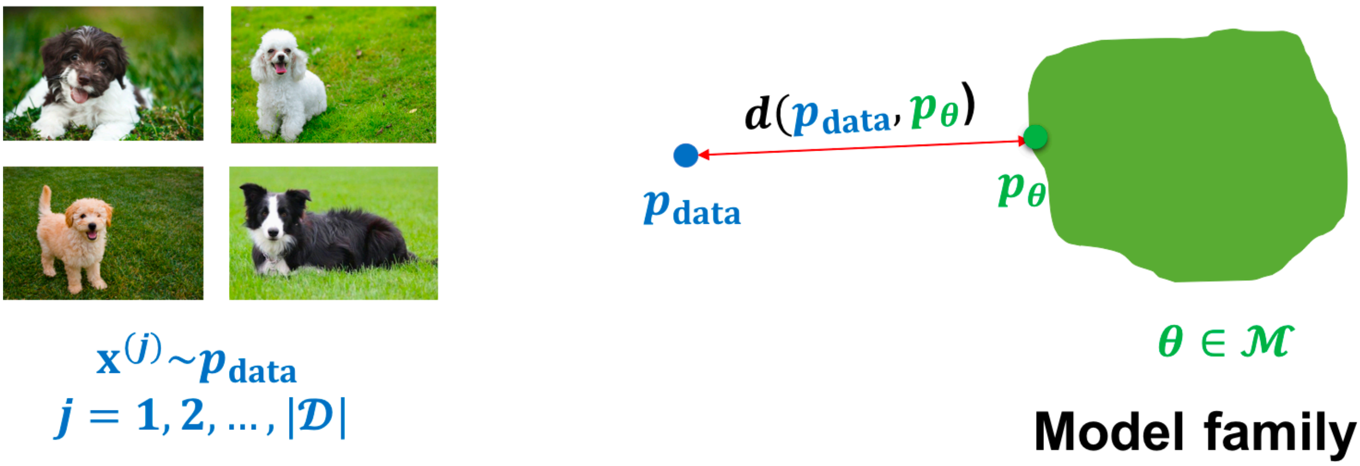
Lec03 - Autoregressive Models

如何训练一个生成模型?

1. 给定一批训练集（例如一批小狗图片）
2. 我们需要利用这批图像学习一个概率分布$p(x)$，其具有以下性质
    - **可生成性**：如果我们从这个概率分布中采样$x_{new} \sim p(x)$，那么这个样本x_{new}应该看起来像条狗。
    - **概率评估**：如果一个输入x看起来像条狗，那么$p(x)$应该比较大，反之亦然。
    - **无监督表示学习**：我们应该能够学习到这些图像的共同之处是什么（例如耳朵、尾巴...）
3. 两个问题：**如何表示**$p(x)$、**如何学习**$p(x)$

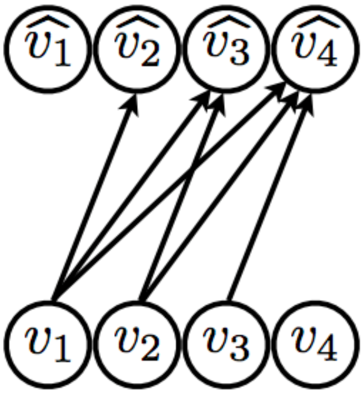


以MNIST训练集为例

1. 给定一批数据集$\mathcal{D}$表示手写数字。

2. 每个图片有28×28个像素点，每个像素点都是一个伯努利变量(0表示black，1表示white)。
3. 我们想要利用像素$x \in \{0,1\}^{784}$学习一个概率分布函数$p(x) = p(x_1, \ldots, x_{784})$，使得对于从$p(x)$中抽取出来的样本$x$，它看起来像个数字。
4. **思路**：定义一个模型族(Model Family)$\{p_\theta(x)\,,\,\theta \in \Theta\}$，然后从中间挑选出来一个比较好的适配训练集$\mathcal{D}$的模型。
5. **问题：如何将**$p_\theta(x)$**参数化?**

全可视Sigmoid信念网络(FVSBN)



