循环神经网络

引入RNN：

引入一个问题：我们不仅可以接收一个序列作为输入，而是还可能期望继续猜测这个序列的后续。此时，序列的相对顺序不能被打乱。

之前卷积神经网络可以有效地处理空间信息，本章的循环神经网络（recurrentneuralnetwork，RNN）则可以更好地处理序列信息。循环神经网络通过引入状态变量存储过去的信息和当前的输入，从而可以确定当前的输出。

为什么要引入RNN：

在比如文本翻译的任务里面，所有的单词的相对顺序，整个序列的顺序是重要的，不能被打乱。而且顺序可以提供很多有用的信息。RNN就是为了处理这种跟序列顺序有很大关系的问题。

8.1序列模型

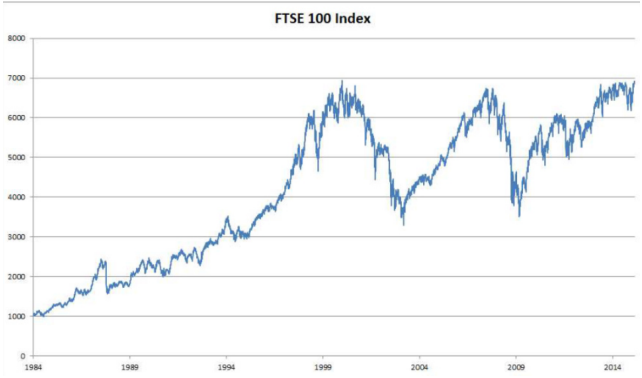
序列数据的解释：按照时间顺序或者某种逻辑顺序排列的数据集合。

- 每个数据点通常依赖于前一个数据点，或者至少与之前的数据点有关。

8.1.1问题引入

引入一个问题 来直观感受序列数据。

比如现在已经有了一个股票的历史价格，现在要求预测时刻t的价格。用 x_t 表示价格，即在时间步（time step） $t \in \mathbb{Z}^+$ 时，观察到的价格 x_t 。



假设一个交易员想在 t 日的股市中表现良好，于是通过以下途径预测 x_t 。

$$x_t \sim P(x_t \mid x_{t-1}, \cdots, x_1). \tag{1}$$

8.1.2自回归模型

- 引入自回归模型：

  为了实现这个预测，交易员可以使用回归模型，比如之前的线性回归。但是有一个问题：**输入数据的数量，输入 $x_{t-1}, \ldots, x_1$ 本身因t而异**。换句话说，输入的数据的大小是会增加的。
  为了处理这个问题，提出了两种策略。

- 自回归模型：

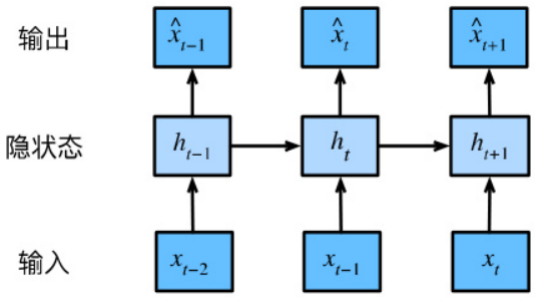
第一种策略，假设在现实情况下相当长的序列 $x_{t-1}, \ldots, x_1$ 可能是不必要的，因此我们只需要满足某个长度为 τ 的时间跨度，即使用观测序列 $[x_{t-1}, \ldots, x_{t-\tau}.]$ 。

当下获得的最直接的好处就是参数的数量总是不变的，至少在 $t > \tau$ 时如此，这就使我们能够训练一个上面提及的深度网络。这种模型被称为自回归模型（autoregressivemodels），因为它们是对自己执行回归。

简单来说，自回归模型在每一次预测的时候，仅仅使用当前在时间维度上最近的τ次数据。

- 隐变量自回归模型：
  第二种策略的解释：

  引入新的定义：h_t表示在t时刻对于**过去观测的总结**；$\hat{x}_t$表示在t时刻的预测结果。



  说明：保留一些对过去观测的总结 h_t ，并且同时更新预测 $\hat{x}_t$ 和总结 h_t 。
  两个公式：$\hat{x}_t = P(x_t \mid h_t)$ ，(基于当前的x_t、h_t预测出来x_t)。$h_t = g(h_{t-1}, x_{t-1})$， （基于某种方式更新$h_t$）。

  因为h_t是一个自己总结的变量，并不是观测直接得到的，所以这种策略叫隐变量自回归模型。

所掌握的数据来预测新的动力学。统计学家称不变的动力学为静止的（stationary）。因此，整个序列的估计值都将通过以下的方式获得：

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_{t-1}, \ldots, x_1). \tag{8.1.2}$$

