浙江工业大学

人工智能及其应用实验报告

(2021 级)

实验三：基于神经网络的模式识别实验

学生姓名：温家伟

学生学号：202103151422

学科专业：大数据分析 2101 班

所在学院：理学院

提交日期：2024 年 1 月 3 日

目录

1 实验目的

理解 BP、LSTM 神经网络的结构和原理，掌握反向传播学习算法对神经元的训练过程，了解反向传播公式。通过构建 BP、LSTM 网络模式识别实例，熟悉前馈网络、反馈网络和循环神经网络的原理及结构。

2 实验原理

BP 学习算法是通过反向学习过程使误差最小，其算法过程从输出节点开始，反向地向第一隐含层 (即最接近输入层的隐含层) 传播由总误差引起的权值修正。BP 网络不仅含有输入节点和输出节点，而且含有一层或多层隐 (层) 节点。输入信号先向前传递到隐节点，经过作用后，再把隐节点的输出信息传递到输出节点，最后给出输出结果。

长短期记忆（Long short-term memory, LSTM）是一种特殊的 RNN，主要是为了解决长序列训练过程中的梯度消失和梯度爆炸问题。简单来说，就是相比普通的 RNN，LSTM 能够在更长的序列中有更好的表现。

3 实验条件

华为平台，mindspore 框架

4 实验内容

1. 针对教材例 8.1，设计一个三层的 BP 网络结构模型，并以教材图 8.5 为训练样本数据，图 8.6 为测试数据。（1）给出训练成功后的连接权值和阈值，以及测试结果。（2）通过 BP 网络各项参数的不同设置，观察 BP 算法的学习效果，并比较 BP 网络各项参数变化对于训练结果的影响。

2. 自行搜索时序数据，或华为云平台提供的时序数据（文本、数字、语音等均可），设计一个 LSTM 网络框架，完成简单的序列标注、分类、翻译等工作（挑一项任务完成即可）。

