关于实现

2021-10-29

1. 关于实现方式

词向量是一种语言模型，将词映射为向量。词向量模型可以使用多种方式实现，例如neural networks、co-occurrence matrix,、probabilistic models等。

Word2Vec也是一种词向量，通过Skip-gram和CBOW两种方式构建训练集，可以选择Hierarchical Softmax或Negative Sampling两种方法对复杂度进行优化。Word2Vec实现方法概括：

1. gensim

   快速感受一下什么是词向量

2. 四合一：Skip_gram+HS、Skip_gram+NEG、CBOW+HS、CBOW+NEG

   比较正版，贴近实际，经过对比，Skip_gram+NEG速度快效果好

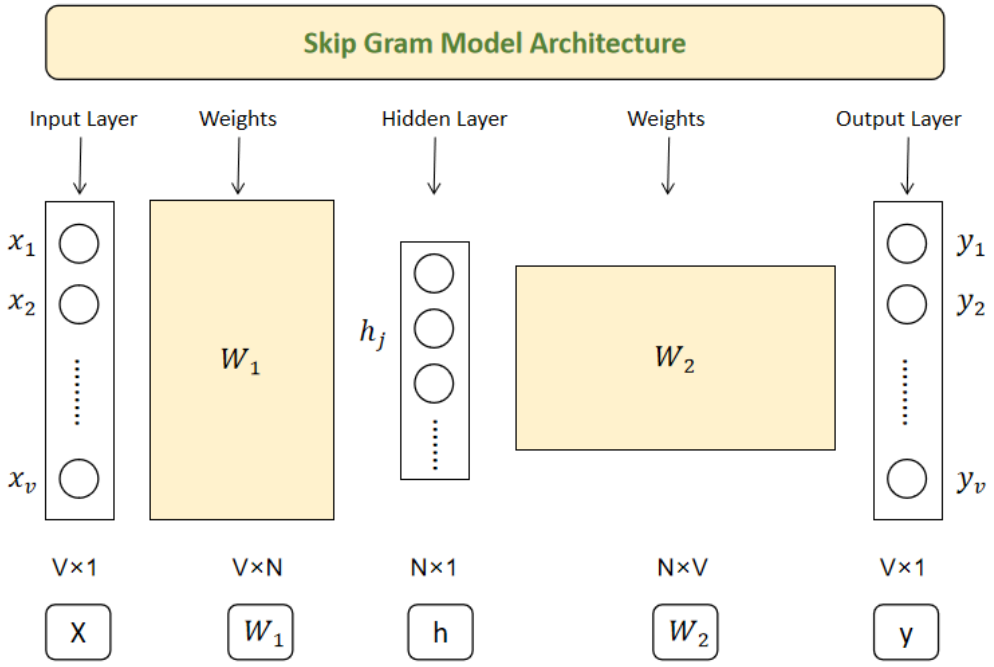
3. 两层神经网络，纯pyhton版（自行实现BP，略麻烦）、pytorch版（one-hot输入、embedding输入）

   自娱自乐，加深对词向量与神经网路的理解。

4. 单层神经网络

   茴香豆的茴有几种写法?

2. 基于神经网络的实现



术语描述：

1. V：词表大小（或语料库中不同单词的数目）
2. N：词向量维度
3. X：输入单词，使用**one-hot编码**表示
4. w：原始单词
5. $\text{Context}(w_i)_c$：单词w_i的第c个周围词，其中，$1 \leq c \leq C$
6. W_1：输入层与隐藏层之间的权重
7. W_2：隐藏层与输出层之间的权重
8. y：每个单词的概率，基于softmax计算

前向计算：

1. $\mathbf{h} = \mathbf{W_1}^{\mathrm{T}}\mathbf{X}$
2. $\mathbf{u} = \mathbf{W_2}^{\mathrm{T}}\mathbf{h}$
3. $\mathbf{y} = \text{softmax}(\mathbf{u})$

损失函数：

1. 给定中心词w_i，预测为周围词w_j的概率

$$p(w_j \mid w_i) = y_j = \frac{e^{u_j}}{\sum_{k=1}^{V} e^{u_k}} \tag{4}$$

2. 给定中心词w_i，期望最大化目标

$$\prod_{c=1}^{C} p\left(\text{Context}(w_i)_c \mid w_i\right) = \prod_{c=1}^{C} \frac{e^{u_{ic}}}{\sum_{k=1}^{V} e^{u_k}} \tag{5}$$

其中，ic表示单词w_i对应的第c个周围词在词表中的索引

3. 定义损失函数：越小越好

$$\left(\sum_{c=1}^{C} \quad e^{u_{ic}} \right)$$

$$E = -\log\left\{\prod_{c=1} \frac{e}{\sum_{k=1}^{V} e^{u_k}}\right\} \tag{6}$$

反向传播：

略，参考其它文档。

3. nn.Linear VS ns.Embedding

Word2Vec 论文中给出的架构其实就一个单层神经网络，那么为什么不直接用 nn.Linear() 来训练呢？nn.Linear() 不是也能训练出一个 weight 吗？

答案是可以的。当然可以直接使用 nn.Linear()，只不过输入要改为 one-hot Encoding，而不能像 nn.Embedding() 这种方式直接传入一个 index。另外，需要设置 bias=False，因为我们只需要训练一个权重矩阵，不训练偏置。

4. 参考文档

1. (68条消息) Pytorch实现word2vec(Skip-gram训练方式)_Delusional的博客-CSDN博客