注意，如果我们处理的是离散的对象（如单词），而不是连续的数字，则上述的考虑仍然有效。唯一的差别是，对于离散的对象，我们需要使用分类器而不是回归模型来估计$P(x_t \mid x_{t-1}, \ldots, x_1)$。

8.1.3马尔可夫模型

- 马尔可夫模型认为：
  一个序列当前的值，只跟最近的τ个序列的值有关。
  比如当τ取值为2时，表示序列里面第四个元素只与第二个和第三个元素有关系。
  而跟第一个元素没有关系。

如果 $\tau = 1$ ，得到一个一阶马尔可夫模型$P(x)$ 由下式给出：

$$P(x_1, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_{t-1}) \text{当} P(x_1 \mid x_0) = P(x_1). \tag{2}$$

当假设 x_t 仅是离散值时，模型表现很好，因为在这种情况下，使用动态规划可以沿着马尔可夫链精确地计算结果。例如，我们可以高效地计算 $P(x_{t+1} \mid x_{t-1})$

$$\begin{aligned} P(x_{t+1} \mid x_{t-1}) &= \frac{\sum_{x_t} P(x_{t+1}, x_t, x_{t-1})}{P(x_{t-1})} \\ &= \frac{\sum_{x_t} P(x_{t+1} \mid x_t, x_{t-1}) P(x_t, x_{t-1})}{P(x_{t-1})} \\ &= \sum_{x_t} P(x_{t+1} \mid x_t) P(x_t \mid x_{t-1}) \end{aligned} \tag{3}$$

利用这一事实，我们只需要考虑过去观察中的一个非常短的历史：
$P(x_{t+1} \mid x_t, x_{t-1}) = P(x_{t+1} \mid x_t)$。

- 上面公式里面：
  $P(x_{t+1}, x_t, x_{t-1})$表示三个数字同时在对应时刻出现的概率。
  公式的推导过程主要用到了条件概率的公式：
  $P(A|B) = \frac{P(AB)}{P(B)}$

> 这部分，目前只是看了公式，并不是非常理解能够干什么，而马尔可夫模型内容深究其实比较多，如果之后遇到了，有实际需要再去研究。
> SASREC的论文里面，只是提到说马尔可夫模型只考虑最近的n个序列元素对此刻要预测的元素的影响。所以可以处理稀疏数据集的问题。

8.1.4因果关系

原则上，将 $P(x_1, \ldots, x_T)$ 倒序展开也没什么问题。因为基于条件概率公式，我们可以写出：
$P(x_1, . . , x_T) = \prod_{t=T}^{1} P(x_t \mid x_{t+1}, . . , x_T)$.

事实上，如果基于一个马尔可夫模型，我们还可以得到一个反向的条件概率分布。但是在许多情况下，数据存在一个自然的方向，即在时间上是前进的。很明显，未来的事件不能影响过去。
因此，如果我们改变 x_t 可能会影响未来发生的事情 x_{t+1} ，但不能反过来。如果我们改变 x_t ，基于过去事件得到的分布不会改变。因此，解释 $P(x_{t+1} \mid x_t)$ 应该比解释 $P(x_t \mid x_{t+1})$ 更容易。例如，在某些情况下，对于某些可加性噪声e，显然我们可以找到 $x_{t+1} = f(x_t) + \epsilon$ ，而反之则不行。

> 记得SASREC里面也有因果关系的模块。
> 之后可以结合书上对于这部分的解释再读一下。

8.1.5复现MLP代码 进行序列模型的训练 和 预测

- 问题：
  输入为[1,1000]的任何一个整数。输出为$y = sin(0.01 * x) + 噪音$
- 模型：
  首先模型研究了tau等于四的情况。
  也就是研究给定$x_{i-4}, x_{i-3}, x_{i-2}, x_{i-1}$对应的结果，来预测$x_i$对应的结果。

  模型使用：
  一个拥有两个全连接层的多层感知机，ReLU激活函数和平方损失。

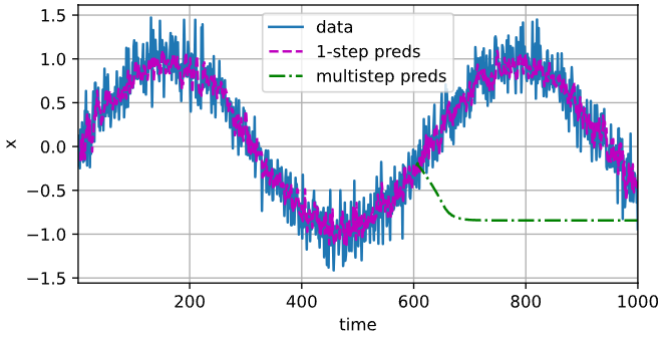
  模型代码：

```
# 初始化网络权重的函数
def init_weights(m):
      if type(m) == nn.Linear:
            nn.init.xavier_uniform_(m.weight)

# 一个简单的多层感知机
def get_net():
      net = nn.Sequential(nn.Linear(4, 10),    #?一个线性层 4个输入 10 个输出
                                    nn.ReLU(), #?relu 激活函数
                                    nn.Linear(10, 1)) #?最后的线性层 10个输入 1个输出
      net.apply(init_weights)
      return net

# 平方损失。注意：MSELoss计算平方误差时不带系数1/2
loss = nn.MSELoss(reduction='none')
```

