Word2Vec 实验报告

实验要求

实现 Word2Vec 并使用 SGD 进行模型学习，获得词向量

- **Word2Vec 模型实现 (word2vec.py)：完成后，通过运行 python word2vec.py 进行测试。**
    1. 完成 sigmoid 函数
    2. 完成其中的 softmax 和 negative sampling 函数的 LossAndGradient 函数
    3. 完成 skig-gram 函数
- **在 sgd.py 中完成 SGD 优化器的实现。完成后，通过运行 python sgd.py 进行测试。**
- **加载实际数据并训练词向量：我们使用 SST 数据集来训练词向量，然后将其应用于情感分析任务（不需要写代码）。**
    1. 获取数据集：运行 sh get datasets.sh
    2. 进行训练：运行 python run.py
       注意：训练过程可能需要很长时间，具体取决于机器速度（大约需要一个小时）。经过 40,000 次迭代，训练完成，脚本将对词向量进行可视化并把图像输出到 word vectors.png 。写一份报告介绍你的观察结果。

环境

Windows11、Pycharm、jupyter、matplotlib、numpy、scikit-learn、python=3.7

实验过程

1. **Word2Vec 模型实现 (word2vec.py)**
    1. 完成 sigmoid 函数

       由 sigmoid 函数公式：

$$S(x) = \frac{1}{1+e^{-x}}$$

       可得代码实现：

```python
def sigmoid(x):
    """
    Compute the sigmoid function for the input here.
    Arguments:
    x -- A scalar or numpy array.
    Return:
    s -- sigmoid(x)
    """

    ### YOUR CODE HERE
    s = 1. / (1. + np.exp(-x))
    ### END YOUR CODE

    return s
```

    2. 完成其中的 softmax 和 negative sampling 函数的 LossAndGradient 函数

- 完成 softmax 函数的 LossAndGradient 函数

  在 Word2Vec 中，条件概率分布是通过采用向量点积并应用 softmax 函数给出的，公式如下：

$$P(O=o \mid C=c)=\frac{\exp \left(\boldsymbol{u}_o^{\top} \boldsymbol{v}_c\right)}{\sum_{w \in \text{Vocab}} \exp \left(\boldsymbol{u}_w^{\top} \boldsymbol{v}_c\right)}$$

  由于 one-hot 向量只有有标记的维度为1，其余维度均为0，于是下式等价：

$$-\sum_{w \in \text { Vocab }} y_w \log \left(\hat{y}_w\right)=-\log \left(\hat{y}_o\right)$$

  于是可得采用 naive-softmax 的损失函数：

$$\boldsymbol{J}_{\text{naive-softmax }}\left(\boldsymbol{v}_c, o, \boldsymbol{U}\right)=-\log P(O=o \mid C=c)$$

  $\boldsymbol{J}_{\text{naive-softmax }}\left(\boldsymbol{v}_c, o, \boldsymbol{U}\right)$ 对 v_c 和 U 的偏导数分别如下：

$$\frac{\partial \boldsymbol{J}}{\partial \boldsymbol{v}_c}=U(\hat{\boldsymbol{y}}-\boldsymbol{y})$$
$$\frac{\partial \boldsymbol{J}}{\partial \boldsymbol{U}}=\boldsymbol{v}_c(\hat{\boldsymbol{y}}-\boldsymbol{y})$$

