**CS 224n: Natural Language Processing with Deep Learning** **(Due: 01/21/20)**

# Submission Assignment #2 (word2vec (43 Points))

*Instructor:* Christopher Manning *Name:* Jianpan Gun

## Problem 1: Understanding word2vec (3+5+5+3+4+3=23 points)

Let's have a quick refresher on the `word2vec`algorithm. The key insight behind `word2vec`is that 'a word is known by the company it keeps'. Concretely, suppose we have a 'center' word c and a contextual window surrounding c. We shall refer to words that lie in this contextual window as 'outside words'. For example, in Figure 1 we see that the center word c is 'banking'. Since the context window size is 2, the outside words are 'turning', 'into', 'crises', and 'as'.

The goal of the skip-gram `word2vec`algorithm is to accurately learn the probability distribution $P(O|C)$. Given a specific word o and a specific word c, we want to calculate $P(O=o|C=c)$, which is the probability that word o is an 'outside' word for c, i.e., the probability that o falls within the contextual window of c.

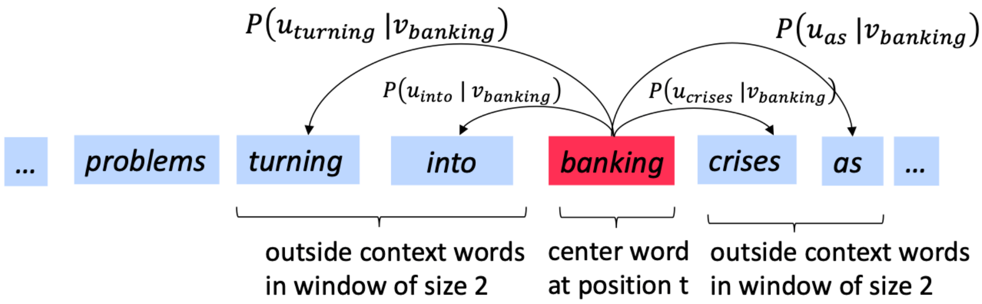


Figure 1: The `word2vec`skip-gram prediction model with window size 2.

In `word2vec`, the conditional probability distribution is given by taking vector dot-products and applying the softmax function:

$$P(O=o|C=c) = \frac{\exp\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right)}{\sum_{w\in \mathrm{V_{ocab}}} \exp\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c\right)} \tag{1}$$

Here, $\boldsymbol{u_o}$ is the 'outside' vector representing outside word $o$, and $\boldsymbol{v_c}$ is the 'center' vector representing center word $c$. To contain these parameters, we have two matrices, $\boldsymbol{U}$ and $\boldsymbol{V}$. The columns of $\boldsymbol{U}$ are all the 'outside' vectors $\boldsymbol{u_w}$. The columns of $\boldsymbol{V}$ are all of the 'center' vectors $\boldsymbol{v_w}$. Both $\boldsymbol{U}$ and $\boldsymbol{V}$ contain a vector for every $w \in$ Vocabulary.

Recall from lectures that, for a single pair of words $c$ and $o$, the loss is given by:

$$J_{\text{naive-softmax}}\left(\boldsymbol{v}_c, o, \boldsymbol{U}\right) = -\log P(O=o|C=c) \tag{2}$$

Another way to view this loss is as the cross-entropy between the true distribution $\boldsymbol{y}$ and the predicted distribution $\hat{\boldsymbol{y}}$. Here, both $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$ are vectors with length equal to the number of words in the vocabulary. Furthermore, the $k^{th}$ entry in these vectors indicates the conditional probability of the $k^{th}$ word being an 'outside word' for the given $c$. The true empirical distribution $\boldsymbol{y}$ is a one-hot vector with a 1 for the true outside word $o$, and 0 everywhere else. The predicted distribution $\hat{\boldsymbol{y}}$ is the probability distribution $P(O|C=c)$ given by our model in equation (1).