5 实验结果

5.1 BP 手写数字

手写数字识别的实验结果如下：

图 1: 界面示意图

图 2: 界面示意图

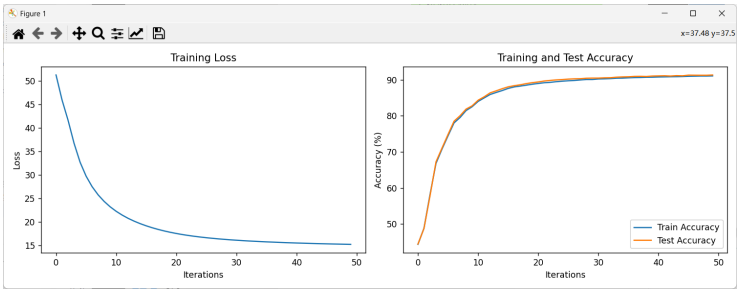

图 3: 训练集上的损失及训练集上的准确率

5.2　双向 LSTM 实现命名实体识别

双向 LSTM 实现命名实体识别的实验结果如下：

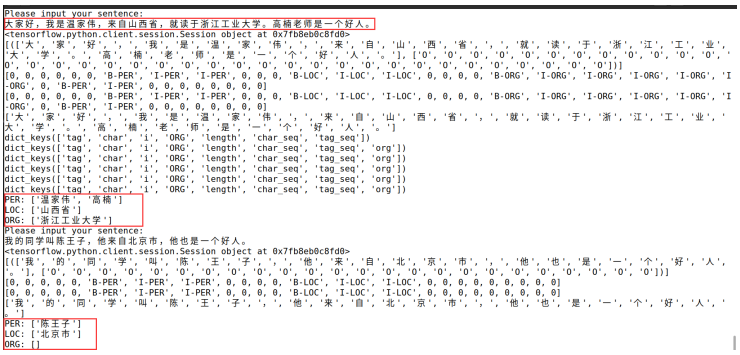

图 4: 训练集上的损失及训练集上的准确率

6 附录

6.1 手写数字

6.1.1 GUI.py

```
from tkinter import *
from PIL import ImageTk, Image
import numpy as np
from PIL import ImageGrab
from Method import predict

window = Tk()
window.title("Handwritten digit recognition")

# 加载背景图片
bg_image = Image.open("background.png")
bg_photo = ImageTk.PhotoImage(bg_image)

# 创建标签并设置背景图像
bg_label = Label(window, image=bg_photo)
bg_label.place(x=0, y=0, relwidth=1, relheight=1)

l1 = Label()
```

```
19
20      def MyProject():
21          global l1
22
23          widget = cv
24          #
                设置画布的坐标（加入参数index=55就对齐了，不知道为什么）
25          index = 55
26          x = window.winfo_rootx() + widget.winfo_x() +
                index
27          y = window.winfo_rooty() + widget.winfo_y() +
                index
28          x1 = x + widget.winfo_width() + index
29          y1 = y + widget.winfo_height() + index
30          # print(x, y, x1, y1)
31          # 从画布上提取图像并修改像素大小为 (28 X 28)
32          img = ImageGrab.grab().crop((x, y, x1,
                y1)).resize((28, 28))
33          # 灰度处理（感觉没必要，本身就是黑白）
34          img = img.convert('L')
35          # 提取图像的像素矩阵，并转换为(1, 784)的矩阵
36          x = np.asarray(img)
37          vec = np.zeros((1, 784))
38          k = 0
39          for i in range(28):
40              for j in range(28):
41                  vec[0][k] = x[i][j]
42                  k += 1
43          # print(vec)
44
45          print(vec)
46          # 加载之前存储的两层神将网络权重
47          Theta1 = np.loadtxt('Theta1.txt')
```

```
48          Theta2 = np.loadtxt('Theta2.txt')
49
50          # 调用识别功能
51          pred = predict(Theta1, Theta2, vec / 255)
52          # 展示结果
53          l1 = Label(window, text="识别数字 = " +
                str(pred[0]), font=('黑体', 20))
54          l1.place(x=200, y=450)
55
56
57      lastx, lasty = None, None
58
59
60      # 清除界面功能实现
61      def clear_widget():
62          global cv, l1
63          cv.delete("all")
64          l1.destroy()
65
66
67      # 开始识别
68      def event_activation(event):
69          global lastx, lasty
70          cv.bind('<B1-Motion>', draw_lines)
71          lastx, lasty = event.x, event.y
72
73
74      # 展示界面
75      def draw_lines(event):
76          global lastx, lasty
77          x, y = event.x, event.y
78          cv.create_line((lastx, lasty, x, y),
              width=30, fill='white', capstyle=ROUND,
```

```
                smooth=TRUE, splinesteps=12)
79          lastx, lasty = x, y
80
81
82      # 标题展示
83      L1 = Label(window, text="温家伟的BP网络",
            font=('华文新魏', 30), fg="blue")
84      L1.place(x=140, y=10)
85
86      # 清楚界面按钮
87      b1 = Button(window, text="1. 清除",
            font=('方正姚体', 20), bg="orange",
            fg="black", command=clear_widget)
88      b1.place(x=120, y=370)
89
90      # 识别数字按钮
91      b2 = Button(window, text="2. 识别",
            font=('方正姚体', 20), bg="white", fg="red",
            command=MyProject)
92      b2.place(x=320, y=370)
93
94      # 设置手写界面参数
95      cv = Canvas(window, width=350, height=290,
            bg='black')
96      cv.place(x=120, y=70)
97
98      cv.bind('<Button-1>', event_activation)
99      window.geometry("600x500")
100     window.mainloop()
```