  可得代码实现：

```
def naiveSoftmaxLossAndGradient(
        centerWordVec,
        outsideWordIdx,
        outsideVectors,
        dataset
):
    """ Naive Softmax loss & gradient function for word2vec models

    Implement the naive softmax loss and gradients between a center
word's
    embedding and an outside word's embedding. This will be the
building block
    for our word2vec models.

    Arguments:
    centerWordVec -- numpy ndarray, center word's embedding
                    (v_c in the pdf handout)
    outsideWordIdx -- integer, the index of the outside word
                    (o of u_o in the pdf handout)
    outsideVectors -- outside vectors (rows of matrix) for all words
in vocab
                      (U in the pdf handout)
    dataset -- needed for negative sampling, unused here.

    Return:
    loss -- naive softmax loss
    gradCenterVec -- the gradient with respect to the center word
vector
                     (dJ / dv_c in the pdf handout)
    gradOutsideVecs -- the gradient with respect to all the outside
word vectors
                    (dJ / dU)
    """
```

```python
    ### YOUR CODE HERE

    ### Please use the provided softmax function (imported earlier in this file)
    ### This numerically stable implementation helps you avoid issues pertaining
    ### to integer overflow.

    # 矩阵维度
    # centerWordVec:  (embedding_dim,1)
    # outsideVectors: (vocab_size,embedding_dim)
    # 参数初始化
    loss = 0.0
    gradCenterVec = np.zeros(centerWordVec.shape)
    gradOutsideVecs = np.zeros(outsideVectors.shape)

    temp = np.matmul(outsideVectors, centerWordVec)
    probability = softmax(temp)
    loss = -np.log(probability[outsideWordIdx])  # 等价
    d_temp = probability.copy()
    d_temp[outsideWordIdx] = d_temp[outsideWordIdx] - 1  # outside词向量的期望减去当前词向量，即y_hat - y
    gradCenterVec = np.matmul(outsideVectors.T, d_temp)
    gradOutsideVecs = np.outer(d_temp, centerWordVec)
    ### END YOUR CODE

    return loss, gradCenterVec, gradOutsideVecs
```

- 完成 negative sampling 函数的 LossAndGradient 函数

  sigmoid 函数公式如下：

$$\sigma(\boldsymbol{x})=\frac{1}{1+e^{-\boldsymbol{x}}}=\frac{e^{\boldsymbol{x}}}{e^{\boldsymbol{x}}+1}$$

其对向量 $\boldsymbol{x}$ 的导数如下：

$$\sigma'(\boldsymbol{x})=\begin{bmatrix}\sigma\left(x_1\right)\left(1-\sigma\left(x_1\right)\right)\\ \sigma\left(x_2\right)\left(1-\sigma\left(x_2\right)\right)\\ \cdots\\ \sigma\left(x_m\right)\left(1-\sigma\left(x_m\right)\right)\end{bmatrix}$$

由于采用 negative sampling 的损失函数公式如下：

$$\boldsymbol{J}_{\text{neg-sample}}\left(\boldsymbol{v}_c, o, \boldsymbol{U}\right)=-\log\left(\sigma\left(\boldsymbol{u}_o^{\top}\boldsymbol{v}_c\right)\right)-\sum_{k=1}^{K}\log\left(\sigma\left(-\boldsymbol{u}_k^{\top}\boldsymbol{v}_c\right)\right)$$

求出其对 $\boldsymbol{v}_c$ 和 $\boldsymbol{u}_o, \boldsymbol{u}_k$ 的偏导数：

$$\begin{aligned}\frac{\partial J_{\text{neg-sample}}}{\partial \boldsymbol{v}_c}&=\left(\sigma\left(\boldsymbol{u}_o^T\boldsymbol{v}_c\right)-1\right)\boldsymbol{u}_o+\sum_{k=1}^{K}\left(1-\sigma\left(-\boldsymbol{u}_k^T\boldsymbol{v}_c\right)\right)\boldsymbol{u}_k\\ \frac{\partial J_{\text{neg-sample}}}{\partial \boldsymbol{u}_o}&=\left(\sigma\left(\boldsymbol{u}_o^T\boldsymbol{v}_c\right)-1\right)\boldsymbol{v}_c\\ \frac{\partial J_{\text{neg-sample}}}{\partial \boldsymbol{u}_k}&=\left(1-\sigma\left(-\boldsymbol{u}_k^T\boldsymbol{v}_c\right)\right)\boldsymbol{v}_c\end{aligned}$$

