一、实验设置

1.1 模型选择

本次实验选取 **Skip-gram** 词向量模型进行词嵌入学习。Skip-gram模型的主要目标是基于一个给定的中心词预测其上下文词汇。与CBOW模型不同，Skip-gram通过输入一个词来预测其周围的词，这使得它在处理稀有词汇时通常表现得更好，适用于大规模语料库的词向量训练。

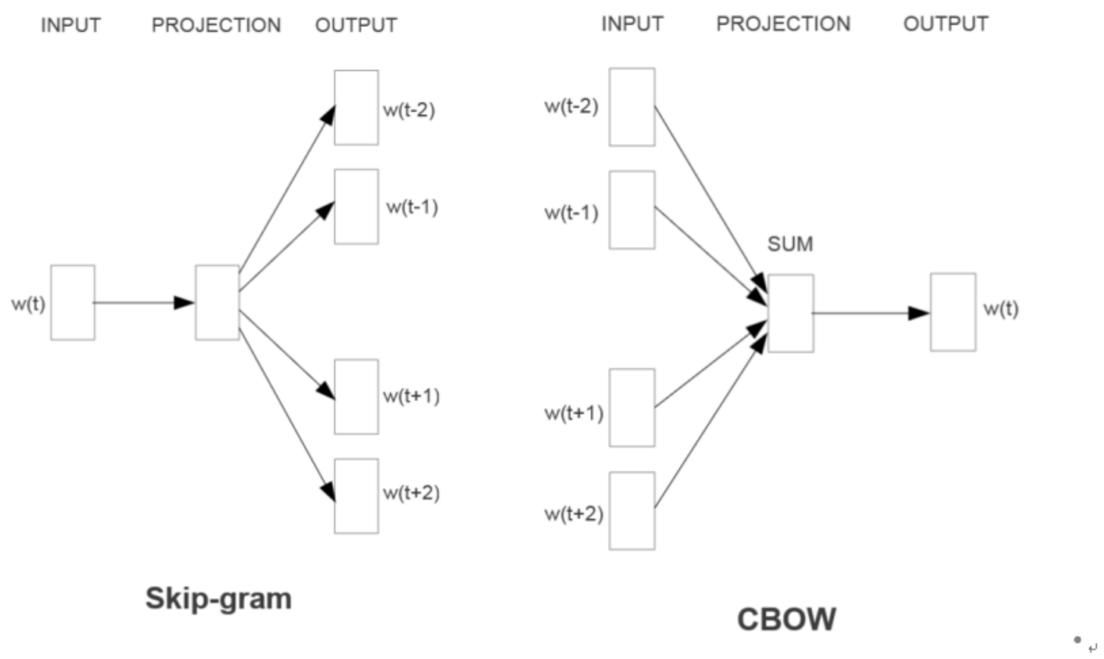

图 1 Skip-gram和CBOW模型图

1.2 参数设置

本次实验中的参数设置如下：

参数名	参数值
词向量维度 (Embedding size)	300
窗口大小 (Context window size)	7（代码中写的是其中一边的长度3）
训练迭代次数 (Epochs)	50
学习率 (learning_rate)	0.01
批量大小 (Batch size)	1024
优化算法 (Optimization algorithm)	Adam

二、实验介绍

2.1 数据处理

2.1.1 中文数据处理

由于已经分好词，去掉标点符号即可

```
text_file_path = './data/zh.txt'

def remove_chinese_punctuation(text):
    punctuation = r'[，。！？“”‘’（）【】『』；：、《》,.!?]'
    return re.sub(punctuation, '', text)

def preprocess_data(text_file_path):
  # load the text
  with open(text_file_path, 'r') as file:
    text = file.read().replace('\n', ' ')
  text = remove_chinese_punctuation(text)
  tokens = text.split()
  return tokens

tokens = preprocess_data(text_file_path)

# Example tokens: ['目前', '粮食', '出现', '阶段性', '过剩', '恰好', '可以', '以', '粮食', '换']
```

去重且为词添加索引，去重后13263

```
def vocal_index(tokens):
  # get the unique token
  vocabulary = list(OrderedDict.fromkeys(tokens))
  # map word to it's corresponding index
  word2index = {word: index for index, word in enumerate(vocabulary)}
  # map index to it's corresponding word
  index2word = {index: word for index, word in enumerate(vocabulary)}
  return vocabulary, word2index, index2word

# Vocabulary size: 13263
```

滑动Window生成pair

```
def generate_context_pairs(tokens,context_window_size):
  context_target_pairs = []
  for i in range(context_window_size, len(tokens) - context_window_size):
    # center word
    center = tokens[i]
    context = tokens[i-context_window_size:i+context_window_size+1]
    context.remove(center)
    for word in context:
        context_target_pairs.append((center, word))
  return context_target_pairs
# Example pairs: [('阶段性', '目前'), ('阶段性', '粮食'), ('阶段性', '出现'), ('阶段性', '过剩'), ('阶段性', '恰好'), ('阶段性', '可以'), ('过剩', '粮食'), ('过剩', '出现'), ('过剩', '阶段性'), ('过剩', '恰好')]
```

