从one-hot到embedding

P9工作法
分布式架构、上百人团队管理、全球支付实践与AI技术

上文P9工作法：分析了如何用one-hot表达一个词，以及它的优缺点。这里接着分析如何从one-hot进化到embedding。

如何预测下一个词

如何能够预测句子中的下一个词，能够想到的最简单的办法是不是计算下一个词出现的概率。所以出现了 N-gram 模型。

N-gram

这个名字看起来不明觉厉，但究其本质就是统计学计算概率。如果N等于1，就是Unigram，比如我爱中国，那么我，爱，中国每个词都是序列。如果N等于2，那就是Bigram，我爱，爱中国就是一个序列。如果N等于3，那就是Trigram，那么我爱中国就是一个序列。N也可以大于3，那就以此类推。

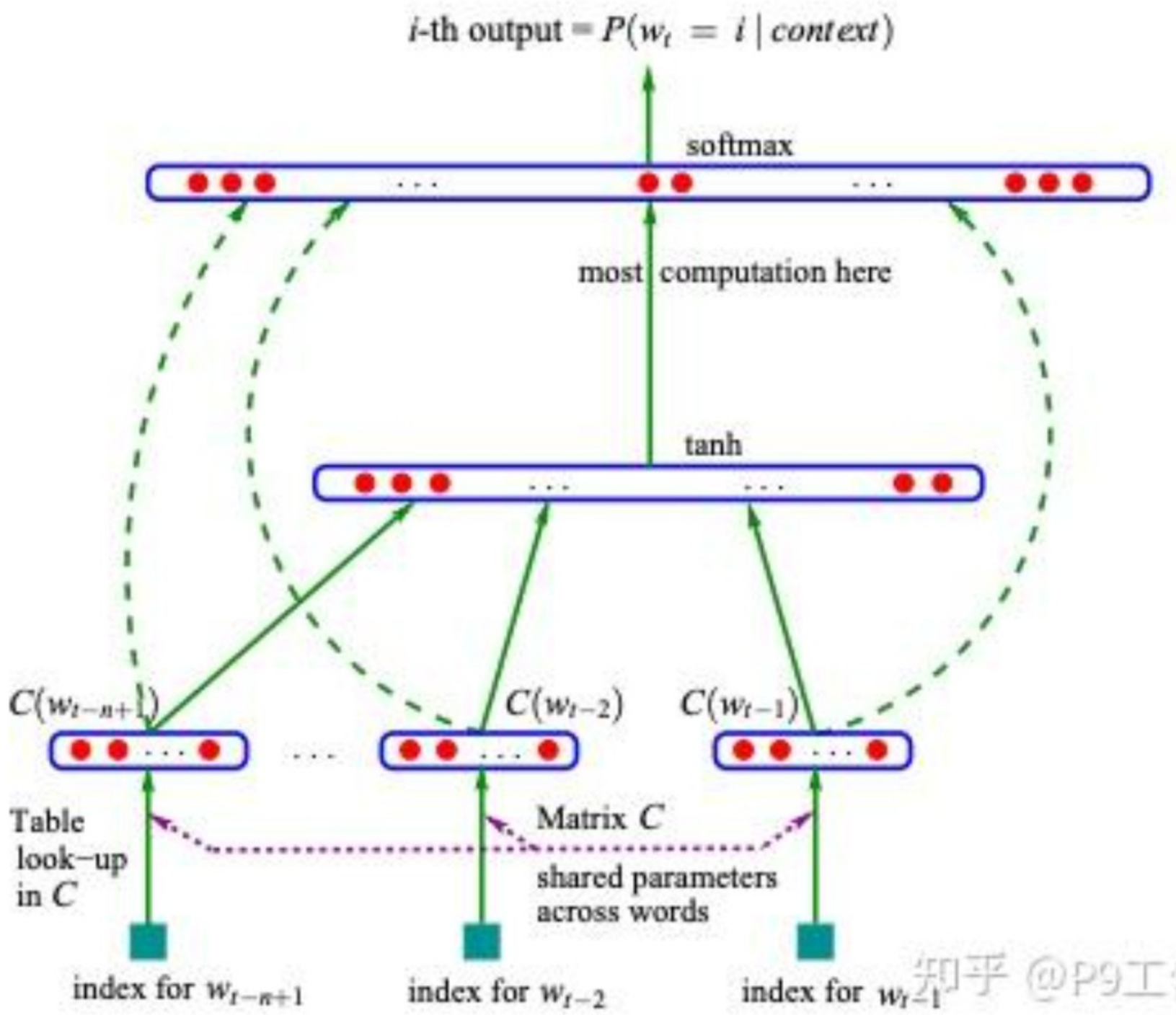
N-gram就是一个概率计算，比如在Bigram 中，就是去计算后一个词是中国的概率。这个概率就是通过大量的文本中统计计算得来，看哪个词出现的概率最高。

