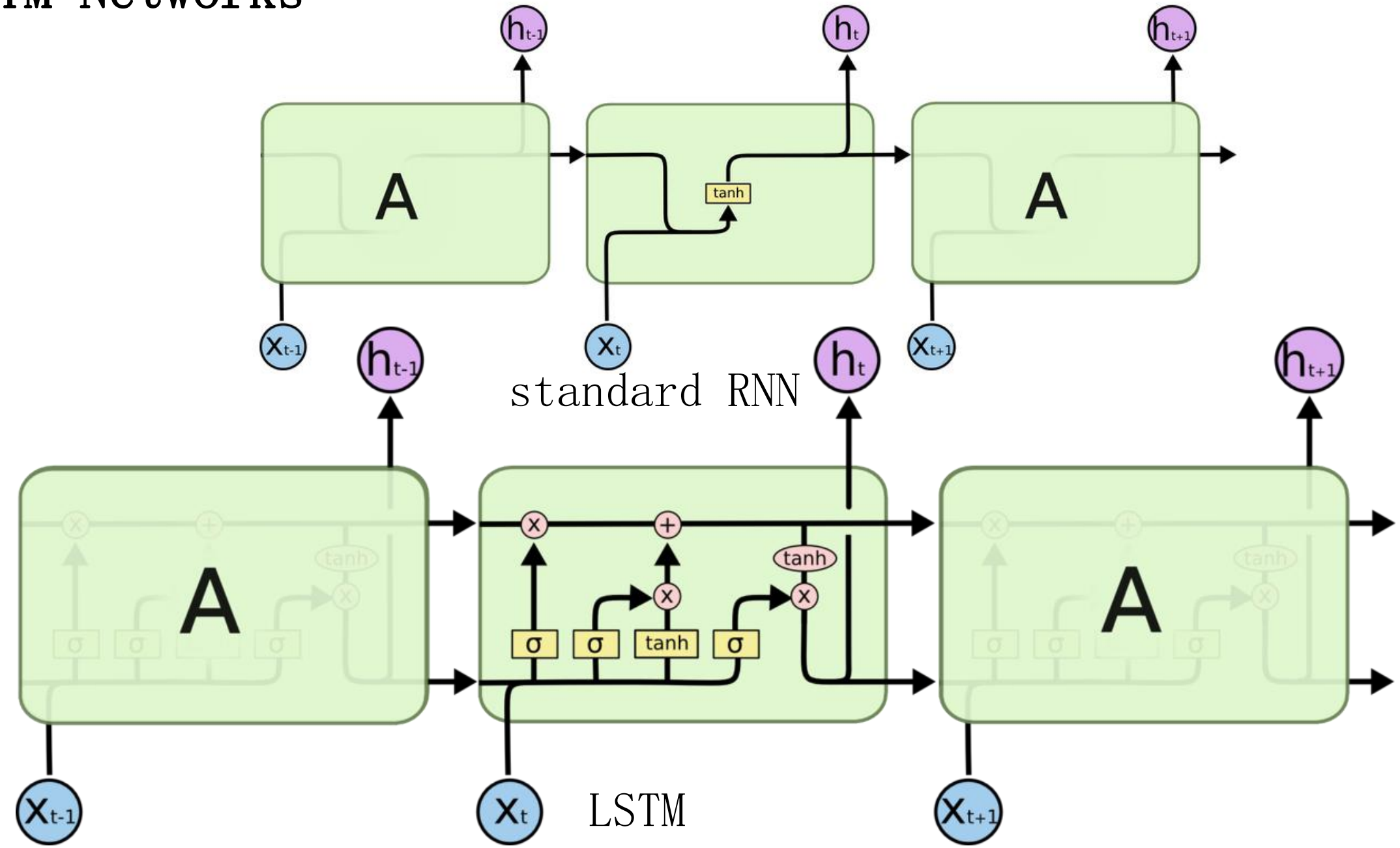
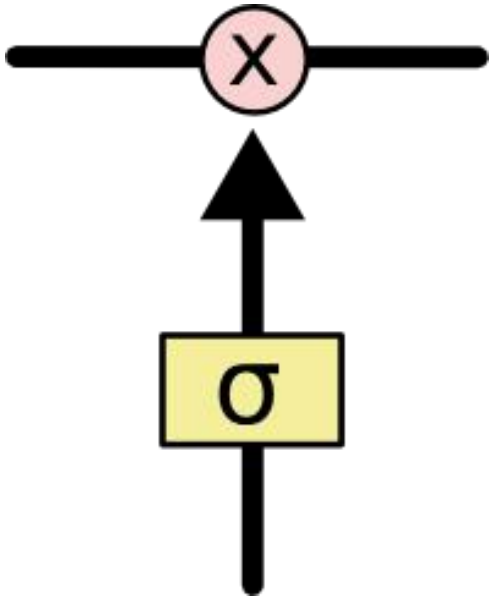


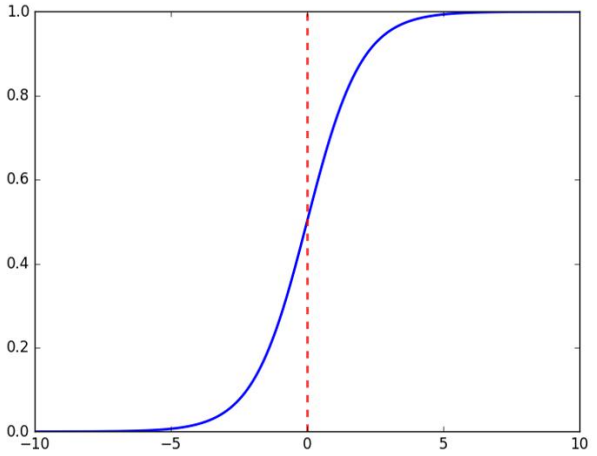
图 5.10　LSTM 的抽象网络结构

LSTM Networks



Step-by-Step

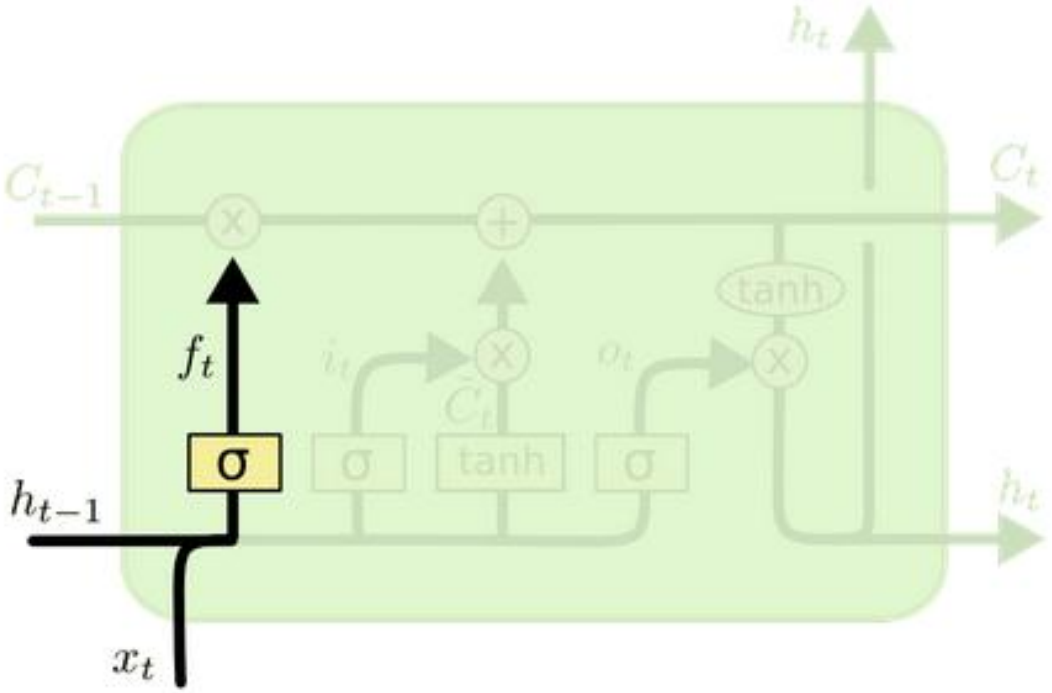




$$\sigma(x) = \frac{1}{(1+e^{-x})}$$

“门”是一种让信息选择式通过的方法，包含一个 sigmoid 神经网络层和一个 逐点乘法操作，sigmoid层输出0到1之间的概率，描述每个部分有多少量可以通过。