FVSBN

采用光栅扫描的方法从左上到右下把像素变成一批有顺序的变量$x_1,...,x_{784}$

利用链式法则，先不考虑条件独立的情况

$$p(x_1,...,x_{784})=p(x_1)\cdot p(x_2|x_1)\cdot p(x_3|x_1,x_2)\cdots p(x_n|x_1,...,x_{n-1})$$

但是这显然太过于复杂，因此我们假设：

$$p(x_1,...,x_{784})=p_{CPT}(x_1;\alpha^1)\cdot p_{logit}(x_2|x_1;\boldsymbol{\alpha}^2)\cdot p_{logit}(x_3|x_1,x_2;\boldsymbol{\alpha}^3)\cdots p_{logit}(x_n|x_1,...,x_{n-1};\boldsymbol{\alpha}^n)$$

其中p_{CPT}代表你可与你单独储存这一个参数，$\boldsymbol{\alpha}^n$表示一个含n个参数的向量。

我们继续分析上式：

- $p_{CPT}(X_1=1;\alpha^1)=\alpha^1, p(X_1=0)=1-\alpha^1$这是第一个像素的先验分布，也就是一个单一的数值。
- $p_{logit}(X_n=1|x_1,...,x_{n-1};\boldsymbol{\alpha}^n)=\sigma(\boldsymbol{\alpha}_0^n+\boldsymbol{\alpha}_1^n x_1+\cdots+\boldsymbol{\alpha}_{n-1}^n x_{n-1})=\sigma(\boldsymbol{\alpha}_0^n+\sum_{i=1}^{n-1}\boldsymbol{\alpha}_i^n x_i)=\hat{x}_n$

这是一种模型假设。我们使用逻辑回归来基于前面的像素预测下一个像素对应的概率。这被称为**自回归模型**。

如何计算$p(x_1,...,v_{784})$？将所有的条件概率系数都相乘就可以了。

$$p(X_1=0,X_2=1,X_3=1,X_4=0)=(1-\hat{x}_1)\times\hat{x}_2\times\hat{x}_3\times(1-\hat{x}_4)$$
$$=(1-\hat{x}_1)\times\hat{x}_2(X_1=0)\times\hat{x}_3(X_1=0,X_2=1)\times(1-\hat{x}_4(X_1=0,X_2=1,X_3=1))$$

如何从$p(x_1,...,v_{784})$中采样？

- 从$p(x_1)$中采样$\bar{x}_1\sim p(x_1)$ `np.random.choice([1,0],p=[x_1,1-x_1])`
- 从$p(x_2)$中采样$\bar{x}_2\sim p(x_2|x_1=\bar{x}_1)$
- 从$p(x_3)$中采样$\bar{x}_3\sim p(x_3|x_1=\bar{x}_1,x_2=\bar{x}_2)$

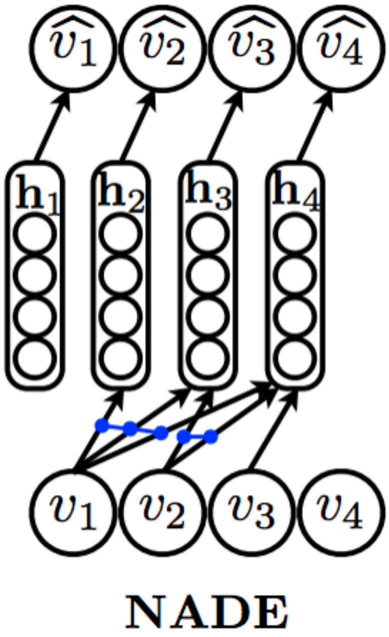
参数总数：$1+2+3+\cdots+n\approx\frac{n^2}{2}$

特点：

1. **参数数量**：$O(n^2)$，随着变量数量平方增长。
2. **生成方式**：顺序采样，每一步依赖前序变量。
3. **局限性**：表达能力有限（仅线性组合），适合简单数据（如二值图像）。

神经网络自回归密度估计(NADE)

改进上述模型



- 不使用单纯的逻辑回归，而是在中间添加一个神经网络层，即：

$$\boldsymbol{h}_i = \sigma(A_i \mathbf{x}_{<i} + \mathbf{c}_i)$$

$$\hat{\mathbf{x}}_i = p(x_i | x_1, \ldots, x_{i-1}; A_i, \mathbf{c}_i, \boldsymbol{\alpha}_i, b_i) = \sigma(\boldsymbol{\alpha}_i \mathbf{h}_i + b_i)$$