6.1.2 Main.py

```
1       from scipy.io import loadmat
```

```
2       from Method import initialise, predict
3       from Optim import *
4       from draw import drawpic, draw_accuracy_curve
5
6       if __name__ == '__main__':
7           '''
8           导入数据集，划分为60，000个训练样本，10，000个测试样本
9           '''
10          # 加载数据文件
11          data = loadmat('mnist-original.mat')
12          # 提取数据的特征矩阵，并进行转置
13          X = data['data']
14          X = X.transpose()
15          #
                然后将特征除以255，重新缩放到[0,1]的范围内，以避免在计算过程中溢出
16          X = X / 255
17          # 从数据中提取labels
18          y = data['label']
19          y = y.flatten()
20          # 将数据分割为60,000个训练集
21          train_size = 60000
22          X_train = X[:train_size, :]
23          y_train = y[:train_size]
24          # 和10,000个测试集
25          test_size = 10000
26          X_test = X[train_size:train_size + test_size,
                :]
27          y_test = y[train_size:train_size + test_size]
28          '''
29          构建三层全连接神经网络的参数
30          '''
31          # 输入层，隐藏层，输出层节点个数
```

```
32          input_layer_size = 784  # 图片大小为 (28 X
               28) px 所以设置784个特征
33          hidden_layer_size = 16
34          num_labels = 10  # 拥有十个标准为 [0, 9]
               十个数字
35          # 初始化层之间的权重 Thetas
36          initial_Theta1 =
               initialise(hidden_layer_size,
               input_layer_size)  #
               输入层和隐藏层之间的权重
37          initial_Theta2 = initialise(num_labels,
               hidden_layer_size)  #
               隐藏层和输出层之间的权重
38          # 设置神经网络的参数
39          initial_nn_params =
               np.concatenate((initial_Theta1.flatten(),
               initial_Theta2.flatten()))
40          '''
41          进行神经网络的训练
42          '''
43          # 设置学习率和迭代次数
44          alpha = 0.01
45          max_iter = 50
46          lambda_reg = 0.1  # 避免过拟合
47          # 训练神经网络，根据函数选择优化方法
48          initial_nn_params =
               MiniBGD(initial_nn_params,
               input_layer_size, hidden_layer_size,
               num_labels, X_train, y_train,
49                                        lambda_reg,
                                             max_iter,
                                             alpha, X_test,
                                             y_test)
```

```
50
51          # 重新分割，获得三个层次之间两两的权重
52          Theta1 =
                np.reshape(initial_nn_params[:hidden_layer_size
                * (input_layer_size + 1)], (
53               hidden_layer_size, input_layer_size + 1))
                     # shape = (100, 785)
54          Theta2 =
                np.reshape(initial_nn_params[hidden_layer_size
                * (input_layer_size + 1):],
55                                 (num_labels,
                                      hidden_layer_size +
                                      1))   # shape = (10,
                                      101)
56          # 测试集的准确度
57          pred = predict(Theta1, Theta2, X_test)
58          print('Test Set Accuracy:
                {:f}'.format((np.mean(pred == y_test) *
                100)))
59          # 训练集的准确度
60          pred = predict(Theta1, Theta2, X_train)
61          print('Training Set Accuracy:
                {:f}'.format((np.mean(pred == y_train) *
                100)))
62
63          # 将Theta参数保存在txt文件中，用作后续程序识别
64          np.savetxt('Theta1.txt', Theta1, delimiter='
                ')
65          np.savetxt('Theta2.txt', Theta2, delimiter='
                ')
66
67          # 寻找最佳超参数
68          # data = []
```

```
69          # best_asem = []
70          # for alpha in [0.5, 0.1, 0.01, 0.001]:
71          #     for lambda_reg in [0.01, 0.1, 1.0]:
72          #         for max_iter in [10, 50, 100, 200]:
73          #             initial_nn_params =
            MiniBGD(initial_nn_params,
            input_layer_size, hidden_layer_size,
            num_labels, X_train,
74          #
            y_train,
75          #
            lambda_reg, max_iter, alpha, X_test,
            y_test)
76          #             Theta1 =
            np.reshape(initial_nn_params[:hidden_layer_size
            * (input_layer_size + 1)], (
77          #                 hidden_layer_size,
            input_layer_size + 1))  # shape = (100,
            785)
78          #             Theta2 =
            np.reshape(initial_nn_params[hidden_layer_size
            * (input_layer_size + 1):],
79          #
            (num_labels, hidden_layer_size + 1))  #
            shape = (10, 101)
80          #             # 测试集的准确度
81          #             pred = predict(Theta1, Theta2,
            X_test)
82          #
            print(f'alpha={alpha},lambda_reg={lambda_reg},max_iter={max_iter}
83          #             print('Accuracy:
            {:f}'.format((np.mean(pred == y_test) *
            100)))
```

```
84          #             data.append((alpha, lambda_reg,
                max_iter, np.mean(pred == y_test) * 100))
85          # drawpic(data)
86          # draw_accuracy_curve(data)
```