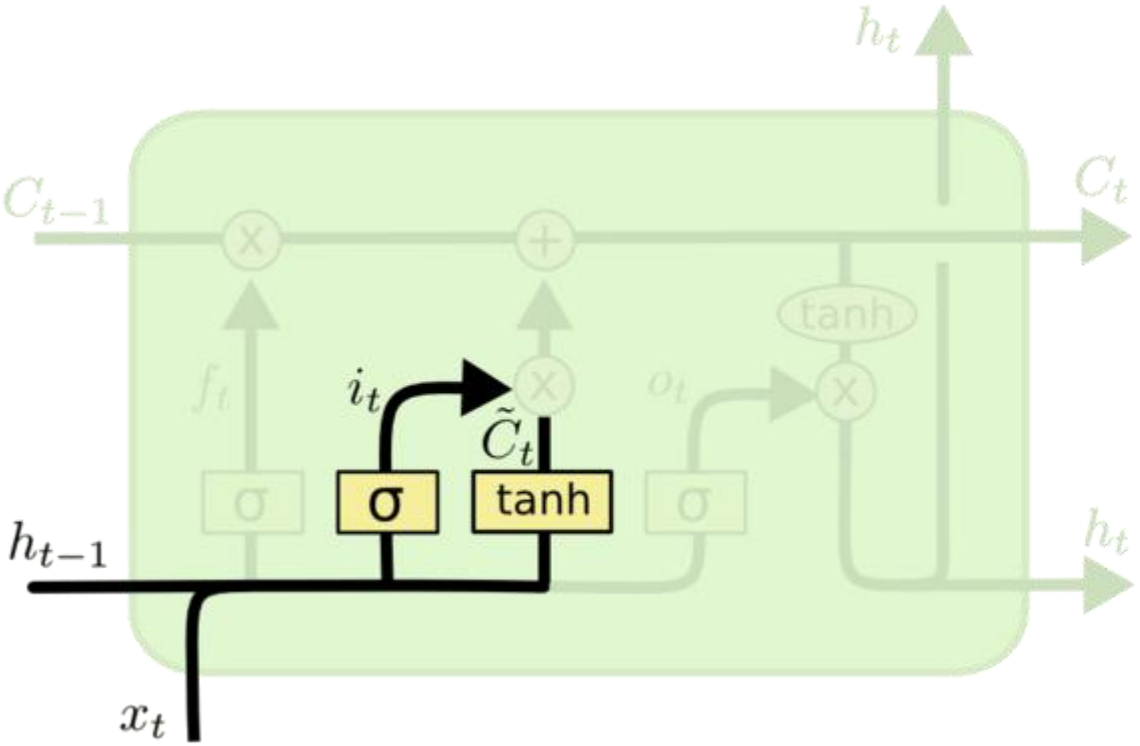
Step-1：决定丢弃信息



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

第一步是决定从细胞状态中丢弃什么信息。这个决定通过一个称为 forget gate（遗忘门）完成。该门会读取 h_{t-1}和 x_t，输出一个在 0 到 1 之间的数值给每个在细胞状态 C_{t-1} 中的数字。

Step-2：确定什么样的新信息被存放在细胞状态中



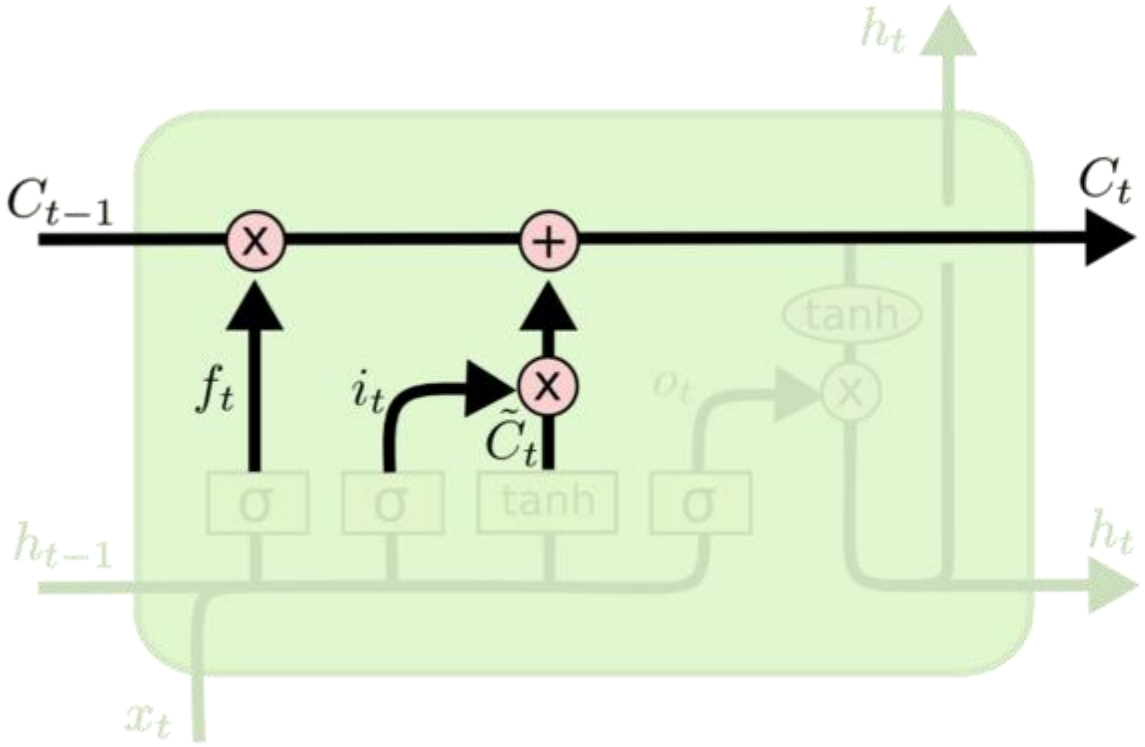
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

1. sigmoid层决定什么样的值需要更新
2. tanh层创建一个新的候选值向量$\tilde{C}_t$

上述两步是为更新做准备

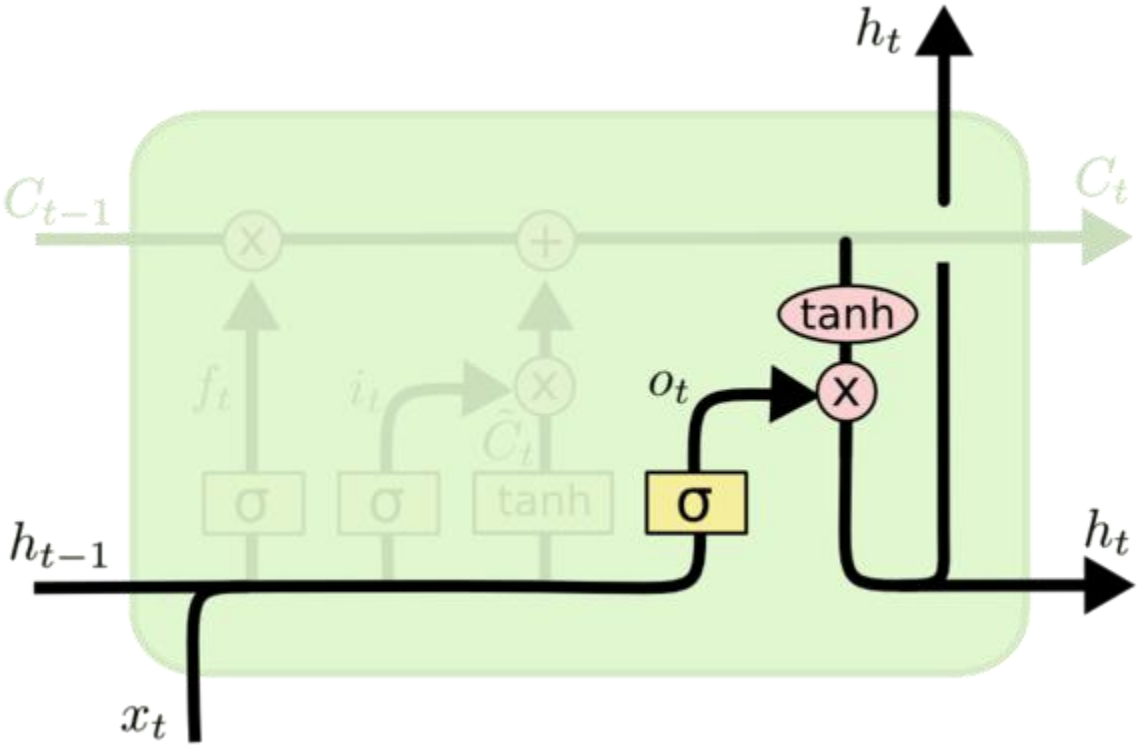
Step-3：更新细胞状态



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

1. 更新C_{t-1}为C_t
2. 旧状态与f_t 相乘， 丢弃掉我们确定要丢弃的信息
3. 加上$i_t * \tilde{C}_t$， 就是新的候选值

Step-4：基于细胞状态得到输出



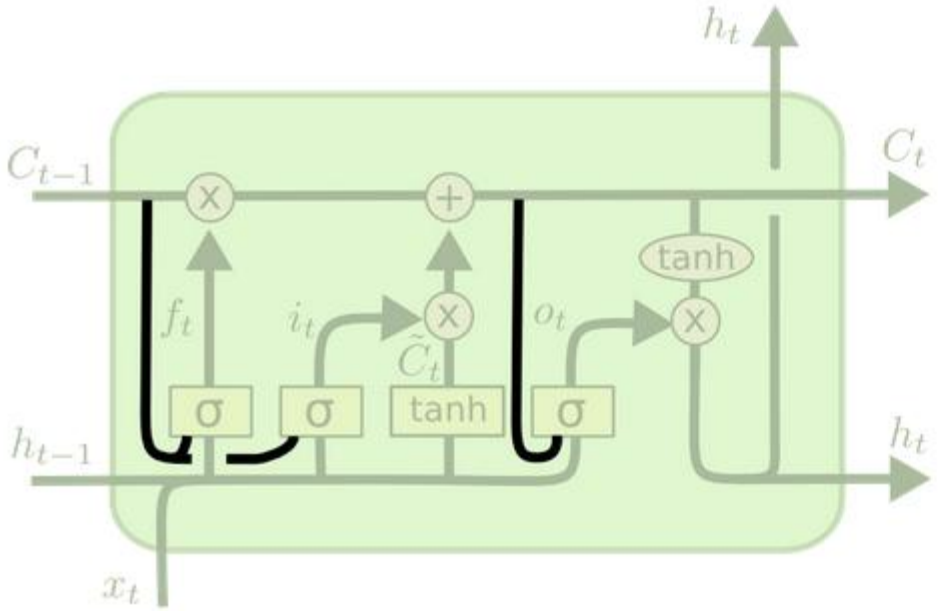
$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