这种方法有很大的问题，第一个是计算量zx超级大，第二个就是预测不准，因为下一个词的出现如果不看语义只是看频率就可能有问题，而且如果在训练的统计文本中没出现过的就无法预测（为了应对这个问题也有一些技术来缓解，这就是平滑技术，比如加法平滑，Kneser-Ney等）。

NNLM模型

NNLM（Neural Network Language Model）是由 Yoshua Bengio 等人在 2003 年的一篇论文中提出，论文地址：jmlr.org/papers/volume3...

原理图如下：



即通过输入一串文本，通过神经网络能够预测出来下一个词的概率。至于什么是神经网络可以看这个专栏的系列文章：

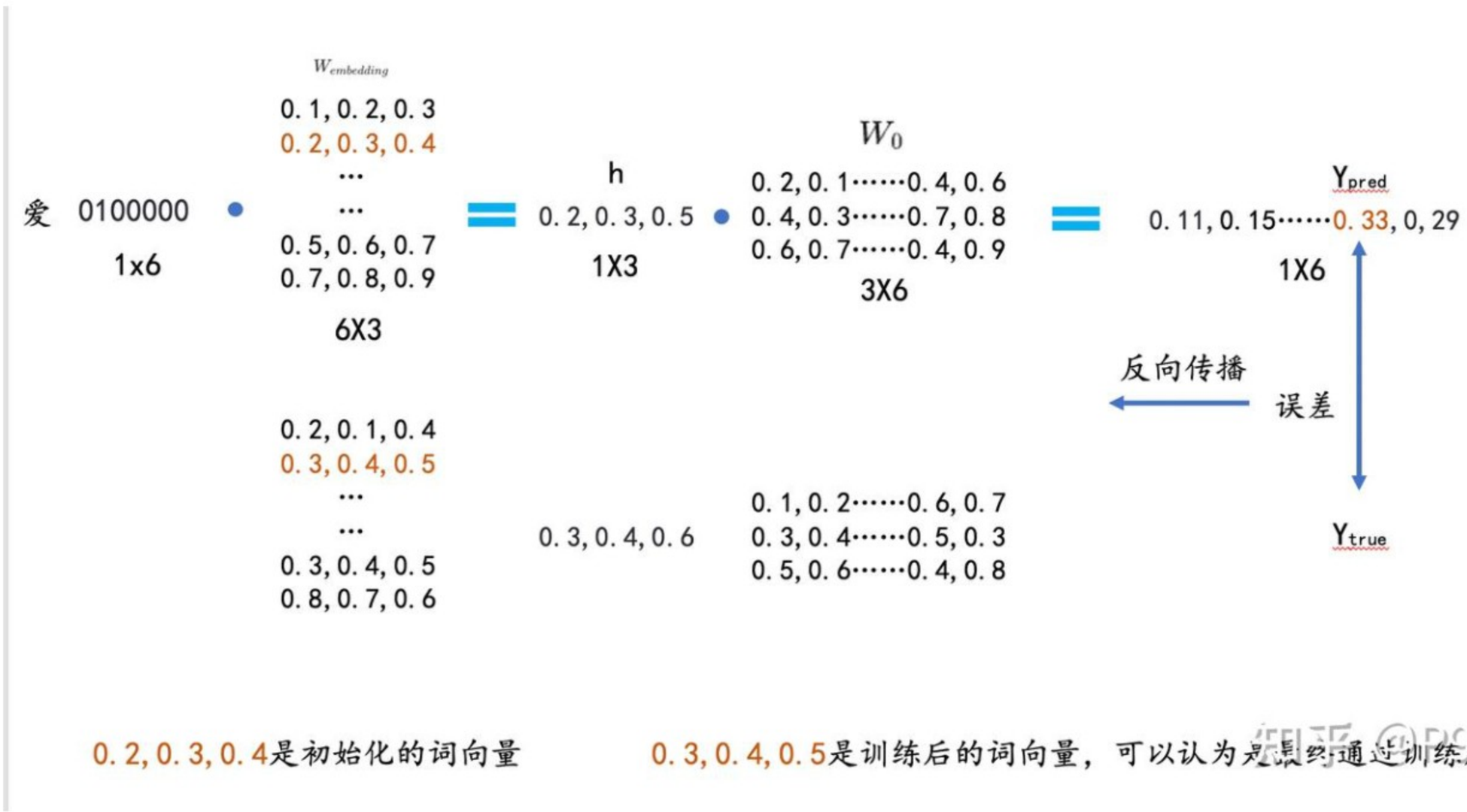
P9工作法：架构师与AI狭路相逢

P9工作法：看全局知AI

P9工作法：函数是神经网络的本质

P9工作法：三步构建神经网络

通过“神经网络”的计算就能够得到下一个单词应该是什么。但可以用一个计算过程（以"我爱我的祖国"为例）来比较直白的理解这个过程。



1、源头的文本输入就是 one-hot 编码的向量，如爱的编码是[0,1,0,0,0,0]

2、假设有一个6*D维度的矩阵（为什么是6，因为one-hot的维度是6，这样向量与矩阵才能够相乘），将编码与矩阵相乘（因为one-hot编码特殊性，其实就是从矩阵中找相应的行，one-hot第二维是1，那就是找第二行），然后会得到一个D维度的向量，也就是图上的h