6.1.3 draw.py

```
 1      import matplotlib.pyplot as plt
 2      from mpl_toolkits.mplot3d import Axes3D
 3
 4
 5      # data = [
 6      #     (0.5, 0.01, 10, 94.09),
 7      #     (0.5, 0.01, 50, 94.63),
 8      #     (0.5, 0.01, 100, 94.43),
 9      #     (0.5, 0.01, 200, 94.56),
10      #     (0.5, 0.1, 10, 92.85),
11      #     (0.5, 0.1, 50, 93.10),
12      #     (0.5, 0.1, 100, 92.97),
13      #     (0.5, 0.1, 200, 92.60),
14      #     (0.5, 1.0, 10, 86.64),
15      #     (0.5, 1.0, 50, 87.19),
16      #     (0.5, 1.0, 100, 86.01),
17      #     (0.5, 1.0, 200, 87.96),
18      #     (0.1, 0.01, 10, 93.22),
19      #     (0.1, 0.01, 50, 94.39),
20      #     (0.1, 0.01, 100, 94.71),
21      #     (0.1, 0.01, 200, 95.00),
22      #     (0.1, 0.1, 10, 93.37),
23      #     (0.1, 0.1, 50, 92.91),
24      #     (0.1, 0.1, 100, 92.96),
25      #     (0.1, 0.1, 200, 92.90),
26      #     (0.1, 1.0, 10, 87.17),
```

```
27      #     (0.1, 1.0, 50, 87.92),
28      #     (0.1, 1.0, 100, 88.19),
29      #     (0.1, 1.0, 200, 88.09),
30      #     (0.01, 0.01, 10, 90.60),
31      #     (0.01, 0.01, 50, 93.02),
32      #     (0.01, 0.01, 100, 94.11),
33      #     (0.01, 0.01, 200, 94.66),
34      #     (0.01, 0.1, 10, 94.60),
35      #     (0.01, 0.1, 50, 93.70),
36      #     (0.01, 0.1, 100, 92.98),
37      #     (0.01, 0.1, 200, 92.82),
38      #     (0.01, 1.0, 10, 89.76),
39      #     (0.01, 1.0, 50, 88.19),
40      #     (0.01, 1.0, 100, 88.14),
41      #     (0.01, 1.0, 200, 88.17),
42      #     (0.001, 0.01, 10, 88.50),
43      #     (0.001, 0.01, 50, 89.95),
44      #     (0.001, 0.01, 100, 91.36),
45      #     (0.001, 0.01, 200, 92.43),
46      #     (0.001, 0.1, 10, 92.48),
47      #     (0.001, 0.1, 50, 92.65),
48      #     (0.001, 0.1, 100, 92.63),
49      #     (0.001, 0.1, 200, 92.71),
50      #     (0.001, 1.0, 10, 92.64),
51      #     (0.001, 1.0, 50, 90.94),
52      #     (0.001, 1.0, 100, 88.85),
53      #     (0.001, 1.0, 200, 88.26)
54      # ]
55      def drawpic(data):
56          alphas = [item[0] for item in data]
57          lambda_regs = [item[1] for item in data]
58          max_iters = [item[2] for item in data]
59          accuracies = [item[3] for item in data]
```

```
60
61          # 创建一个三维散点图
62          fig = plt.figure()
63          ax = fig.add_subplot(111, projection='3d')
64          ax.scatter(alphas, lambda_regs, max_iters,
                c=accuracies, cmap='viridis', marker='o')
65
66          # 设置坐标轴标签
67          ax.set_xlabel('Alpha')
68          ax.set_ylabel('Lambda Reg')
69          ax.set_zlabel('Max Iter')
70
71          # 添加颜色条
72          cbar = plt.colorbar(ax.scatter(alphas,
                lambda_regs, max_iters, c=accuracies,
                cmap='viridis', marker='o'))
73          cbar.set_label('Accuracy')
74
75          # 显示图表
76          plt.show()
77          data.sort(key=lambda x: (x[0], x[1], x[2]))
                # 按照alpha、lambda_reg和max_iter排序
78
79
80      def draw_accuracy_curve(data):
81          alphas = [item[0] for item in data]
82          lambda_regs = [item[1] for item in data]
83          max_iters = [item[2] for item in data]
84          accuracies = [item[3] for item in data]
85
86          # 创建一个新的图形
87          plt.figure()
88
```

```
89          # 绘制折线图
90          plt.plot(range(len(accuracies)), accuracies,
                marker='o', linestyle='-')
91          plt.xticks(range(len(accuracies)),
                [f'{alpha}, {lambda_reg}, {max_iter}' for
                alpha, lambda_reg, max_iter in
92                                              zip(alphas,
                                                    lambda_regs,
                                                    max_iters)],
                                                    rotation=45)
93          plt.xlabel('Hyperparameters (alpha,
                lambda_reg, max_iter)')
94          plt.ylabel('Accuracy')
95          plt.title('Accuracy vs. Hyperparameters')
96
97          # 显示图表
98          plt.tight_layout()
99          plt.show()
```

6.1.4 Accuracy.py

```
1       import numpy as np
2       from Method import predict
3
4
5       def accuracy(initial_nn_params, input_layer_size,
            hidden_layer_size, num_labels, X_train,
            y_train, X_test, y_test):
6           # 重新分割，获得三个层次之间两两的权重
7           Theta1 =
                np.reshape(initial_nn_params[:hidden_layer_size
                * (input_layer_size + 1)], (
```

```
8               hidden_layer_size, input_layer_size + 1))
                    # shape = (100, 785)
9           Theta2 =
                np.reshape(initial_nn_params[hidden_layer_size
                * (input_layer_size + 1):],
10                              (num_labels,
                                    hidden_layer_size +
                                    1))   # shape = (10,
                                    101)
11          # 训练集的准确度
12          train_pred = predict(Theta1, Theta2, X_train)
13          train_accuracy = np.mean(train_pred ==
                y_train) * 100
14
15          # 测试集的准确度
16          test_pred = predict(Theta1, Theta2, X_test)
17          test_accuracy = np.mean(test_pred == y_test)
                * 100
18
19          return train_accuracy, test_accuracy
```

