Word Vectors

2023-03-28

Outline

- Task Introduction
- Algorithm Introduction
  - Skip-gram
  - Negative Sampling
- Some findings in the code
- Experiment Result
- Thoughts & Pretrained Model Results

Outline

- Task Introduction
- Algorithm Introduction
  - Skip-gram
  - Negative Sampling
- Some findings in the code
- Experiment Result
- Thoughts & Pretrained Model Results

1 Task Introduction

- Definition of **Word Vector Models**
- Part I
  - Train **Skip-gram** Word2Vec Model
- Part II
  - Sentiment Classification
  - With **Softmax Regression** & **SGD**

Outline

- Task Introduction
- Algorithm Introduction
  - Skip-gram
  - Sample Rate & Negative Sampling
- Some findings in the code
- Experiment Result
- Thoughts & Pretrained Model Results

2.1 Skip-gram

Idea: Use the **similarity** of the word to calculate the probability

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

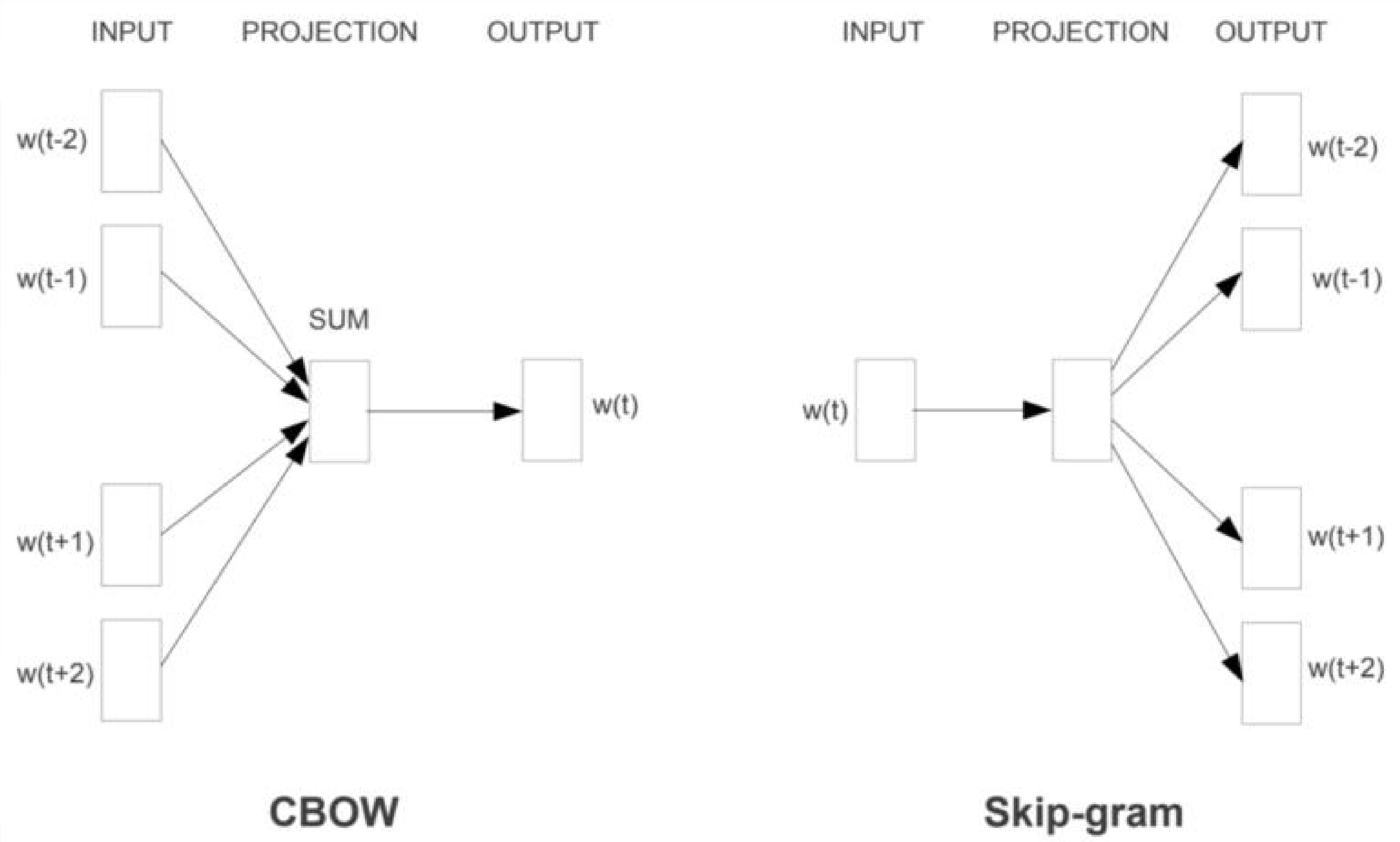
Pros:

1. Maps discrete word symbols to **continuous** vectors in a relatively **low dimensional** space
2. allow the model to **capture similarity** between words

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \leq j \leq m, j \neq 0} \log P(w_{t+j} | w_t; \theta)$$