3、将h 纳入神经网络结构中去训练，可以认为 W_0 就是神经网络（其实是神经网络的参数），通过这个神经网络最后的softmax函数，就能够找到词汇表中概率最大的哪个词。

4、将该预测的词（假设是 上 这个词），那需要与实际词 狗 做损失函数，然后反向传播（一个算法）去重新训练模型。

5、当模型训练好以后，就能够较好地预测下一个词了。

什么是词向量

NNLM的副产物

N-gram到NNLM其实是将文本处理迈进了一大步。其中最关键的一点，就是能够找到一种方法，能够把one-hot编码用起来，能够进入模型进行训练。可以看到上述的矩阵其实是一个非常好的东西，可以将one-hot编码变成D维的向量。那么是不是可以用D维向量来直接表达文字呢，去替代one-hot编码呢。

结论显然是可以的，这个D维向量就其实就是词向量。而且通过NNLM学习出来，只是我们利用的不是神经网络的预测能力，而是提取训练的参数。

为什么叫词嵌入

将one-hot编码嵌入到一个矩阵中，就得到一个向量，这就是词嵌入的形象理解。这个矩阵就是一个超级大的空间，把一个个词嵌入到这个空间中。可以想象为用数学表示了一个宇宙，词就是星星一样嵌在这个空间中，所以词嵌入这个翻译真的太形象了。