6.1.5 Method.py

```
1       import numpy as np
2       from matplotlib import pyplot as plt
3
4
5       # 初始化权重，b->a，给b层中加入一个偏置节点
6       def initialise(a, b):
7           epsilon = 0.15
8           c = np.random.rand(a, b + 1) * (
9               # 随机初始化权重在 [-epsilon, +epsilon]
                    范围内
```

```
10                  2 * epsilon) - epsilon
11          return c
12
13
14      def loss(Theta1, Theta2, y_vect, a3, lamb, m):
15          # 计算损失值
16          # 正则化方法选择
17          # L2正则化
18          L2_Reg = (lamb / (2 * m)) * (
19                  np.sum(np.square(Theta1[:, 1:])) +
                        np.sum(np.square(Theta2[:, 1:])))
20          # L1正则化
21          L1_Reg = (lamb / (2 * m)) * (
22                  np.sum(np.abs(Theta1[:, 1:])) +
                        np.sum(np.abs(Theta2[:, 1:]))
23          )
24          Reg = L2_Reg
25          # 交叉熵损失（Cross-Entropy Loss）
26          J = (1 / m) * (np.sum(np.sum(-y_vect *
                np.log(a3) - (1 - y_vect) * np.log(1 -
                a3)))) + Reg
27          # 均方误差损失（Mean Squared Error Loss）
28          # J = (1 / (2 * m)) * np.sum(np.square(a3 -
                y_vect)) + Reg
29          return J
30
31
32      # 进行一次正向传播来得到结果，用于预测准确率
33      def predict(Theta1, Theta2, X):
34          m = X.shape[0]
35          one_matrix = np.ones((m, 1))
36          X = np.append(one_matrix, X, axis=1)  #
                给第一层加入偏置参数
```

```
37          z2 = np.dot(X, Theta1.transpose())
38          a2 = 1 / (1 + np.exp(-z2))  # 使用Sigmoid函数激活第二层
39          one_matrix = np.ones((m, 1))
40          a2 = np.append(one_matrix, a2, axis=1)  # 给第二层加入偏置参数
41          z3 = np.dot(a2, Theta2.transpose())
42          a3 = 1 / (1 + np.exp(-z3))  # 激活第三层
43          p = (np.argmax(a3, axis=1))  # 输出预测的分类
44          return p
45
46
47      def plot_loss_and_accuracy(loss_history, train_accuracy_history, test_accuracy_history):
48          # 绘制损失曲线
49          plt.figure(figsize=(12, 4))
50          plt.subplot(1, 2, 1)
51          plt.plot(range(len(loss_history)), loss_history, label='Loss')
52          plt.xlabel('Iterations')
53          plt.ylabel('Loss')
54          plt.title('Training Loss')
55
56          # 绘制准确度曲线
57          plt.subplot(1, 2, 2)
58          plt.plot(range(len(train_accuracy_history)), train_accuracy_history, label='Train Accuracy')
59          plt.plot(range(len(test_accuracy_history)), test_accuracy_history, label='Test Accuracy')
60          plt.xlabel('Iterations')
61          plt.ylabel('Accuracy (%)')
```

```
62          plt.title('Training and Test Accuracy')
63          plt.legend()  # 添加图例
64
65          plt.tight_layout()
66          plt.show()
```

6.1.6 Model.py

```
1       import numpy as np
2       from Method import loss
3
4
5       # 进行一次向前传播和向后传播，并返回损失和梯度
6       def neural_network(nn_params, input_layer_size,
            hidden_layer_size, num_labels, X, y, lamb):
7           # 分割获得三个层次之间两两的权重
8           Theta1 =
                np.reshape(nn_params[:hidden_layer_size *
                (input_layer_size + 1)],
9                                   (hidden_layer_size,
                                        input_layer_size + 1))
10          Theta2 =
                np.reshape(nn_params[hidden_layer_size *
                (input_layer_size + 1):],
11                                  (num_labels,
                                        hidden_layer_size + 1))
12
13          # 向前传播
14          m = X.shape[0]
15          one_matrix = np.ones((m, 1))
16          X = np.append(one_matrix, X, axis=1)  #
                向输入层添加偏置单元，使之成为偏差节点
17          a1 = X
```

```
18          z2 = np.dot(X, Theta1.transpose())
19          a2 = 1 / (1 + np.exp(-z2))  # 采用Sigmoid函数对隐藏层进行激活
20          one_matrix = np.ones((m, 1))
21          a2 = np.append(one_matrix, a2, axis=1)  # 向隐藏层添加偏置单元，使之成为偏差节点
22          z3 = np.dot(a2, Theta2.transpose())
23          a3 = 1 / (1 + np.exp(-z3))  # 采用Sigmoid函数对输出层进行激活
24          # 将标签改为一个长度为10的布尔向量，在向量的10个布尔数值里，哪个数等
25          y_vect = np.zeros((m, 10))
26          for i in range(m):
27              y_vect[i, int(y[i])] = 1
28
29          # 计算损失值
30          J = loss(Theta1, Theta2, y_vect, a3, lamb, m)
31
32          # 向后传播
33          Delta3 = a3 - y_vect
34          Delta2 = np.dot(Delta3, Theta2) * a2 * (1 -
                a2)
35          Delta2 = Delta2[:, 1:]
36
37          # 计算梯度
38          Theta1[:, 0] = 0
39          Theta1_grad = (1 / m) *
                np.dot(Delta2.transpose(), a1) + (lamb /
                m) * Theta1
40          Theta2[:, 0] = 0
41          Theta2_grad = (1 / m) *
                np.dot(Delta3.transpose(), a2) + (lamb /
                m) * Theta2
```

```
42          grad = np.concatenate((Theta1_grad.flatten(),
                Theta2_grad.flatten()))
43
44          return J, grad
```

