姓名:
学号:

中国科学院大学 NLP 课程

第二次作业

实验的主要代码我已经上传至我的 GitHub 仓库中。我采用的是 PyTorch 框架。中文的 vocab 词向量与下标对应关系保存在"zh_vocab _idx2token.pkl" 和"zh_vocab_token2idx.pkl" 两个文件中，用 pickle 库保存的二进制文件，分别是 python 中的一维 list 格式和 dict 格式。英文的 vocab 词向量与下标对应关系保存在"en_vocab _idx2token.pkl" 和"en_vocab_token2idx.pkl" 两个文件中，用 pickle 库保存的二进制文件，分别是 python 中的一维 list 格式和 dict 格式。中文的词向量嵌入层保存在"zh_net.pt" 中的第一个嵌入层中，模型架构如代码2.1所示，num_embeddings 的值为 49324，embed_size 的值为默认值 100。英文的词向量嵌入层保存在"en_net.pt" 中的第一个嵌入层中，模型架构如代码2.1所示，num_embeddings 的值为 43638，embed_size 的值为默认值 100。

1 数据集

1.1 数据集来源

由于老师提供的数据集过小，且我有 gpu 服务器资源，所以我采用了比较大的数据集进行训练。对于中文数据集，我采用的是kaggle 上的人民日报语料库。这个语料库收集了人民日报的文章报道等。英文数据集我采用了kaggle 上的 CNN 新闻数据集。

1.2 数据预处理

人民日报语料库下载下来后是按照天数每天的头版新闻一个 txt 文件，倘若对整个数据集进行训练的话语料规模过于庞大，尤其是在处理 vocab 这一步时因为内存受限制无法处理比较大规模的语料。所以我选取数据集的两个 20% 分成两个子集，分别作为训练集和验证集。将隶属于训练集的 txt 文件和隶属于验证集的 txt 文件分别合并成一个文件，并且在合并时每行一句话。对于每句话我都用 jieba 库进行分词处理，将一句话变成由空格分开的各个词组成。去除每行话中没有什么意义的停顿词（stopwords），这里的停顿词用了哈工大的中文停顿词语料库，并去除较短的行。数据预处理代码如下所示。

```
1 def makeChineseDataFile(stopWords: set, dataSetPath: str = './data/RenMin_Daily',
      trainSavePath: str = './data/renmin.txt',\
2     tesSavePath: str = './data/renmin_test.txt', percent = 0.1 ):
3     """将所有文件合并到一个文件中"""
4     fileNames = os.listdir(dataSetPath)
5     numOfFiles = int(len(fileNames) * percent)
6     with open(trainSavePath, 'w') as trainFile:
7         for i in range(numOfFiles):
```

```
            fileName = fileNames[i]
            with open(os.path.join(dataSetPath, fileName), 'r') as f1:
                for line in f1:
                    line = line.strip()
                    # 空行
                    if line == '':
                        continue
                    newLine = []
                    for word in jieba.cut(line, cut_all= False):
                        # 去除停顿词
                        word = word.strip()
                        if word != '' and word not in stopWords:
                            newLine.append(word)
                    # 太短的行不要
                    if len(newLine) < 4:
                        continue
                    newLine = " ".join(newLine)
                    newLine = newLine.strip()
                    newLine += '\n'
                    trainFile.write(newLine)

    with open(tesSavePath, 'w') as testFile:
        for i in range(numOfFiles+2, numOfFiles * 2 + 2):
            fileName = fileNames[i]
            with open(os.path.join(dataSetPath, fileName), 'r') as f1:
                for line in f1:
                    line = line.strip()
                    # 空行
                    if line == '':
                        continue
                    newLine = []
                    for word in jieba.cut(line, cut_all= False):
                        # 去除停顿词
                        word = word.strip()
                        if word != '' and word not in stopWords:
                            newLine.append(word)
                    # 太短的行不要
                    if len(newLine) < 4:
                        continue
                    newLine = " ".join(newLine)
                    newLine = newLine.strip()
                    newLine += '\n'
                    testFile.write(newLine)
```