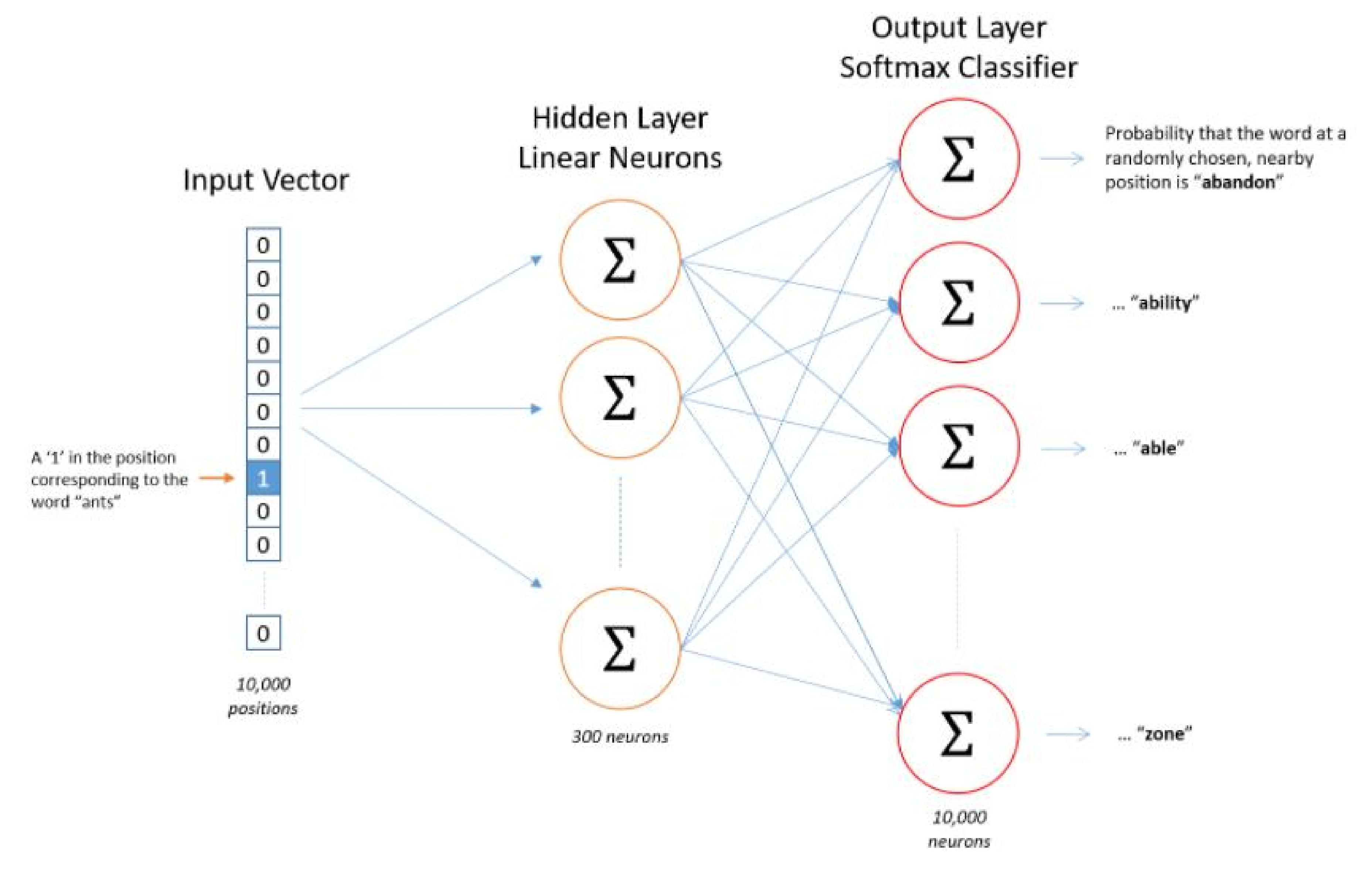
2.1 Skip-gram

CBOW V.S. Skip-gram



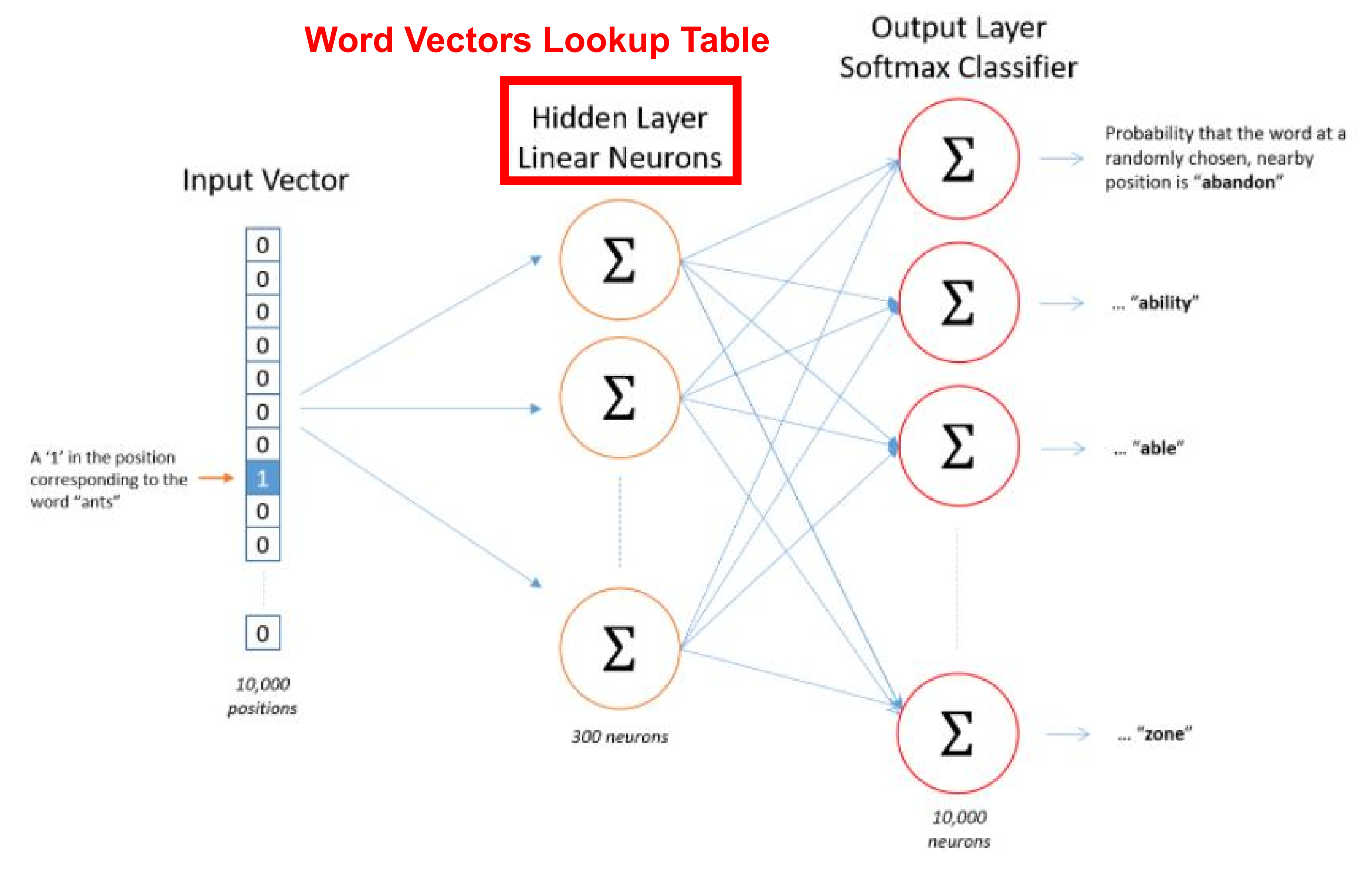
2.1 Skip-gram

Architecture of the neural network



2.1 Skip-gram

Architecture of the neural network



2.1 Skip-gram

Gradients

Objective:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0}\log P(w_{t+j}|w_t;\theta)$$

For center words:
(center word)

$$\frac{\partial J(\theta)}{\partial V_c} = -u_o + \sum_{w=1}^{V} P(w|c)u_w$$

For other words:
(context word)

$$\frac{\partial J(\theta)}{\partial u_w} = P(w|c)v_c = \frac{\exp(u_w^T v_c)}{\sum_{i=1}^{V}\exp(u_i^T v_c)}v_c$$