1. 首先运行一个sigmoid层来确定细胞状态的哪个部分将输出
2. 接着用tanh处理细胞状态在将它和sigmoid门的输出相乘，输出我们确定输出的那部分。

LSTM的各种变体

Gers&Schmidhuber（2000）提出添加“窥视孔（Peephole）连接”。这意味着我们让门层观察细胞状态。

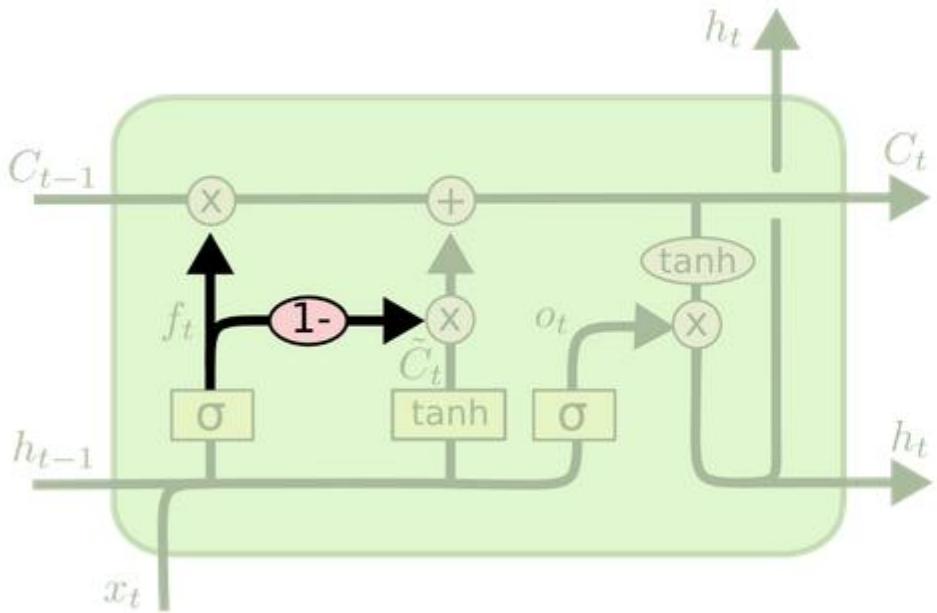


$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

上图为所有的门增加窥视孔，但许多论文会给出一些窥视孔，而不是全部。

LSTM的各种变体

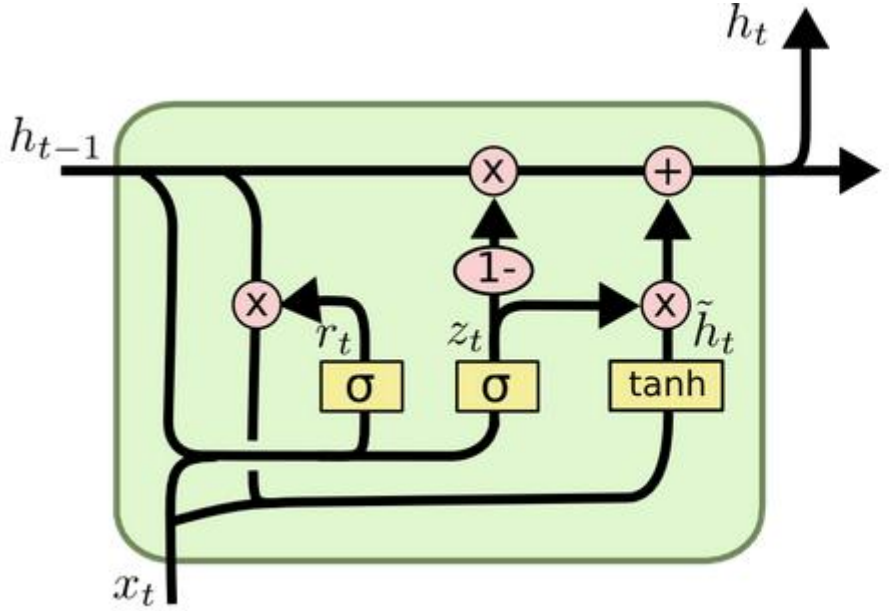
耦合遗忘门和输入门。我们不是单独决定忘记什么和应该添加什么新信息，而是一起做出这些决定。



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

LSTM的各种变体

Cho, et al. (2014)提出门控递归单元（Gated Recurrent Unit, GRU）。将遗忘门和输入门合成了一个单一的更新门。同时还混合了细胞状态和隐藏状态和其他一些改动。最终的模型比标准的 LSTM模型要简单，效果和LSTM差不多，但是参数少了1/3，不容易过拟合。



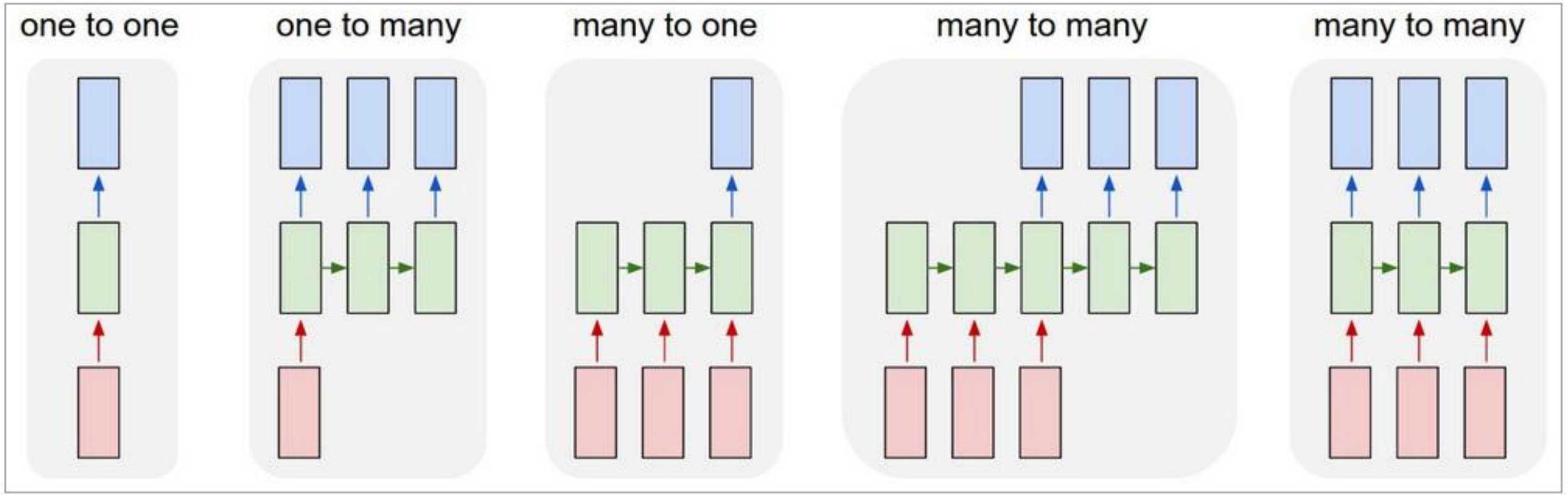
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

Operating on sequences of vectors



(1.e.g. image classification). (2.e.g. image captioning). (3.e.g. sentiment analysis).
（4.e.g. Machine Translation）. (5.e.g. video classification)

应用

5.5.1 Many to one

循环神经网络不仅能够输入序列、输出序列、还能够输入序列，输出单个向量。只需要在输出的序列里面取其中的一个就可以，通常是取最后一个。这样的结构被称为Many to one，那么这种结构能够做什么任务呢？

第一个任务是情感分析，将一句话作为序列输入网络，输出只取最后一个，根据输出判断这句话的态度是消极的还是积极的；第二个任务是关键字提取，原理也是一样的，将一句话作为序列输入网络，输出只取最后一个，不同的是用它来表示这句话的关键字。具体的结构可以用图5.30所示。

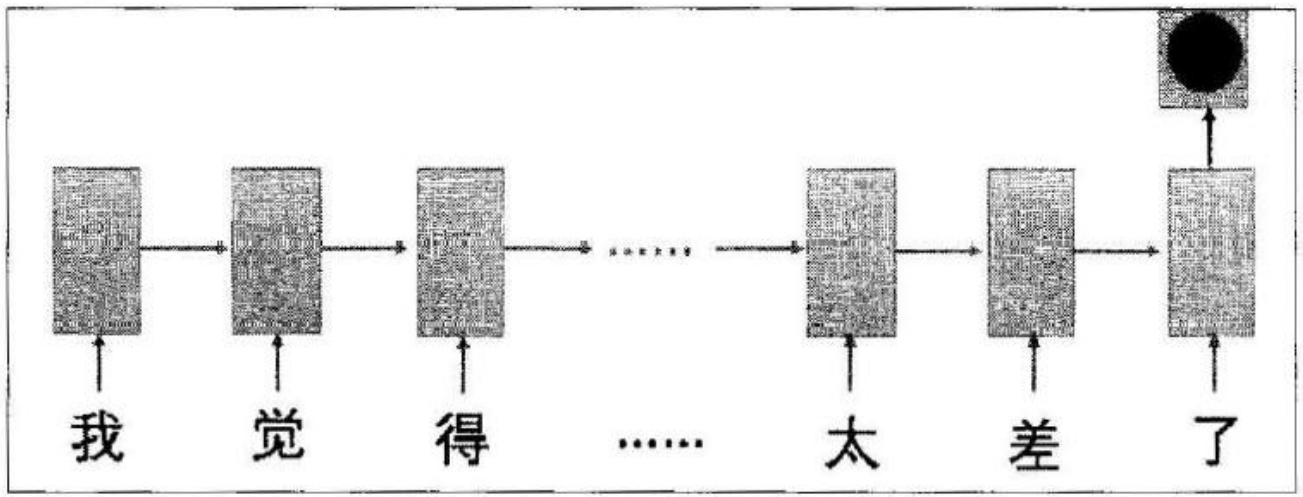


图 5.30 Many to one

应用

5.5.2 Many to Many（shorter）

第二种结构就是输入和输出都是序列，但是输出的序列比输入的序列短。这种类型的结构通常会在语音识别中遇到，因为一段话如果用语音表达往往会比这段话更长，可以看看图 5.31。