Listing 1.1: 中文语料处理程序

而英文语料库则是 csv 格式的，我用 pandas 读取 csv 文件，并只选择文章栏目整理成 txt 文件，同样是每行一句话，且去处了标点符号等。

```python
def makeEnglishDataFile(stopWords: set, dataSetPath: str = './data/cnn.csv', trainSavePath: str = './data/cnn.txt',\
    tesSavePath: str = './data/cnn_test.txt', percent = 0.2 ):
    cnnNews = pd.read_csv(dataSetPath)["Article text"]
    cnnNews = cnnNews.dropna()
    cnnNews = cnnNews.tolist()
    numOfFiles = int(len(cnnNews) * percent)
    with open(trainSavePath, 'w') as trainFile:
        for i in range(numOfFiles):
            paragraph = cnnNews[i]
            paragraph = paragraph.split('.')
            for line in paragraph:
                line = line.strip()
                newLine = []
                for word in line.split():
                    word = word.replace('(CNN)', '')
                    word = word.replace(',', '')
                    word = word.replace('!', '')
                    word = word.replace('?', '')
                    word = word.replace(';', '')
                    word = word.replace(':', '')
                    word = word.replace('(', '')
                    word = word.replace(')', '')
                    word = word.replace('[', '')
                    word = word.replace(']', '')
                    word = word.replace('{', '')
                    word = word.replace('}', '')
                    word = word.replace('—', '')
                    word = word.replace('–', '')
                    word = word.replace('\'', '')
                    word = word.replace('\"', '')
                    word = word.replace('‘', '')
                    word = word.replace('’', '')
                    word = word.strip()
                    if word != '' and word not in stopWords:
                        newLine.append(word)
                # 太短的行不要
                if len(newLine) < 4:
                    continue
                newLine = " ".join(newLine)
                newLine = newLine.strip()
                newLine += '\n'
                trainFile.write(newLine)

    with open(tesSavePath, 'w') as trainFile:
        for i in range(numOfFiles + 2, 2 * numOfFiles + 2):
            paragraph = cnnNews[i]
            paragraph = paragraph.split('.')
```

```
            for line in paragraph:
                line = line.strip()
                newLine = []
                for word in line.split():
                    word = word.replace('(CNN)', '')
                    word = word.replace(',', '')
                    word = word.replace('!', '')
                    word = word.replace('?', '')
                    word = word.replace(';', '')
                    word = word.replace(':', '')
                    word = word.replace('(', '')
                    word = word.replace(')', '')
                    word = word.replace('[', '')
                    word = word.replace(']', '')
                    word = word.replace('{', '')
                    word = word.replace('}', '')
                    word = word.replace('—', '')
                    word = word.replace('–', '')
                    word = word.replace('\'', '')
                    word = word.replace('\"', '')
                    word = word.replace('`', '')
                    word = word.replace('  ', '')
                    word = word.strip()
                    if word != '' and word not in stopWords:
                        newLine.append(word)
                # 太短的行不要
                if len(newLine) < 4:
                    continue
                newLine = " ".join(newLine)
                newLine = newLine.strip()
                newLine += '\n'
                trainFile.write(newLine)
```

Listing 1.2: 英文语料处理程序

1.3 制作 dataloader

我将 txt 文件中的所有单词装进 vocab，然后将 vocab 中的低频词（此处我选择了出现少于 10 次的词为低频次）统一改成 <unk>，避免少见词计算的浪费。并将 vocab 中的总词数记录进 log 中，因为在之后进行验证时我需要将之前训练好的嵌入层数据加载进来，所以需要在 log 中记录 vocab 收录的词量大小。然后分别用 list 和 dict 记录 index 和 word 的互相对应转换关系。我对于在 txt 文件中的每个单词作为中心词，随机大小选取上下文窗口大小做成上下文词，并随机采样 k 倍上下文词数量的无关词作为负采样词（此处 k 选择 5）。然后做成 dataset，并进一步做成 dataloader。将 dataloader 每个 batch 中的上下文词数量和窗口数量词由之前的随机选取大小改成定长大小以方便后期乘法计算，变成每个 dataloader 有着 batchSize 大小个的（中心词, 上下文和负采样词, 实值 mask, 上下文或负采样的 label）的形式，其中实值 mask 为 1 时带表这个词是上下文词或者负采样词，为 0 代表这个词没有意义是凑长度用的。上下文或