For outside words:
(context word)

$$\frac{\partial J(\theta)}{\partial u_o} = P(o|c)v_c - v_c = \left(\frac{\exp(u_o^T v_c)}{\sum_{i=1}^{V}\exp(u_i^T v_c)} - 1\right)v_c$$

2.2 Subsampling & Negative Sampling

Problems:

Lots of weights to train → slow to train & data hungry

$$|V| \times d$$

V-the vocabulary size, d-the dimension of the vector

Solutions:

1. Subsampling Frequent Words

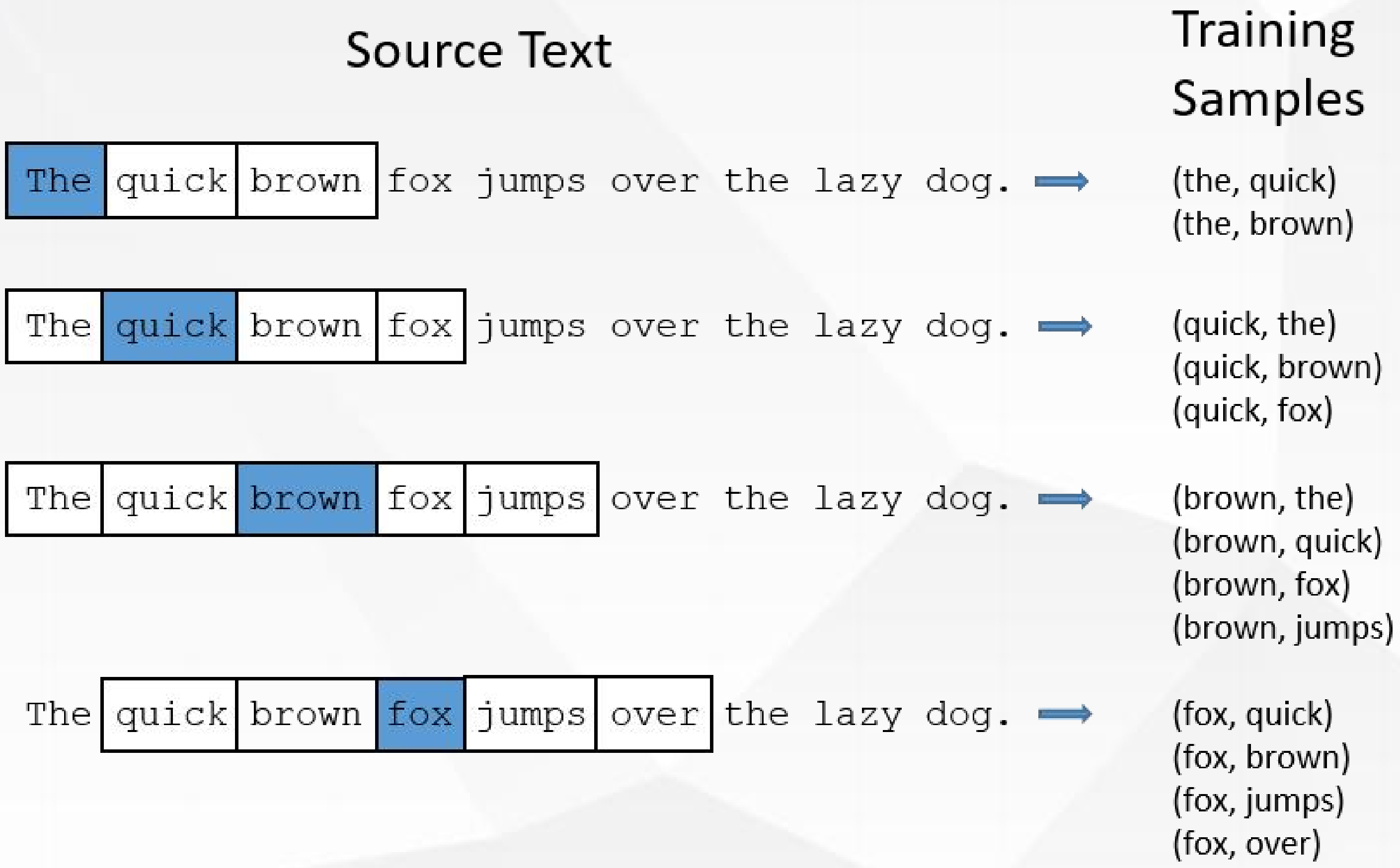
Delete a word in the training text by a chance related to its frequency

2. Negative Sampling

Each training sample only modify a small percentage of the weights, rather than all of them

2.2.1 Subsampling

Idea: Delete a word in the training text by a chance related to its frequency



Common words will:

1. Give less meanings

2. Appear too many times

2.2.1 Subsampling

Idea: Delete a word in the training text by a chance related to its frequency

The code given us has the sampling rate:

$$P_{reject}(w_i) = \max\left(0, 1 - \sqrt{\frac{10^{-5} \times M}{freq(w_i)}}\right)$$

M-the total number of the word tokens in the document

freq(x)-the count of the word token in the document

2.2.2 Negative Sampling

Idea: train **binary logistic regressions** for a true pair (center word and word in its context window) versus several noise pairs (the center word paired with a random word)

$$J(\theta) = -\log(\sigma(u_o^T v_c)) - \sum_{w=1}^{K} \log(\sigma(-u_w^T v_c))$$

Picking Probability:

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{M} \left(f(w_j)^{3/4}\right)}$$

M-the total number of the word tokens in the document

f(x)-the count of the word token in the document

2.2.2 Negative Sampling

Gradients

Objective:

$$J(\theta) = -\log(\sigma(u_o^T v_c)) - \sum_{w=1}^{K} \log(\sigma(-u_w^T v_c))$$

For center words:
(center word)

$$\frac{\partial J(\theta)}{\partial v_c} = \left[\sigma(u_o^T v_c) - 1\right] u_o + \sum_{w=1}^{K} \left[1 - \sigma(-u_w^T v_c)\right] u_w$$

For other words:
(context word)

$$\frac{\partial J(\theta)}{\partial u_w} = \left[1 - \sigma(-u_w^T v_c)\right] v_c$$

For outside words:
(context word)

$$\frac{\partial J(\theta)}{\partial u_o} = \left[\sigma(u_o^T v_c) - 1\right] v_c$$

Outline

- Task Introduction
- Algorithm Introduction
  - Skip-gram
  - Negative Sampling
  - Algorithm Pipeline
- Some findings in the code
- Experiment Result
- Thoughts & Pretrained Model Results

3.1 Findings in Algorithm Pipeline

Preprocessing (pre-splitted data)

Word2Vec Training:

- **Minibatch (size-50)**
- **Window Size resample (speed up training?)**

Sentiment Classification:

- **Minibatch (size-50)**
- **Find best regularization value**

3.2 Findings in the preprocess

data_utils.py

```python
def getRandomContext(self, C=5):
    allsent = self.allSentences()
    sentID = random.randint(0, len(allsent) - 1)    # 随机选了一个句子
    sent = allsent[sentID]
    wordID = random.randint(0, len(sent) - 1)       # 随机选了一个单词

    context = sent[max(0, wordID - C):wordID]
    if wordID+1 < len(sent):
        context += sent[wordID+1:min(len(sent), wordID + C + 1)]

    centerword = sent[wordID]
    context = [w for w in context if w != centerword]   # 提取centerwords

    if len(context) > 0:
        return centerword, context
    else:
        return self.getRandomContext(C) # 为空则直接重新生成一遍
```

3.2 Findings in the preprocess

data_utils.py

```python
def getRandomContext(self, C=5):
    allsent = self.allSentences()
    sentID = random.randint(0, len(allsent) - 1)    # 随机选了一个句子
    sent = allsent[sentID]
    wordID = random.randint(0, len(sent) - 1)       # 随机选了一个单词

    context = sent[max(0, wordID - C):wordID]
    if wordID+1 < len(sent):
        context += sent[wordID+1:min(len(sent), wordID + C + 1)]

    centerword = sent[wordID]
    context = [w for w in context`if w != centerword]   # 提取centerwords

    if len(context) > 0:
        return centerword, context
    else:
        return self.getRandomContext(C) # 为空则直接重新生成一遍
```

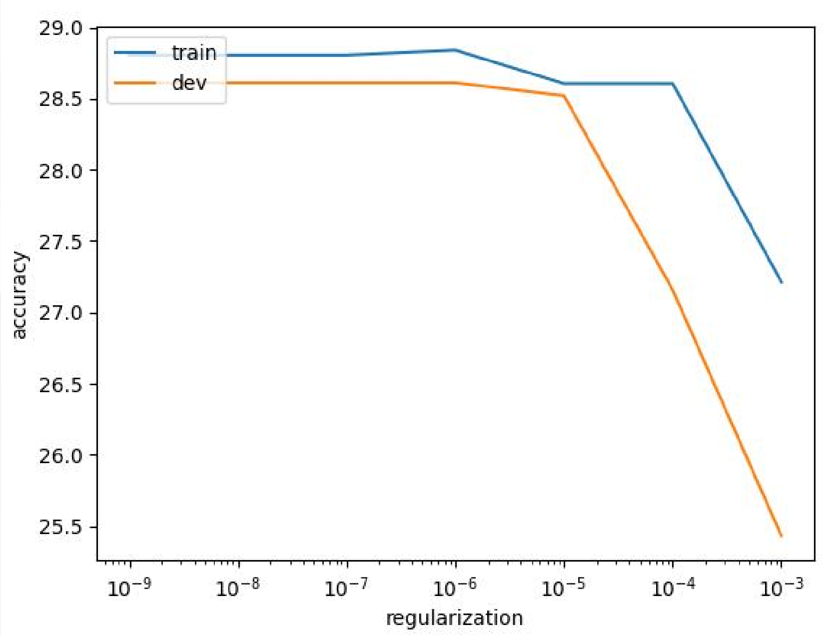
应该保留context word中和center word相同的词？

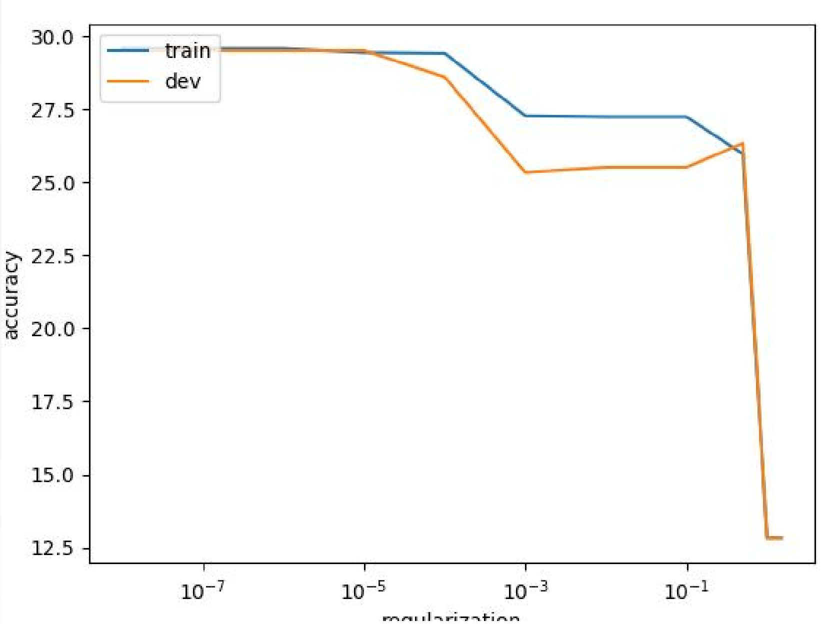
Outline

- Task Introduction
- Algorithm Introduction
  - Skip-gram
  - Negative Sampling
- Some findings in the code
- Experiment Result
- Thoughts & Pretrained Model Results