注：此处的神经网络层$\mathbf{h}_i$并不一定要加一个σ函数，这么做的原因只是为了加入一些非线性元素。

- 这里的$\mathbf{h}_i$是一个隐藏层，其输入是一个权重矩阵A_i，两个向量$\mathbf{x}_{<i}, \mathbf{c}_i$

$$\mathbf{h}_3 = \sigma\left(\begin{bmatrix} \vdots & \vdots \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} \vdots \end{bmatrix}\right)$$

- **共享权重矩阵以减少参数数量。**也就是说这里的W是一个大的参数矩阵，每一个$\mathbf{h}_i$利用的参数都是这个大矩阵的一部分。随着i的增大，最终使用整个参数矩阵。同时所有的偏置向量$\mathbf{c}$都相同。

$$\mathbf{h}_i = \sigma(W_{\cdot,<i} \mathbf{x}_{<i} + \mathbf{c})$$

$$\mathbf{h}_2 = \sigma\left(\begin{bmatrix} \vdots \\ w_1 \\ \vdots \end{bmatrix} x_1 + \mathbf{c}\right)$$

$$\mathbf{h}_3 = \sigma\left(\begin{bmatrix} \vdots & \vdots \\ w_1 & w_2 \\ \vdots & \vdots \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \mathbf{c}\right)$$

$$\mathbf{h}_4 = \sigma\left(\begin{bmatrix} \vdots & \vdots & \vdots \\ w_1 & w_2 & w_3 \\ \vdots & \vdots & \vdots \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \mathbf{c}\right)$$

- 参数总数：$W \in \mathcal{R}^{d \times n}$并且$\boldsymbol{\alpha}_i, b_i \in \mathcal{R}^{d+1}$，因此参数总量为$O(nd)$

- **优势**：
  - 参数数量降至$o(nd)$（d为隐藏层维度），计算效率更高。
  - 支持连续和离散变量（通过softmax或高斯混合）。

一般离散分布

如果我想建模**彩色图像**，就不能使用简单的二进制变量了。因为此时这个离散变量有多个不同的取值，例如RGB中一个通道的取值范围。

- **Solution**：把二分类变成多分类。假设变量$X_i \in \{1, \ldots, K\}$，则

$$\mathbf{h}_i = \sigma(W_{\cdot,<i}\mathbf{x}_{<i} + \mathbf{c})$$

$$p(x_i|x_1, \ldots, x_{i-1}) = Cat(p_i^1, \ldots, p_i^K)$$

$$\hat{\mathbf{x}}_i = \left(p_i^1, \ldots, p_i^K\right) = softmax\left(X_i\mathbf{h}_i + \mathbf{b}_i\right)$$

其中$softmax$函数是$sigmoid$函数的高维形式，它可以把一个含有K个元素的向量转变成一个含有K个概率值的向量。

$$softmax(\mathbf{a}) = softmax(a^1, \ldots, a^K) = \left(\frac{e^{a^1}}{\sum_i e^{a^i}}, \ldots, \frac{e^{a^K}}{\sum_i e^{a^i}}\right)$$

RNADE

如果我的输入是一批**连续变量**，就不能使用上述方法了。因为上述方法都是基于**离散变量输入**的情况。

- **Solution**：让$\hat{\mathbf{x}}_i$参数化连续概率分布。例如把K个高斯分布混合起来，即

$$p(x_i|x_1, \ldots, x_{i-1}) = \sum_{j=1}^{K} \frac{1}{K}\mathcal{N}(x_i; \mu_i^j, \sigma_i^j)$$

$$\mathbf{h}_i = \sigma(W_{\cdot,<i}\mathbf{x}_{<i} + \mathbf{c})$$

$$\hat{\mathbf{x}}_i = (\mu_i^1, \ldots, \mu_i^K, \sigma_i^1, \ldots, \sigma_i^K) = f(\mathbf{h}_i)$$

