语言模型

语言模型定义:

语言模型（LM）的经典定义是一种对令牌序列（token）的概率分布。

设有一个令牌集的词汇表 $\mathcal{V}$。语言模型 p 为每个令牌序列 $x_1, \ldots, x_L \in \mathcal{V}$ 分配一个概率（介于 0 和 1 之间的数字）：

$$p(x_1, \ldots, x_L)$$

> **Tip**
>
> **语言模型的核心作用是根据概率分布判断一个句子是否合理。在生成任务中，它通过评估不同序列的概率，选择"更合理"的序列作为输出。 直观上，一个好的语言模型应具有语言能力和世界知识。**

自回归语言模型:

将序列 $x_{1:L}$ 的联合分布 $p(x_{1:L})$ 的常见写法是使用概率的链式法则：

$$p(x_{1:L}) = p(x_1)\, p(x_2 \mid x_1)\, p(x_3 \mid x_1, x_2) \cdots p(x_L \mid x_{1:L-1}) = \prod_{i=1}^{L} p(x_i \mid x_{1:i-1})$$

别地，需要理解 $p(x_i \mid x_{1:i-1})$ 是一个给定前面的记号 $x_{1:i-1}$ 后，下一个记号 x_i 的条件概率分布。在数学上，任何联合概率分布都可以通过这种方式表示。

然而，自回归语言模型的特点是它可以利用例如前馈神经网络等方法有效计算出每个条件概率分布 $p(x_i \mid x_{1:i-1})$。在自回归语言模型 p 中生成整个序列 $x_{1:L}$，我们需要一次生成一个令牌（token），该令牌基于之前已生成的令牌进行计算获得：

$$\text{for } i = 1, \ldots, L: \quad x_i \sim p(x_i \mid x_{1:i-1})^{1/T}$$

其中 $T \geq 0$ 是一个控制我们希望从语言模型中得到多少随机性的温度参数：

- $T = 0$：确定性地在每个位置 i 选择最可能的令牌 x_i
- $T = 1$：正常采样，直接使用原始概率分布。
- $T > 1$：增加随机性，降低高概率词的优势，使低概率词有更大的机会被选中：增加随机性，降低高概率词的优势，使低概率词有更大的机会被选中
- $T = \infty$：从整个词汇表上的均匀分布中采样

具体来说，这个温度参数会应用于每一步的条件概率分布 $p(x_i \mid x_{1:i-1})$，将其幂变为 $1/T$。这意味着：

- 当 T 值较高时，我们会获得更平均的概率分布，生成的结果更具随机性；
- 反之，当 T 值较低时，模型会更倾向于生成概率较高的令牌。

即：

$$p_T(x_i \mid x_{1:i-1}) \propto p(x_i \mid x_{1:i-1})^{1/T}$$

然而，如果我们仅将概率提高到 $1/T$ 的次方，概率分布可能不会加和到 1。我们可以通过重新标准化分布来解决这个问题

具体步骤如下：

（1）计算未归一化的退火概率

对于每个词 x_i，计算：

$$p_T(x_i \mid x_{1:i-1}) \propto p(x_i \mid x_{1:i-1})^{1/T}$$

（2）归一化

将所有词的概率归一化，使得它们的总和为 1：

$$p_T(x_i \mid x_{1:i-1}) = \frac{p(x_i \mid x_{1:i-1})^{1/T}}{\sum_j p(x_j \mid x_{1:i-1})^{1/T}}$$

对于非自回归的条件生成，**更一般地，我们可以通过指定某个前缀序列 x_i（称为提示）并采样其余的 $x_{i+1:L}$（称为补全）来进行条件生成。**例如$T = 0$产生的：

$$\underbrace{the, mouse, ate}_{\text{prompt}} \overset{T=0}{\rightsquigarrow} \underbrace{the, cheese.}_{\text{completion}}$$

信息理论：

语言模型的发展可以追溯到克劳德·香农，他在 **1948 年的论文《通信的数学理论》** 中奠定了信息论的基础。

其中最重要的概念是 **熵（Entropy）**，用于衡量概率分布的信息量和不确定性。

一、香农熵（Shannon Entropy）

定义：香农熵是用来衡量一个随机变量的不确定性或信息量的指标。

公式：

$$H(p) = \sum_x p(x) \log \frac{1}{p(x)} = -\sum_x p(x) \log p(x)$$

- $p(x)$：每个符号（如字母、单词等）出现的概率。
- $H(p)$：香农熵，单位是 **比特（bits）** 或 **奈特（nats）**，取决于对数的底数。