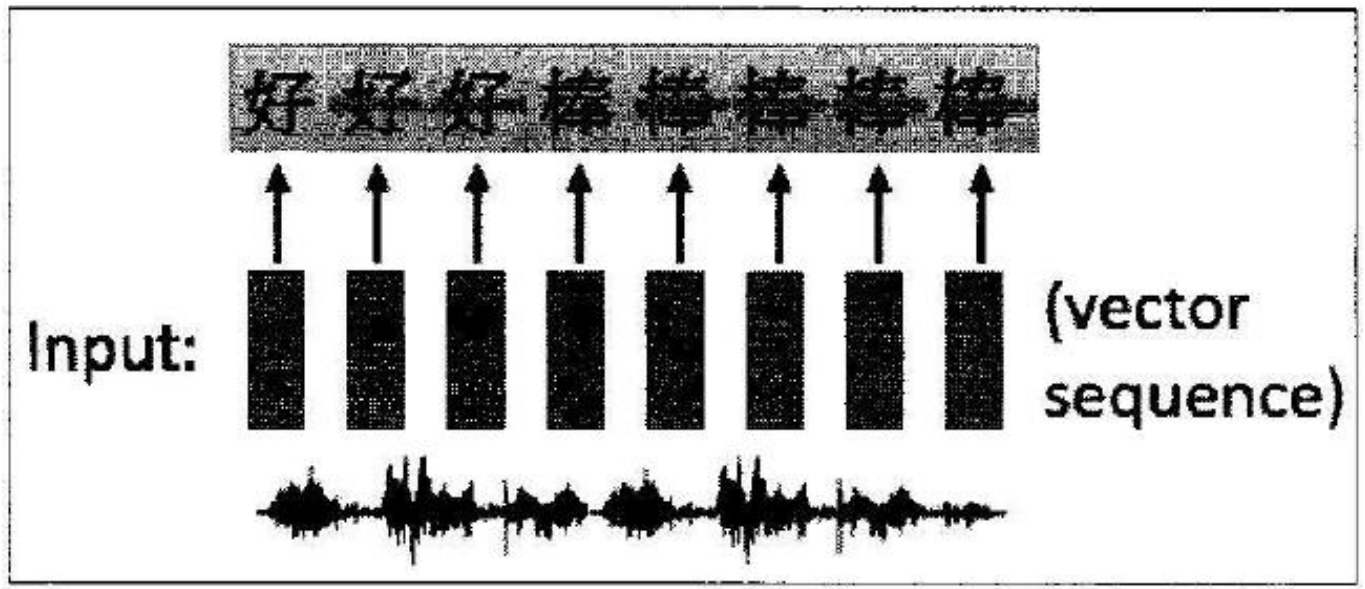


图 5.31 Many to Many

应用

5.5.3 Seq2seq

第三种情况是输出的长度不确定，这种情况一般是在机器翻译的任务中出现，将一句中文翻译成英文，那么这句英文的长度有可能会比中文短，也有可能会比中文长，所以这时候输出的长度就不确定了，需要用序列到序列的模型来解决这个问题，具体如图5.32所示。