- 最后的结果：



结果说明：

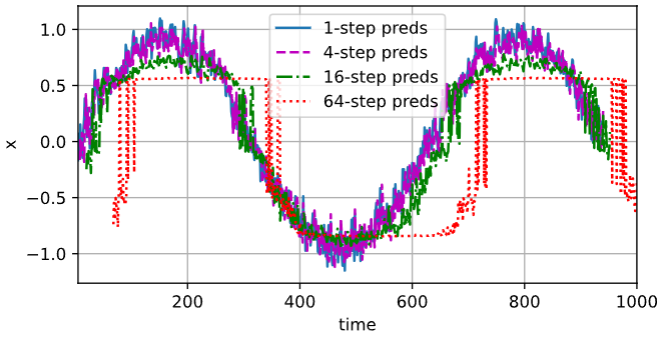
- data：蓝色曲线就是 真实的结果。
- 1-step preds：逻辑是训练出来模型之后，在预测的时候，依然使用真实值。比如在预测x_i的时候，直接使用$x_{i-4}, x_{i-3}, x_{i-2}, x_{i-1}$对应的真实结果进行预测。
- multistep preds：很明显上面的 1-step测试结果，在实际中是不合理的。合理的情况应该是由 1 2 3 4预测出来5，再用2 3 4 5预测出来6。使用的所有数据都应该是预测的结果。

$$\hat{x}_{605} = f(x_{601}, x_{602}, x_{603}, x_{604}),$$
$$\hat{x}_{606} = f(x_{602}, x_{603}, x_{604}, \hat{x}_{605}),$$
$$\hat{x}_{607} = f(x_{603}, x_{604}, \hat{x}_{605}, \hat{x}_{606}),$$
$$\hat{x}_{608} = f(x_{604}, \hat{x}_{605}, \hat{x}_{606}, \hat{x}_{607}),$$
$$\hat{x}_{609} = f(\hat{x}_{605}, \hat{x}_{606}, \hat{x}_{607}, \hat{x}_{608}),$$

multistep preds就是这样的过程。
可以看到在600步之后的表现很糟糕。

- 糟糕的原因：
假设在步骤1之后，我们积累了一些错误$\epsilon_1 = \bar{\epsilon}$。于是，步骤2的输入被扰动了$\epsilon_1$，结果积累的误差是依照次序的$\epsilon_2 = \bar{\epsilon} + c\epsilon_1$，其中$c$为某个常数，后面的预测误差依此类推。因此误差可能会相当快地偏离真实的观测结果。

- 上面代码里面tau等于4，也复现了tau为16 64的情况：
文章中的这里使用的方式也是在训练出来模型之后，预测的时候直接根据需要的y的真实值进行预测。



从结果来看 k > 4 之后的表现就没有很好了。

8.2文本预处理

文本预处理的过程、词元话的过程，进行了复现。

- 为什么需要进行预处理：
  在处理文章的时候，首先需要将文章里面的所有单词都提取出来。
  其次，因为模型只能处理数字，所以要建立单词和数字之间的映射。
- 映射的建立：
  根据每一个单词在文章里面出现过的频率，次数高的索引小，次数小的索引大。
  - 我们实现了一个Vocab类。
    定义变量的时候，传进去列表，列表里面存储所有的单词列表（次数没有限制）。
    就得到了映射关系。
    举例来说：定义：`vocab = Vocab(list)`
    就可以使用`vocab['the']`得到the这个单词对应的索引。
    使用`vocab.to_tokens(1)`得到索引为1的单词是什么。
  - 具体的代码细节，在上面的链接里面有解释。
- 文本是序列数据的一种最常见的形式之一。·为了对文本进行预处理，我们通常将文本拆分为词元，构建词表将词元字符串映射为数字索引I，并将文本数据转换为词元索引以供模型操作。

8.3语言模型和数据集

- 语言模型的目标：
  估计序列的联合概率：$P(x_1, x_2, \ldots, x_T)$.
- 只需要每一次预测出来 $x_t \sim P(x_t \mid x_{t-1}, \ldots, x_1)$ ，一个理想的语言模型就能够基于模型本身生成自然文本。
  目前，距离设计出这样的系统还很遥远，因为它需要“理解”文本，而不仅仅是生成语法合理的内容。

8.3.1学习语言模型

- 我们面临的首要问题是**如何对一个文档、词元序列**进行建模。
  上述中我们已经分析过：$P(x_1, x_2, \ldots, x_T) = \prod_{t=1}^{T} P(x_t \mid x_1, \ldots, x_{t-1})$..