感知词向量

Google 在 2013 年发布了 Word2Vec 的 C 实现，而且利用他们新闻数据训练了一个Word2Vec的预训练模型，可以直接下载体验。

即你直接输入一个词，通过这个模型可以直接获取到Google已经训练好的词向量。

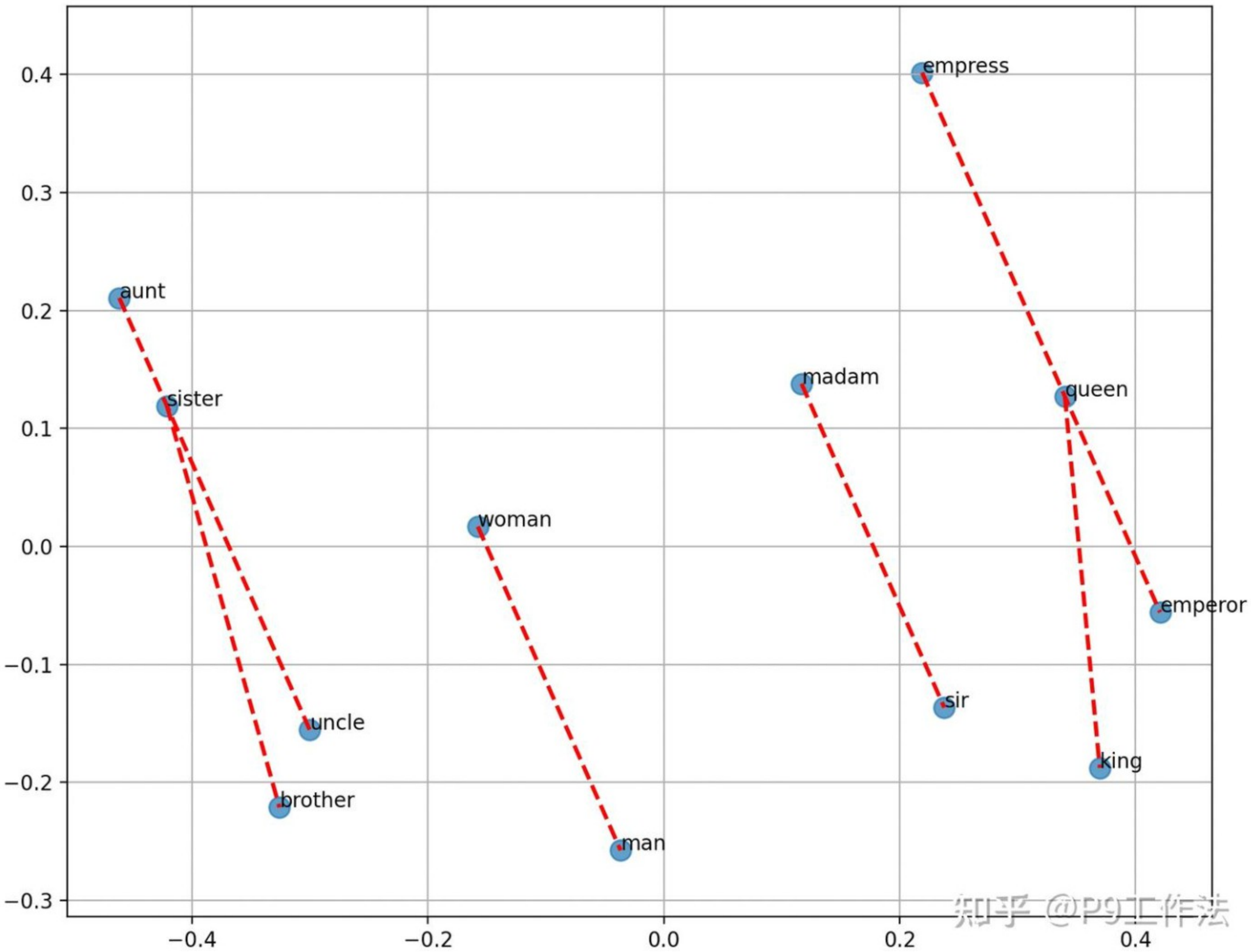
```python
import gensim
import numpy as np

# 加载预训练的 Word2Vec 模型
model_path = 'path/to/GoogleNews-vectors-negative300.bin.gz'  # 替换为你的模型文件路径
model = gensim.models.KeyedVectors.load_word2vec_format(model_path, binary=True)

# 获取“中国”的词向量
try:
    china_vector = model['中国']
    print("词向量维度:", china_vector.shape)
    print("'中国'的词向量:")
    print(china_vector)
except KeyError:
    print("单词 '中国' 不在词汇表中。")
```


是一个300维的向量：[0.12345678, -0.98765432, ...]

训练出来的词向量通过可视化后，会发现语义相近的会在一起，距离较近。而且可以通过加减法对语义进行运算，比如皇后等于皇帝-男人+女人。这也就是说，词向量具有了一定的语义表达。