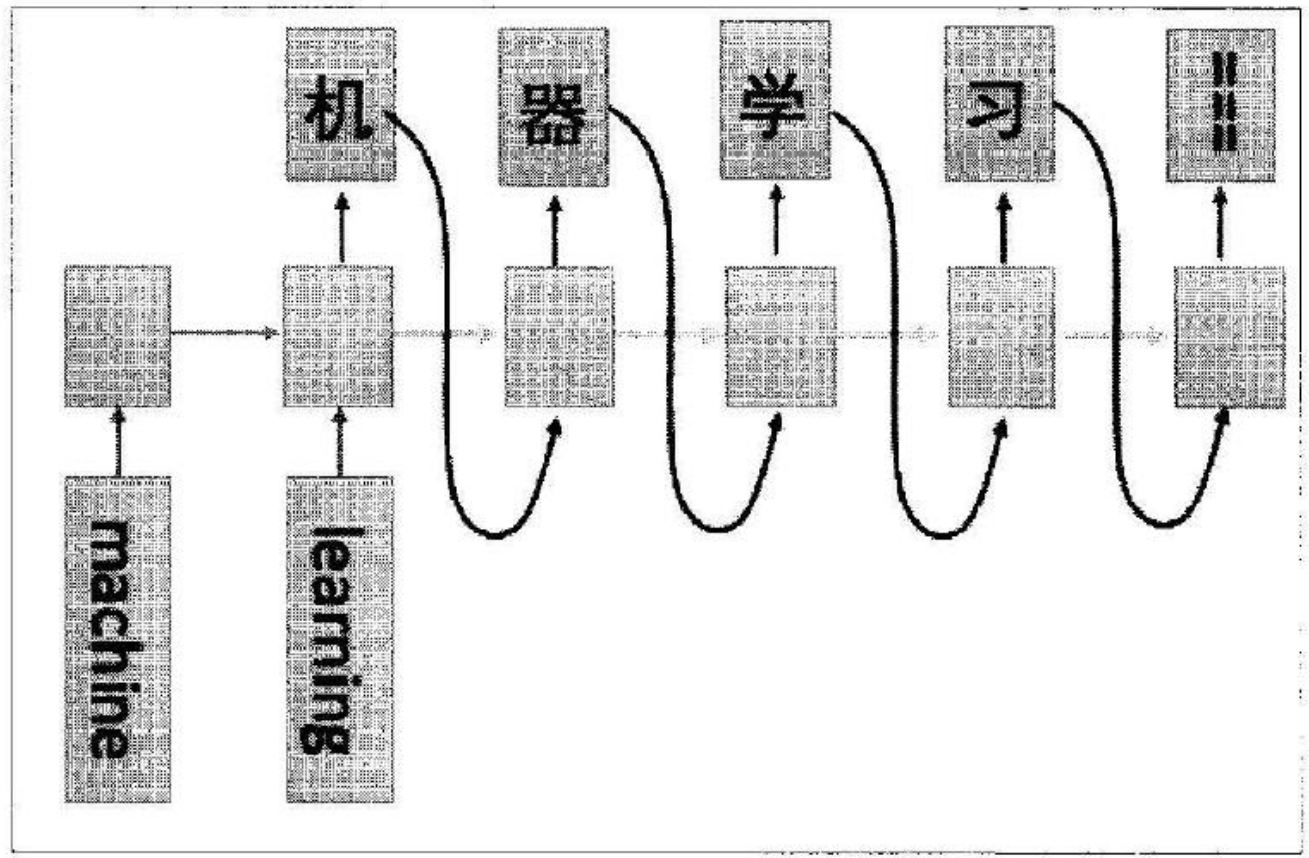


图 5.32 序列到序列模型

应用

聊天机器人和问答系统也都是同样的原理，将句子输入，输出是根据前面的输入来得到，如图 5.33所示。

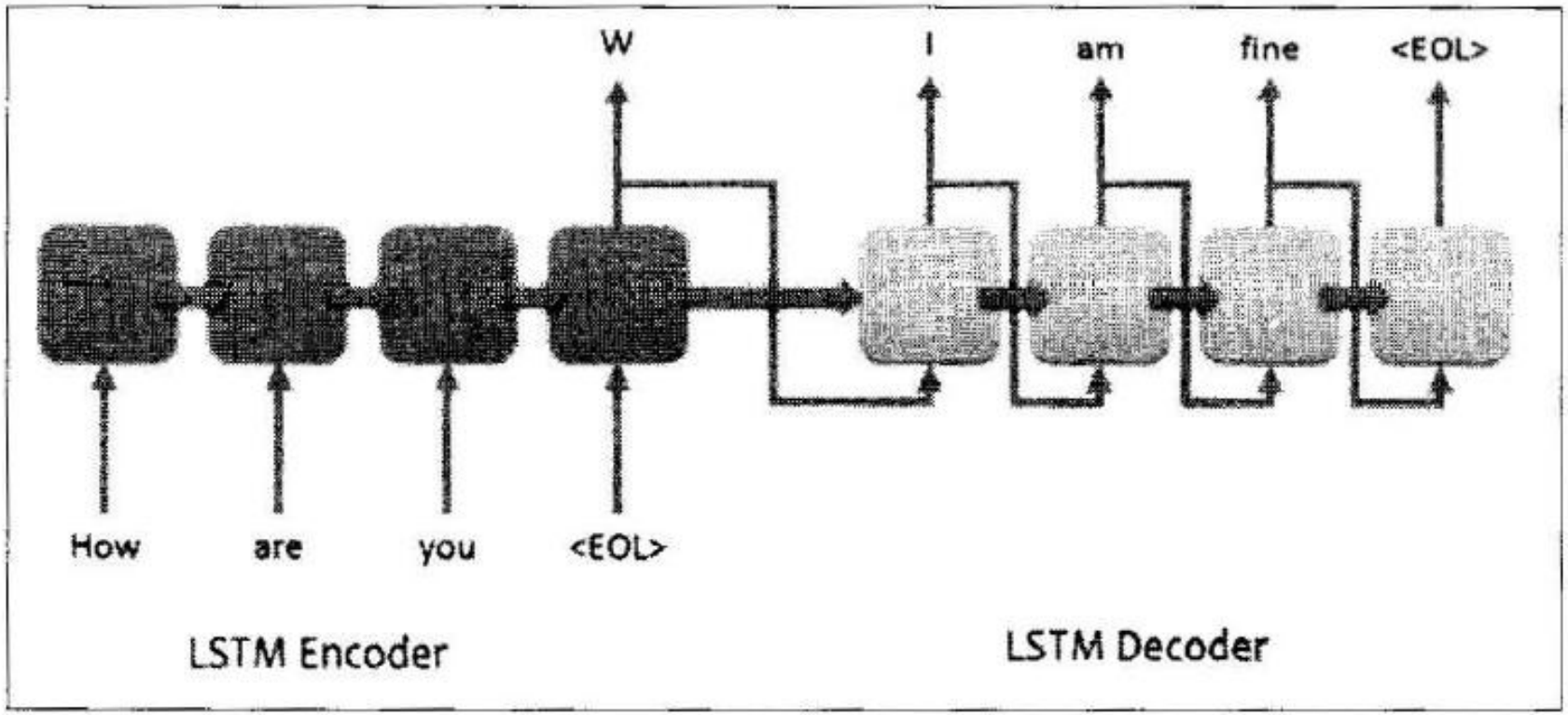


图 5.33　聊天机器人原理

应用

5.5.4 CNN+RNN

RNN 和 CNN 还能够联合在一起完成图像描述任务，简而言之，就是通过预训练的卷积神经网络提取图片特征，接着通过循环网络将特征变成文字描述，具体细节就不再展开介绍了，如图 5.34所示。

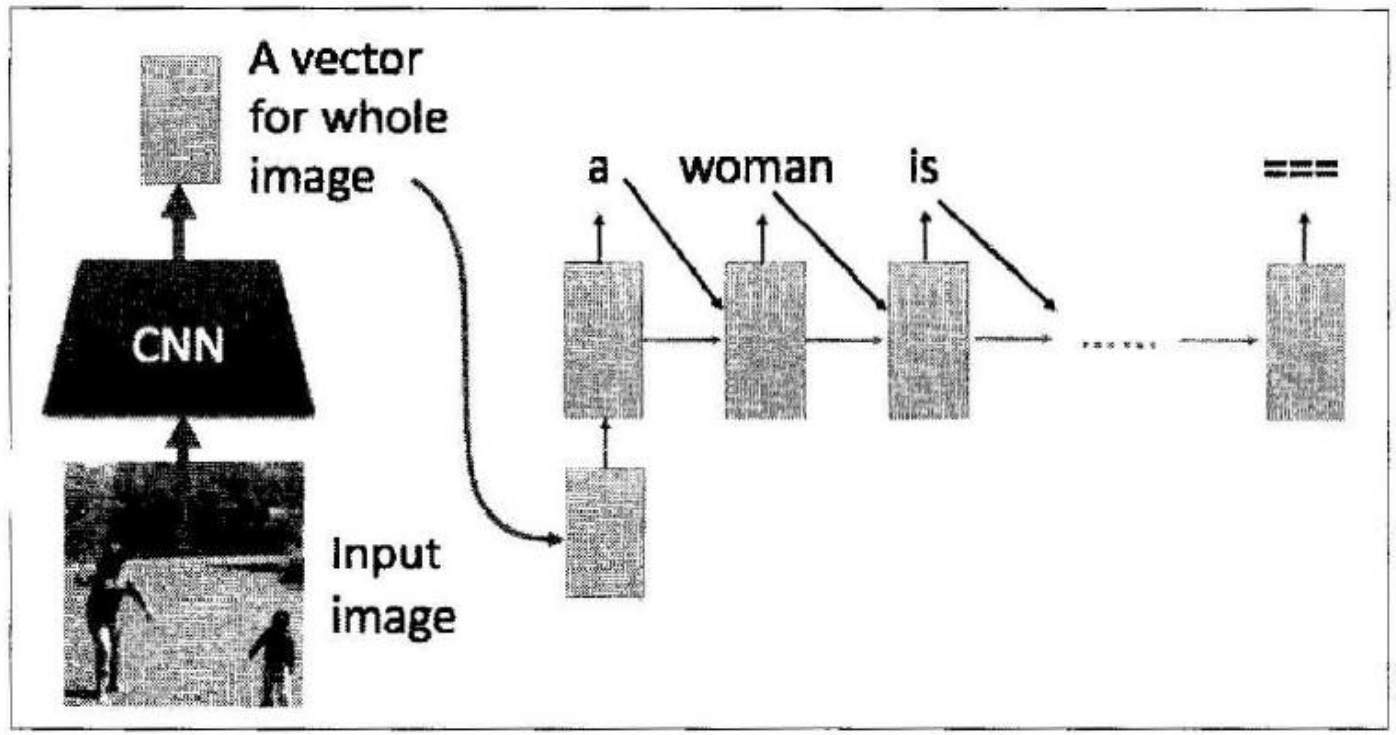
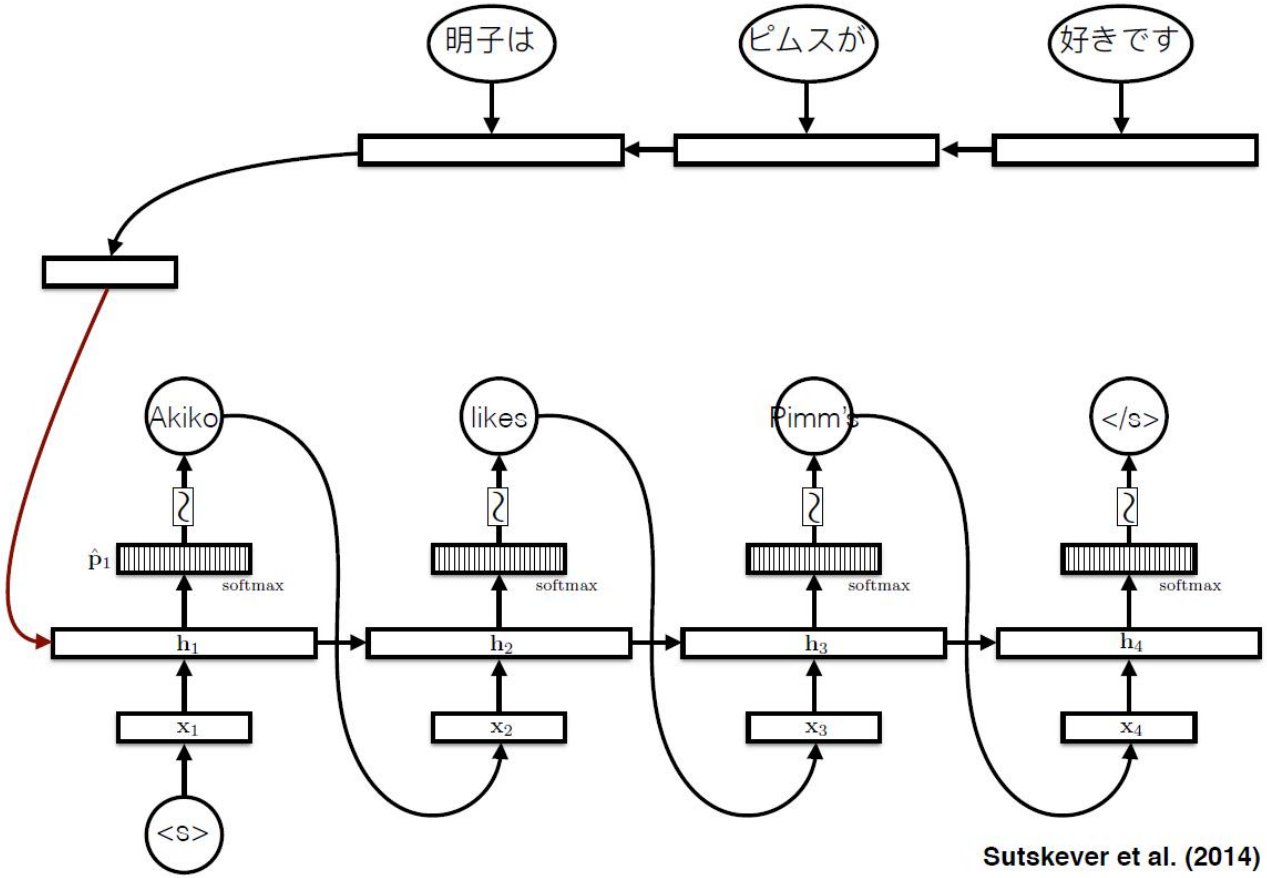
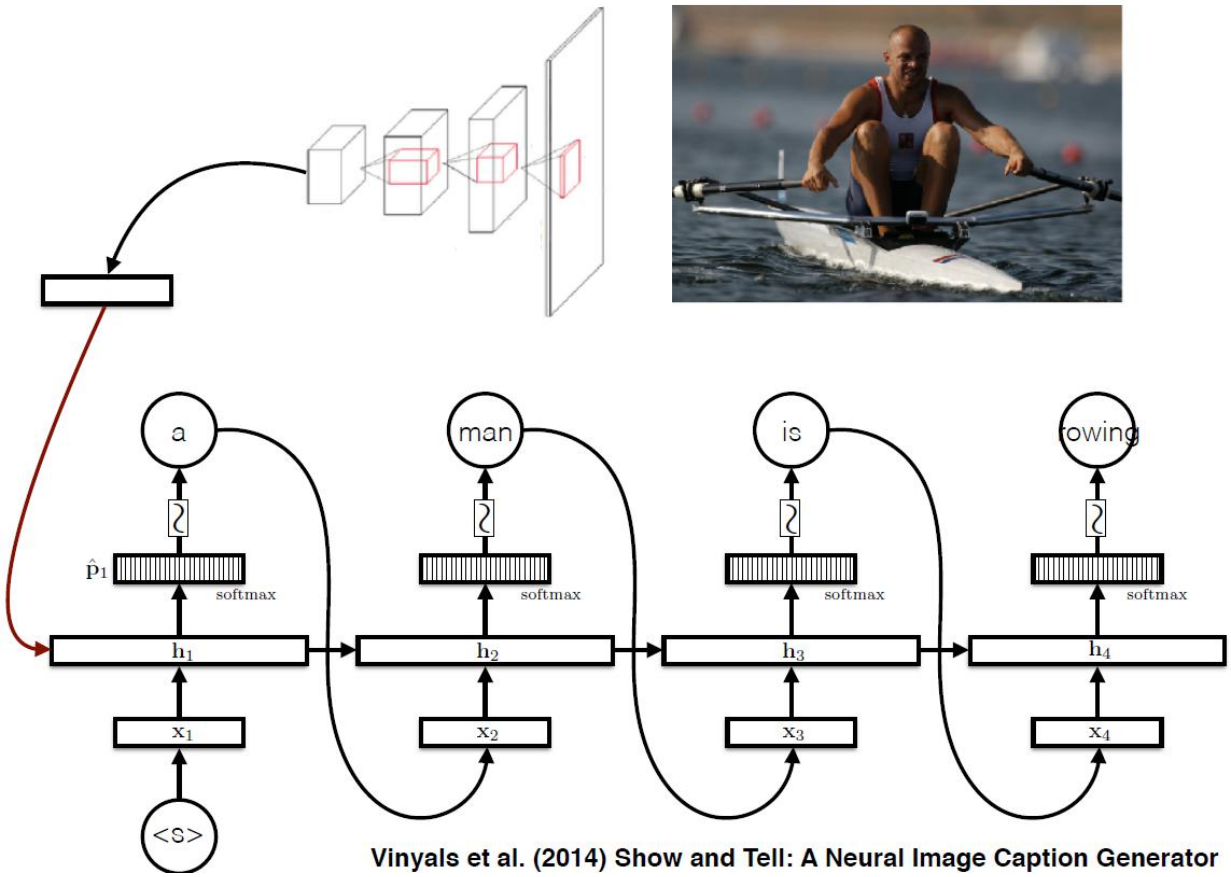