  比如：$P(deep, learning, is, fun) = P(deep)P(learning|deep)P(is|deep, learning)P(funIdeep, learning, is)$.

  在这样的公式里面，为了计算最后的概率，我们就需要计算上面每一项的条件概率。
- 假如我们使用$\hat{P}(\text{letting} \mid \text{deep}) = \frac{n(\text{deep,letting})}{n(\text{deep})}$进行估计。其中n(x,y)表示出现x,y连续对的次数。
  但是连续单词的出现频率非常小。所以上面的计算结果也会很小。
  而且如果用三个或者三个以上的单词组合，出现的次数会更小。
  - 常见的解决这个方案的策略：采用拉普拉斯平滑。
    在所有的计数中添加一个小常量。

$$\hat{P}(x) = \frac{n(x) + \epsilon_1/m}{n + \epsilon_1},$$

$$\hat{P}(x' \mid x) = \frac{n(x, x') + \epsilon_2 \hat{P}(x')}{n(x) + \epsilon_2},$$

$$\hat{P}(x'' \mid x, x') = \frac{n(x, x', x'') + \epsilon_3 \hat{P}(x'')}{n(x, x') + \epsilon_3}.$$

    在上面公式里面，n表示训练集里面 n 表示训练集中的单词总数， m 表示唯一单词的数量。其中， ϵ_1, ϵ_2 和 ϵ_3 是超参数。以 ϵ_1 为例：当 $\epsilon_1 = 0$ 时，不应用平滑；当 ϵ_1 接近正无穷大时， $\hat{P}(x)$ 接近均匀概率分布 $1/m$。

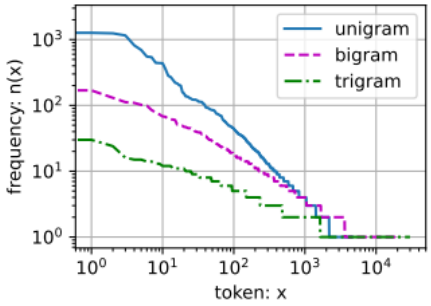
- 上述模型很容易无效
  原因：首先，我们需要存储所有的计数；其次，这完全忽略了单词的意思。例如，“猫”（cat）和“猫科动物”（feline）可能出现在相关的上下文中，但是想根据上下文调整这类模型其实是相当困难的。最后，长单词序列大部分是没出现过的，因此一个模型如果只是简单地统计先前“看到”的单词序列频率，那么模型面对这种问题肯定是表现不佳的。

8.3.2马尔可夫模型与 n 元语法

- 这里了解一下马尔可夫模型的讨论，应用于语言建模。如果
  $P(x_{t+1} \mid x_t, \ldots, x_1) = P(x_{t+1} \mid x_t)$ ，则序列上的分布满足一阶马尔可关性质。
- 阶数越高，对应的依赖关系就越长。这种性质推导出了许多可以应用于序列建模的近似公式：
  $$P(x_1, x_2, x_3, x_4) = P(x_1)P(x_2)P(x_3)P(x_4),$$
  $$P(x_1, x_2, x_3, x_4) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_2)P(x_4 \mid x_3),$$
  $$P(x_1, x_2, x_3, x_4) = P(x_1)P(x_2 \mid x_1)P(x_3 \mid x_1, x_2)P(x_4 \mid x_2, x_3)$$
- 涉及一个、两个和三个变量的概率公式分别被称为一元语法（unigram）、二元语法（bigram）和三元语法（trigram）模型。

8.3.3自然语言统计

对于`时光机器数据集`，我们统计了它里面所有的单个单词的出现频率、二元语法、三元语法的出现频率(具体代码在ipynb里面)。统计之后绘制图形发现：



其中蓝色为一元，紫色二元，绿色三元。

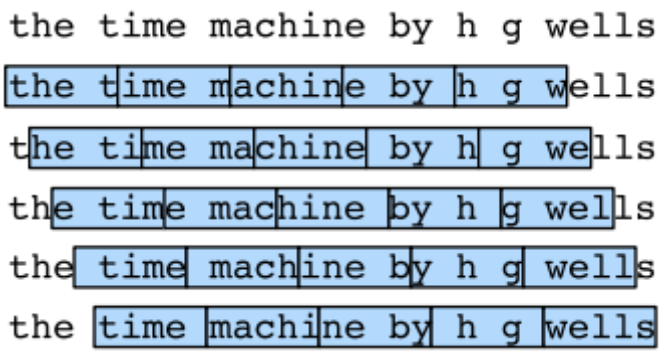
- 大致遵循**双对数**坐标图上的一条直线。
  这意味频率满足齐普夫定律，即第i个最常用单词的频率n_i为： $n_i \propto \frac{1}{i^\alpha}$,等价于
  $\log n_i = -\alpha \log i + c$,.其中 α 是刻画分布的指数，c 是常数。**这告诉我们想要通过计数统计和平滑来建模单词是不可行的，因为这样建模的结果会大大高估尾部单词的频率，也就是所谓的不常用单词。**
- 发现：
  - 词表中 n 元组的数量并没有那么大，这说明语言中存在相当多的结构，这些结构给了我们应用模型的希望；
  - 很多 n 元组很少出现，这使得拉普拉斯平滑非常不适合语言建模。