**(a)** (3 points) **Show that the naive-softmax loss given in Equation (2) is the same as the cross-entropy loss between $\boldsymbol{y}$ and $\hat{\boldsymbol{y}}$; i.e., show that** $-\sum_{w\in V_{ocab}} y_w \log\left(\hat{y}_w\right) = -\log\left(\hat{y}_o\right)$

**A**: Due to the $\boldsymbol{y_i}$ is a one-hot vector, that is, $\mathbb{I}_i$, which means that only have i-th dim is 1. So,

$$\mathcal{L}_{ce} = -\sum_{w \in V_{ocab}} y_w \log(\hat{y}_w) = -\boldsymbol{y}_o \log(\hat{\boldsymbol{y}}_o) = -\log(\hat{y}_o) \tag{3}$$

**(b)** (5 points) **Compute the partial derivative of $J_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})$ with respect to $\boldsymbol{v_c}$. Please write your answer in terms of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, and $\boldsymbol{U}$**

$$\begin{aligned}
\frac{\partial \mathcal{J}_{\text{naive-softmax}}}{\partial \boldsymbol{v}_c}(\boldsymbol{v}_c, o, \boldsymbol{U}) &= \frac{\partial}{\partial \boldsymbol{v}_c} - \log \frac{\exp\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right)}{\sum_{w \in \mathrm{V_{ocab}}} \exp\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c\right)} = -\boldsymbol{u}_o + \frac{\partial}{\partial \boldsymbol{v}_c} \log \sum_{w \in \mathrm{V_{ocab}}} \exp\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c\right) \\
&= -\boldsymbol{u}_o + \frac{1}{\sum_{w \in \mathrm{V_{ocab}}} \exp\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c\right)} \sum_{w \in \mathrm{V_{ocab}}} \exp\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c\right) \boldsymbol{u}_w \\
&= -\boldsymbol{y}^\top \boldsymbol{U} + \sum_{w \in \mathrm{V_{ocab}}} P(O = w | C = c) \boldsymbol{u}_w \\
&= -\boldsymbol{y}^\top \boldsymbol{U} + \hat{\boldsymbol{y}}^\top \boldsymbol{U} = (\hat{\boldsymbol{y}} - \boldsymbol{y})^\top \boldsymbol{U}
\end{aligned} \tag{4}$$

**(c)** (5 points) **Compute the partial derivatives of $J_{\text{naive-softmax}}(\boldsymbol{v}_c, o, \boldsymbol{U})$ with respect to each of the 'outside' word vectors, $\boldsymbol{u_w}$'s. There will be two cases: when $w = o$, the true 'outside' word vector; and $w \neq o$, for all other words. Please write you answer in terms of $\boldsymbol{y}$, $\hat{\boldsymbol{y}}$, and $\boldsymbol{v_c}$.**

$$\frac{\partial \mathcal{J}_{\text{naive-softmax}}}{\partial \boldsymbol{u}_w}(\boldsymbol{v}_c, o, \boldsymbol{U}) = \frac{\partial}{\partial \boldsymbol{u}_w} - \log \frac{\exp\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right)}{\sum_{w \in \mathrm{V_{ocab}}} \exp\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c\right)} = -\frac{\partial \boldsymbol{u}_o^\top \boldsymbol{v}_c}{\partial \boldsymbol{u}_w} + \frac{\partial}{\partial \boldsymbol{u}_w} \log \sum_{w \in \mathrm{V_{ocab}}} \exp\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c\right) \tag{5}$$

If $w = o$:

$$= -\boldsymbol{v}_c + \frac{\exp\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right) \boldsymbol{v}_c}{\sum_{w \in \mathrm{V_{ocab}}} \exp\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c\right)} = (P(O = o | C = c) - 1) \boldsymbol{v}_c \tag{6}$$

If $w \neq o$:

$$= \frac{\exp\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c\right) \boldsymbol{v}_c}{\sum_{w \in \mathrm{V_{ocab}}} \exp\left(\boldsymbol{u}_w^\top \boldsymbol{v}_c\right)} = P(O = w | C = c) \boldsymbol{v}_c \tag{7}$$