4.1 Experiment Results

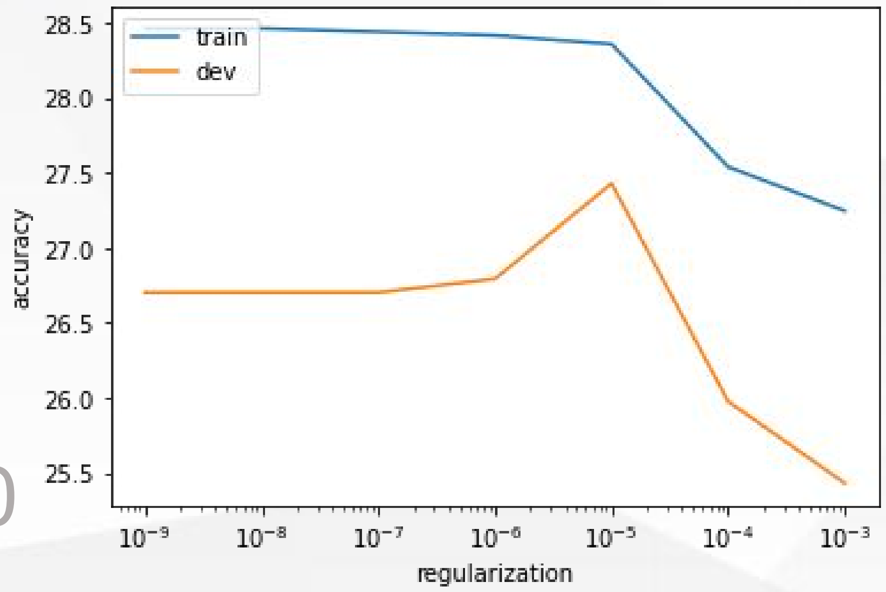
C 5 dim 10

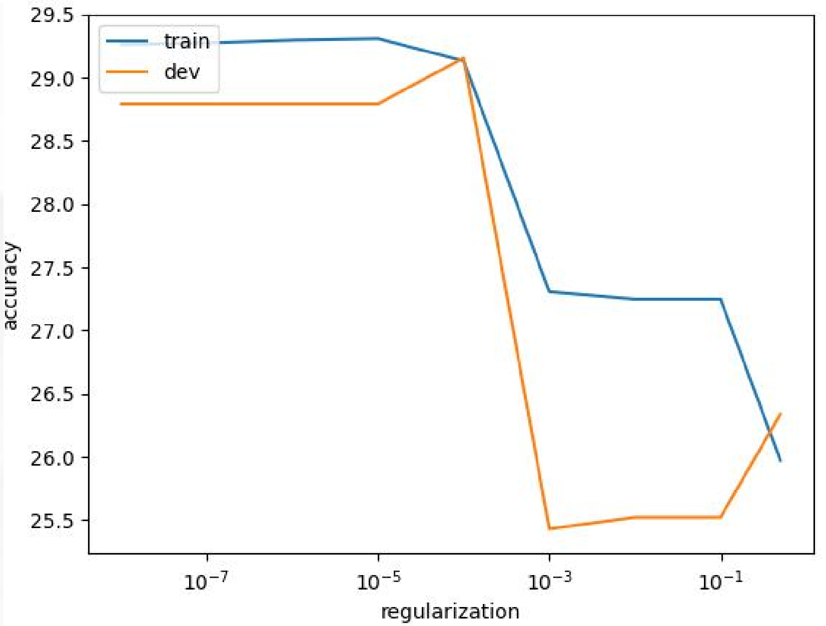




C 6 dim 10

C 9 dim 10

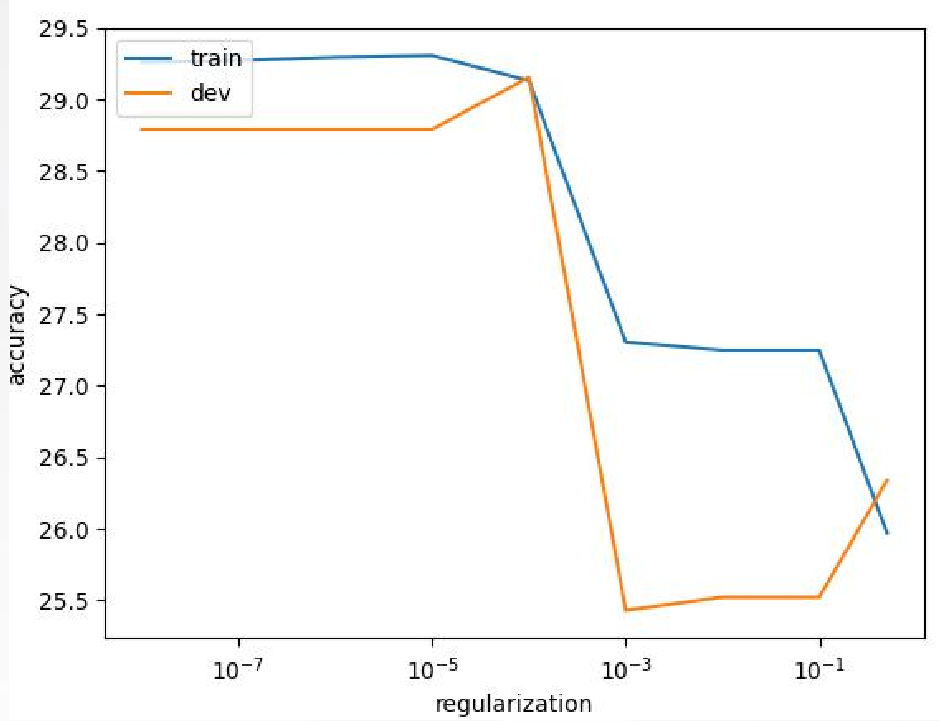




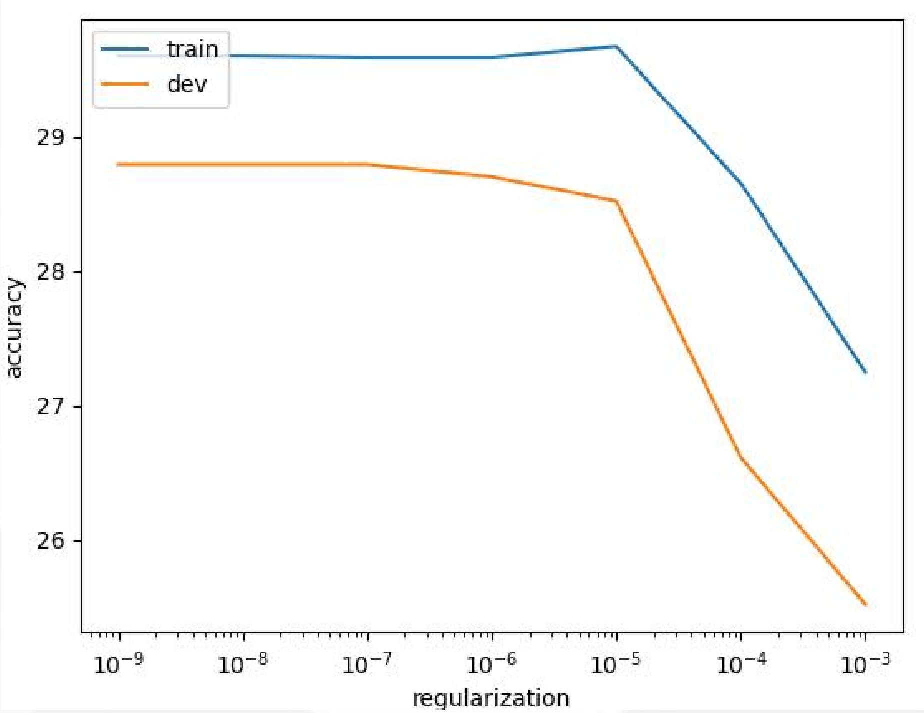
C 10 dim 10

4.1 Experiment Results

C 10 dim 10

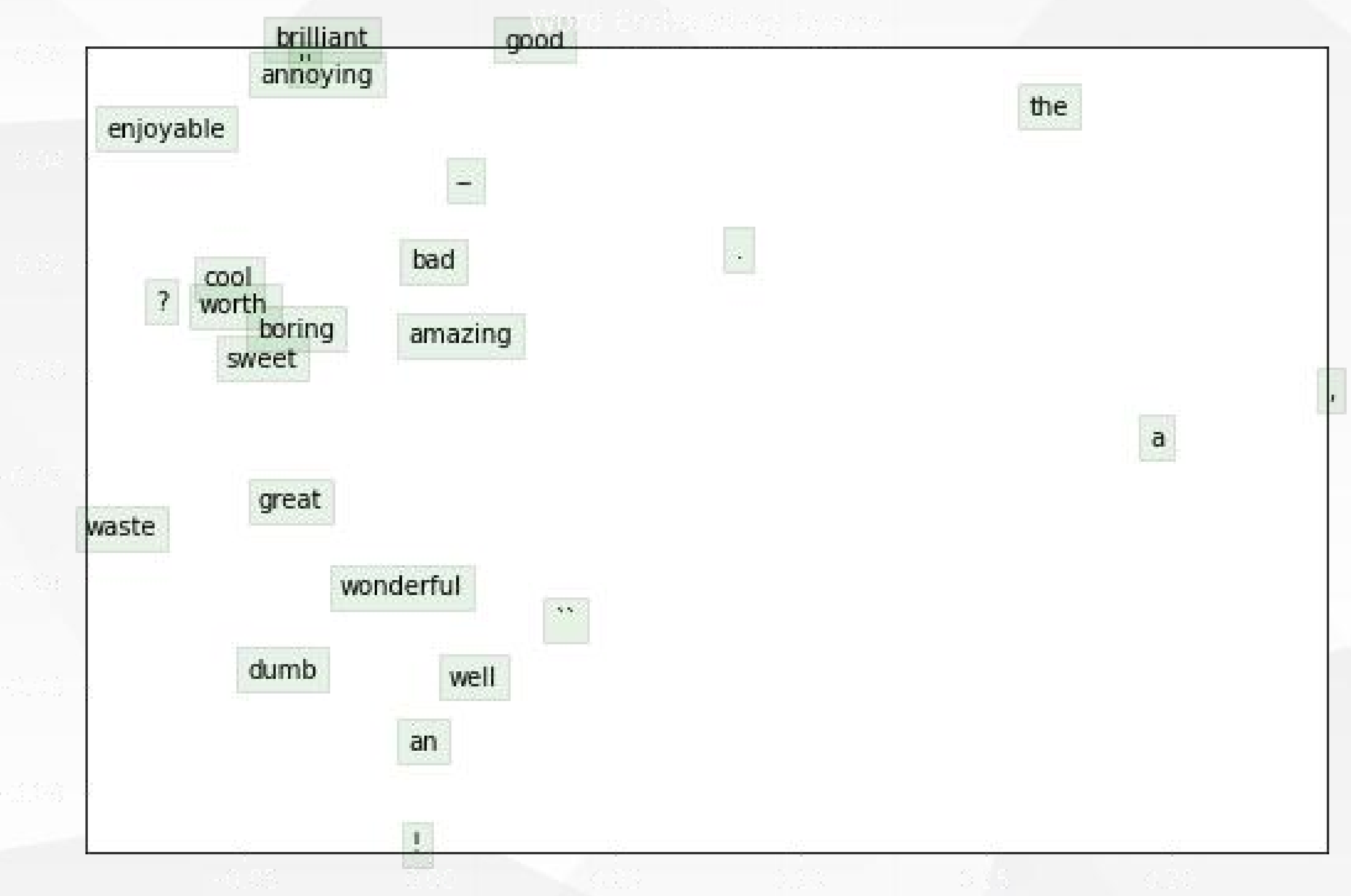


C 10 dim 20



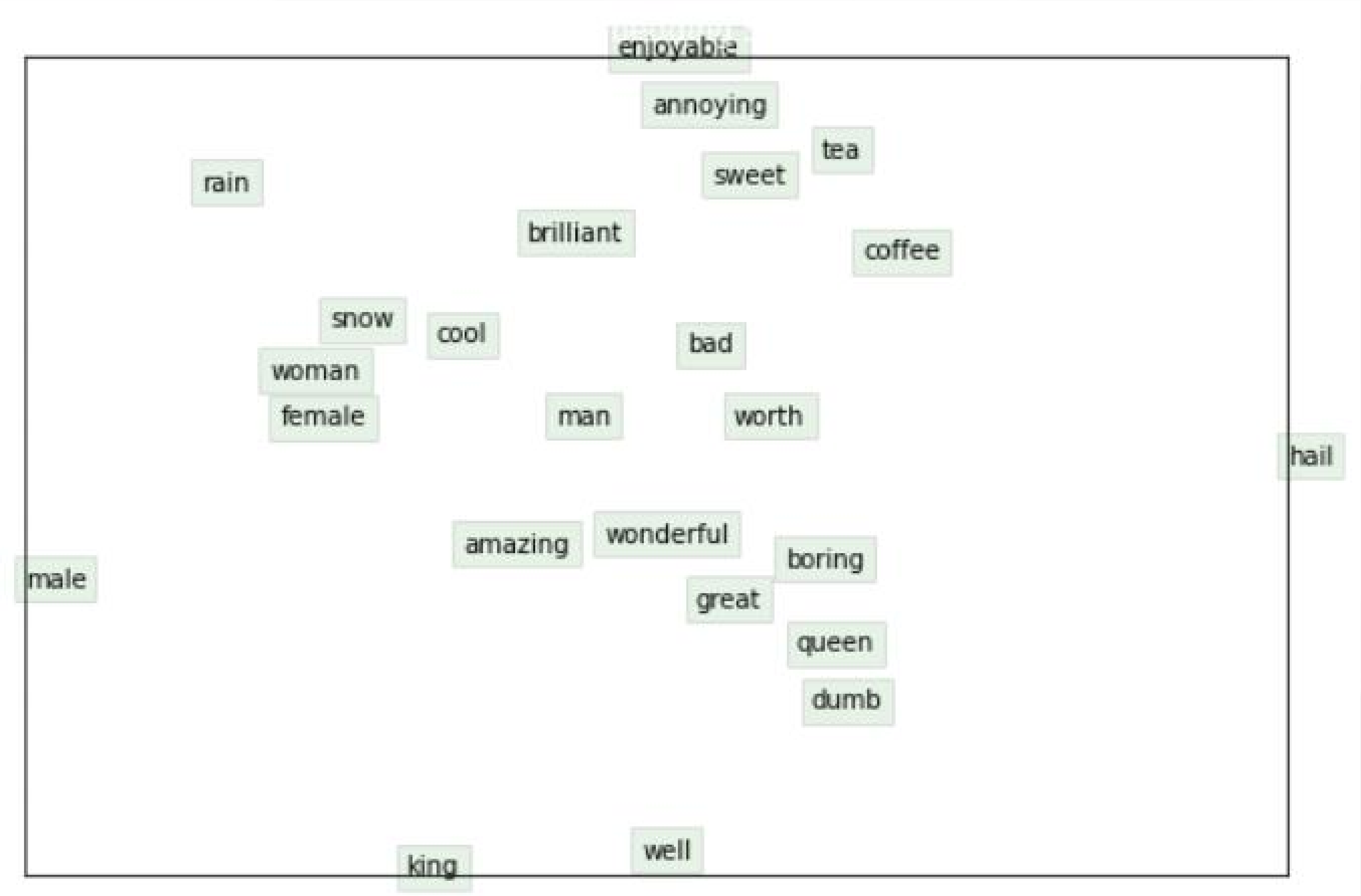
4.1 Experiment Results

C 5 dim 10