> - 通过上面的一些讨论，以及对数据的处理。
>   最终我们还是否定掉了提出来的在基于概率的公式上添加拉普拉斯平滑。
>   接下来我们将使用深度学习模型来处理这个问题。

8.3.4读取长序列数据

- 现在的问题：
  - 因为序列数据本质上是连续的，比如一篇文章。
    所以我们将原始的序列进行拆分，每一个被拆分的结果是连续的。
    这样的结果方便模型读取。
  - 现在我们的策略：
    假设我们将使用神经网络来训练语言模型，模型中的网络一次处理具有预定义长度（例如 n 个时间步）的一个小批量序列。

    如何随机生成一个小批量数据的特征和标签以供读取?
- 下图画出来了，在时间步数为5的时候，从原始序列划分出来子序列的一些情况：



  解释：时间步为5的意思是对于当前要预测的位置，我们考虑前面5个元素都是什么。在这个图片里面，每一个时间步都固定了里面的长度是5，而不是固定一定是1个单词。

  另外，起始位置(也就是偏移量)是不同的，我们可以从随机偏移量开始划分序列，以同时获得 `覆盖性` 和 `随机性` 。
- 接下来我们讨论一下 `随机采样` 和 `顺序分区` 。

8.3.4.1随机采样的实现

对于语言建模，目标是基于到目前为止我们看到的词元来预测下一个词元，因此标签是**移位了一个词元的原始序列。**

因为这个实现比较简单，这里直接说明随机采样的结果。

- 数据：
  原始序列直接使用(0到34)。规定时间步数为5。
  所以一共可以生成 $\lfloor (35-1)/5 \rfloor = 6$ “特征一标签”子序列对。
  如果设置小批量大小为2，我们只能得到3个小批量。
  也就是说一个批量里面会有两个"特征-标签"子序列对。
- 随机采样的结果：

```
X:  tensor([[14, 15, 16, 17, 18],
        [29, 30, 31, 32, 33]])
Y: tensor([[15, 16, 17, 18, 19],
        [30, 31, 32, 33, 34]])
X:  tensor([[19, 20, 21, 22, 23],
        [ 9, 10, 11, 12, 13]])
Y: tensor([[20, 21, 22, 23, 24],
        [10, 11, 12, 13, 14]])
X:  tensor([[ 4,  5,  6,  7,  8],
        [24, 25, 26, 27, 28]])
Y: tensor([[ 5,  6,  7,  8,  9],
        [25, 26, 27, 28, 29]])
```

  一共有三对XY，也就是三个小批量。
  每一个X里面是两个特征，每一个Y里面是两个标签。上面有说过标签就是移位一个词元的原始序列。

8.3.4.2顺序采样的实现

- 顺序分区就是直接将原始序列按照顺序划分一个个"特征-标签"对。
  然后按照顺序放到最后的返回结果里面。
  代码实现也比较简单。
- 顺序分区的结果：

```
X:  tensor([[ 1,  2,  3,  4,  5],
        [17, 18, 19, 20, 21]])
Y: tensor([[ 2,  3,  4,  5,  6],
        [18, 19, 20, 21, 22]])
X:  tensor([[ 6,  7,  8,  9, 10],
        [22, 23, 24, 25, 26]])
Y: tensor([[ 7,  8,  9, 10, 11],
        [23, 24, 25, 26, 27]])
X:  tensor([[11, 12, 13, 14, 15],
        [27, 28, 29, 30, 31]])
Y: tensor([[12, 13, 14, 15, 16],
        [28, 29, 30, 31, 32]])
```

  两个相邻小批量中的子序列，在原始序列上也是相邻的。

8.3.4.3

- `顺序分区` 和 `随机采样` 唯一的区别就在于：
  在数据划分之后，两个相邻的小批量中的子序列在原始序列上是否也相邻。
- 上述的代码，书籍作者将两个整合到一个类里面。
  这里的代码在后面的章节中会用到。特别是注意力机制。

8.4循环神经网络(在公式层面理解RNN)

隐藏层和隐状态说明：

- 隐藏层是神经网络里面，除了输入、输出之外的其他层次。
- 隐状态是专门在rnn里面设计出来的一种新的变量。

两者没有任何关系。

8.4.1之前的神经网络(没有隐状态H)

