自然语言理解初步第一次大作业实验报告

姓名：卜梦煜 学号：1120192419 班级：07111905

一、实验名称

分词与词性标注

二、实验要求

- 实现基于词典的分词方法和统计分词方法：两类方法中实现一种即可；
- 对分词结果进行词性标注，也可以在分词的同时进行词性标注；
- 对分词及词性标注结果进行评价，包括4个指标：正确率、召回率、F1值和效率；
- （选做）命名实体识别：对句子中出现的人名、地名、机构名进行识别。

三、实验环境

- python 3.8.5
- 数据集：corpusZh（https://github.com/liwenzhu/corpusZh）

四、实验内容

4.1 中文分词

4.1.1 预处理

（1）预处理任务

1、由于分词选用`2-gram`模型，因此需统计单个词出现的频数`wfreq`、相邻两个词出现的频数`bifreq`。

2、对原始数据集，需经过去词性处理，作为分词测试集的答案；再经过拼接处理形成完整地句子，作为分词测试集。

（2）预处理代码

```
# 对词频与相邻两个词的词频做统计
def preprocess3(trainSet):
    wfreq["BOS"] = wfreq["EOS"] = 0
    for sent in trainSet:
        wfreq["EOS"] += 1
        wfreq["BOS"] += 1
        for i in range(len(sent)):
            # 处理unigram
            if sent[i] in wfreq.keys():
                wfreq[sent[i]] += 1
            else:
                wfreq[sent[i]] = 1
            # 处理bigram
            if i == 0:
                token = "BOS & " + sent[i]
            else:
                token = sent[i-1] + " & " + sent[i]
            if token in bifreq.keys():
                bifreq[token] += 1
```

```
            else:
                bifreq[token] = 1
            if i == len(sent) - 1:
                token = sent[i] + " & EOS"
                if token in bifreq.keys():
                    bifreq[token] += 1
                else:
                    bifreq[token] = 1

# 将set中词列表的词拼接成完整句子testSet，用于分词测试
def preprogress4(set, testSet):
    for i in range(len(set)):
        sent = ""
        for ci in set[i]:
            sent += ci
        testSet.append(sent)
```

4.1.2 基于词典的分词方法

（1）算法设计

采用逆向最大匹配算法（BMM）进行中文分词。将句子分词任务转换为按一定顺序在字典中做查找的任务。具体来说，BMM算法从左到右将待分词文本中的几个连续字符与词表匹配，如果匹配上，则切分出一个词。

算法具体步骤如下：

1、输入词典dict，待切分句子sentence；

2、从待切分句子末尾开始向前截取长度为l（l为sentence长度）的子句，进行分词；

3、判断子句是否在词典中。若在，则保存子句，并从sentence中删除这个子句，转到2；若不在，则判断子句长度是否为1，若为1，则将单字保存，从sentence中删除单字，转到2；若不为1，则将子句最左边一个字删除，形成新的子句，转到3。

（2）程序结构

```
def BMM(dict, sentence, ans):
    r = len(sentence)
    while r > 0:
        flag = 1
        for l in range(r):
            if sentence[l: r] in dict.keys():
                flag = 0
                ans.append(sentence[l: r])
                r = l
                break
        if flag == 1:
            ans.append(sentence[r-1])
            r -= 1
```

4.1.3 基于统计的分词方法

（1）算法设计

采用`2-gram`语言模型+`viterbi`算法进行中文分词。将分词任务转换为对所有可能的分词方式计算在`2-gram`语言模型下的概率的问题。具体来说，该方法先统计待切分句子中所有在词典中出现的词，得到所有可能的切分方式，然后在`2-gram`模型下计算，将概率最大的切分Seg作为句子的切分结果。计算公式如下：

$$P(Seg_k) = P(w_1)\prod_{i=2}^{n} P(w_{i-1}|w_i)$$

实际应用中，为避免数据过小的问题，将计算乘积转换为计算取对数求和，即实际公式为：

$$P(Seg_k) = log(P(w_1)) + \sum_{i=2}^{n} log(P(w_{i-1}|w_i))$$

`viterbi`算法是一种动态规划算法。分析知，本任务满足最优子结构与重叠子问题性质，即整个句子的最优切分可由其前缀的最优切分得到，而求解整个句子最优切分的算法对求解子问题也适用。因此可利用`viterbi`算法加速。