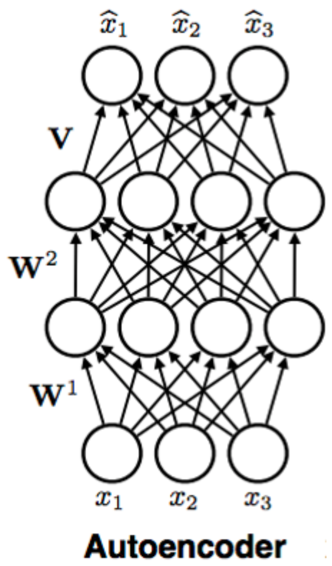
- $\hat{\mathbf{x}}_i$定义了每一个高斯分布$\mathcal{N}(\mu^j, \sigma^j)$的均值和方差，我们可以用exp函数来确保其非负。

自回归模型 vs. 自编码器

1. **核心目标**：
   - **自回归模型**
     - **目标**：显式建模数据的联合概率分布$p(x)$，通过链式法则分解为条件概率的乘积。
     - **用途**：用于直接生成新数据（通过顺序采样）和密度估计（计算样本的似然概率）。
   - **自编码器**
     - **目标**：学习数据的低维潜在表示（编码），然后通过解码器重构输入数据。

- **用途**：主要用于特征学习、降维或去噪，而非显式建模概率分布。传统自编码器无法直接生成新数据。

2. **生成能力**：
   - **自回归模型**：生成过程是**顺序依赖**的。例如，生成图像时需按像素顺序（如从左到右、从上到下）逐个预测，每一步依赖已生成的部分。这种显式的概率分解使其天然支持生成任务（如PixelRNN、WaveNet）。
   - **自编码器**：传统自编码器**无法直接生成数据**，因为缺乏对潜在空间的概率建模。若需生成，需结合额外技术（如变分自编码器VAE引入概率假设，或MADE通过掩码强制自回归结构）。
3. **结构与训练**：
   - **自回归模型**
     - **结构**：通常使用神经网络建模条件概率$p(x_i|x_{<i})$。例如，NADE用共享权重的神经网络逐步预测每个变量。
     - **训练**：通过最大化数据的对数似然（即最小化负对数似然损失），直接优化生成分布与真实分布的匹配。
   - **自编码器**
     - **结构**：包含编码器（压缩输入为潜在表示）和解码器（从潜在表示重构输入）。
     - **训练**：通过最小化重构误差（如均方误差或交叉熵），但未显式建模$p(x)$，导致无法评估新样本的概率。



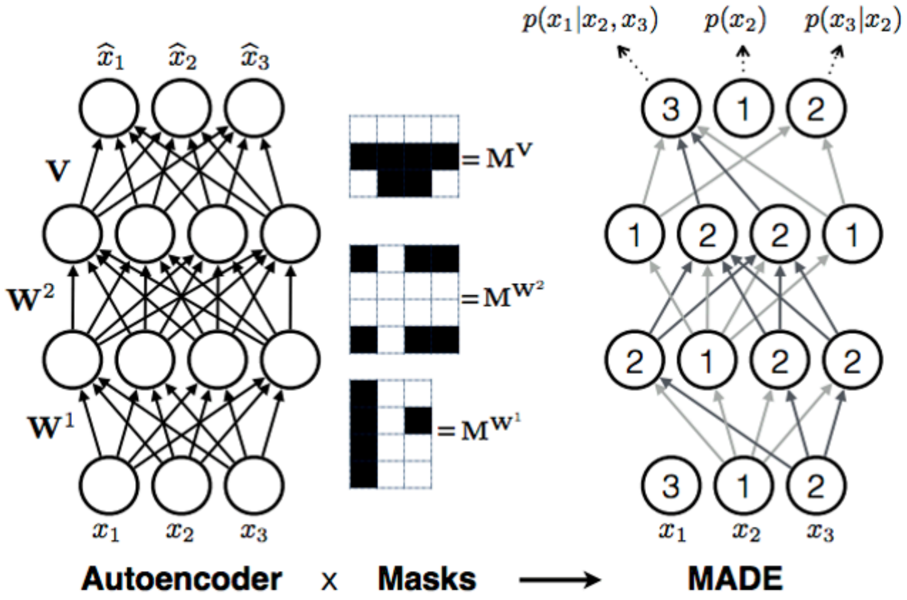
表面上，FVSBN和NADE都和自编码器很相似，但是他们之间有一个最大的**不同点**：对于自回归模型而言，它的输入是**有顺序的**，但是对于自编码器而言，它的输入是**没有顺序的**。