首先回顾一下只有单隐藏层的多层感知机。

- 从输入层→隐藏层：
  设隐藏层的激活函数为 $\boldsymbol{\phi}$，给定一个小批量样本 $\mathbf{X} \in \mathbb{R}^{n \times d}$，其中批量大小为$n$，输入维度为 d，则隐藏层的输出 $\mathbf{H} \in \mathbb{R}^{n \times h}$ 通过下式计算：$\mathbf{H} = \phi(\mathbf{X}\mathbf{W}_{xh} + \mathbf{b}_h)$。
  解释：隐藏层权重参数为 $\mathbf{W}_{xh} \in \mathbb{R}^{d \times h}$，偏置参数为 $\mathbf{b}_h \in \mathbb{R}^{1 \times h}$，以及隐藏层次神经元的数目为 h。
- 从隐藏层→输出层：
  将隐藏变量 $\mathbf{H}$ 用作输出层的输入。输出层的结果通过$\mathbf{O} = \mathbf{H}\mathbf{W}_{hq} + \mathbf{b}_q$，计算。

  解释：$\mathbf{O} \in \mathbb{R}^{n \times q}$ 是输出变量，$\mathbf{W}_{hq} \in \mathbb{R}^{h \times q}$ 是权重参数，$\mathbf{b}_q \in \mathbb{R}^{1 \times q}$ 是输出层的偏置参数。

8.4.2有隐状态的循环神经网络

- 假设我们在时间步t有小批量输入 $\mathbf{X}_t \in \mathbb{R}^{n \times d}$ 。
  解释：在每一个batch里面，有n个序列样本；时间步是t，意思是是序列里面的第t个位置、第t个时刻。d表示的是k步预测中k的大小，如果d = 5，就表示当前的预测是通过前面最近的5个元素预测得到的。
  t是当前的时间步，d是整个预测里面的时间步数。
  - 时间步和时间步数：
    时间步：当前的样本在整个数据里面的位置。
    时间步数：k步预测里面的k。也就是在预测当前位置的输出的时候，我们会用到多少个之前的元素。

  用 $\mathbf{H}_t \in \mathbb{R}^{n \times h}$ 表示时间步t的隐藏变量。
  解释：n还是表示每一个batch里面有n个样本。h是隐变量H本身的维度。对h解释：n只是为了加速运算才设置的一个维度，如果每一个batch_size都是1，隐变量只是一个向量，h就是这个向量的维度。
  - 隐变量的计算：
    在RNN里面，我们还需要保存前一个时间步的隐藏变量 $\mathbf{H}_{t-1}$ ，引入新的权重参数 $\mathbf{W}_{hh} \in \mathbb{R}^{h \times h}$ ，这个参数的作用是表示如何在当前时间步中使用前一个时间步的隐藏变量。

    参数都说明完毕之后，可以引出公式：当前时间步隐藏变量由当前时间步的输入与前一个时间步的隐藏变量一起计算得出

    $$\mathbf{H}_t = \phi(\mathbf{X}_t \mathbf{W}_{xh} + \mathbf{H}_{t-1} \mathbf{W}_{hh} + \mathbf{b}_h). \tag{4}$$

  - 计算出来了当前时间步的隐变量之后，就可以计算当前位置的输出：

    $$\mathbf{O}_t = \mathbf{H}_t \mathbf{W}_{hq} + \mathbf{b}_q. \tag{5}$$

    t表示时间步是t，也就是现在要预测的是第t个位置、时刻的元素。(公式里面的参数的解释 和上面MLP里面的公式一致，这里不再解释)。

    值得一提的是，即使在不同的时间步，循环神经网络也总是使用这些模型参数。因此，循环神经网络的参数开销不会随着时间步的增加而增加。

    上面的W_{xh}, W_{hh}, W_{Hq}整个模型里面都是不变的。
- 隐变量的计算相比上文的MLP里面输出层的输出，添加了$\mathbf{H}_{t-1} \mathbf{W}_{hh}$ 。
  这一项的实际意义是表示相邻时间步的隐藏变量之间的关系。
- 对于隐变量和RNN理解方面的解释：

  隐变量捕获并保留了序列直到其当前时间步的历史信息，就如当前时间步下神经网络的状态或记忆，因此这样的隐藏变量被称为隐状态。
  因为在当前时间步中，隐状态使用的定义与前一个时间步中使用的定义相同。
  所以隐变量每一次的计算和之前都是一样的，所以这个计算是循环的。