算法具体步骤如下：

1、建立二元切分词图。建立一个有向无环图，图中的结点为任意一个可能的候选词语，图中的边代表相邻两个词语的续接关系。二元切分词图的每一条边的权值表示二元词语转移概率$P(w_{i-1}|w_i)$。任何一种切分的方式可以表示为二元切分词图上的一条起始结点到结束结点的路径。路径上所有边的概率之积就是该切分结果对应的二元语法模型概率。

2、对待切分句子做全切分。按照词图的拓扑序，从句首到句尾，每个汉字为一个节点。

3、`viterbi`算法求最优路径。

所需辅助数组如下：

```
vit[i]：待切分句子子句[0: i+1]在2-gram模型下的最大可能概率。
now[i]：以i为结尾的词。
pre[i]：以i为结尾的词的前一个词。
prepos[i]：以i为结尾的词的前一个词结尾的下标。
```

转移方程如下：

$$vit[i] = \max_k(vit[i], vit[i-l] + log(P(ci_k|now[i-l])))$$

其中$l = len(ci_k)$，$P(ci_k|now[i-l]) = \frac{bifreq[ci_k\&now[i-l]]}{wfreq[now[i-l]]}$，$bifreq[c1\&c2]$表示训练集中词c1词c2相邻的次数，$wfreq[c1]$表示训练集中词c1出现的次数。