- 在**自回归模型**中，$\hat{x}_1$不依赖于任何输入x，因此在生成样本的时候我们**不需要任何输入**。并且$\hat{x}_2$仅仅依赖于x_1，而与其后面的$x_2, x_3, ...$无关。
- 在**自编码器**中，每一个输入都对每一个输出有影响，具体见**上图**即可。

自编码器的优势：我们可以用一个n个输出的简单神经网络来学习所有的参数，而NADE需要n次参数传递，自编码器具有**明显速度优势**。

密度估计掩码自动编码器(MADE)

由于**自编码器无法直接生成数据**，我们需要通过掩码(mask)在自编码器中强制实现自回归结构，因此实现并行计算条件概率。



原理：

- 对神经网络权重矩阵施加掩码，确保第i个输出仅依赖前$i = 1$个输入。
- 例如，若输入顺序为x_1, x_2, x_3，则x_3的预测仅能使用x_1, x_2

优势：

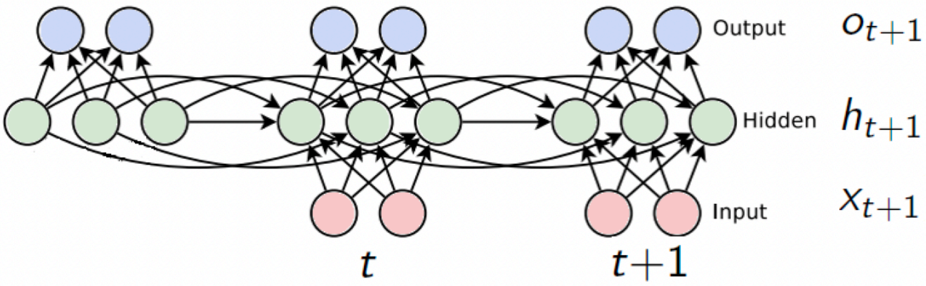
- 参数共享，训练速度大幅提升。
- 支持多变量条件分布（如彩色图像的RGB通道）。

应用：高维数据生成（如图像、语音）。

循环神经网络(RNN)

亟待解决的问题：当我们需要建模$p(x_t|x_{1:t-1}; \boldsymbol{\alpha}^t)$的时候，其“历史信息”$x_{1:t-1}$会越来越长，越来越多。

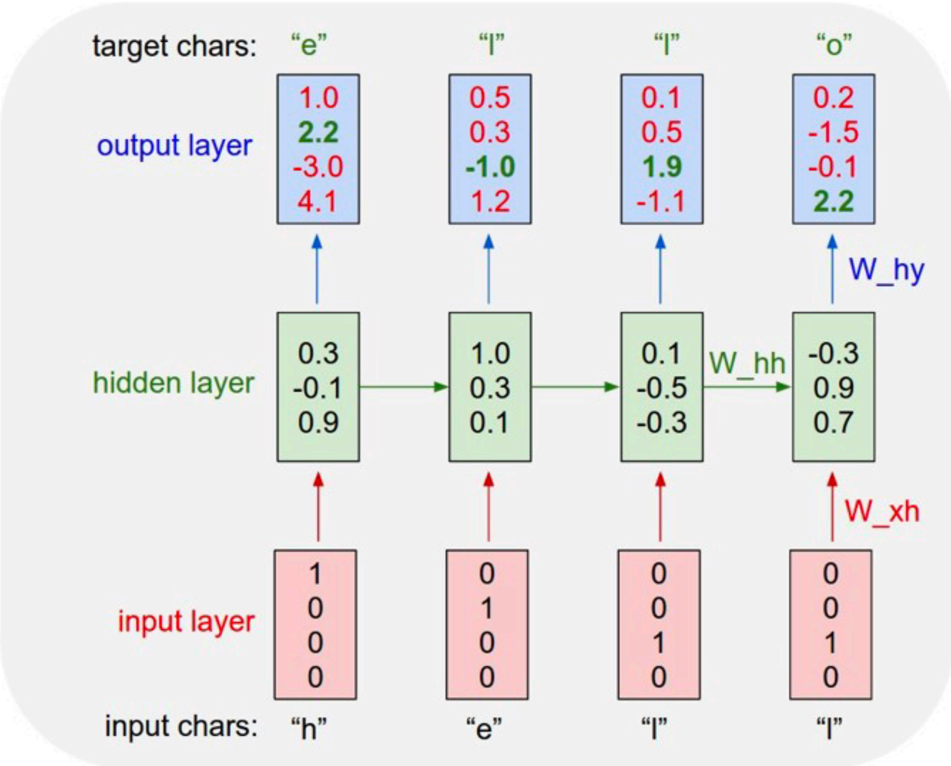
解决方案：保存对所有信息的“**摘要**”或“**汇总**”，并且不断**更新**它。