6.1.7 Optim.py

```
1       import random
2
3       import numpy as np
4
5       from Model import neural_network
6       from Accuracy import accuracy
7       from Method import plot_loss_and_accuracy
8
9
10      def BGD(nn_params, input_layer_size,
            hidden_layer_size, num_labels, X, y,
            lambda_reg, iter_num, alpha_rate, X_test,
11              y_test):
12          # 创建空列表来存储损失和准确度
13          loss_history = []
14          train_accuracy_history = []  #
                用于存储训练集准确度的历史数据
15          test_accuracy_history = []  #
                用于存储测试集准确度的历史数据
16
17          for i in range(iter_num):
18              cost, grad = neural_network(nn_params,
                    input_layer_size, hidden_layer_size,
                    num_labels, X, y, lambda_reg)
19              nn_params -= alpha_rate * grad
20              loss_history.append(cost)
```

```
21
22              # 计算并记录训练集和测试集准确度
23              train_accuracy, test_accuracy =
                    accuracy(nn_params, input_layer_size,
                    hidden_layer_size, num_labels, X, y,
24                                                      X_test,
                                                           y_test)
25              train_accuracy_history.append(train_accuracy)
26              test_accuracy_history.append(test_accuracy)
27
28              print(f"Iteration {i}: Cost {cost}")
29              print('Training Set Accuracy:
                    {:f}'.format(train_accuracy))
30              print('Test Set Accuracy:
                    {:f}'.format(test_accuracy))
31
32          plot_loss_and_accuracy(loss_history,
                train_accuracy_history,
                test_accuracy_history)
33          return nn_params
34
35
36      def SGD(nn_params, input_layer_size,
            hidden_layer_size, num_labels, X, y,
            lambda_reg, iter_num, alpha_rate, X_test,
37              y_test):
38          batch_size = 1
39          m = X.shape[0]
40          # 创建空列表来存储损失和准确度
41          loss_history = []
42          train_accuracy_history = []  #
                用于存储训练集准确度的历史数据
```

```
43          test_accuracy_history = []  # 用于存储测试集准确度的历史数据
44
45          for i in range(iter_num):
46              indices = list(range(m))
47              random.shuffle(indices)
48              totalcost = 0
49              for j in range(0, m, batch_size):
50                  batch_indices = indices[j:j + batch_size]
51                  X_batch = X[batch_indices]
52                  y_batch = y[batch_indices]
53
54                  cost, grad = neural_network(nn_params, input_layer_size, hidden_layer_size, num_labels, X_batch, y_batch, lambda_reg)
55
56                  nn_params -= alpha_rate * grad
57                  totalcost += cost
58              train_accuracy, test_accuracy = accuracy(nn_params, input_layer_size, hidden_layer_size, num_labels, X, y, X_test, y_test)
59
60              train_accuracy_history.append(train_accuracy)
61              test_accuracy_history.append(test_accuracy)
62              loss_history.append(totalcost / m)
63              print(f"Iteration {i}: Cost {totalcost / m}")
64              print('Training Set Accuracy: {:f}'.format(train_accuracy))
```

```
65              print('Test Set Accuracy:
                    {:f}'.format(test_accuracy))
66
67          plot_loss_and_accuracy(loss_history,
                train_accuracy_history,
                test_accuracy_history)
68
69          return nn_params
70
71
72      def OGD(nn_params, input_layer_size,
            hidden_layer_size, num_labels, X, y,
            lambda_reg, iter_num, alpha_rate, X_test,
73              y_test):
74          batch_size = 32
75          m = X.shape[0]
76          # 创建空列表来存储损失和准确度
77          loss_history = []
78          train_accuracy_history = []  #
                用于存储训练集准确度的历史数据
79          test_accuracy_history = []  #
                用于存储测试集准确度的历史数据
80
81          for i in range(iter_num):
82              indices = list(range(m))
83              totalcost = 0
84              for j in range(0, m, batch_size):
85                  batch_indices = indices[j:j +
                        batch_size]
86                  X_batch = X[batch_indices]
87                  y_batch = y[batch_indices]
88
```

```
89                  cost, grad =
                        neural_network(nn_params,
                        input_layer_size,
                        hidden_layer_size, num_labels,
                        X_batch, y_batch,
90                                              lambda_reg)
91                  totalcost += cost
92                  nn_params -= alpha_rate * grad
93              loss_history.append(totalcost / m)
94              train_accuracy, test_accuracy =
                    accuracy(nn_params, input_layer_size,
                    hidden_layer_size, num_labels, X, y,