So,

$$= (\hat{\boldsymbol{y}} - \boldsymbol{y})^\top \boldsymbol{v}_c \tag{8}$$

**(d)** (3 Points) **The sigmoid function is given by Equation 9:**

$$\sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + 1} \tag{9}$$

**Please compute the derivative of $\sigma(x)$ with respect to x, where x is a scalar. Hint: you may want to write your answer in terms of $\sigma(x)$.**

$$\frac{\partial \sigma(x)}{\partial x} = \frac{\partial}{\partial x} \frac{e^x}{e^x + 1} = \frac{e^x \dot{(}e^x + 1) - e^x \dot{(}e^x + 1)'}{(e^x + 1)^2} = \frac{e^{2x}}{(e^x + 1)^2} = \sigma(x)(1 - \sigma(x)) \tag{10}$$

**(e)** (4 points) **Now we shall consider the Negative Sampling loss, which is an alternative to the Naive Softmax loss. Assume that K negative samples (words) are drawn from the vocabulary. For simplicity of notation we shall refer to them as $w_1, w_2, \ldots, w_K$ and their outside vectors as $u_1, \ldots, u_K$. Note that $o \notin w_1, \ldots, w_K$. For a center word c and an outside word $o$, the negative sampling loss function is given by:**

$$\boldsymbol{J}_{\text{neg-sample}}(\boldsymbol{v}_c, o, \boldsymbol{U}) = -\log\left(\sigma\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right)\right) - \sum_{k=1}^{K} \log\left(\sigma\left(-\boldsymbol{u}_k^\top \boldsymbol{v}_c\right)\right) \tag{11}$$

**for a sample $w_1, w_2, \ldots, w_K$, where $\sigma(\cdot)$ is the sigmoid function.**

**Please repeat parts (b) and (c), computing the partial derivatives of $\boldsymbol{J}_{\textbf{neg-sample}}$ with respect to $\boldsymbol{v_c}$, with respect to $\boldsymbol{u_o}$, and with respect to a negative sample $\boldsymbol{u_k}$. Please write your answers in terms of the vectors $\boldsymbol{u_o}$,$\boldsymbol{v_c}$, and $\boldsymbol{u_k}$, where $k \in [1, k]$. After you've done this, describe with one sentence why this loss function is much more efficient to compute than the naive-softmax loss. Note, you should be able to use your solution to part (d) to help compute the necessary gradients here.**

$$\begin{aligned}\frac{\partial \mathcal{J}_{\text{neg-sample}}}{\partial \boldsymbol{v}_c}\left(\boldsymbol{v}_c, o, \boldsymbol{U}\right) &= \frac{\partial}{\partial \boldsymbol{v}_c}[-\log\left(\sigma\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right)\right) - \sum_{k=1}^{K} \log\left(\sigma\left(-\boldsymbol{u}_k^\top \boldsymbol{v}_c\right)\right)] = -\frac{\sigma\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right)\left(1-\sigma\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right)\right)\boldsymbol{u}_o}{\sigma\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right)} \\ &+ \sum_{k=1}^{K}\left(1-\sigma\left(-\boldsymbol{u}_k^\top \boldsymbol{v}_c\right)\right)\boldsymbol{u}_k = -(1-\sigma\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right))\boldsymbol{u}_o + \sum_{k=1}^{K}\left(1-\sigma\left(-\boldsymbol{u}_k^\top \boldsymbol{v}_c\right)\right)\boldsymbol{u}_k\end{aligned} \tag{12}$$

