Word2Vec CBOW实现

彭博 519030910366

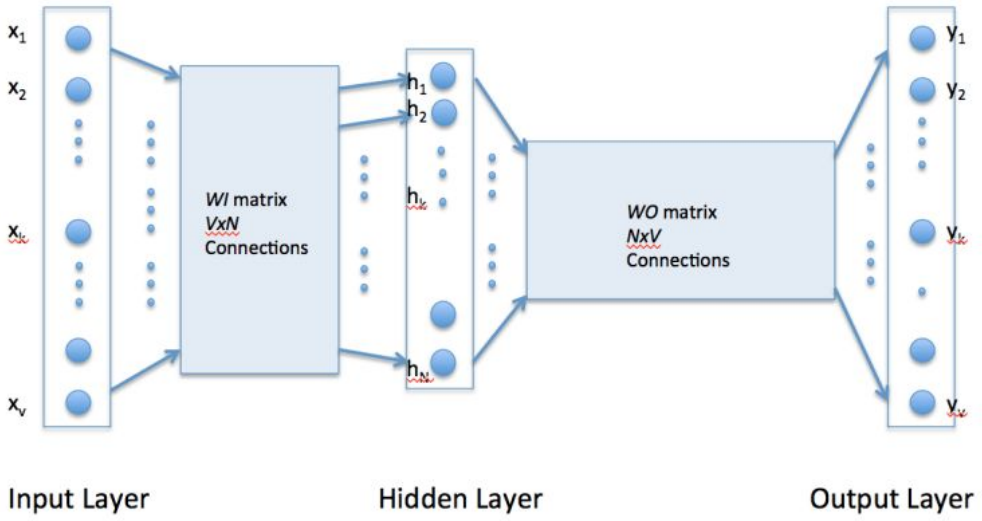
只需补全给定代码中的`train_one_step`函数，下面简要介绍Word2Vec和CBOW原理

CBOW

为了更接近自然语言的本质，希望在训练过程中蕴含上下文信息，但是由于引入上下文在过去的模型中容易导致词的向量表示维度过大，为后续运算带来了很大的麻烦，因此引入CBOW，使得输入向量可以满足：

1. 携带上下文信息
2. 稠密

在介绍CBOW之前，首先简要介绍Word2Vec模型，如下图所示



输入层X的大小V即训练所用到的词表大小，隐藏层神经元数目为N，（训练时设定），输出层Y大小同样为V，模型主要流程为：输入层X*WI矩阵，其中WI矩阵的大小为V*N, 得到隐藏层H， 隐藏层H*WO得到输出层U, 输出U后需要进行softmax层得到最终的预测。

训练网络过程中，计算出softmax后的向量和ground truth的one hot vector的loss，然后BP反向更新权重。

CBOW结构和Word2Vec基本一致，不同之处仅仅是此时的输入层X并不是来源于一个单词而是多个单词的结合。其中的多个单词主要由一个window采样得到，（可以想象成一个窗在整个文段中移动然后每次移动后把窗中的内容放入一个词袋，作为一个target vector的一个上下文组