转移方程的含义是，对于以sentence[i]为结尾的所有可能的切分词，选取前一个词转移到该词的最大概率对应的词作为该节点的切分方式。

求得概率最大的最优值后，从末尾按照$pre = prepos[now]$回溯遍历一遍，即可获得最优解，即待切分句子的切分方式。

(2) 程序结构

```
# 对sentence中所有可能出现的词做统计，放在列表ci中
def cut(wfreq, sentence, ci):
    for i in range(len(sentence)):
        if sentence[i] not in ci:
            ci.append(sentence[i])
        for j in range(i+2, len(sentence)):
            if sentence[i: j] in wfreq.keys() and sentence[i: j] not in ci:
                ci.append(sentence[i: j])

# 分词viterbi算法
def viterbiFC(wfreq, bifreq, sentence, ci, result1, result):    # sentence为原始句子，ci为句中所有可能的词
    vit = [-5e100 for i in range(len(sentence))]
    pre = [0 for i in range(len(sentence))]
    prepos = [0 for i in range(len(sentence))]
    now = [0 for i in range(len(sentence))]
    for i in range(len(sentence)):
        for item in ci:
            if item[-1] == sentence[i] and len(item) <= i + 1 and item == sentence[i-len(item)+1: i+1]:
                if len(item) == i + 1:
                    if ("BOS & "+item) not in bifreq.keys():
                        nfreq = math.log10(1 / (wfreq["BOS"] + len(wfreq.keys())))
                    else:
                        nfreq = math.log10((bifreq["BOS & "+item] + 1) / (wfreq["BOS"] + len(wfreq.keys())))
                else:
                    if (str(now[i - len(item)]) + " & " + item) not in bifreq.keys():
                        if now[i - len(item)] not in wfreq.keys():
                            nfreq = MINN
                        else:
                            nfreq = math.log10(1 / (wfreq[now[i - len(item)]] + len(wfreq.keys())))
                    else:
                        nfreq = math.log10((bifreq[str(now[i-len(item)]) + " & " + item] + 1) /
                                           (wfreq[now[i - len(item)]] + len(wfreq.keys())))
                    nfreq += vit[i - len(item)]
                if i == len(sentence) - 1:
                    if (str(item) + " & EOS") not in bifreq.keys():
                        if item not in wfreq.keys():
                            nfreq += MINN
                        else:
                            nfreq += math.log10(1 / (wfreq[item] + len(wfreq.keys())))
                    else:
                        nfreq += math.log10((bifreq[item + " & EOS"] + 1) /
                                            (wfreq[item] + len(wfreq.keys())))
                if nfreq > vit[i]:
                    prepos[i] = i - len(item)
                    pre[i] = now[i-len(item)] if len(item) < i + 1 else "BOS"
                    now[i] = item
```

```
                    vit[i] = nfreq
    # 回溯记录分词结果
    result2 = []
    result2.append(len(sentence)-1)
    temp = len(sentence)-1
    while pre[temp] != "BOS":
        temp = prepos[temp]
        result2.append(temp)
    result2.append(-1)
    temp = []
    for i in range(1, len(result2)):
        result1.append([result2[len(result2)-i]+2, result2[len(result2)-i-1]+1])
        temp.append(sentence[result2[len(result2)-i]+1: result2[len(result2)-i-
1]+1])
    result.append(temp)

# 计算单个句子的准确率，可共用
def calAccuracyForSentence(predict, ans):
    sum = 0
    for item in ans:
        if item in predict:
            sum += 1
    return sum, len(ans), len(predict)

# 分词
def fenCi(testSet, ans, result):
    f1 = f2 = f3 = 0  # precise, recall, f1_score
    for i in range(len(testSet)):
        ci = []
        result1 = []
        cut(wfreq, testSet[i], ci)
        viterbiFC(wfreq, bifreq, testSet[i], ci, result1, result[i])
        t1, t2, t3 = calAccuracyForSentence(result[i], ans[i])
        f1 += t1
        f2 += t2
        f3 += t3
    precise = f1 / f3
    recall = f1 / f2
    f1_score = 2 * precise * recall / (precise + recall)
    return precise, recall, f1_score
```

4.2 词性标注

4.2.1 预处理

（1）预处理任务

1、从训练集学习模型参数，包括A、B、C、D、E。A为词性转移矩阵，`A[i][j]`表示词性i转移到词性j的频数；B为词性频度表，`B[i]`表示词性i出现的频数；C为词语/词性频度表，`C[i][j]`表示词i词性为j的频数；D为词性出现在句首的频度表，`D[i]`表示词性i出现在句首的频度；E为词性种类表，用于统计出现的所有词性。

2、对原始数据集做处理，去掉词性形成新的集合，作为测试集。

(2) 预处理代码

```
# 原始语料统计ABCDE
def preprocess1(trainSet):
    pre = [0, 0]
    for lines in trainSet:
        line = lines.replace(" [", "").replace("] ", "")
        line = line[:-1].split()
        for i in range(len(line)):
            pair = line[i].split("/")
            if len(pair) == 1 or pair[1] == "":
                continue

            # 词性种类
            if pair[1] not in E:
                E.append(pair[1])

            # 词性出现在句首的频度
            if i == 0:
                if pair[1] not in D.keys():
                    D[pair[1]] = 0
                D[pair[1]] += 1

            # 词性频度表
            if pair[1] not in B.keys():
                B[pair[1]] = 0
            B[pair[1]] += 1

            # 词语/词性频度表
            if pair[0] not in C.keys():
                C[pair[0]] = {}
            if pair[1] not in C[pair[0]].keys():
                C[pair[0]][pair[1]] = 0
            C[pair[0]][pair[1]] += 1

            # 统计词性转移矩阵
            if i != 0:
                if pre[1] not in A.keys():
                    A[pre[1]] = {}
                if pair[1] not in A[pre[1]].keys():
                    A[pre[1]][pair[1]] = 0
                A[pre[1]][pair[1]] += 1
            pre[0] = pair[0]
            pre[1] = pair[1]

# 原始语料去词性，将句子处理成词列表
# 包括带词性的set和不带词性的testSet，用于词性标注及其正确率检测
def preprocess2(set, testSet):
    for i in range(len(set)):
        set[i] = set[i].replace(" [", "").replace("] ", "")[:-1].split()
        line = set[i]
        sentence = []
        for i in range(len(line)):
            pair = line[i].split("/")
            if len(pair) == 1 or pair[1] == "":
                continue
            sentence.append(pair[0])
```

```
        testSet.append(sentence)
```