  所以基于循环计算的隐状态神经网络被命名为循环神经网络。
  而执行这样循环计算的层次叫做循环层。
- 画图解释整个过程：

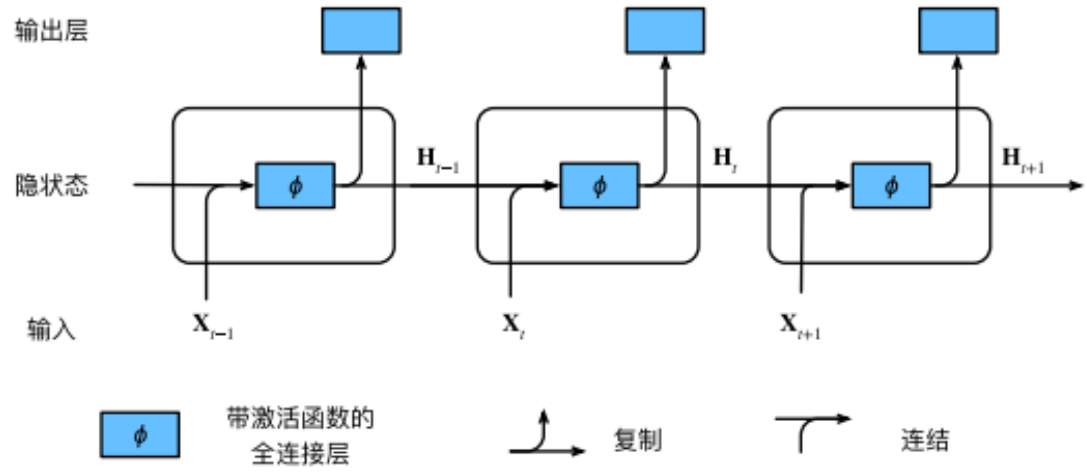
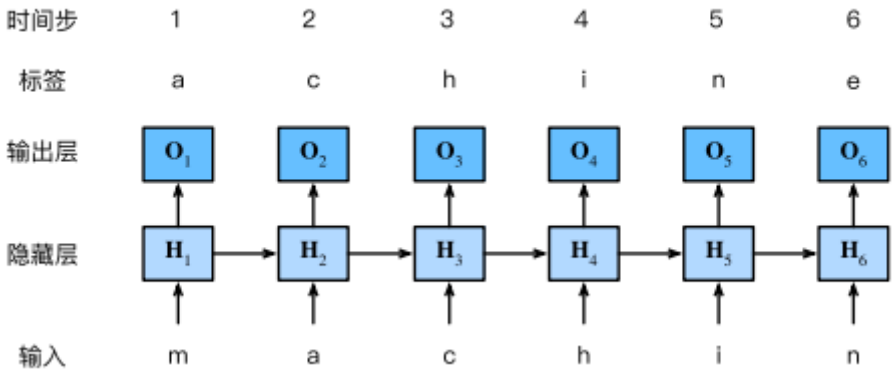


图8.4.1: 具有隐状态的循环神经网络

1.拼接当前时间步t的输入 $\mathbf{X}_t$ 和前一时间步 $t-1$ 的隐状态$\mathbf{H}_{t-1}$计算出来当前的隐状态 $\mathbf{H}_t$.
2.通过计算出来的当前隐状态，送给全连接层，计算出来当前的输出$\mathbf{O}_t$.

8.4.3字符级语言模型

- 上面我们给的例子里面，文本是被划分为了一个有一个单词。Vocac中也是 单词 和 索引 的映射。
  这里介绍**基于字符级语言模型的RNN**，使用 当前前面的字符 来预测当前应该输出哪个字符。
- 结构：(图片里面输入是"machin"，标签是"achine")



- 最后判断输出应该是哪个字母的实现：
  字母一共只有26种。（在下一节的 从零实现RNN 因为有空格和knd两个符号，所以一共只有28种）
  RNN最后的输出层输出出来28种字母的分布情况，使用softmax操作，选择概率最大的即可。

8.4.4困惑度

> 困惑度 是之后用来衡量RNN模型的一个很重要的标准。
> 具体实现在8.5节中也有代码。

- 如何度量语言模型的质量，使用**困惑度**，也是后续部分里面用来评估RNN模型的重要标准。
- **这里需要信息论的知识。**
  如果想要压缩文本，我们可以根据当前词元集预测的下一个词元。
  一个更好的语言模型应该能让我们更准确地预测下一个词元。因此，它应该允许我们在压缩序列时花费更少的比特。所以我们可以通过一个序列中所有的 n 个词元的交叉熵损失的平均值来衡量:
  $\frac{1}{n}\sum_{t=1}^{n}-\log P(x_t \mid x_{t-1},\cdots,x_1)$,
- 其中 P 由语言模型给出， x_t 是在时间步 t 从该序列中观察到的实际词元。这使得不同长度的文档的性能具有了可比性。因为历史原因，自然语言处理的科学家更喜欢使用一个叫做困惑度。

$\exp\left(-\frac{1}{n}\sum_{t=1}^{n}\log P(x_t \mid x_{t-1}, \ldots, x_1)\right)$.