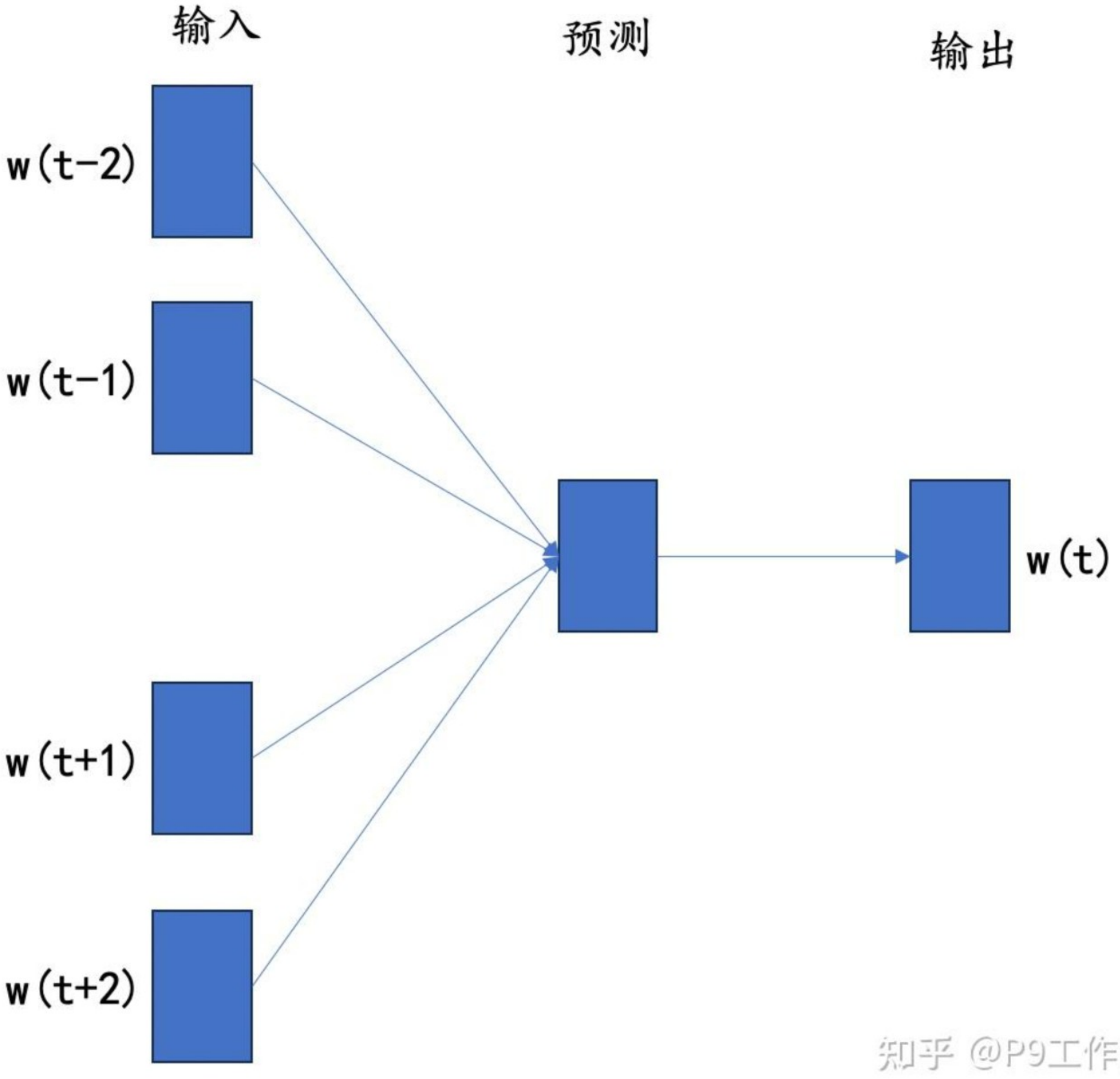
而且，词向量明显是维度较小（Google的Word2Vec也就300维），而且也不稀疏（不全是0），这对于计算和语义表达都挺好。

如何训练词向量

通过上面的分析可以看到词向量其实是可以通过神经网络模型训练出来的。但模型有两种，一种是CBOW，一种是Skip-gram。但直白的意思就是观其伴，而知其义。搜集大量的文本，比如新闻，书籍，网页等，然后利用神经网络模型来训练得到词向量。