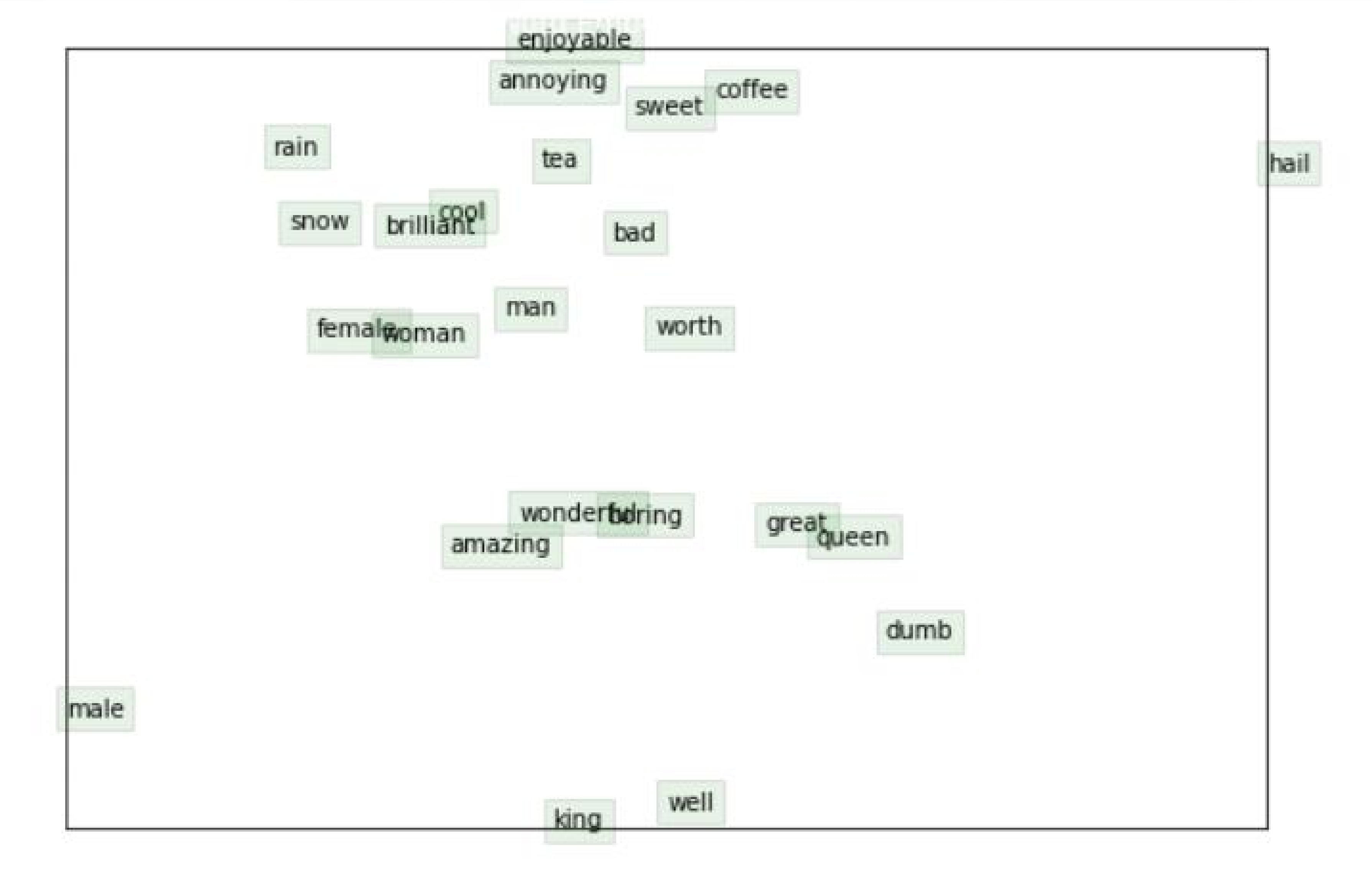
4.1 Experiment Results

C 7 dim 10



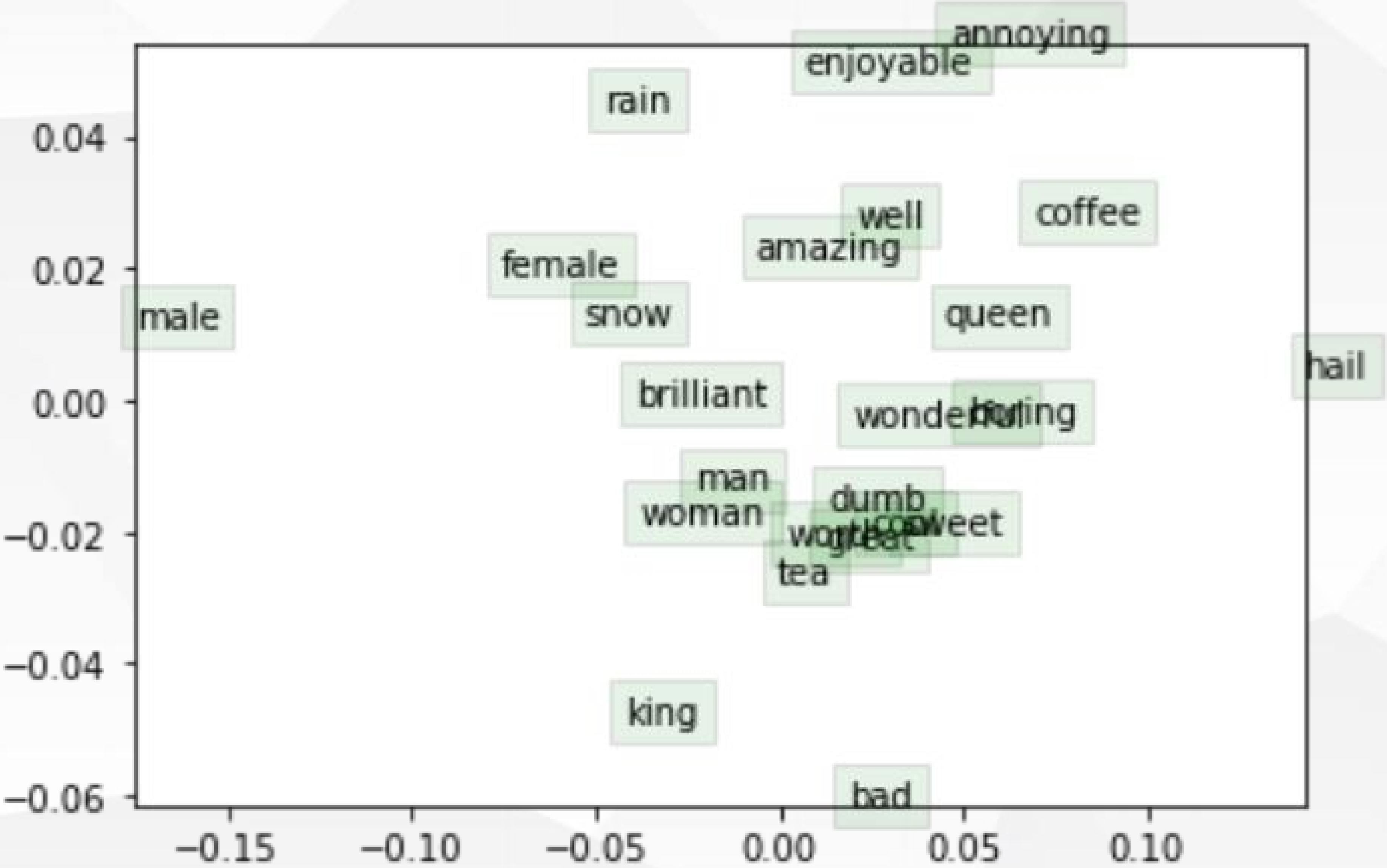
4.1 Experiment Results

C 8 dim 10



4.1 Experiment Results

C 10 dim 20



Outline

- Task Introduction
- Algorithm Introduction
  - Skip-gram
  - Negative Sampling
- Some findings in the code
- Experiment Result
- Thoughts & Pretrained Model Results

5.1 Thoughts

Regularization Coefficient