负采样的 label 为 1 时带表这个值是上下文，为 0 代表负采样词。具体代码如下所示。

```
def load_data_loader(batch_size, max_window_size, num_noise_words, \
    trainDatasetPath, testDatasetPath ):
    """下载PTB数据集，然后将其加载到内存中"""
    num_workers = 0
    trainSentences = read_txt(trainDatasetPath)
    testSentences = read_txt(testDatasetPath)
    allVocab = Vocab(trainSentences + testSentences, min_freq=10)
    trainSubsampled, trainCounter = subsample(trainSentences, allVocab)
    testSubsampled, testCounter = subsample(testSentences, allVocab)
    trainCorpus = [allVocab[line] for line in trainSubsampled]
    testCorpus = [allVocab[line] for line in testSubsampled]
    trainAllCenters, trainAllContexts = get_centers_and_contexts(
        trainCorpus, max_window_size)
    trainAllNegatives = get_negatives(
        trainAllContexts, allVocab, trainCounter, num_noise_words)
    testAllCenters, testAllContexts = get_centers_and_contexts(
        testCorpus, max_window_size)
    testAllNegatives = get_negatives(
        testAllContexts, allVocab, testCounter, num_noise_words)

    class renMinDataset(torch.utils.data.Dataset):
        def __init__(self, centers, contexts, negatives):
            # 长度必然相等，centers每个元素是文本中每一个被选出来的词的中心词（自己），
            # context每个元素是上下文词（一个装有上下文的1D list）
            # negatives每个元素是负采样词（装有负采样元素的K倍长于上下文词的1D list）
            assert len(centers) == len(contexts) == len(negatives)
            self.centers = centers
            self.contexts = contexts
            self.negatives = negatives

        def __getitem__(self, index):
            # 取一次取一个tuple，分别是一个中心词，一个1D的上下文list，
            # 一个是上下文list K倍长的1D list
            return (self.centers[index], self.contexts[index],
                    self.negatives[index])

        def __len__(self):
            return len(self.centers)
    trainDataset = renMinDataset(trainAllCenters, trainAllContexts, trainAllNegatives)
    testDataset = renMinDataset(testAllCenters, testAllContexts, testAllNegatives)
    trainDataIter = torch.utils.data.DataLoader(
        trainDataset, batch_size, shuffle=True,
        collate_fn=batchify, num_workers=num_workers)
    testDataIter = torch.utils.data.DataLoader(
        testDataset, batch_size, shuffle=True,
        collate_fn=batchify, num_workers=num_workers)
# 在创建DataLoader类的对象时，collate_fn函数会将batch_size个样本整理成一个batch样本，便于批量
```

```
    训练。
    return trainDataIter, testDataIter, allVocab
```

Listing 1.3: 做 dataloader 的代码

2 网络与训练

2.1 网络

这里我选择了用两层嵌入层进行操作，它们都被保存在一个 net 里，其中一个嵌入层用来嵌入中心词，一个嵌入层用来嵌入上下文词。这样方便代码编程与计算表达。与之前的网络不同，这个网络中的两个嵌入层并没有直接的前后数值传递关系，放在一个网络里只是为了方便后来的梯度下降等。每个词向量设其有 100 个纬度，网络具体如下所示。

```
def getNet(num_embeddings, embed_size = 100):
    net = nn.Sequential(nn.Embedding(num_embeddings=num_embeddings,
                                     embedding_dim=embed_size),
                        nn.Embedding(num_embeddings=num_embeddings,
                                     embedding_dim=embed_size))
    return net
```

Listing 2.1: 两个嵌入层 net

2.2 训练

在之前的数据集处理部分，我将数据集分为了训练集合与测试集两部分，事实证明这很有必要。因为 word2vec 模型比较简单，就是嵌入层矩阵，所以极易过拟合。一般在 2-3 个 epoch 后就会过拟合，所以通过验证集合检测是否过拟合非常合适。

我选择跳元模型进行计算，并在验证集达到最小损失时保存嵌入层参数和 vocab 中的单词与下标的对应关系。因为只需要判断该上下文词是不是中心词的正确上下文词，所以可以使用二元交叉熵损失函数作为损失函数。训练代码整体如下。

```
def train(net, lossFunction, trainDataIter, testDataIter, lr, num_epochs, device='cuda'):
    def init_weights(m):
        if type(m) == nn.Embedding:
            nn.init.xavier_uniform_(m.weight)
    net.apply(init_weights)
    loss = lossFunction
    net = net.to(device)
    minLossTrain = 1666666
    optimizer = torch.optim.Adam(net.parameters(), lr=lr)
    # 规范化的损失之和，规范化的损失数
    for epoch in range(num_epochs):
        logger.info(f'The {epoch + 1}th epoch')
        num_batches = len(trainDataIter)
```

```
        epochLoss = 0
        net.train()
        logger.info(f'epoch {epoch + 1} start train')
        for i, batch in enumerate(trainDataIter):
            optimizer.zero_grad()
            center, context_negative, mask, label = [data.to(device) for data in batch]
            pred = skip_gram(center, context_negative, net[0], net[1])
            l = (loss(pred.reshape(label.shape).float(), label.float(), mask)
                     / mask.sum(axis=1) * mask.shape[1])
            l = l.sum()
            epochLoss += l / num_batches
            if i % 20 == 0:
                print(f'epoch {epoch + 1}, batch {i}/{num_batches}. The loss is {l}')
            l.backward()
            optimizer.step()
        if epochLoss < minLossTrain:
            minLossTrain = epochLoss
            logger.info(f'***epoch {epoch + 1} get MIN train loss, is {minLossTrain}')
        else:
            logger.info(f'epoch {epoch + 1} loos is {epochLoss}')
        test(net, loss, testDataIter, epoch, device)
```