2.1.2 英文数据处理

与中文数据处理不同的是英文标号和停词的预处理。

```
text_file_path = './data/en.txt'

def preprocess_data(text_file_path):
  with open(text_file_path, 'r') as file:
    text = file.read().replace('\n', ' ')
  text = text.lower()
  # remove punctuations like '(),[].:?'
  text = text.translate(str.maketrans("", "", string.punctuation))
  tokens = word_tokenize(text)
  # remove stop words ('at','so','an','or')
  stop_words = set(stopwords.words('english'))
  tokens_after_processing = []

  for token in tokens:
    if token not in stop_words and token.isnumeric() == False:
      tokens_after_processing.append(token)
  return tokens_after_processing
```

2.2 模型加载

浅层的Skip-gram仅线性层，这里的模型没有采用one-hot，直接使用embeddings

```
class Skip_Gram_Model(nn.Module):
    def __init__(self, vocabulary_size, embedding_size):
        super(Skip_Gram_Model, self).__init__()
        self.embeddings = nn.Embedding(vocabulary_size, embedding_size)
        self.linear = nn.Linear(embedding_size, vocabulary_size)

    def forward(self, context_word):
        # Get the embedding of the context word not one hot encoded
        output = self.embeddings(context_word)
        # Just one Linear layer
        output = self.linear(output)
        return output
```

```
# model = Skip_Gram_Model(len(vocabulary),
embedding_size=embedding_size).to(device)
# Shape : (13263, 300)
```

2.3 模型训练

直接开始训练，tqdm显示进度，并观察loss

```
for epoch in tqdm(range(epochs)):
    total_loss = 0 # restart loss to 0
    # iterate over batch size
    for i in range(0,len(X_train),batch_size):
        x = X_train[i:i+batch_size]
        # print(x)
        # for x_i in x:
        #     print(index2word[x_i.item()])
        y_true = y_train[i:i+batch_size]
        # print(y_true)
        optimizer.zero_grad()
        y_pred = model(x)
        loss = loss_function(y_pred, y_true.view(-1))
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    # print the loss value for each epoch
    print(f'Epoch num: {epoch+1}, loss value: {total_loss:.3f}')
```

2.4 结果保存

保存结果：直接保存词向量矩阵或者保存每个词的embedding的结果

```
# save the model
torch.save(model.state_dict(), 'Skip_Gram_Model.pth')

# Save the embedding result
# Get the embedding matrix
embedding = model.embeddings.weight.data.cpu().numpy()
print(embedding.shape)
# save the embedding directly
with open('embeddings_matrix.txt', 'w') as f:
  for i in range(len(vocabulary)):
    vector_str = ' '.join(map(str, embedding[i]))
    f.write(f"{vector_str}\n")

# def save_embedding(embedding, vocabulary, file_path):
model = Skip_Gram_Model(len(vocabulary),
embedding_size=embedding_size).to(device)
model.load_state_dict(torch.load('Skip_Gram_Model.pth'))

# Create a dictionary to map words to their corresponding vectors
word_to_vec = {}
for i, word in enumerate(vocabulary):
    word_to_vec[word] = embedding[i].tolist()
# Save the word embeddings as a JSON file
```

```
with open('word_embeddings.json', 'w') as f:
    json.dump(word_to_vec, f, indent=4)
```