$$\begin{aligned}\frac{\partial \mathcal{J}_{\text{neg-sample}}}{\partial \boldsymbol{u}_o}\left(\boldsymbol{v}_c, o, \boldsymbol{U}\right) &= \frac{\partial}{\partial \boldsymbol{u}_o}[-\log\left(\sigma\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right)\right) - \sum_{k=1}^{K} \log\left(\sigma\left(-\boldsymbol{u}_k^\top \boldsymbol{v}_c\right)\right)] = -\frac{\sigma\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right)\left(1-\sigma\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right)\right)\boldsymbol{v}_c}{\sigma\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right)} \\ &= -(1-\sigma\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right))\boldsymbol{v}_c\end{aligned} \tag{13}$$

$$\frac{\partial \mathcal{J}_{\text{neg-sample}}}{\partial \boldsymbol{u}_k}\left(\boldsymbol{v}_c, o, \boldsymbol{U}\right) = \frac{\partial}{\partial \boldsymbol{u}_k}[-\log\left(\sigma\left(\boldsymbol{u}_o^\top \boldsymbol{v}_c\right)\right) - \sum_{k=1}^{K} \log\left(\sigma\left(-\boldsymbol{u}_k^\top \boldsymbol{v}_c\right)\right)] = \left(1-\sigma\left(-\boldsymbol{u}_k^\top \boldsymbol{v}_c\right)\right)\boldsymbol{v}_c \tag{14}$$

**(f)** (3 points) **Suppose the center word is $c = wt$ and the context window is $w_{t-m}, \ldots, w_{t-1}, w_t, w_{t+1}, \ldots, w_{t+m}$, where m is the context window size. Recall that for the skip-gram version of `word2vec`, the total loss for the context window is:**

$$J_{\text{skip-gram}}\left(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U}\right) = \sum_{\substack{-m\le j\le m \\ j\ne 0}} \boldsymbol{J}\left(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}\right) \tag{15}$$

**Here, $J\left(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}\right)$ represents an arbitrary loss term for the center word $c = wt$ and outside word $w_{t+j}$. $J\left(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}\right)$ could be $J_{\textbf{neg-sample}}\left(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}\right)$ or $J_{\textbf{naive-softmax}}\left(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}\right)$, depending on your implementation.**

**Write down three partial derivatives:**

(I) $\partial \boldsymbol{J}_{\text{skip-gram}}\ \left(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U}\right)/\partial \boldsymbol{U}$

(II) $\partial \boldsymbol{J}_{\text{skip-gram}}\ \left(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U}\right)/\partial \boldsymbol{v}_c$

(III) $\partial \boldsymbol{J}_{\text{skip-gram}}\ \left(\boldsymbol{v}_c, w_{t-m}, \ldots w_{t+m}, \boldsymbol{U}\right)/\partial \boldsymbol{v}_w$, when $w \neq c$

**Write your answers in terms of $\partial J\left(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}\right)/\partial U$ and $\partial J\left(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}\right)/\partial v_c$. This is very simple. Each solution should be one line. Once you're done: Given that you computed the derivatives of $J\left(\boldsymbol{v}_c, w_{t+j}, \boldsymbol{U}\right)$ with respect to all the model parameters $\boldsymbol{U}$ and $\boldsymbol{V}$ in parts (a) to (c), you have now computed the derivatives of the full loss function $J_{skip-gram}$ with respect to all parameters. You're ready to implement `word2vec`!**

(I)

For native-softmax:

$$\frac{\partial J_{\text{skip-gram}}}{\partial \boldsymbol{U}}\left(\boldsymbol{v}_c, o, \boldsymbol{U}\right) = \sum_{\substack{-m\le j\le m \\ j\ne 0}} \left(\hat{\boldsymbol{y}} - \boldsymbol{y}\right)^\top \boldsymbol{v}_j \tag{16}$$

For neg-sample:

$$\frac{\partial J_{\text{skip-gram}}}{\partial \boldsymbol{U}}\left(\boldsymbol{v}_c, o, \boldsymbol{U}\right) = \sum_{\substack{-m\le j\le m \\ j\ne 0}} (1-\sigma\left(\boldsymbol{u}_o^\top \boldsymbol{v}_j\right))\boldsymbol{v}_j \tag{17}$$