图 5.34 RNN 与 CNN 共同完成图像描述任务



Sutskever et al. (2014)

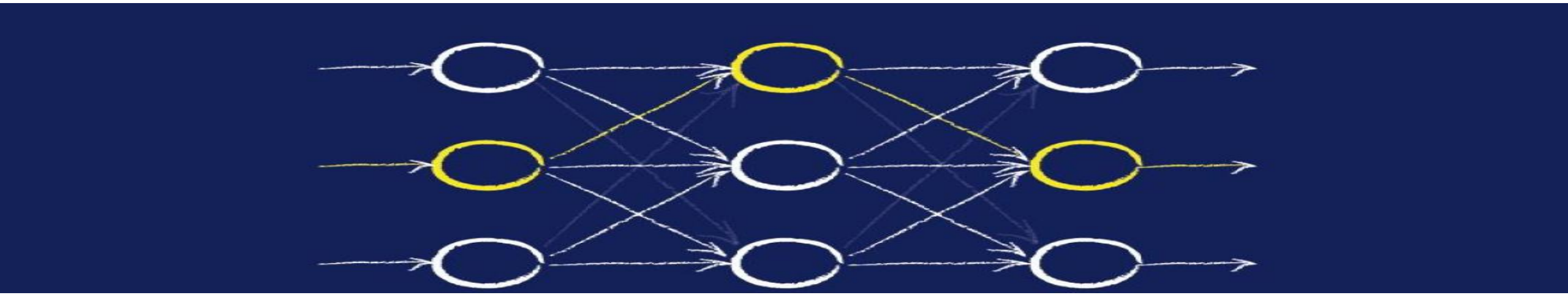
Machine translation



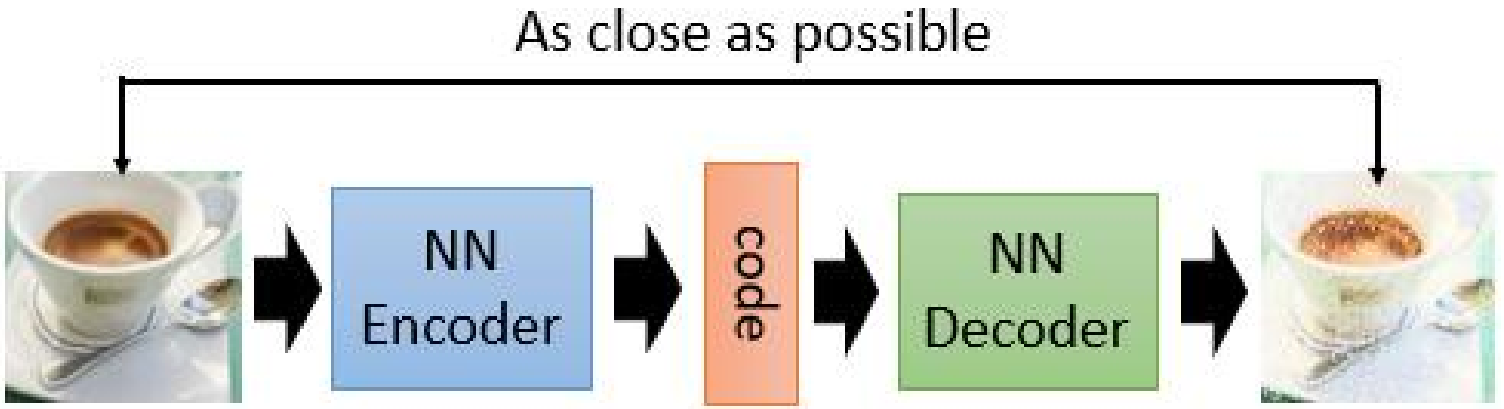
Vinyals et al. (2014) Show and Tell: A Neural Image Caption Generator

Image caption

自编码器与生成对抗网络



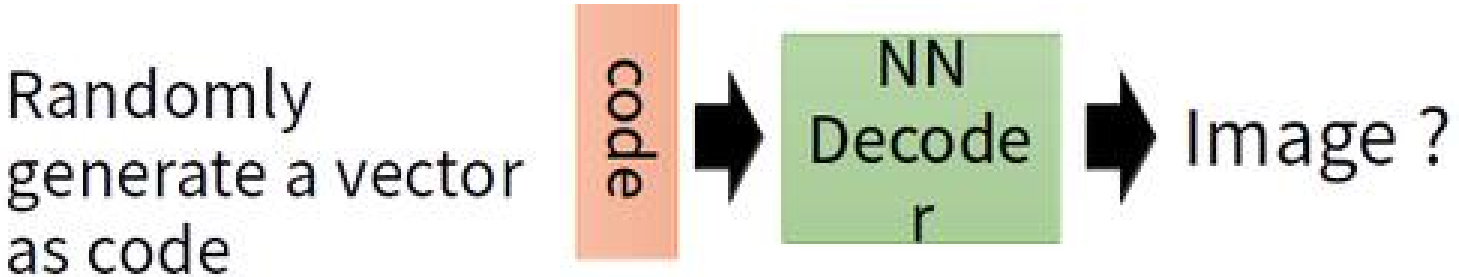
自编码器（autoencoder）是神经网络的一种，经过训练后能尝试将输入复制到输出。**自编码器**（autoencoder）内部有一个隐藏层 $\boldsymbol{h}$，可以产生**编码**（code）表示输入。该网络可以看作由两部分组成：一个由函数 $\boldsymbol{h} = f(\boldsymbol{x})$ 表示的编码器和一个生成重构的解码器 $\boldsymbol{r} = g(\boldsymbol{h})$。