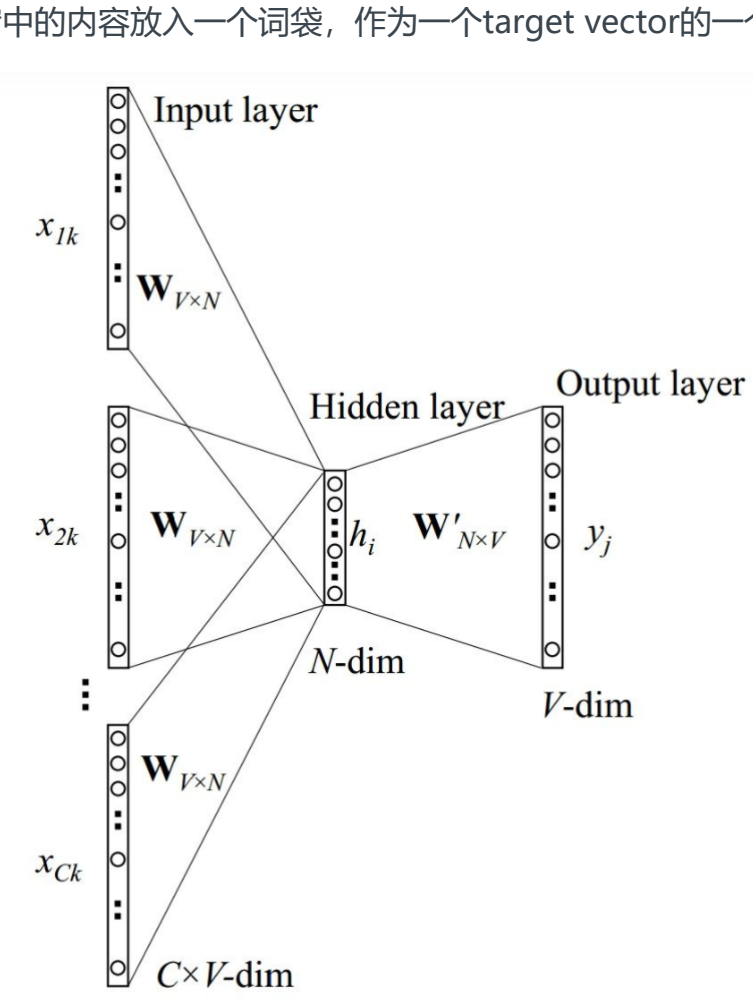


Figure 2: Continuous bag-of-word model

主要公式推导如下：

- 假设一个context_token里面有C个词(token), $t_1, t_2, \ldots, t_c$
- 输入$X = \sum_{i=1}^{c} t_i$
- 计算隐藏层$H = \frac{WI \cdot X}{C}$
- 输出层 $U = WO \cdot H$
- 预测概率 $Y = Softmax(U)$
- 损失函数 $E = -logp(w_O|w_I)$
- 权重更新公式
  - $v'_{w_j}(\text{ new }) = v'_{w_j}(old) - \eta \cdot e_j \cdot \mathbf{h} \quad \text{for } j = 1, 2, \cdots, V$
  - $v_{w_I,c}^{(new)} = v_{w_I,c}^{(old)} - \frac{1}{C} \cdot \eta \cdot \mathrm{EH}^T \quad \text{for } c = 1, 2, \cdots, C.$

代码实现

主要思路就是按照提示的四个步骤依次构建输入的one-hot向量（是context_tokens的叠加），前向传播计算隐藏层H，计算输出层U ,计算softmax， 然后计算损失(loss和EH），最后更新W1 W2参数。

```
        def train_one_step(self, context_tokens: List[str], target_token: str, learning_rate: float) -> float:
            """
            Predict the probability of the target token given context tokens.

            :param context_tokens:  List of tokens around the target token
            :param target_token:    Target (center) token
            :param learning_rate:   Learning rate of each step
            :return:    loss of the target token
            """
            # ==== TODO: Construct one-hot vectors ====
            V = len(self.vocab)
            input_X = np.zeros(V) # (V,)
            for token in context_tokens:
                token_index = self.vocab.token_to_idx(token)
                input_X[token_index] += 1
            target_index = self.vocab.token_to_idx(target_token)
            target = one_hot(V, target_index) # (V,)

            # ==== TODO: Forward step ====
            H = np.dot(input_X, self.W1) / len(context_tokens) # (N,)
            U = np.dot(H, self.W2) # (V,)
            predict_target = softmax(U) # (V,)

            # ==== TODO: Calculate loss ====
            loss = - np.log(np.dot(predict_target, target))
            EH = np.dot(self.W2, np.transpose(predict_target - target)) # (N,)

            # ==== TODO: Update parameters ====
            self.W1 -= input_X[:, None] * EH * learning_rate / len(context_tokens)
            self.W2 -= np.outer(H, predict_target - target) * learning_rate

            return loss
```

输出结果

test1

```
Token number: 50
Vocab size: 21
Epoch 1, loss: 2.96. Cost 0.0 min
Epoch 2, loss: 1.99. Cost 0.0 min
Epoch 3, loss: 1.46. Cost 0.0 min
Epoch 4, loss: 1.16. Cost 0.0 min
Epoch 5, loss: 0.94. Cost 0.0 min
Epoch 6, loss: 0.82. Cost 0.0 min
Epoch 7, loss: 0.74. Cost 0.0 min
Epoch 8, loss: 0.70. Cost 0.0 min
Epoch 9, loss: 0.82. Cost 0.0 min
Epoch 10, loss: 1.09. Cost 0.0 min
[('she', 1.0), ('now', 0.9166363811901059), ('i', 0.7916710150878732), ('he',
0.7600563117313468), ('will', 0.7217754180931802)]
```

test2

```
Token number: 205068
Vocab size: 17832
(省略训练部分详细输出)
 ...
Epoch 1, loss: 8.54. Cost 10.7 min
 ...
Epoch 2, loss: 7.71. Cost 10.8 min
 ...
Epoch 3, loss: 7.47. Cost 10.9 min
 ...
Epoch 4, loss: 7.32. Cost 10.9 min
 ...
Epoch 5, loss: 7.22. Cost 11.0 min
 ...
Epoch 6, loss: 7.14. Cost 10.8 min
 ...
Epoch 7, loss: 7.07. Cost 9.7 min
 ...
Epoch 8, loss: 7.02. Cost 9.4 min
 ...
Epoch 9, loss: 6.97. Cost 10.2 min
 ...
Epoch 10, loss: 6.92. Cost 9.7 min
 ...
Save model to ckpt
Cost 104.2 min
[('i', 0.9999999999999999), ('we', 0.91241699447562 82), ('levy',
0.9051181652637497), ('baboon', 0.8826970358235133), ('adrift', 0.87427171941447 37),
('you', 0.8730728436218511), ('jim', 0.8700385565478237), ('lounge',
0.8690574718409811), ('they', 0.8339341158975251), ('discord', 0.8330665504068829)]
```

test3

```
Load model from ckpt
spearman correlation: 0.397
pearson correlation: 0.551
```