- 困惑度的最好的理解是“下一个词元的实际选择数的调和平均数”。

困惑度的最好的理解是“下一个词元的实际选择数的调和平均数”。我们看看一些案例。

- 在最好的情况下，模型总是完美地估计标签词元的概率为1。在这种情况下，模型的困惑度为1。
- 在最坏的情况下，模型总是预测标签词元的概率为0。在这种情况下，困惑度是正无穷大。
- 在基线上，该模型的预测是词表的所有可用词元上的均匀分布。在这种情况下，困惑度等于词表中唯一词元的数量。事实上，如果我们在没有任何压缩的情况下存储序列，这将是我们能做的最好的编码方式。因此，这种方式提供了一个重要的上限，而任何实际模型都必须超越这个上限。

在接下来的小节中，我们将基于循环神经网络实现字符级语言模型，并使用困惑度来评估这样的模型。

- 在接下来的小节中，我们将基于循环神经网络实现字符级语言模型，并使用困惑度来评估这样的模型。

8.5 从零实现循环神经网络

这部分复现了作者的代码：链接。
具体的细节都在链接里面，这里只记录一些比较关键的点。

1.变量batch_size，num_steps分别作为批量大小(一个批量里面处理的样本个数)，时间步数(预测时使用多少个前面的信息)

2.vocab词表使用**字符级别**。除了a~z还有<unk>和空格两种字符。

3.每一个字符会首先通过vocab转换为对应的索引，再通过索引转换为**one_hot_vector**。

4.每一个小批量的数据是一个**二维张量**，(批量大小，时间步数)。首先我们进行转置，之后使用上面的独热编码，得到最后的数据是一个**三维张量**，(时间步数，批量大小，词表大小)。

5.整个模型的输入维度和输出维度是一样的，num_inputs = num_outputs = vocab_size。

6.变量num_hiddens表示隐状态的维度。这个变量也是**隐藏层的神经单元数目**。

7.代码里面首先使用了顺序采样处理出来train_iter，其次又用了随机采样。顺序采样的困惑度会略低。

8.使用了**梯度裁剪**，使用原因、方式可见下文。

9.提及一句，这一章节，我们的所有实现，过程都是单隐藏层。整个神经网络的结构就是输入层、单隐藏层、输出层。在之后的章节中，会讨论多层RNN的意义。

10.其他具体的实现细节这里不在赘述，可见实现代码。

8.5.1梯度裁剪

- 为什么RNN里面需要梯度裁剪：
  因为对于长度为N的序列，迭代计算T个时间步上面的梯度，将会在反向传播的过程中产生长度为$O(T)$的矩阵惩罚链。
  问题：如果T比较大，就会导致数值很不稳定，可能导致 梯度爆炸 或者 梯度消失 。
  因此RNN模型往往都会采取其他方式来支持稳定的训练。
- > 在书籍321页，这里讲解了有关梯度裁剪的一些知识。
  > 因为第一次接触，并没有完全看懂。仅仅记录一下我们下面的rnn里面要用到的一种梯度裁剪方式。

- 原先的更新：
  使用 $\eta > 0$ 作为学习率时，在一次迭代中，我们将 $\mathbf{X}$ 更新为 $\mathbf{x} - \eta \mathbf{g}$。
  现在我们采用新的方案：
  通过将梯度g投影回给定半径（例如 θ ）的球来裁剪梯度 $\mathbf{g}$ 。

  公式：

$$\mathbf{g} \leftarrow \min\left(1, \frac{\theta}{\|\mathbf{g}\|}\right)\mathbf{g}. \tag{6}$$

- 这样做之后梯度范数永远不会超过θ，并且更新后的梯度完全与g的原始方向对齐。
  它还有一个副作用，即**限制任何给定的小批量数据**对参数向量的影响，这赋予了模型一定程度的稳定性。
  梯度裁剪提供了一个快速修复梯度爆炸的方法，虽然它并不能完全解决问题，但它是众多有效的技术之一。
- 这里的梯度裁剪的代码都在grad_clipping函数里面。

8.6总结

- 首先说明了，什么是序列数据。为什么处理序列数据需要新的模型。
- 8.1.2介绍了处理序列模型问题。介绍了自回归模型和隐变量自回归模型。
  也介绍了马尔可夫模型。
- 讲解了词表是如何得到的。
  如何**获取长序列数据问题**。随机采样和顺序分区。
  也说明了马尔可夫模型不适用的原因。
- 讲解了RNN的公式，熟悉了MLP的公式。
  掌握整个RNN的推导流程。
- 进行了从零开始的代码实现。
- 还讲解了梯度裁剪。
- 在书籍里面 最后还有两节。
  分别讲述了RNN的简洁实现和如何通过时间进行反向传播。