95                                                      X_test,
                                                          y_test)
96              train_accuracy_history.append(train_accuracy)
97              test_accuracy_history.append(test_accuracy)
98              print(f"Iteration {i}: Cost {totalcost /
                    m}")
99              print('Training Set Accuracy:
                    {:f}'.format(train_accuracy))
100             print('Test Set Accuracy:
                    {:f}'.format(test_accuracy))
101         plot_loss_and_accuracy(loss_history,
                train_accuracy_history,
                test_accuracy_history)
102
103         return nn_params
104
105
106     def MiniBGD(nn_params, input_layer_size,
            hidden_layer_size, num_labels, X, y,
            lambda_reg, iter_num, alpha_rate, X_test,
107                 y_test):
```

```
108         batch_size = 64
109         m = X.shape[0]
110         loss_history = []
111         train_accuracy_history = []  #
                用于存储训练集准确度的历史数据
112         test_accuracy_history = []  #
                用于存储测试集准确度的历史数据
113         for i in range(iter_num):
114             # 随机打乱数据和标签，以创建随机的小批次
115             indices = list(range(m))
116             random.shuffle(indices)
117             totalcost = 0
118             for j in range(0, m, batch_size):
119                 batch_indices = indices[j:j +
                        batch_size]
120                 X_batch = X[batch_indices]
121                 y_batch = y[batch_indices]
122
123                 cost, grad =
                        neural_network(nn_params,
                        input_layer_size,
                        hidden_layer_size, num_labels,
                        X_batch, y_batch,
124                                                 lambda_reg)
125                 nn_params -= alpha_rate * grad
126                 totalcost += cost
127             loss_history.append(totalcost /
                    batch_size)
128             train_accuracy, test_accuracy =
                    accuracy(nn_params, input_layer_size,
                    hidden_layer_size, num_labels, X, y,
129                                                     X_test,
                                                        y_test)
```

```
130             train_accuracy_history.append(train_accuracy)
131             test_accuracy_history.append(test_accuracy)
132             print(f"Iteration {i}: Cost {totalcost /
                    batch_size}")
133             print('Training Set Accuracy:
                    {:f}'.format(train_accuracy))
134             print('Test Set Accuracy:
                    {:f}'.format(test_accuracy))
135         plot_loss_and_accuracy(loss_history,
                train_accuracy_history,
                test_accuracy_history)
136
137         return nn_params
138
139
140     def Momentum(nn_params, input_layer_size,
            hidden_layer_size, num_labels, X, y,
            lambda_reg, iter_num, alpha_rate, X_test,
141                  y_test):
142         beta = 0.9
143         # 初始化动量向量
144         v = np.zeros(nn_params.shape)
145         # 创建空列表来存储损失和准确度
146         loss_history = []
147         train_accuracy_history = []  #
                用于存储训练集准确度的历史数据
148         test_accuracy_history = []  #
                用于存储测试集准确度的历史数据
149         m = X.shape[0]  # 总样本数
150         batch_size = 64
151         for i in range(iter_num):
152             totalcost = 0
153             # 随机打乱数据和标签，以创建随机的小批次
```

```
154             indices = list(range(m))
155             random.shuffle(indices)
156             for j in range(0, m, batch_size):
157                 batch_indices = indices[j:j +
                        batch_size]
158                 X_batch = X[batch_indices]
159                 y_batch = y[batch_indices]
160                 cost, grad =
                        neural_network(nn_params,
                        input_layer_size,
                        hidden_layer_size, num_labels,
                        X_batch, y_batch,
161                                                 lambda_reg)
162                 # 更新动量
163                 v = beta * v + alpha_rate * grad
164                 # 更新参数
165                 nn_params -= v
166                 totalcost += cost
167
168             # 计算并记录训练集和测试集准确度
169             train_accuracy, test_accuracy =
                    accuracy(nn_params, input_layer_size,
                    hidden_layer_size, num_labels, X, y,
170                                                     X_test,
                                                        y_test)
171             train_accuracy_history.append(train_accuracy)
172             test_accuracy_history.append(test_accuracy)
173             loss_history.append(totalcost /
                    batch_size)
174             print(f"Iteration {i}: Cost {totalcost /
                    batch_size}")