4.2.2 基于隐马尔可夫模型与Viterbi算法的词性标注

（1）算法设计

词性标注是序列标注模型，可通过HMM模型的解决。将词性序列作为隐藏序列，将词语序列作为观测序列，则可通过viterbi算法预测最优的词性序列。

算法具体步骤如下：

1、预处理，学习模型参数。

2、使用viterbi算法预测。

将待标注序列视为从句首到句尾的一条路径，通过viterbi算法获取概率最大的路径。输入为分词结果序列ci。

所需辅助数组如下：

```
    vit[i][j]：表示第i+1个词词性为j的概率。
    pre[i][j]：表示第i+1个词词性为j时第i个词最大概率的词性。
```

状态转移方程如下：

$$vit[i][j] = \max_k(vit[i][j], vit[i-1][k] + log(P_{trans} * P_{emit}))$$

其中k为词所有可能的词性，$P_{trans} = \frac{A[k][j]}{B[k]}$，表示前后两个词为词性k转移到词性j的概率，$P_{emit} = \frac{C[ci[i]][j]}{B[j]}$，表示当前词为词性j的概率。

求得概率最大的最优值后，从末尾按照$pre_{词性} = pre[i][now_{词性}]$回溯遍历一遍，即可获得最优解，即句子的词性标注序列。

（2）程序结构

```
# 词性标注viterbi
def viterbiCXBZ(ci, result):
    vit = {}    # vit[词序号][词性]：当前词为某种词性的概率
    pre = {}    # pre[词序号][词性]：当前词为某种词性是前一个词最大概率的词性
    for i in range(len(ci)):
        vit[i] = {}
        pre[i] = {}
        if ci[i] not in C.keys():
            C[ci[i]] = {}
            for type in E:
                C[ci[i]][type] = 1e-10
        for type in C[ci[i]].keys():
            vit[i][type] = -1e50
            pre[i][type] = (i - 1) if i != 0 else 0
    for i in range(len(ci)):
        if i == 0:
            for type in C[ci[i]].keys():
                if type not in D.keys():
                    D[type] = 1
                freq = math.log10(C[ci[i]][type] * D[type])
                vit[0][type] = freq
```

```
        else:
            for type2 in C[ci[i]].keys():
                freq = -1e100
                preType = 0
                for type1 in C[ci[i-1]].keys():
                    if type2 not in A[type1].keys():
                        A[type1][type2] = 1
                    freq1 = vit[i-1][type1] + math.log10(
                        A[type1][type2] / B[type1] * C[ci[i]][type2] / B[type2])
                    if freq1 > freq:
                        freq = freq1
                        preType = type1
                vit[i][type2] = freq
                pre[i][type2] = preType
    temp = len(ci) - 1
    while temp >= 0:
        maxx = -5e100
        record = 0
        for type in vit[temp].keys():
            if vit[temp][type] > maxx:
                maxx = vit[temp][type]
                record = type
        result.append(ci[temp] + "/" + record)
        temp -= 1
    result = result.reverse()

# 计算单个句子的准确率
def calAccuracyForSentence(predict, ans):
    sum = 0
    for item in ans:
        if item in predict:
            sum += 1
    return sum, len(ans), len(predict)

# 词性标注
def CXBZ(testSet, ans):
    f1 = f2 = f3 = 0
    resultCXBZ = [[] for i in range(len(testSet))]
    for i in range(len(testSet)):
        viterbiCXBZ(testSet[i], resultCXBZ[i])
        t1, t2, t3 = calAccuracyForSentence(resultCXBZ[i], ans[i])
        f1 += t1
        f2 += t2
        f3 += t3
    precise = f1 / f3
    recall = f1 / f2
    f1_score = 2 * precise * recall / (precise + recall)
    return precise, recall, f1_score
```