意义：

- **熵越小**：序列结构性越强，不确定性越低，编码所需长度越短。
- **熵越大**：信息不确定性越高，需要更多比特来表示。

直观理解：

- $\log \frac{1}{p(x)}$ 可以看作是**表示一个概率为 $p(x)$ 的元素 x 所需的编码长度。**
- 出现频率高的词（$p(x)$ 大），编码更短；
- 出现频率低的词（$p(x)$ 小），编码更长。

二、交叉熵（Cross-Entropy）

定义：交叉熵用于衡量两个概率分布之间的差异。在 NLP 中常用来评估模型生成的分布 $q(x)$ 与真实分布 $p(x)$ 的匹配程度。

公式：

$$H(p, q) = -\sum_x p(x) \log q(x)$$

- $p(x)$：真实分布（ground truth）
- $q(x)$：模型生成的分布
- $H(p, q)$：交叉熵，表示模型对真实分布的拟合程度

用途：

- 交叉熵越小，表示模型输出的分布越接近真实分布。
- 常被用作训练语言模型的目标函数（优化目标是最小化交叉熵）。

三、交叉熵与香农熵的关系

- 当 $q(x) = p(x)$ 时，交叉熵等于香农熵：

$$H(p, p) = -\sum_{x} p(x) \log p(x) = H(p)$$

- 交叉熵可以看作是对香农熵的一种扩展，用于比较两个分布的差异。

n-gram模型:

定义:在 **n-gram 模型** 中，对下一个词 x_i 的预测只依赖于最近的 $n-1$ 个词，而不是整个上下文历史：

$$p(x_i \mid x_{1:i-1}) = p(x_i \mid x_{i-(n-1):i-1})$$

即： 当前词的概率仅依赖于前面最近的 $n-1$ 个词。

n 的影响：

如果 n 太小：

- 模型只能捕获局部依赖
- 无法建模长距离依赖关系

如果 n 太大：

- 所有可能的 n-gram 组合数量爆炸性增长
- 在真实语料中几乎所有的合理组合都没有出现过（数据稀疏问题）
- 导致概率估计不可靠

> **Note**
>
> n-gram 模型通过限制上下文长度来简化语言建模任务。n 太小会丢失长距离依赖；n 太大会导致统计稀疏、无法准确估计概率。

神经网络:

基本概念：神经语言模型是语言建模领域的重要进展。2003 年，**Bengio 等人**首次提出了使用神经网络建模条件概率分布的方法：

$$p(x_i \mid x_{i-(n-1):i-1}) = \text{some-neural-network}(x_{i-(n-1):i})$$

上下文长度仍受 n 的限制，但**神经网络的非线性建模能力使更大 n 的语言模型变得可行。**

与 N-gram 模型的对比：

特点	N-gram 模型	神经语言模型
上下文长度	受限于 n	同样受限于 n（早期）
概率估计方式	统计频率	神经网络输出
对稀疏数据的鲁棒性	差（依赖平滑技术）	更好（可泛化未见过的组合）
计算资源需求	低	高（训练成本大）

尽管神经语言模型在相同数据量下优于 N-gram 模型，但由于训练开销大，N-gram 在接下来的十年仍然占据主导地位。

> **Note**
>
> **从 N-gram 到神经语言模型，再到 RNN 和 Transformer，语言模型逐步从局部依赖走向全局建模，并借助深度学习实现了更强的语言理解和生成能力。**