Listing 2.2: word2vec 训练代码

3 验证

在训练完毕后我写了一个函数来判断词向量的相关性，以找到最相似的同类词。首先创建一个新的拥有两个嵌入层的网络，然后从 log 文件夹中加载网络参数，并加载之前保存的 vocab 中单词与下标的对应关系。采用词向量的正弦值来判断两个词向量的相似程度，测试代码如下所示：

```
def get_similar_tokens(query_token, k, embed, token_to_idx, idx_to_token):
    W = embed.weight.data
    x = W[token_to_idx[query_token]]
    # 计算余弦相似性。增加1e-9以获得数值稳定性
    cos = torch.mv(W, x) / torch.sqrt(torch.sum(W * W, dim=1) *
                                      torch.sum(x * x) + 1e-9)
    topk = torch.topk(cos, k=k+1)[1].cpu().numpy().astype('int32')
    for i in topk[1:]:  # 删除输入词
        print(f'cosine sim={float(cos[i]):.3f}: {idx_to_token[i]}')

if __name__ == '__main__':
    en, zh = 0, 1
    logDir = ['./log/cnn', './log/renmin'][en]
    with open(os.path.join(logDir, 'vocab_idx2token.pkl'), 'rb') as f:
        idx_to_token = pickle.load(f)
    with open(os.path.join(logDir, 'vocab_token2idx.pkl'), 'rb') as f:
        token_to_idx = pickle.load(f)
```

```
    embed = train.getNet(43638)
    embed.load_state_dict(torch.load(os.path.join(logDir ,'net.pt')))
    embed = embed[0]

    toFind = input("please inpu the word you want to find in word2vec\n")
    print("10 most similar words in word2vec is:")
    get_similar_tokens(toFind, 10, embed, token_to_idx, idx_to_token)
```

Listing 3.1: word2vec 测试代码

对于中文的 word2vec，我们输入测试词“上海”输出最相似的十个词，结果如图3.1所示。在输出的十个词中武汉、西安、嘉定、广州、厦门、青岛等为国内的其他城市名，华虹为上海的知名企业，青浦、松江、奉贤则是上海的区，这些词的词性和上下文搭配的单词均与“上海”相似，结果符合 word2vec 相似的定义。

```
~/ucas/nlp/homework2 main !2 python3 ./predictSim.py
Please input the word you want to find similar words in word2vec
上海
10 most similar words in word2vec is:
cosine sim=0.681: 武汉
cosine sim=0.650: 西安
cosine sim=0.637: 嘉定
cosine sim=0.627: 广州
cosine sim=0.625: 华虹
cosine sim=0.623: 青浦
cosine sim=0.610: 厦门
cosine sim=0.606: 松江
cosine sim=0.604: 奉贤
cosine sim=0.601: 青岛
```

Figure 3.1: 中文 word2vec 对于词“上海”找寻相近词

对于英文的 word2vec，我们输入测试词“Spain”也就是国家西班牙的英文名，输出最相似的十个词向量对应的单词，结果如图3.2所示。在输出的十个单词中“Portugal”（葡萄牙）、“Croatia”（克罗地亚）、“Netherlands”（荷兰）、“Uruguay”（乌拉圭）为国家名，“Bilbao”（毕尔巴鄂）、“Norwich”（诺维奇）、“Milan”（米兰）、“Porto”（波尔图）、“SaintDenis”（圣但尼）为城市名称，而“Inter”则是为不太相关的词。这些词与单词“Spain”词性相近，上下文搭配的词趋同，这与 word2vec 的目标一致。

```
~/ucas/nlp/homework2 main !2 python3 ./predictSim.py
Please input the word you want to find similar words in word2vec
Spain
10 most similar words in word2vec is:
cosine sim=0.830: Portugal
cosine sim=0.830: Bilbao
cosine sim=0.823: Croatia
cosine sim=0.799: Norwich
cosine sim=0.798: Netherlands
cosine sim=0.790: Inter
cosine sim=0.784: Milan
cosine sim=0.784: Porto
cosine sim=0.783: Uruguay
cosine sim=0.779: SaintDenis
```

Figure 3.2: 英文 word2vec 对于词“Spain”找寻相近词

4 一些思考与发现

- 因为在前期制作 vocab 的过程中需要使用全部 word 进行索引与下标的匹配与统计，所以需要消耗大量的**内存**，反而对显存的消耗不大。
- 因为嵌入层架构简单，所以在训练时极易过拟合，一般两三个 epoch 就可以达到最佳。注意减少 epoch 数量或者用验证集测试结果选择最佳参数保存。
- 前期对数据集进行预处理和清洗以得到合适的数据集非常重要。