4.3 结果评价

4.3.1 分词结果评价

选取语料库中2000行数据进行评估，用准确率、召回率、f1、效率四个指标衡量分词算法性能，结果如下：

准确率	召回率	f1	效率
93.67%	96.26%	94.95%	5.66KB/s

4.3.2 词性标注结果评价

选取语料库中2000行数据进行评估，用准确率、召回率、f1、效率四个指标衡量词性标注算法性能，结果如下：

准确率	召回率	f1	效率
94.26%	94.25%	94.26%	25.71KB/s

4.3.3 总体结果评价

选取语料库中前10000行做训练，10000~12000行数据进行评估。对分词结果做词性标注，对总体结果（预处理、分词、词性标注）做评价。用准确率、召回率、f1、效率四个指标衡量词性标注性能，结果如下：

准确率	召回率	f1	效率
90.54%	93.04%	91.77%	3.43KB/s

4.4 命名实体识别

对于命名实体识别任务，选取人民日报语料库为数据集。对每个词用one-hot表示法构建词向量，利用批量梯度下降法做训练，在用二分类器对每个词做预测，识别实体。最后计算准确率、召回率、f1。由于时间原因，仅对机构词做了识别。

4.4.1 词频统计

统计出训练集中出现的词的频率，ns、nt标记为实体，将出现频率最高的300个实体词（出现次数9次以上）和1200个高频词（出现次数75次以上）提取出存入词向量文件“词向量.txt”。

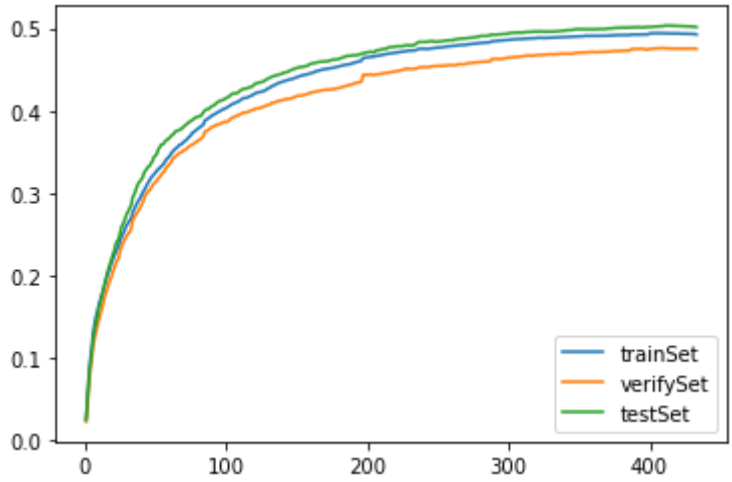
4.4.2 数据处理

从“词向量.txt”中读取词向量，将训练集、验证集、测试集的数据转化成Ci[ ]、Val[ ]（分别表示“词”和“是否是实体”）存储。同时设定正样本、负样本按1:1提取，从而加速梯度下降、提高模型准确率。

4.4.3 梯度下降

采用sigmoid函数为激励函数，选取批量梯度下降为优化算法寻找最优解，每10次epoch上梯度下降后计算一次训练集、验证集、测试集的f1_measure，并将准确率记录。当准确率近似不变时停止迭代，此时的测试集的准确率为模型准确率。

4.4.4 准确率计算



	训练集	验证集	测试集
预测正确实体数	12060	3936	2725
预测实体总数	29675	9905	6159
实际实体总数	19290	6670	4707
查准率	40.64%	39.74%	44.24%
查全率	62.52%	59.01%	57.89%
f1	49.26%	47.49%	50.16%