(II)

For native-softmax:

$$\frac{\partial J_{\text{skip-gram}}}{\partial \boldsymbol{v}_c}\left(\boldsymbol{v}_c, o, \boldsymbol{U}\right) = \sum_{\substack{-m\le j\le m \\ j\ne 0}} \left(\hat{\boldsymbol{y}} - \boldsymbol{y}\right)^\top \boldsymbol{U} \tag{18}$$

For neg-sample:

$$\frac{\partial J_{\text{skip-gram}}}{\partial \boldsymbol{v}_c}(\boldsymbol{v}_c, o, \boldsymbol{U}) = \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \left(-(1-\sigma\left(\boldsymbol{u}_o^\top \boldsymbol{v}_j\right))\boldsymbol{u}_o + \sum_{k=1}^{K}\left(1-\sigma\left(-\boldsymbol{u}_k^\top \boldsymbol{v}_j\right)\right)\boldsymbol{u}_k\right) \tag{19}$$

(III)

$$\frac{\partial J_{\text{skip-gram}}}{\partial \boldsymbol{v}_w}(\boldsymbol{v}_c, o, \boldsymbol{U}) = 0 \tag{20}$$

## Problem Implementing word2vec (20 points)

(12+4+4=20 points)

**(a)** (12 points) **We will start by implementing methods in** `word2vec.py`. **First, implement the** `sigmoid` **method, which takes in a vector and applies the sigmoid function to it. Then implement the softmax loss and gradient in the** `naiveSoftmaxLossAndGradient` **method, and negative sampling loss and gradient in the** `negSamplingLossAndGradient` **method. Finally, fill in the implementation for the skip-gram model in the** `skipgram` **method. When you are done, test your implementation by running python** `word2vec.py`

Listing 1: word2vec.py function.

```python
def sigmoid(x):
    # \sigma(x)=\frac{1}{1+e^{-x}}=\frac{e^{x}}{e^{x}+1}
    s = 1 / (1 + np.exp(-x))
    return s

def naiveSoftmaxLossAndGradient(centerWordVec, outsideWordIdx, outsideVectors, dataset):
    y_hat = softmax(np.dot(centerWordVec, outsideVectors.T))

    loss = -np.log(y_hat[outsideWordIdx]) # ref to equation(2)
    y = np.zeros_like(y_hat)
    y[outsideWordIdx] = 1
    gradCenterVec = np.dot((y_hat - y).T, outsideVectors) # ref to equation(4)
    gradOutsideVecs = np.dot(
        (y_hat - y).reshape(-1, 1), centerWordVec.reshape(1, -1)
    )  # (T, 1) * (1, H) ref to equation(8)

    return loss, gradCenterVec, gradOutsideVecs

def negSamplingLossAndGradient(
    centerWordVec, outsideWordIdx, outsideVectors, dataset, K=10
):
    negSampleWordIndices = getNegativeSamples(outsideWordIdx, dataset, K)
    indices = [outsideWordIdx] + negSampleWordIndices

    negSampleWordIndices = np.array(negSampleWordIndices)

    pos = sigmoid(np.dot(outsideVectors[outsideWordIdx], centerWordVec))
    neg = sigmoid(-np.dot(outsideVectors[negSampleWordIndices], centerWordVec))
    loss = -np.log(pos) - np.log(neg).sum() # ref to equation(11)
    gradCenterVec = -np.dot((1 - pos), outsideVectors[outsideWordIdx]) + np.dot(
        1 - neg, outsideVectors[negSampleWordIndices]
    ) # ref to equation(12)
    gradOutsideVecs = np.zeros_like(outsideVectors)
    gradOutsideVecs[outsideWordIdx] = - np.dot(1 - pos, centerWordVec) # ref to equation(13)
    gradOutside = np.dot((1 - neg).reshape(-1, 1), centerWordVec.reshape(1, -1))
    for idx, neg_idx in enumerate(negSampleWordIndices):
        gradOutsideVecs[neg_idx] += gradOutside[idx] # ref to equation(14)
```

```python

    return loss, gradCenterVec, gradOutsideVecs

def skipgram(
    currentCenterWord,
    windowSize,
    outsideWords,
    word2Ind,
    centerWordVectors,
    outsideVectors,
    dataset,
    word2vecLossAndGradient=naiveSoftmaxLossAndGradient,
):
    loss = 0.0
    gradCenterVecs = np.zeros(centerWordVectors.shape)
    gradOutsideVectors = np.zeros(outsideVectors.shape)

    centerWordIdx = word2Ind[currentCenterWord]
    centerWordVector = centerWordVectors[centerWordIdx]
    for outsideWord in outsideWords:
        outsideWordIdx = word2Ind[outsideWord]
        tmp_loss, gradCenterVec, gradOutsideVector = word2vecLossAndGradient(
            centerWordVector, outsideWordIdx, outsideVectors, dataset
        )
        loss += tmp_loss
        gradCenterVecs[centerWordIdx, :] += gradCenterVec # ref to equation(18,19)
        gradOutsideVectors += gradOutsideVector

    return loss, gradCenterVecs, gradOutsideVectors
```