$$L(\boldsymbol{x}, g(f(\boldsymbol{x})))$$

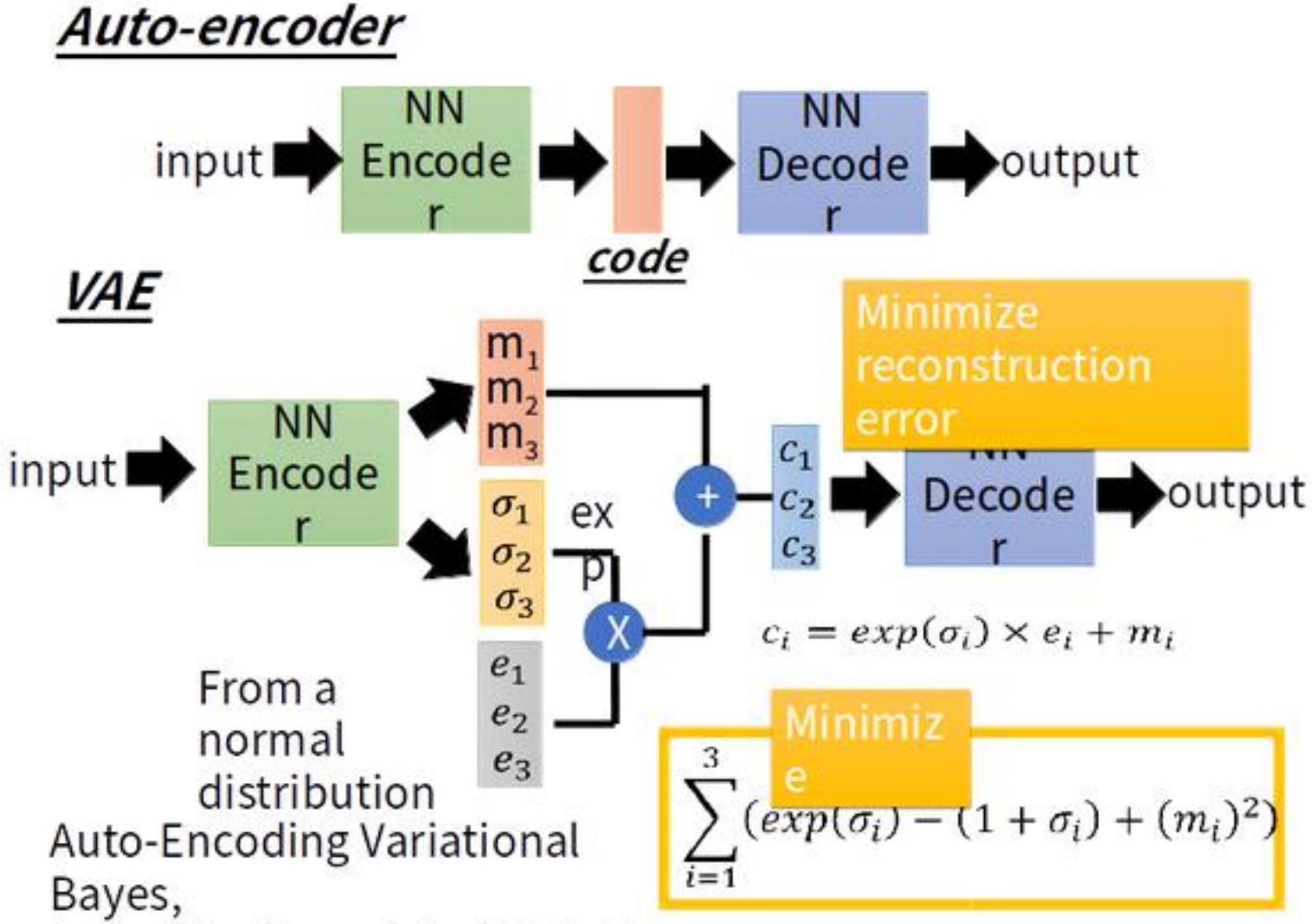
从图6.1中能够看到两个部分：第一个部分是编码器（Encoder），第二个部分是解码器（Decoder），编码器和解码器都可以是任意的模型，通常使用神经网络模型作为编码器和解码器。输入的数据经过神经网络降维到一个编码（code），接着又通过另外一个神经网络去解码得到一个与输入原数据一模一样的生成数据，然后通过比较这两个数据，最小化它们之间的差异来训练这个网络中编码器和解码器的参数。当这个过程

训练完之后，拿出这个解码器，随机传入一个编码（code），通过解码器能够生成一个和原数据差不多的数据，图6.2就是希望能够生成一张差不多的图片。



传统自编码器被用于降维或特征学习。近年来，自编码器与潜变量模型理论的联系将自编码器带到了生成式建模的前沿。

变分自动编码器VAE(Variational Auto－encoder)

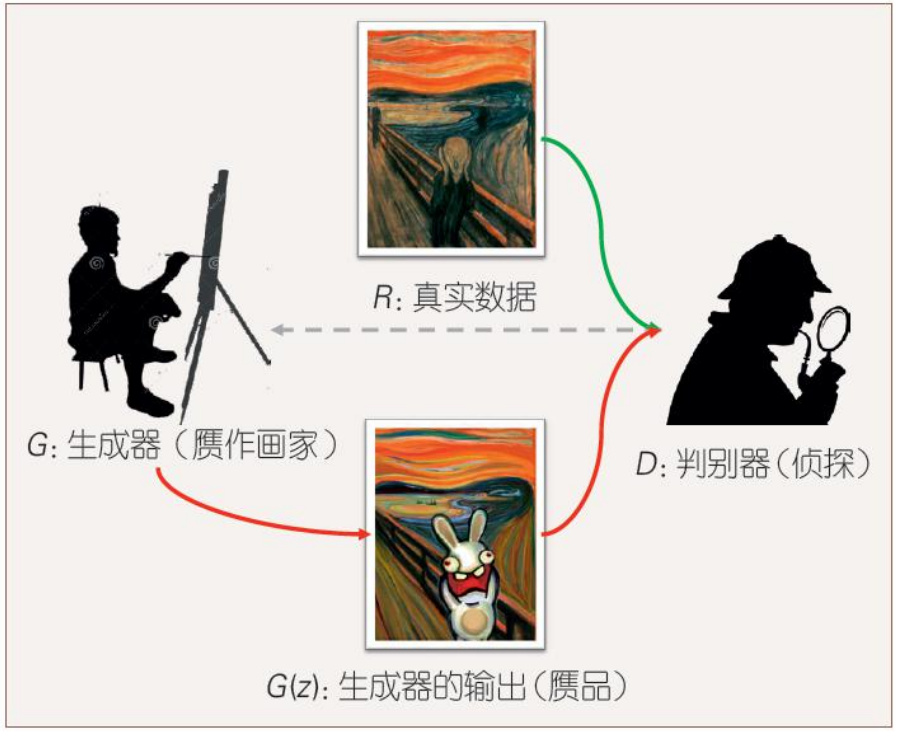


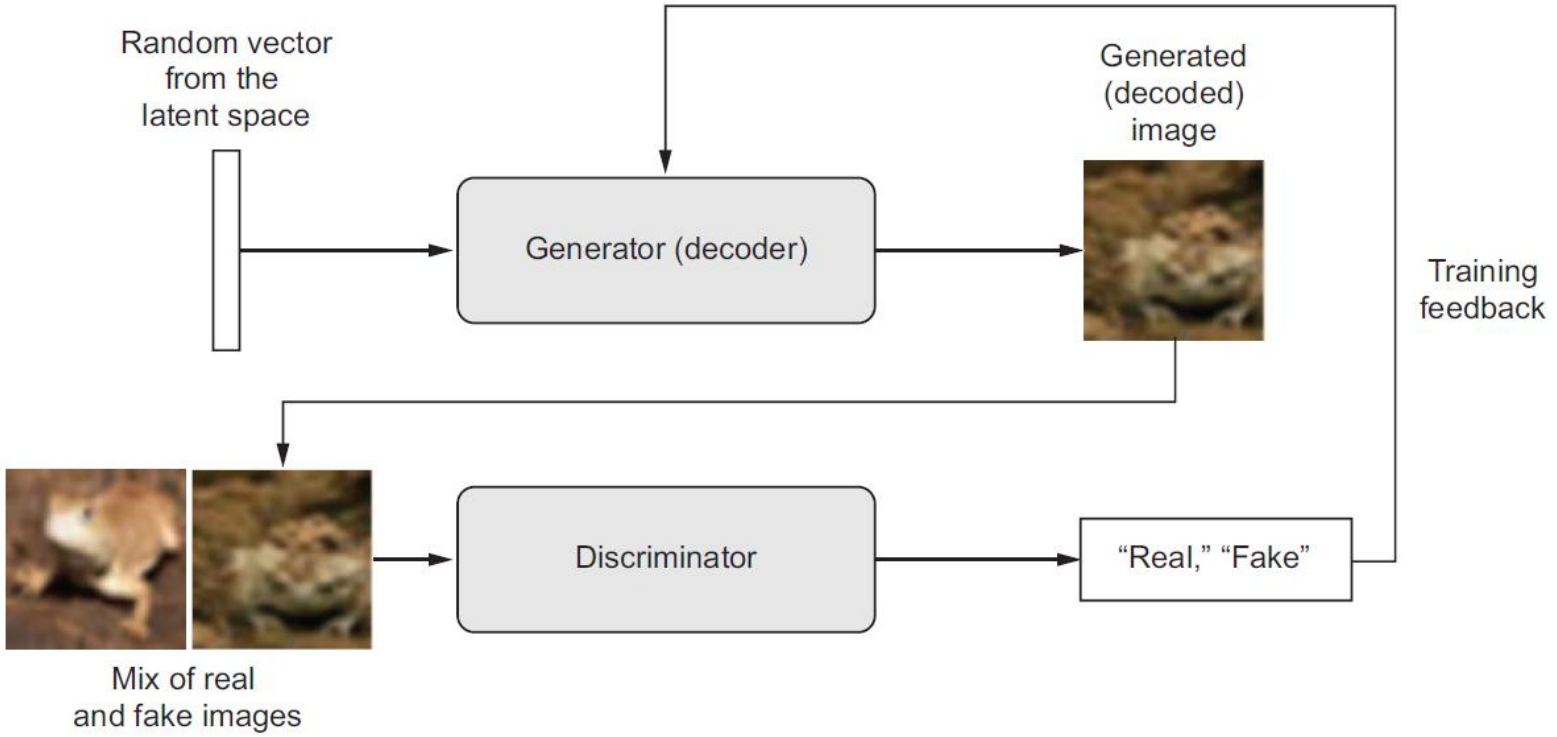
Auto-Encoding Variational Bayes,

Generative Adversarial Nets（生成对抗网络）

2014 年，深度学习三巨头之一 Ian Goodfellow 提出了生成对抗网络（Generative Adversarial Networks, GANs）这一概念，刚开始并没有引起轰动，直到 2016 年，学界、业界对它的兴趣如“井喷”一样爆发，多篇重磅文章陆续发表，Lecun 这样形容 GANs “adversarial training is the coolest thing since sliced bread”。2016 年 12 月 NIPS 大会上，Goodfellow 做了关于 GANs 的专题报告，使得 GANs 成为了当今最热门的研究领域之一，接下来介绍如今深度学习界的明星——生成对抗网络。