CBOW

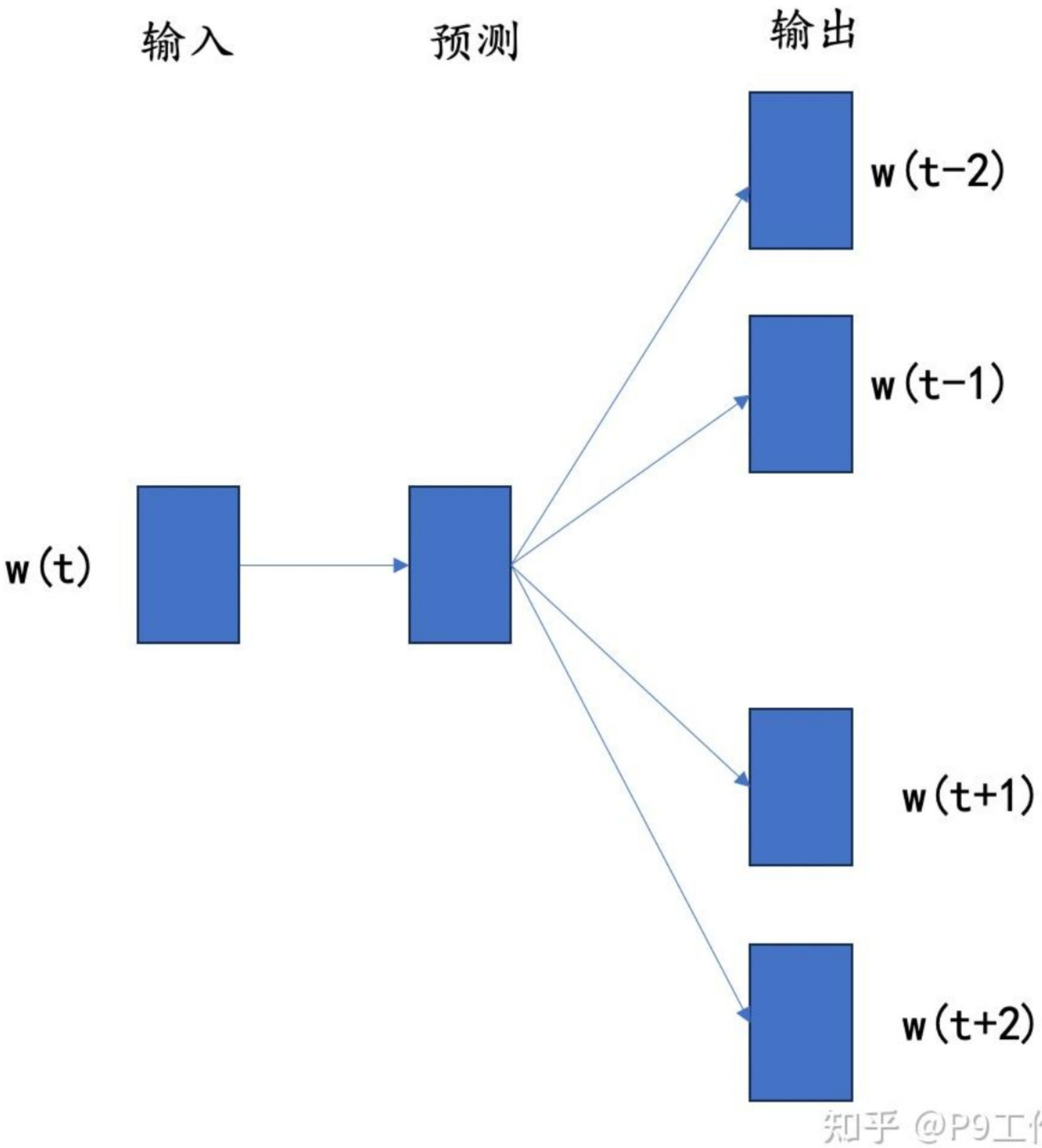
词袋模型，是Continuous Bag of Words的缩写，通过上下文单词预测目标单词，如下图所示。



比如上面的例子中，一只狗躺在地毯上，如果把狗这个词遮住，把前面的“一，只”和后面的“躺，在” 给到模型作为输入，去预测狗这个词。而“一、只”这个数量2就是滑动窗口，也就是能够感知的上下文大小，这个可以自己设置。

Skip-gram

跳词模型，通过目标单词预测上下文单词。如下图所示



比如上面的例子中，一只狗躺在地毯上，把狗输入进去，预测上下文是“一，只”和“躺，在”。输入到神经网络模型后，通过损失函数定义以及反向传播算法，就能够得到词向量。

但既然是两个模型，肯定有所区别。简单理解（如果要比较精确的理解，需要到介绍神经网络相关文章中，用损失函数定义来理解，这里先按下不表）本质区别是在：

1、Skip-gram的训练时间要比CBOW长，从模型结构就能够看到，CBOW是训练一次可以得到滑动窗口的词向量，而Skip-gram只能得到1个（要记住，我们是要得到词向量，所以CBOW相当于是1个老师去教多个学生）。