可得代码实现：

```
def negSamplingLossAndGradient(
        centerWordVec,
        outsideWordIdx,
        outsideVectors,
        dataset,
        K=10
):
    """ Negative sampling loss function for word2vec models

    Implement the negative sampling loss and gradients for a
centerWordVec
    and a outsideWordIdx word vector as a building block for
word2vec
    models. K is the number of negative samples to take.

    Note: The same word may be negatively sampled multiple times.
For
    example if an outside word is sampled twice, you shall have to
    double count the gradient with respect to this word. Thrice if
    it was sampled three times, and so forth.

    Arguments/Return Specifications: same as
naiveSoftmaxLossAndGradient
    """

    # Negative sampling of words is done for you. Do not modify this
if you
    # wish to match the autograder and receive points!
    negSampleWordIndices = getNegativeSamples(outsideWordIdx,
dataset, K)
    indices = [outsideWordIdx] + negSampleWordIndices

    ### YOUR CODE HERE

    ### Please use your implementation of sigmoid in here.
    # 参数初始化
    loss = 0.0
    gradCenterVec = np.zeros(centerWordVec.shape)
    gradOutsideVecs = np.zeros(outsideVectors.shape)

    u_o = outsideVectors[outsideWordIdx]
    temp_1 = sigmoid(np.dot(u_o, centerWordVec))
    loss -= np.log(temp_1)
    gradCenterVec += u_o * (temp_1 - 1)
    gradOutsideVecs[outsideWordIdx] = centerWordVec * (temp_1 - 1)

    for i in range(K):
        neg_id = indices[i + 1]
        u_k = outsideVectors[neg_id]
        temp_2 = sigmoid(-np.dot(u_k, centerWordVec))
        loss -= np.log(temp_2)
        gradCenterVec += u_k * (1 - temp_2)
        gradOutsideVecs[neg_id] += centerWordVec * (1 - temp_2)
    ### END YOUR CODE
    return loss, gradCenterVec, gradOutsideVecs
```

3. 完成 skig-gram 函数

   由题意，上下文窗口的全部的 loss 如下式：

$$\boldsymbol{J}_{\text{skip-gram}}\left(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U}\right)=\sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \boldsymbol{J}\left(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}\right)$$

   其对 $\boldsymbol{U}, \boldsymbol{v_c}, \boldsymbol{v_w}$ 的偏导数如下：

$$\begin{aligned}
\frac{\partial J_{\text{skip-gram}}}{\partial \boldsymbol{U}} &= \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \frac{\partial J\left(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}\right)}{\partial \boldsymbol{U}} \\
\frac{\partial J_{\text{skip-gram}}}{\partial \boldsymbol{v}_c} &= \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \frac{\partial J\left(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}\right)}{\partial \boldsymbol{v}_c} \\
\frac{\partial J_{\text{skip-gram}}}{\partial \boldsymbol{v}_w} &= 0, \text{ when } w \neq c
\end{aligned}$$

   可得代码实现：

```
def skipgram(currentCenterWord, windowSize, outsideWords, word2Ind,
             centerWordVectors, outsideVectors, dataset,
             word2vecLossAndGradient=naiveSoftmaxLossAndGradient):
    """ Skip-gram model in word2vec

    Implement the skip-gram model in this function.

    Arguments:
    currentCenterWord -- a string of the current center word
    windowSize -- integer, context window size
    outsideWords -- list of no more than 2*windowSize strings, the
outside words
    word2Ind -- a dictionary that maps words to their indices in
              the word vector list
    centerWordVectors -- center word vectors (as rows) for all words in
vocab
                        (V in pdf handout)
    outsideVectors -- outside word vectors (as rows) for all words in
vocab
                    (U in pdf handout)
    word2vecLossAndGradient -- the loss and gradient function for
                               a prediction vector given the
outsideWordIdx
                               word vectors, could be one of the two
                               loss functions you implemented above.

    Return:
    loss -- the loss function value for the skip-gram model
            (J in the pdf handout)
    gradCenterVecs -- the gradient with respect to the center word
vectors
            (dJ / dV in the pdf handout)
    gradOutsideVectors -- the gradient with respect to the outside word
vectors
                        (dJ / dU in the pdf handout)
    """
    loss = 0.0
    gradCenterVecs = np.zeros(centerWordVectors.shape)
    gradOutsideVectors = np.zeros(outsideVectors.shape)

    ### YOUR CODE HERE
    central_id = word2Ind[currentCenterWord]
    centerWordVec = centerWordVectors[central_id]
    for word in outsideWords:
        outside_id = word2Ind[word]
        current_loss, current_gradCenter, current_gradOutside = \
        word2vecLossAndGradient(centerWordVec=centerWordVec,
                                outsideWordIdx=outside_id,
                                outsideVectors=outsideVectors,
                                dataset=dataset)
        loss += current_loss
        gradCenterVecs[central_id] += current_gradCenter
        gradOutsideVectors += current_gradOutside
    ### END YOUR CODE

    return loss, gradCenterVecs, gradOutsideVectors
```