**(b)** (4 points) **Complete the implementation for your SGD optimizer in the sgd method of sgd.py. Test your implementation by running python sgd.py.**

Listing 2: sgd.py function.

```python

def sgd(f, x0, step, iterations, postprocessing=None, useSaved=False, PRINT_EVERY=10):
    # 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.

```

```
        # c, gin, gout = word2vecModel(
        #     centerWord,
        #     windowSize1,
        #     context,
        #     word2Ind,
        #     centerWordVectors,
        #     outsideVectors,
        #     dataset,
        #     word2vecLossAndGradient,
        # )
        # loss += c / batchsize
        # grad[: int(N / 2), :] += gin / batchsize
        # grad[int(N / 2) :, :] += gout / batchsi
        loss, grad = f(x)
        x -= step * grad

        x = postprocessing(x)
        if iter % PRINT_EVERY == 0:
            if not exploss:
                exploss = loss
            else:
                exploss = 0.95 * exploss + 0.05 * 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
```

**(c)** (4 points) **Show time! Now we are going to load some real data and train word vectors with everything you just implemented! We are going to use the** `Stanford Sentiment Treebank (SST)` **dataset to train word vectors, and later apply them to a simple sentiment analysis task. You will need to fetch the datasets first. To do this, run sh get** `datasets.sh`**. There is no additional code to write for this part; just run python** `run.py`**.**

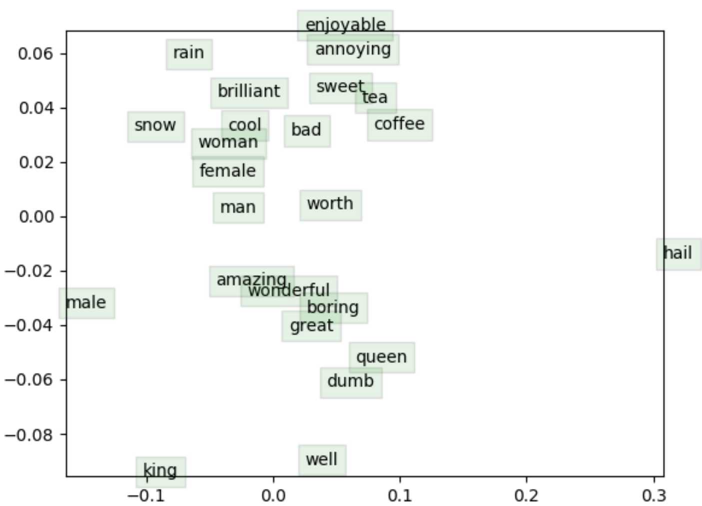


Figure 2: The word vector distribution figure.