神经语言模型的发展历程：

（1）Recurrent Neural Networks (RNNs) 和 LSTMs

- RNN 及其变体 LSTM 允许模型依赖**整个历史上下文** $x_{1:i-1}$，相当于 $n = \infty$
- 条件概率分布建模为：

$$p(x_i \mid x_{1:i-1})$$

- 虽然理论上更强大，但**训练难度较大**，难以处理长距离依赖。

（2）Transformers（2017）

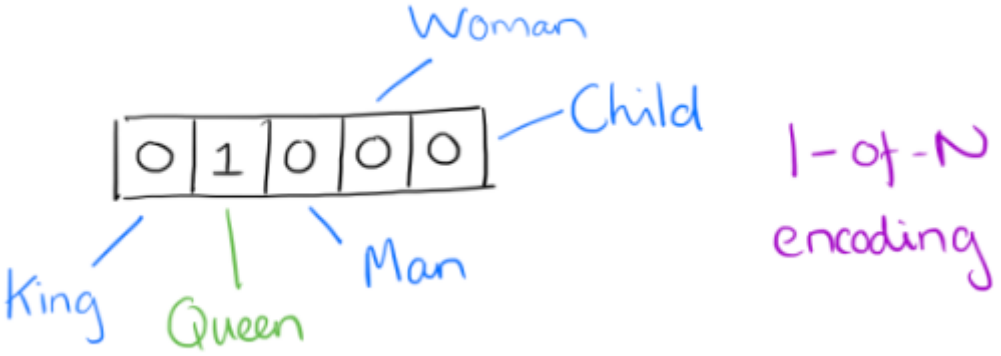
- Transformer 架构于 **2017 年**提出，最初用于机器翻译任务。
- 再次采用固定上下文长度 n，但通过**自注意力机制**大幅提升建模能力。
- 更易于并行化和训练（充分利用 GPU 的计算能力）。
- 当前主流模型（如 GPT-3）使用较大的上下文长度，例如：

$$n = 2048 \quad (\text{GPT-3})$$

Word2Vec

word2vec：

其特点**是能够将单词转化为向量来表示，这样词与词之间就可以定量的去度量他们之间的关系，挖掘词之间的联系。**
最早的词向量采用One-Hot编码，又称为一位有效编码，每个词向量维度大小为整个词汇表的大小，对于每个具体的词汇表中的词，将对应的位置置为1。



采用One-Hot编码方式来表示词向量非常简单，但缺点也是显而易见的，

- 一方面实际使用的**词汇表很大**，经常是百万级以上，这么高维的数据处理起来会消耗大量的计算资源与时间。
- 另一方面，One-Hot编码中所有词向量之间彼此正交，**没有体现词与词之间的相似关系。**

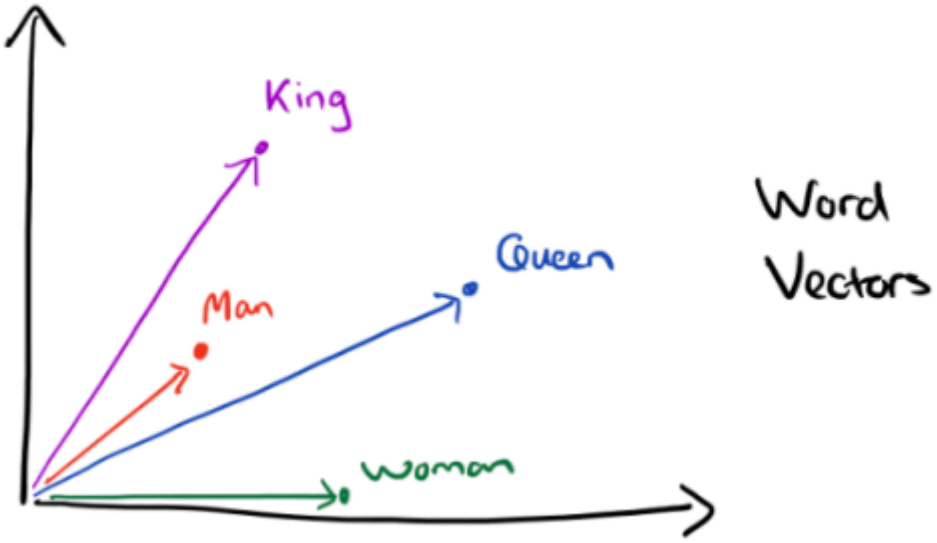
Distributed representation（分布式表示）：

- 它是一种用低维、稠密的实数向量来表示词的方法。
- 相比于高维稀疏的 One-Hot 编码，它能更好地捕捉词语之间的语义关系。
- 在神经网络中，这个词向量其实就是输入层到隐含层之间的权重矩阵 *W* 的每一行。