2. **在sgd.py中完成SGD优化器的实现**

   理解原理后只需要实现两行代码即可：

```
def sgd(f, x0, step, iterations, postprocessing=None, useSaved=False,
        PRINT_EVERY=10):
    """ Stochastic Gradient Descent

    Implement the stochastic gradient descent method in this function.

    Arguments:
    f -- the function to optimize, it should take a single
         argument and yield two outputs, a loss and the gradient
         with respect to the arguments
    x0 -- the initial point to start SGD from
    step -- the step size for SGD
    iterations -- total iterations to run SGD for
    postprocessing -- postprocessing function for the parameters
                      if necessary. In the case of word2vec we will need to
                      normalize the word vectors to have unit length.
    PRINT_EVERY -- specifies how many iterations to output loss

    Return:
    x -- the parameter value after SGD finishes
    """

    # Anneal learning rate every several iterations
    ANNEAL_EVERY = 20000

    if useSaved:
        start_iter, oldx, state = load_saved_params()
        if start_iter > 0:
            x0 = oldx
            step *= 0.5 ** (start_iter / ANNEAL_EVERY)
        if state:
            random.setstate(state)
    else:
        start_iter = 0

    x = x0

    if not postprocessing:
        postprocessing = lambda x: x

    exploss = None

    for iter in range(start_iter + 1, iterations + 1):
        # You might want to print the progress every few iterations.

        loss = None
        ### YOUR CODE HERE
        loss, grad = f(x)
        x = x - grad * step  # 沿着梯度下降
        ### END YOUR CODE

        x = postprocessing(x)
        if iter % PRINT_EVERY == 0:
            if not exploss:
                exploss = loss
            else:
                exploss = .95 * exploss + .05 * loss
            print("iter %d: %f" % (iter, exploss))

        if iter % SAVE_PARAMS_EVERY == 0 and useSaved:
            save_params(iter, x)

        if iter % ANNEAL_EVERY == 0:
            step *= 0.5

    return x
```

3. **加载实际数据并训练词向量**

   在powershell 中执行下列语句：

```
python run.py
```

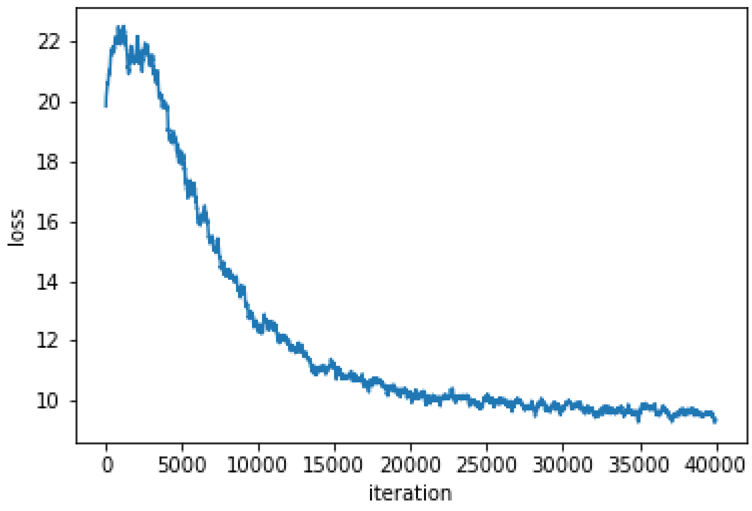
部分训练细节如下：

```
iter 10: 19.824024
iter 20: 20.136629
iter 30: 20.151500
iter 40: 20.211374
iter 50: 20.245379
iter 60: 20.300555
iter 70: 20.322475
iter 80: 20.495801
iter 90: 20.568950
iter 100: 20.531704
......
iter 39900: 9.351048
iter 39910: 9.324637
iter 39920: 9.284225
iter 39930: 9.298478
iter 39940: 9.296606
iter 39950: 9.313374
iter 39960: 9.317475
iter 39970: 9.330720
iter 39980: 9.410215
iter 39990: 9.418270
iter 40000: 9.367644
```