Trade-off between fitting the data well and avoiding overfitting.

We use **L2-norm** as our regularization term, which shrinks the weights towards zero without necessarily making them zero.

Larger RCs **penalize** more on the **model complexity**, which is expected to gain better generalization performance.

5.1 Thoughts

Why two vectors?

1. Simplify the calculation of the gradient for each word
2. Similarity representation: Vector Inner Product

Why performed poorly?

Our model only trained small window size and small vector dimension.

Other common word representations like BOW or N-gram models often have large vector dimension.

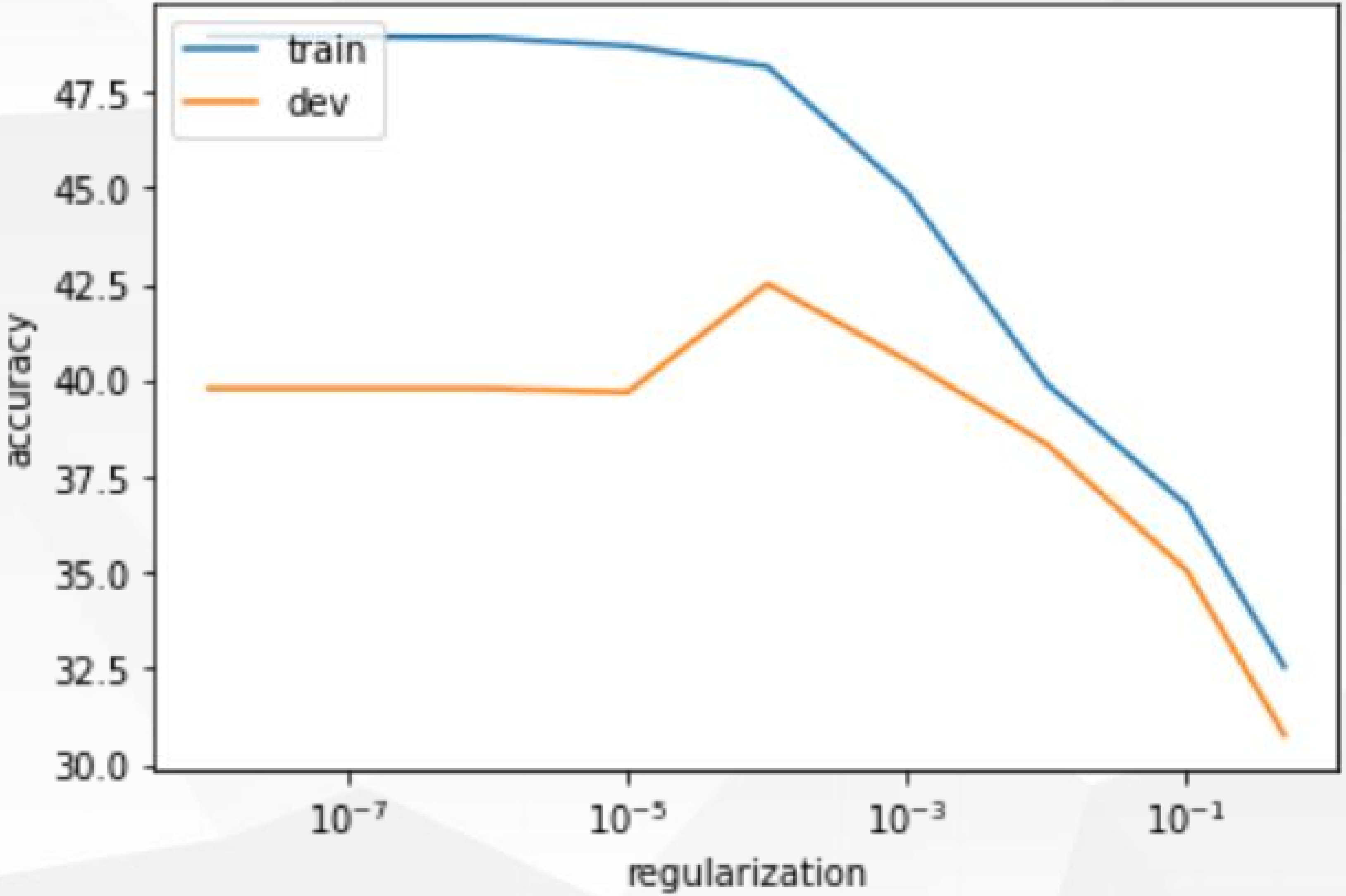
Comparison Experiment:

Try Pre-trained Word2Vec vectors

(or other vector models? E.g. GloVe)

5.2 Pretrained Model Results

Word2Vec (300 dimension) pre-trained on Google news



Thanks