可以解决One-Hot编码存在的问题，它的思路是**通过训练将原来One-Hot编码的每个词都映射到一个较短的词向量上来**，而这个较短的词向量的维度可以由自己在训练时根据任务需要来指定。
例如 以四个维度去对词进行表示，如下：

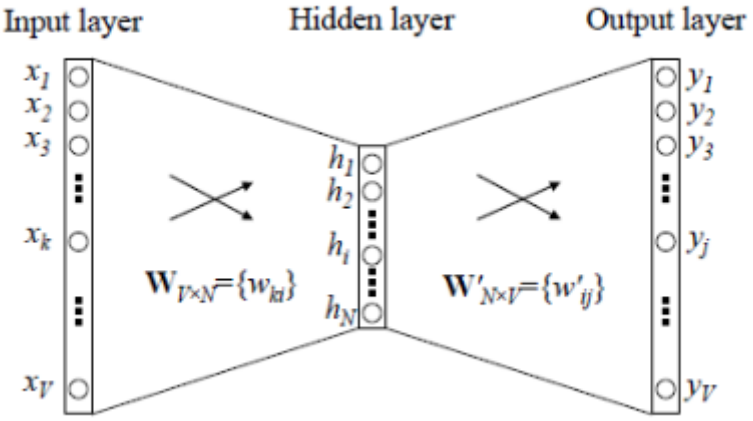
之后也可以使用降维的方式将其从四维变成二维，就可以得到词之间的相关性：

$$\overrightarrow{King} - \overrightarrow{Man} + \overrightarrow{Woman} \approx \overrightarrow{Queen}$$



Word2Vec原理：

Word2Vec 的训练模型本质上是**只具有一个隐含层的神经元网络**（如下图）。

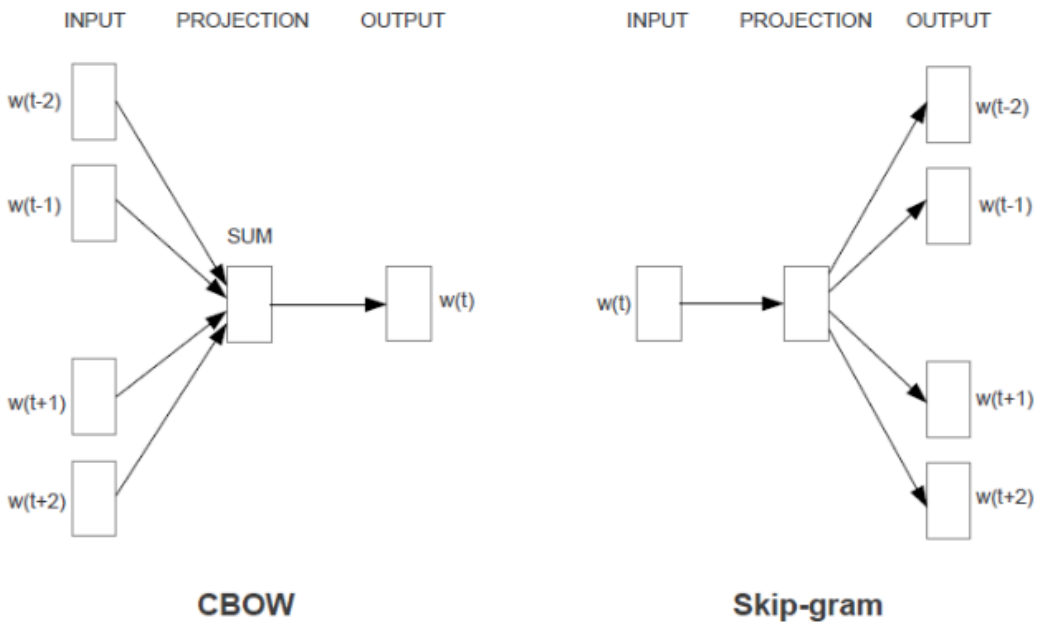


它的输入是采用One-Hot编码的词汇表向量，它的输出也是One-Hot编码的词汇表向量。**使用所有的样本，训练这个神经元网络，等到收敛之后，从输入层到隐含层的那些权重，便是每一个词的采用Distributed Representation的词向量。**