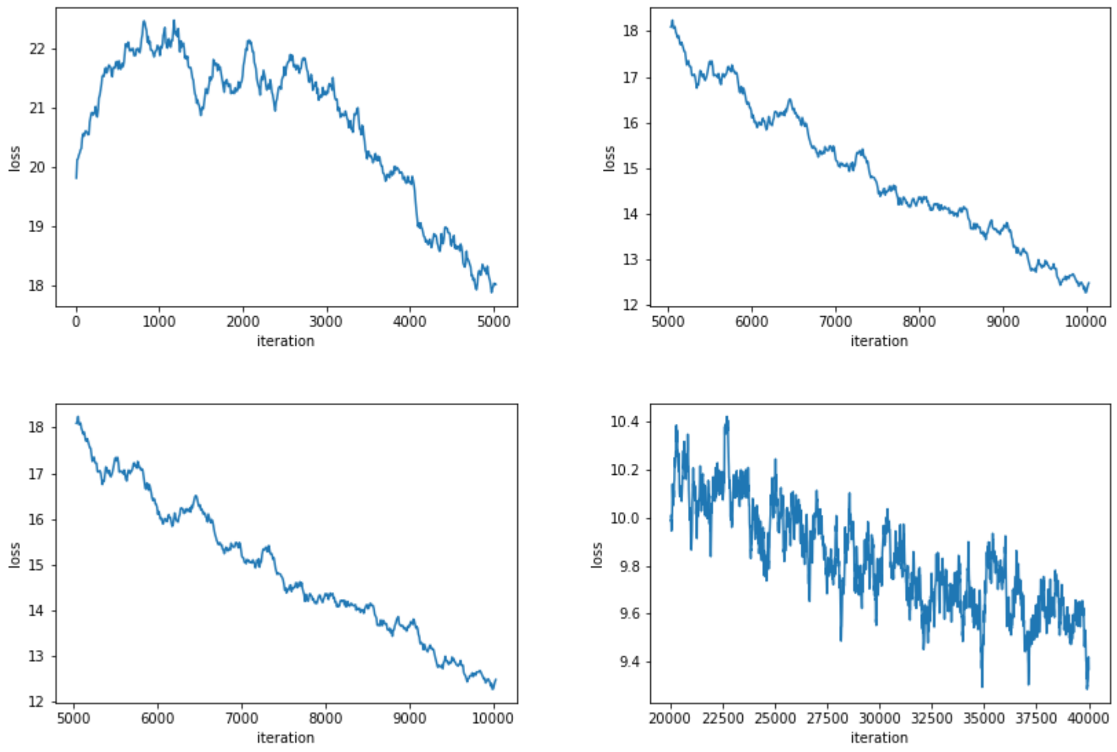
可见，通过 SGD 优化器，随着迭代的不断增加，模型的误差不断下降。

实验结果

训练时，随着迭代次数的增加，得到的误差图像如下：



局部误差图像如下（分别是 0-5, 000，5, 000-10, 000，10, 000-20, 000，20, 000-40, 000）：

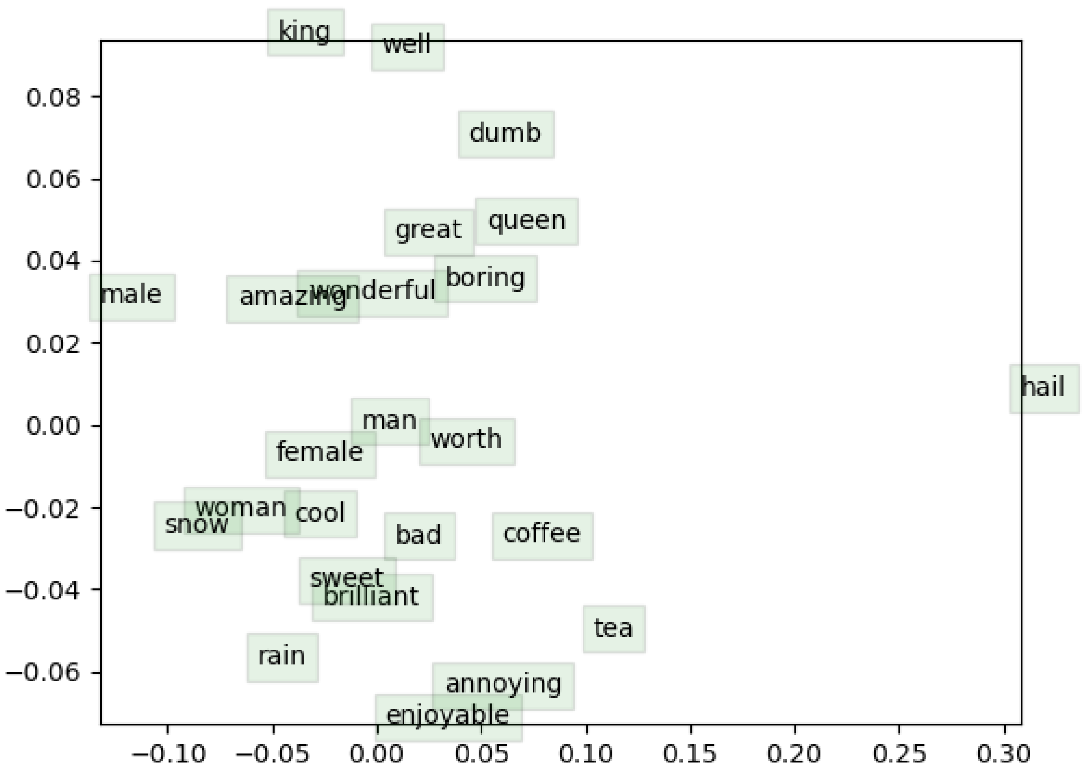


可见：

在迭代次数 0-5, 000 时，SGD 优化器并不是一瞬间便找到使误差下降的梯度，而是**四处探寻**，误差甚至还增加了许多，而后在迭代次数到达 3, 000 左右时，终于找到方向，一步一步减小误差。

从迭代次数 0-20, 000 时，误差下降的速度较快，幅度较大，而在 20, 000-40, 000 时，误差下降速度较慢，幅度较小，这说明模型的性能可能已经达到瓶颈，或是需要更改 hyperparameters 的取值以取得更好的效果。

经历 40, 000 次迭代后，输出的词向量可视化图像如下：



可见：

"amazing", "wonderful", "boring", "great" 等形容词聚集在一起，"coffee", "tea" 等饮品有关的名词聚集在一起，"sweet", "brilliant", "bad" 等形容词聚集在一起，而与其余单词没有什么联系的 "hail" 则在一旁。

论文中提及的 "**Additive Compositionality**" 也可以在图中见得——单词对 ("man", "king") 的斜率与单词对 ("female", "queen") 的斜率几乎相同，证明单词的向量表示间的 "**Linear Translations**" 是可以实现的。

其中，"woman" 与 "snow" 和 "cool" 聚集在一起，由常理似乎也可以讲得通（冷若冰山的女性，英姿飒爽的女性等），但我更倾向认为：由于此图是高维向量投影到二维平面，也许是通过迭代过程，三者的向量表示间在更高的维度中有许多特征相似（这是计算机学习到的，而人类可能无法理解为什么相似，这也是现在的机器/深度学习可解释性研究需要探讨的问题），因而被投影到低维时三者聚集在一起。