$$
\begin{aligned}
\text{Summary update rule:}\quad h_{t+1} &= tanh(W_{hh}h_t + W_{xh}x_{t+1}) \\
\text{Prediction:}\quad o_{t+1} &= W_{hy}h_{t+1} \\
\text{Summary initalization:}\quad h_0 &= \boldsymbol{b}_0
\end{aligned}
$$

- 其中:
  - 隐藏层h_t是在**时间t及以前**的输入的“汇总”。
  - 输出层o_{t-1}为条件概率$p(x_t|x_{1:t-1})$指定参数。

举例：字符RNN



- 假设输入$x_i \in \{h, e, l, o\}$，使用独热编码可得：$h = [1, 0, 0, 0]$, $e = [0, 1, 0, 0], \ldots$
- 使用**自回归模型**：$p(x = hello) = p(x_1 = h) \cdot p(x_2 = 3|x_1 = h) \cdots p(x_5 = o|x_1 = h, x_2 = e, x_3 = l, x_4 = l)$
  - $p(x_2 = e|x_1 = h) = softmax(o_1) = \frac{e^{2.2}}{e^{1.0}+e^{2.2}+e^{-3.0}+e^{4.1}}$
  - $o_1 = W_{hy}h_1$
  - $h_1 = tanh(W_{hh}h_0 + W_{xh}x_1)$

优势

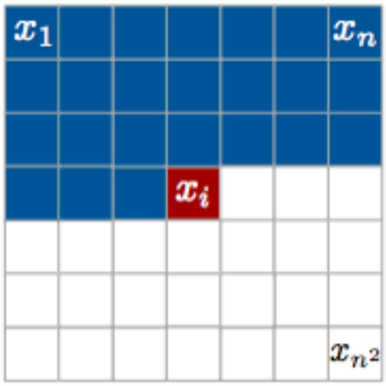
- 可以被用于任意长度的序列。
- 泛化能力强，对于所有的可计算的函数，都可以用有限长度的RNN来计算。

劣势

- 仍然需要排序
- 序贯似然评估（训练很慢）
- 序列生成（在自回归模型中不可避免）
- 难以训练（梯度消失/爆炸）

像素循环神经网络(Pixel RNN)

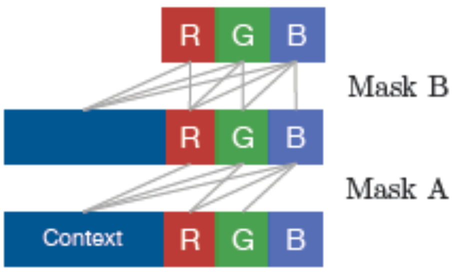
1. 采用光栅扫描的方法从左上到右下把像素变成一批有顺序的变量。



2. 每一个像素的条件概率分布$p(x_t|x_{1:t-1})$需要指明三种颜色：

$$p(x_t|x_{1:t-1}) = p(x_t^{red}|x_{1:t-1}) \cdot p(x_t^{green}|x_{1:t-1}, x_t^{red}) \cdot (x_t^{blue}|x_{1:t-1}, x_t^{red}, x_t^{green})$$

每个条件都是一个有256个可能值的分类随机变量。



基于注意力的模型

注意力机制原理：比较一个**query**向量和一批**key**向量。

1. 通过点乘**query**向量和所有的历史隐藏状态中的**key**向量来发现他们之间的相似度。

2. 构建**注意力分布**，指出历史输入中哪些部分更相关。
3. 利用**加权求和**来构造对于历史输入的“汇总”。
4. 利用这个“汇总”和现在的隐藏状态来预测下一个单词。