五、问题及解决方法

1. 问题：分词与词性标注中，均出现词的出现概率过低，导致长句子运算结果为0的情况。

   解决方法：将求乘法转化为求对数之和，避免出现数据下溢的情况。

2. 问题：分词与词性标注中，均存在未登录词。

   解决方法：采用+1数据平滑方法。对于分词中`wfreq`和`bifreq`均未出现的情况，若直接+1会导致$P(w_{i-1}\&w_i)$值过大，不符合实际，故统一赋值10^{-50}处理。

六、附录（命名实体识别代码）

```
import numpy as np
import torch
import matplotlib.pyplot as plt
import random

def GetSet(file):
    set = open(file, 'r')
    lists = []
    lines = set.readlines()
    for i in range(len(lines)):
        line = lines[i].replace('[', '*')
```

```
        line = line.replace(']ns', '%')
        line = line.replace(']nt', '%')
        line = line.replace(']', '&')
        lists.append(line)
    set.close()
    return lists


def sigmoid(t):
    if t < -10:
        return 0
    elif t > 10:
        return 1
    return 1 / (1 + np.exp(-t))


def get_Ci_Val(Set):
    Ci = []
    Val = []
    for lines in Set:
        l = r = -1
        now = 0
        while 1:
            l = lines.find('*', r + 1)
            temp = lines[now: l].split()
            for item in temp:
                ci = item.split('/')
                if ci[0] in Vec:
                    Ci.append(Vec.index(ci[0]))
                elif ci[1][0] == 'w':
                    Ci.append(300)
                else:
                    Ci.append(1500)
                Val.append(int(ci[1] in ['ns', 'nt']))
            if l == -1:
                break
            r = lines.find('%', l + 1)
            if r == -1:
                break
            now = r + 1
            temp = lines[l+1: r].split()
            for item in temp:
                ci = item.split('/')
                if ci[0] in Vec:
                    Ci.append(Vec.index(ci[0]))
                elif ci[1][0] == 'w':
                    Ci.append(300)
                else:
                    Ci.append(1500)
                Val.append(1)
    return Ci, Val


def get_Ci_Val1(Set):
    Ci = []
    Val = []
    rate = 10
    for lines in Set:
```

```
        l = r = -1
        now = 0
        while 1:
            l = lines.find('*', r + 1)
            temp = lines[now: l].split()
            for item in temp:
                ci = item.split('/')
                if rate <= 0 and ci[1] not in ['ns', 'nt']:
                    continue
                if ci[1] not in ['ns', 'nt']:
                    rate -= 1
                else:
                    rate += 1
                if ci[0] in Vec:
                    Ci.append(Vec.index(ci[0]))
                elif ci[1][0] == 'w':
                    Ci.append(300)
                else:
                    Ci.append(1500)
                Val.append(int(ci[1] in ['ns', 'nt']))
            if l == -1:
                break
            r = lines.find('%', l + 1)
            if r == -1:
                break
            now = r + 1
            temp = lines[l+1: r].split()
            for item in temp:
                ci = item.split('/')
                rate += 1
                if ci[0] in Vec:
                    Ci.append(Vec.index(ci[0]))
                elif ci[1][0] == 'w':
                    Ci.append(300)
                else:
                    Ci.append(1500)
                Val.append(1)
    return Ci, Val


def func(predict, real):
    f1 = 0    # 正确预测数
    f2 = 0    # 模型预测总数
    f3 = 0    # 实体总数
    flag2 = 0
    flag3 = 0
    for i in range(len(predict)):
        if predict[i] == 1 and flag2 == 0:
            f2 += 1
            flag2 = 1
        elif predict[i] == 0:
            flag2 = 0
        if real[i] == 1 and flag3 == 0:
            f3 += 1