三、实验结果

3.1 中文语料的训练结果

完整的训练过程的控制台输出在terminal_output_zh文件中，以下展示部分：

```
Epoch num: 48, loss value: 5039.244
Epoch num: 49, loss value: 5041.375
Epoch num: 50, loss value: 5037.989
(13263, 300)
目前: [-0.40219635 -0.02171995 -0.12586017 -0.00934302  0.18048024 -0.21055649
  0.00478735  0.19280124  0.2181114   0.3015942   0.06994438 -0.03505711
  0.3278358  -0.35538003 -0.09086848 -0.2437857  -0.07224135  0.17155373
  0.07170976  0.09338263 -0.06652712 -0.2955409  -0.05403789  0.37752217
 -0.08122384 -0.5581397  -0.14281435  0.0423794   0.20474607 -0.06344712
 -0.36541715  0.02014033 -0.18652567 -0.0910044  -0.2058427  -0.06988672
 -0.24561019  0.2624486  -0.2245839  -0.19180462 -0.1319362   0.12109792
  0.36338922  0.1892393  -0.07014816 -0.44155324 -0.03858433  0.09114679
  0.02763766 -0.1578046   0.49991378 -0.28081068  0.1660043   0.48031226
 -0.19039261 -0.0248214  -0.00277093 -0.5487965   0.03158586 -0.1517019
  0.19842967 -0.12327971 -0.04291764  0.532777    0.1411565  -0.18081197
 -0.14484817 -0.03330301  0.11980118  0.06557097  0.21597041 -0.04061739
 -0.04666172 -0.1531573  -0.48576555 -0.09773291 -0.4293428   0.1224639
 -0.16469412 -0.03505226 -0.09270269 -0.73508245 -0.18060374  0.26826105
  0.36148208  0.3771731  -0.27127683  0.32914343 -0.06109033  0.03787353
 -0.38042462 -0.1418266  -0.01223918  0.36083585  0.16266876  0.29769364
  0.01360536  0.47889918 -0.20100236 -0.23989992 -0.3498267   0.03210779
 -0.10737366 -0.1352554  -0.34238562 -0.19120145  0.01289883  0.18237217
 -0.09308985 -0.12513484  0.19118215 -0.4156162   0.08926932  0.05074977
  0.22124606  0.10464279  0.04981065  0.17019108  0.23565483  0.12824914
 -0.15639345  0.23674302  0.02119025  0.15651613 -0.04545641  0.13832957
 -0.08716942 -0.01903276 -0.15358849  0.17352004  0.23620862 -0.17929623
 -0.02909085 -0.4009999  -0.03076087 -0.3794585   0.12006786  0.27005592
  0.13108788  0.13658717  0.27861583 -0.06972104  0.02587781  0.08266434
  0.15143839  0.05905696 -0.17594893  0.18138325 -0.09471741  0.4882026
 -0.05934866  0.05239395 -0.20220436 -0.24375722 -0.16542253  0.05381907
  0.09205762 -0.00175219  0.3673719   0.51248455  0.0437794  -0.15644279
 -0.03727709 -0.00264157  0.02739949 -0.05089551 -0.44049978 -0.09785706
 -0.53078973  0.24604018 -0.03924002  0.04957464 -0.20838133  0.21511225
 -0.06962147  0.01151483  0.30712795  0.10832047  0.22305377  0.15414202
```

3.2 英文语料的训练结果

完整的训练过程的控制台输出在terminal_output_en中，以下展示部分：

```
Epoch num: 45, loss value: 3765.751
Epoch num: 46, loss value: 3762.848
Epoch num: 47, loss value: 3762.178
Epoch num: 48, loss value: 3761.691
Epoch num: 49, loss value: 3758.073
Epoch num: 50, loss value: 3758.360
(11291, 300)
present: [ 1.03614055e-01 -1.35926053e-01 -1.75306246e-01 -4.02580481e-03
 -3.82139057e-01  1.66832194e-01 -1.99695230e-01 -1.66145802e-01
  3.88001382e-01 -6.67212754e-02 -3.96802187e-01 -7.62382662e-03
  3.11324477e-01 -1.65829316e-01 -1.92578852e-01  5.34411818e-02
 -2.77796805e-01  1.04505710e-01 -1.85768619e-01  3.78162086e-01
 -2.28868142e-01  1.52830005e-01  5.74965835e-01  3.64147909e-02
 -4.31846023e-01  1.92094520e-01  1.11026607e-01 -9.05587710e-03
  2.75904417e-01 -2.08202213e-01  3.66027355e-01 -4.51218843e-01
 -2.84836106e-02  4.94662732e-01  4.55306657e-02 -8.83704051e-02
 -1.59955136e-02 -9.10326187e-03 -8.02597255e-02 -1.83930561e-01
 -6.72115991e-03  3.72149162e-02 -3.38497059e-03  5.69186270e-01
  1.20090492e-01  4.95481975e-02  3.21050644e-01 -1.17095783e-01
 -2.74860263e-02  1.81735799e-01  2.67118514e-02 -5.76100238e-02
 -3.76364678e-01 -5.06924689e-02 -1.17770076e-01  5.96755818e-02
 -5.57830513e-01 -1.26494214e-01 -2.54078329e-01  1.10976407e-02
  3.72550398e-01  1.69347435e-01 -1.06180787e-01  1.01840205e-01
 -1.99926689e-01 -1.54131241e-02 -2.62782007e-01 -1.17738821e-01
  1.88607752e-01  4.17254269e-02  7.38299415e-02  1.44670501e-01
  1.53679043e-01 -1.70524139e-02  7.30578415e-03  2.22126245e-02
  2.77648270e-02  2.16248780e-01 -9.79273394e-02  1.49461165e-01
 -1.99725345e-01  1.97682127e-01  6.90867156e-02  2.44614705e-02
 -7.43354023e-01 -9.62042585e-02  2.67673343e-01  1.04230247e-01
 -1.49983630e-01  2.82328337e-01  3.15343175e-04 -2.87266552e-01
```