2、在数量少时，Skip-gram 的效果会好于CBOW，而且低频率出现的词用Skip-gram 训练会好一些，因为训练轮次更多。

当然这两个模型也有自己优化迭代的技巧，比如用负样本采样技术，减少最后一层softmax的计算量。但这个属于局部优化，不妨碍理解词向量大局，可以有需要再去学习。

词向量的缺点

由于word2vec 是通过训练样本中的文本来训练出来的词向量，所以缺点可想而知： 1、如果训练包含的文本数量不够多，显然有些词无法得到有效的词向量。

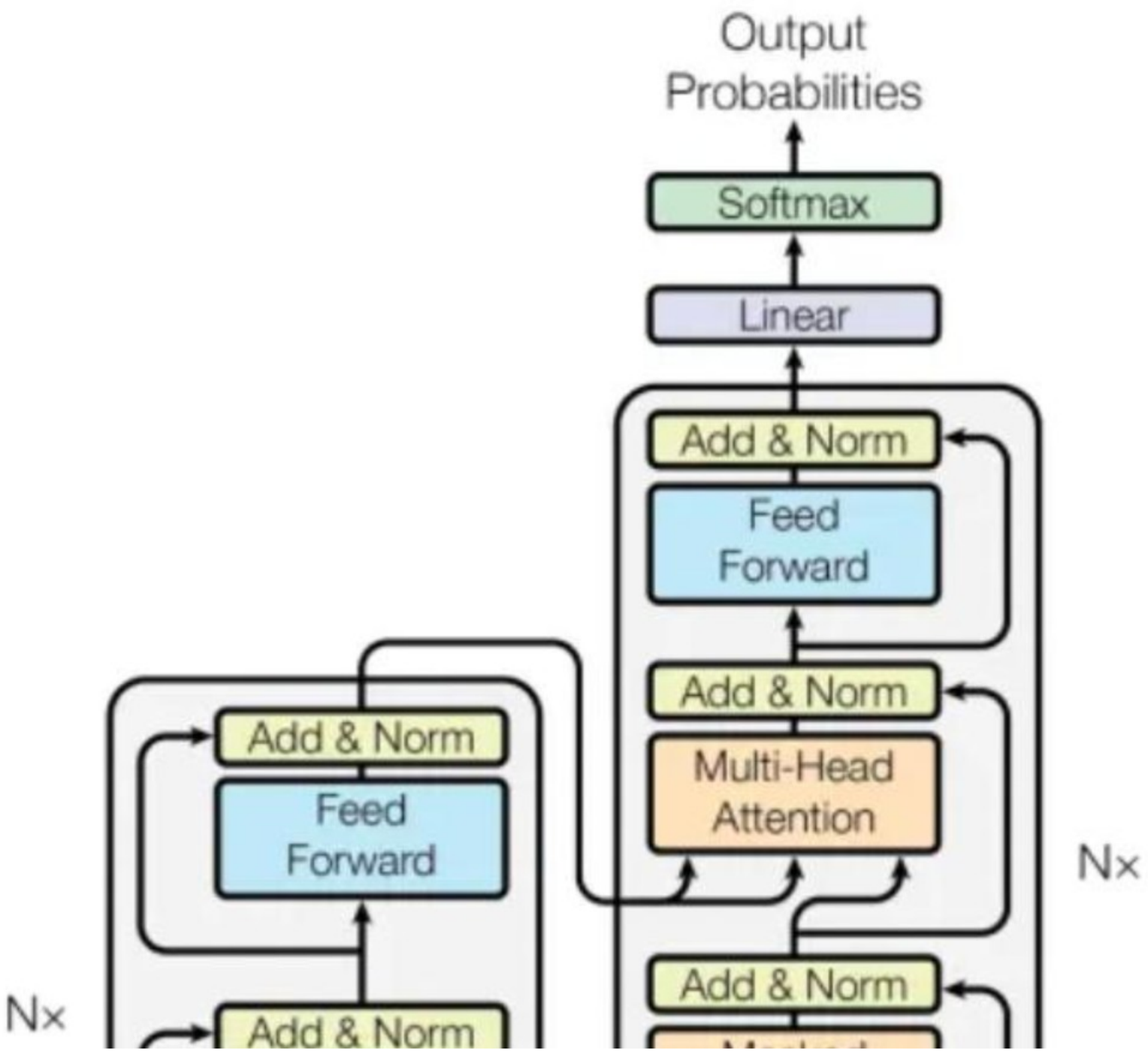
2、继续推论，这个词向量是一个固定的，比如你输入苹果，一定会得到一个词向量。但很明显，苹果有时候表示水果，有时候表示苹果手机，这样一个固定的词向量显然不对。也就是说这样的静态词向量思有问题的，无法表达不同语境下的真正意义。