> **Important**
>
> 因为输入的向量是One-hot的形式 其实输入层和隐含层之间的权重矩阵 W 实际上就等价于所有词的分布式表示。 因为向量和W相乘的时候就相当于是取出了第i行,所以上图中单词的Word embedding后的向量便是矩阵$W_{V\times N}$的第i行的转置。这样就把原本维数为v的词向量变成了维数为N的词向量（N远小于V），并且词向量间保留了一定的相关关系。

CBOW适合于数据集较小的情况，而**Skip-Gram**在大型语料中表现更好。

- 其中CBOW如下图左部分所示，使用围绕目标单词的其他单词（语境）作为输入，在映射层做加权处理后输出目标单词。
- 与**CBOW根据语境预测目标单词**不同，**Skip-gram根据当前单词预测语境**，如下图右部分所示



CBOW:

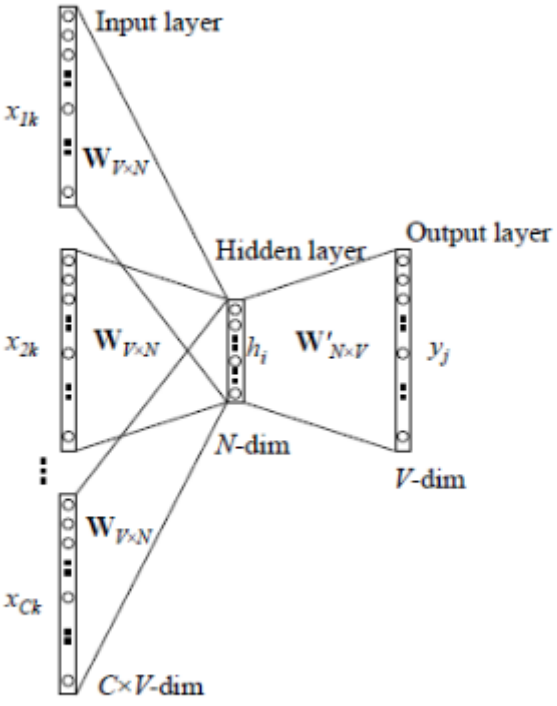


Figure 2: Continuous bag-of-word model

1.输入层：**上下文单词的 One-Hot 编码词向量，V 为词汇表单词个数**，C 为上下文单词个数。以上文那句话为例，这里 $C=4$，所以模型的输入是（is, an, on, the）4 个单词的 One-Hot 编码词向量。

2.初始化一个权重矩阵 $W_{V\times N}$，然后用所有输入的 One-Hot 编码词向量左乘该矩阵，得到维数为 N 的向量 $\omega_1,\omega_2,\ldots,\omega_c$，这里的 N 根据任务需要设置。

3.将所得的向量 $\omega_1,\omega_2,\ldots,\omega_c$ 相加求平均作为隐藏层向量 h。

4.初始化另一个权重矩阵 $W'_{N\times V}$，用隐藏层向量 h 左乘 $W'_{N\times V}$，再经激活函数处理得到 V 维的向量 y，y 的每一个元素代表相对应的每个单词的概率分布。

5.y 中概率最大的元素所指示的单词为预测出的中间词（target word）与 true label 的 One-Hot 编码词向量做比较，误差越小越好（根据误差更新两个权重矩阵）。

在训练前需要定义好损失函数（一般为交叉熵代价函数），采用梯度下降算法更新 W 和 W'。训练完毕后，输入层的每个单词与矩阵 W 相乘得到的向量的就是 Distributed Representation 表示的词向量，也叫做 word embedding。因为 One-Hot 编码词向量中只有一个元素为 1，其他都为 0，所以第 i 个词向量乘以矩阵 W 得到的就是矩阵的第 i 行，所以这个矩阵也叫做 look up table，有了 look up table 就可以免去训练过程，直接查表得到单词的词向量。

Skip-gram:

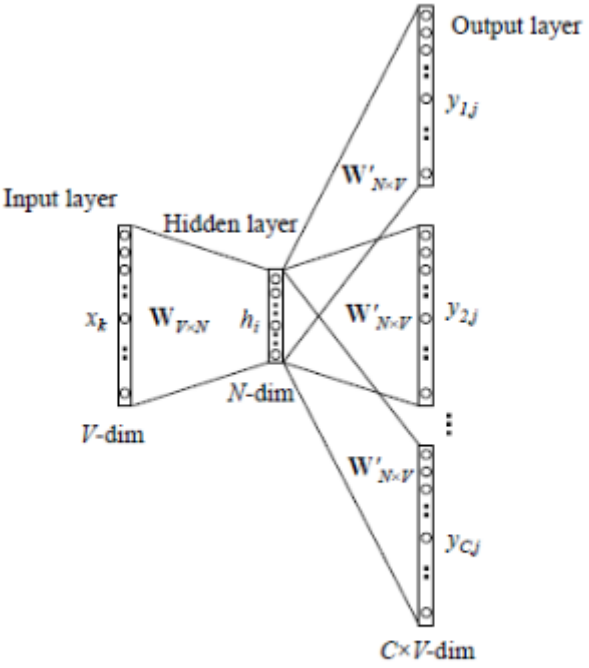


Figure 3: The skip-gram model.

它的做法是，**将一个词所在的上下文中的词作为输出，而那个词本身作为输入，也就是说，给出一个词，希望预测可能出现的上下文的词。通过在一个大的语料库训练，得到一个从输入层到隐含层的权重模型。**

例如，“apple”的上下文词是（‘there’, ‘is’, ‘an’, ‘on’, ‘the’, ‘table’）。那么以 apple 的 One-Hot 词向量作为输入，输出则是（‘there’, ‘is’, ‘an’, ‘on’, ‘the’, ‘table’）的 One-Hot 词向量。训练完成后，就得到了每个词到隐含层的每个维度的权重，就是每个词的向量（和 CBOW 中一样）。

假如有一个句子：

> “There is an apple on the table”

1. 首先选句子中间的一个词作为输入词，例如选取“apple”作为 input word；
2. 有了 input word 以后，再定义一个叫做 skip_window 的参数，它代表着从当前 input word 的一侧（左边或右边）选取词的数量。
    - 如果设置 skip_window=2，那么最终获得窗口中的词（包括 input word 在内）就是：

$$[\backslash is',' an',' apple',' on',' the']$$

    - skip_window=2 代表着选取 input word 左侧 2 个词和右侧 2 个词进入窗口，所以整个窗口大小：

$$\text{span} = 2 \times 2 = 4$$

3. 另一个参数叫 num_skips，它代表着从整个窗口中选取多少个不同的词作为 output word。
    - 当 skip_window=2，num_skips=2 时，将会得到两组 (input word, output word) 形式的训练数据，即：

$$('apple','an'),\quad ('apple','on')$$

神经网络基于这些训练数据中每对单词出现的次数习得统计结果，并输出一个概率分布，这个概率分布代表着我们词典中每个词有多大可能性跟 input word 同时出现。举个例子，如果向神经网络模型中输入一个单词“中国”，那么最终模型的输出概率中，像“英国”，“俄罗斯”这种相关词的概率将远高于像“苹果”，“蝈蝈”这种非相关词的概率。因为“英国”，“俄罗斯”在文本中更大可能在“中国”的窗口中出现。我们将通过给神经网络输入文本中成对的单词来训练它完成上面所说的概率计算。