Generative Adversarial Nets（生成对抗网络）





$$\min_G \max_D V(D,G) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))].$$

生成模型

在生成对抗网络中，不再是将图片输入编码器得到隐含向量然后生成图片，而是随机初始化一个隐含向量，根据变分自动编码器的特点，初始化一个正态分布的隐含向量，通过类似解码的过程，将它映射到一个更高的维度，最后生成一个与输入数据相似的数据，这就是假的图片。这时自动编码器是通过对比两张图片之间每个像素点的差异计算损失函数的，而生成对抗网络会通过对抗过程来计算出这个损失函数，如图6.9所示。

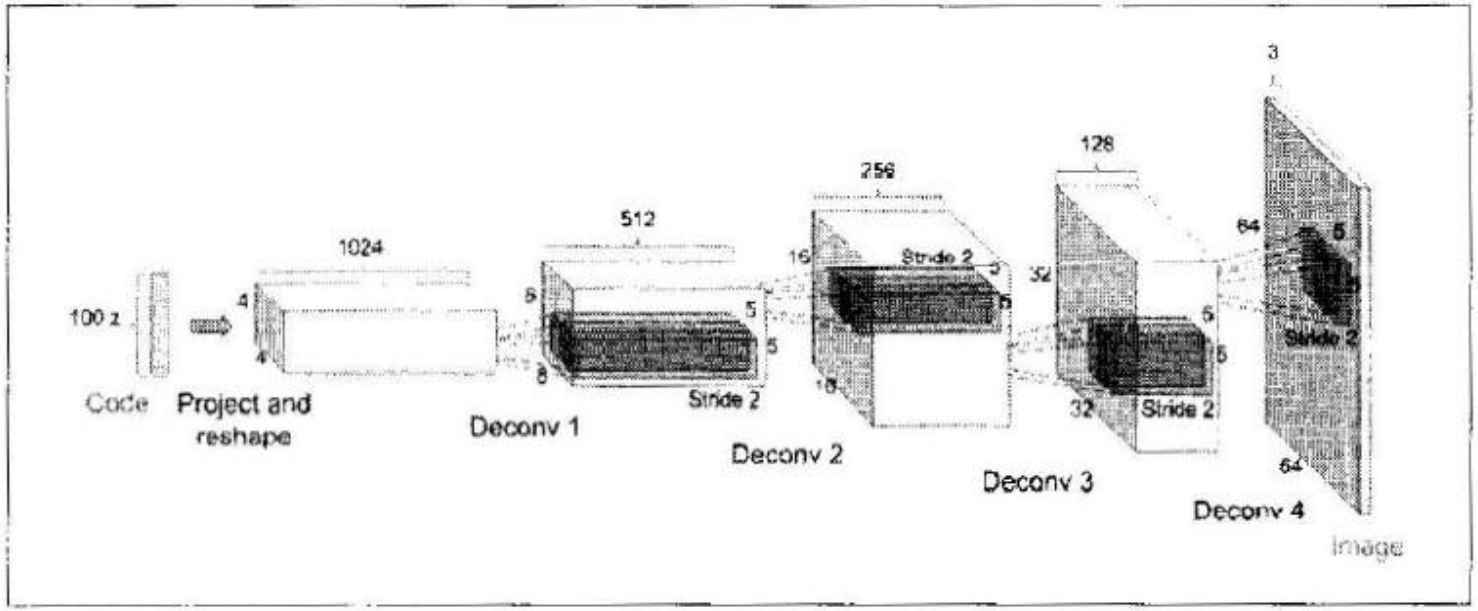


图 6.9　生成模型

对抗模型

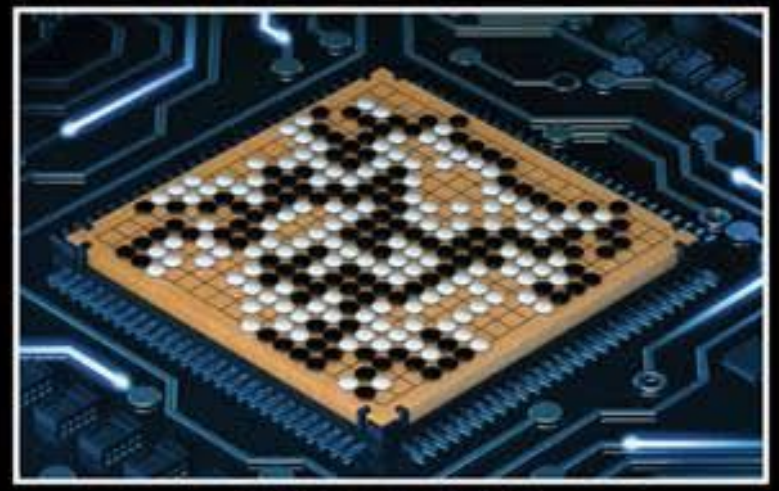
对抗过程简单来说就是一个判断真假的判别器，相当于一个二分类问题，输入一张真的图片希望判别器输出的结果是 1，输入一张假的图片希望判别器输出的结果是 0。这跟原图片的 label 没有关系，不管原图片到底是一个多少类别的图片，它们都统一称为真的图片，输出的 label 是 1，则表示是真实的；而生成图片的 label 是 0，则表示是假的。

在训练的时候，先训练判别器，将假的数据和真的数据都输入给判别模型，这个时候优化这个判别模型，希望它能够正确地判断出真的数据和假的数据，这样就能够得到一个比较好的判别器。

对抗模型

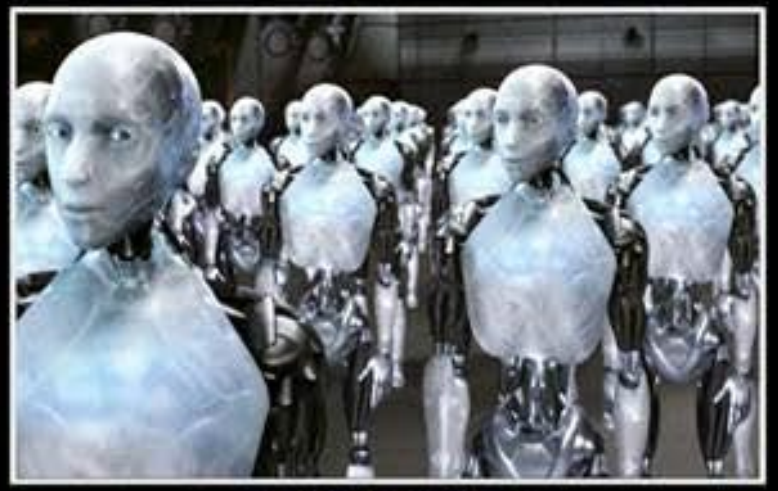
然后开始训练生成器，希望它生成的假的数据能够骗过现在这个比较好的判别器。具体做法就是将判别器的参数固定，通过反向传播优化生成器的参数，希望生成器得到的数据在经过判别器之后得到的结果能尽可能地接近 1，这时只需要调整一下损失函数就可以了，之前在优化判别器的时候损失函数是让假的数据尽可能接近 0，而现在训练生成器的损失函数是让假的数据尽可能接近 1。

这其实就是一个简单的二分类问题，这个问题可以用前面介绍过的很多方法去处理，比如 Logistic 回归、多层感知器、卷积神经网络、循环神经网络等。

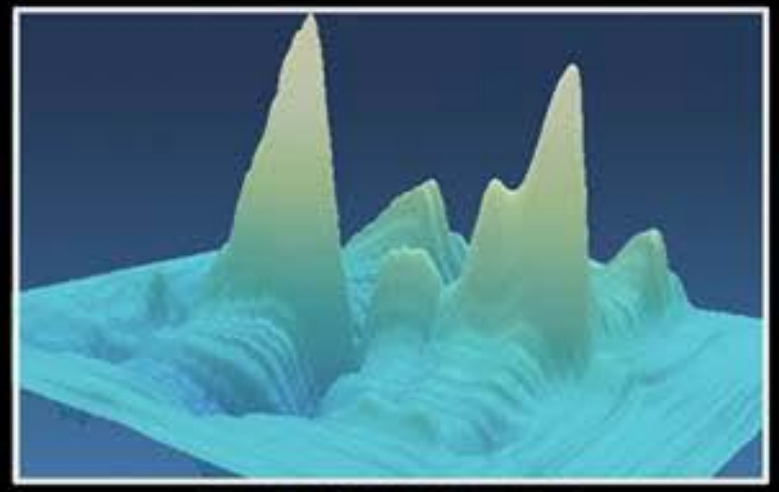


朋友覺得我在

我媽覺得我在



大眾覺得我在



指導教授覺得我在

我以為我在

事實上我在


Thanks!
Obrigado!
Gracias!
谢谢！
Dziękuję!
Danke!
Grazie!
Merci!
شكرا لك !
감사합니다!
Kiitos!