            flag3 = 1
        elif real[i] == 0:
            flag3 = 0
    i = 0
```

```
    length = len(real)
    while i < length:
        if real[i] == 1:
            l = i
            i += 1
            while i < length and real[i] == 1:
                i += 1
            r = i
            flag1 = 1
            for j in range(l, r):
                if predict[j] == 0:
                    flag1 = 0
                    break
            if flag1 == 1:
                if (l > 0 and predict[l-1] == 1) or (r < length and predict[r]
== 1):
                    flag1 = 0
            if flag1 == 1:
                f1 += 1
        else:
            i += 1
    return f1, f2, f3


def cal_accuracy(Ci, Val, theta):
    predict = []
    for i in range(len(Ci)):
        temp = 0
        if i != 0:
            temp += theta[Ci[i-1]]
        temp += theta[Ci[i] + 1501]
        if i != len(Ci)-1:
            temp += theta[Ci[i+1] + 3002]
        res = sigmoid(temp)
        if res >= 0.9:
            predict.append(1)
        else:
            predict.append(0)
    f1, f2, f3 = func(predict, Val)

    print(f1, f2, f3)
    if f1 == 0 and f2 == 0 and f3 == 0:
        return 1
    elif f1 ==0 or f2 == 0 or f3 == 0:
        return 0
    else:
        return 2 * (f1 / f2) * (f1 / f3) / (f1 / f2 + f1 / f3)

file = open('词向量.txt')
trainSet = GetSet('训练集.txt')
verifySet = GetSet('验证集.txt')
testSet = GetSet('测试集.txt')
# 提取词向量
Vec = []

lines = file.readlines()
for item in lines:
    Vec.append(item[: -1])
```

```
    if len(Vec) >= 1500:
        break
# 提取Ci, Val

Ci_train, Val_train = get_Ci_Val1(trainSet)
Ci_trainSet, Val_trainSet = get_Ci_Val(trainSet)
Ci_verifySet, Val_verifySet = get_Ci_Val(verifySet)
Ci_testSet, Val_testSet = get_Ci_Val(testSet)

print(len(Ci_train))
print(len(Val_train))

theta = np.random.randn(4503, 1)
accuracy1 = []
accuracy2 = []
accuracy3 = []

for batch_size in range(10000):
    grad = np.zeros(4503)
    l = len(Ci_train)
    for i in range(l):
        temp = 0
        if i != 0:
            temp += theta[Ci_train[i-1]]
        temp += theta[Ci_train[i] + 1501]
        if i != l-1:
            temp += theta[Ci_train[i+1] + 3002]
        temp = Val_train[i] - sigmoid(temp)
        if i != 0:
            grad[Ci_train[i-1]] += temp * 0.0002
        grad[Ci_train[i] + 1501] += temp * 0.0002
        if i != l-1:
            grad[Ci_train[i+1] + 3002] += temp * 0.0002
    grad = np.transpose([grad])
    theta = theta + grad
    print(np.transpose(theta))
    if batch_size % 10 == 0:
        a = cal_accuracy(Ci_trainSet, Val_trainSet, theta)
        accuracy1.append(a)
        b = cal_accuracy(Ci_verifySet, Val_verifySet, theta)
        accuracy2.append(b)
        c = cal_accuracy(Ci_testSet, Val_testSet, theta)
        accuracy3.append(c)
        print(a, b, c)
file_a1 = open('save_a1.txt', 'w')
file_a2 = open('save_a2.txt', 'w')
file_a3 = open('save_a3.txt', 'w')

print(len(accuracy1))
print(len(accuracy2))
print(len(accuracy3))

for i in accuracy1:
    file_a1.writelines(str(i) + '\n')
for i in accuracy2:
    file_a2.writelines(str(i) + '\n')
for i in accuracy3:
    file_a3.writelines(str(i) + '\n')
```

```
file_theta = open('save_theta1.txt', 'w')
theta1 = np.transpose(theta)
for i in range(len(theta)):
    file_theta.writelines(str(theta[i]) + '\n')
x = [i+1 for i in range(len(accuracy1))]
plt.plot(x, accuracy1, label='trainSet')
plt.plot(x, accuracy2, label='verifySet')
plt.plot(x, accuracy3, label='testSet')
plt.legend(loc='best')
plt.show()
```