4.通过梯度下降和反向传播更新矩阵 W

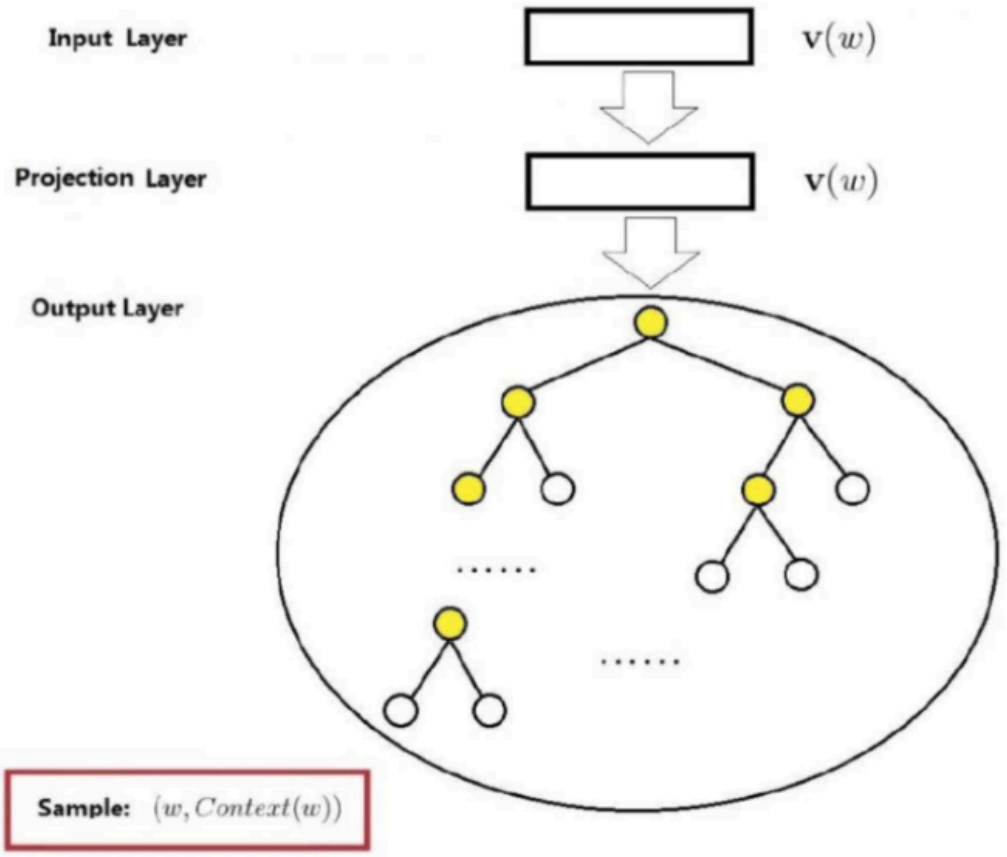
5.矩阵 W 中的行向量即为每个单词的 Word embedding 表示

> **Note**
>
> **CBOW 是“根据上下文猜中间词”，Skip-gram 是“根据中间词猜上下文”；CBOW 更快更稳，Skip-gram 更慢但学到的词向量语义更强。**

Hierarchical Softmax

在传统的词嵌入模型中（如 Word2Vec），Softmax 层的计算量非常大，尤其是在词汇表 V 非常大的情况下。
Hierarchical Softmax 是一种对传统 Softmax 的改进方法，它通过引入 **哈夫曼树（Huffman Tree）** 来显著降低计算复杂度。

1. 第一点是**从输入层到隐藏层的映射**，没有采用原先的与矩阵W相乘然后相加求平均的方法，而是直接对所有输入的词向量**求和。**假设输入的词向量为（0,1,0,0）和（0,0,0,1），那么隐藏层的向量为（0,1,0,1）。
2. 第二点改进是**采用哈夫曼树来替换了原先的从隐藏层到输出层的矩阵W’。**哈夫曼树的叶节点个数为词汇表的单词个数V，一个叶节点代表一个单词，而从根节点到该叶节点的路径确定了这个单词最终输出的词向量



具体来说，这棵哈夫曼树除了根结点以外的所有非叶节点中都含有一个由参数 θ 确定的sigmoid函数，不同节点中的 θ 不一样。训练时隐藏层的向量与这个sigmoid函数进行运算，根据结果进行分类，若分类为负类则沿左子树向下传递，编码为0；若分类为正类则沿右子树向下传递，编码为1。

Negative Sample:

负采样的核心思想是：每次训练时只更新一部分权重，而不是全部输出层权重

换句话说：

- 对于当前的正样本（即上下文词），我们希望神经网络预测出高概率；
- 对于随机选取的一些“负样本”（即非上下文词），我们希望它们的输出概率尽可能低；
- 其他大部分词的权重不参与更新，从而大大减少计算量。

> **Note**
>
> 客观上来说 就是只更新正样本和随机几个负样本的输出概率（权重），而不是把所有的负样本都进行权重更新