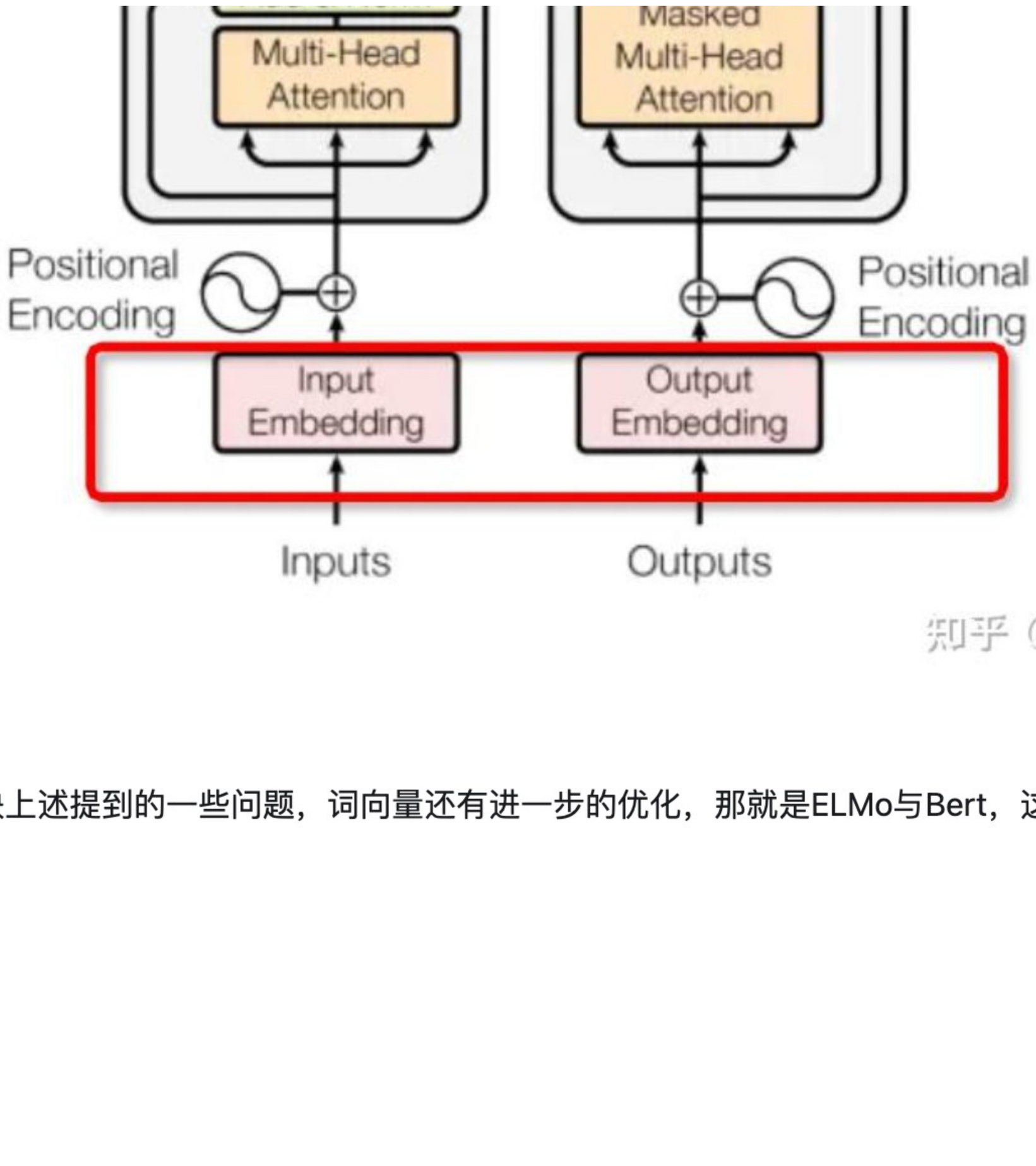
3、在继续深入到原理，可以看到训练是有滑动窗口的，也就是上下文大小是有限的，这从本质上来说也会导致词向量不准确。

embedding总结

事实上，词向量已经比one-hot前进了一大步，one-hot都能够用起来，那如果改进后面的模型自然词向量也能够得到更大的改进（底层来说，反正都要通过神经网络去反向更新，更先进的表示只是减少成本，提升效果，并不是原理上的彻底阻碍）。

所以其实会看到，词向量embedding在后续的transformer模型中，都是最基础的输入信息了。





为了解决上述提到的一些问题，词向量还有进一步的优化，那就是ELMo与Bert，这个单独再开篇章分享。