175             print('Training Set Accuracy:
                    {:f}'.format(train_accuracy))
```

```
176             print('Test Set Accuracy:
                    {:f}'.format(test_accuracy))
177
178         plot_loss_and_accuracy(loss_history,
                train_accuracy_history,
                test_accuracy_history)
179         return nn_params
180
181
182     def Adagrad(nn_params, input_layer_size,
            hidden_layer_size, num_labels, X, y,
            lambda_reg, iter_num, alpha_rate, X_test,
183                 y_test):
184         epsilon = 1e-8
185         # 初始化梯度平方累积
186         G = np.zeros(nn_params.shape)
187         # 创建空列表来存储损失和准确度
188         loss_history = []
189         train_accuracy_history = []  #
                用于存储训练集准确度的历史数据
190         test_accuracy_history = []  #
                用于存储测试集准确度的历史数据
191         m = X.shape[0]  # 总样本数
192         batch_size = 64
193         for i in range(iter_num):
194             totalcost = 0
195             # 随机打乱数据和标签，以创建随机的小批次
196             indices = list(range(m))
197             random.shuffle(indices)
198             for j in range(0, m, batch_size):
199                 batch_indices = indices[j:j +
                        batch_size]
200                 X_batch = X[batch_indices]
```

```
                y_batch = y[batch_indices]
                cost, grad =
                    neural_network(nn_params,
                    input_layer_size,
                    hidden_layer_size, num_labels,
                    X_batch, y_batch,
                                                lambda_reg)
                # 更新梯度平方累积
                G += np.square(grad)
                # 计算适应的学习率
                alpha = alpha_rate / np.sqrt(G +
                    epsilon)
                # 更新参数
                nn_params -= alpha * grad
                totalcost += cost
            # 计算并记录训练集和测试集准确度
            train_accuracy, test_accuracy =
                accuracy(nn_params, input_layer_size,
                hidden_layer_size, num_labels, X, y,
                                                    X_test,
                                                        y_test)
            train_accuracy_history.append(train_accuracy)
            test_accuracy_history.append(test_accuracy)
            loss_history.append(totalcost /
                batch_size)

            print(f"Iteration {i}: Cost {totalcost /
                batch_size}")
            print('Training Set Accuracy:
                {:f}'.format(train_accuracy))
            print('Test Set Accuracy:
                {:f}'.format(test_accuracy))
```

```
221         plot_loss_and_accuracy(loss_history,
                train_accuracy_history,
                test_accuracy_history)
222         return nn_params
223
224
225     def Adam(nn_params, input_layer_size,
            hidden_layer_size, num_labels, X, y,
            lambda_reg, iter_num, alpha_rate, X_test,
226             y_test):
227         beta1 = 0.9
228         beta2 = 0.999
229         epsilon = 1e-8
230         # 初始化一阶矩和二阶矩
231         m = np.zeros(nn_params.shape)
232         v = np.zeros(nn_params.shape)
233         t = 0
234         # 创建空列表来存储损失和准确度
235         loss_history = []
236         train_accuracy_history = []  #
                用于存储训练集准确度的历史数据
237         test_accuracy_history = []  #
                用于存储测试集准确度的历史数据
238         M = X.shape[0]  # 总样本数
239         batch_size = 64
240         for i in range(iter_num):
241             totalcost = 0
242             # 随机打乱数据和标签，以创建随机的小批次
243             indices = list(range(M))
244             random.shuffle(indices)
245             for j in range(0, M, batch_size):
246                 batch_indices = indices[j:j +
                        batch_size]
```

```
247                 X_batch = X[batch_indices]
248                 y_batch = y[batch_indices]
249                 cost, grad =
                        neural_network(nn_params,
                        input_layer_size,
                        hidden_layer_size, num_labels,
                        X_batch, y_batch,
250                                             lambda_reg)
251                 t += 1
252                 # 更新一阶矩和二阶矩
253                 m = beta1 * m + (1 - beta1) * grad
254                 v = beta2 * v + (1 - beta2) * (grad
                        ** 2)
255                 # 偏差修正
256                 m_hat = m / (1 - beta1 ** t)
257                 v_hat = v / (1 - beta2 ** t)
258                 # 计算适应的学习率
259                 alpha = alpha_rate / (np.sqrt(v_hat)
                        + epsilon)
260                 # 更新参数
261                 nn_params -= alpha * m_hat
262                 totalcost += cost
263             # 计算并记录训练集和测试集准确度
264             train_accuracy, test_accuracy =
                    accuracy(nn_params, input_layer_size,
                    hidden_layer_size, num_labels, X, y,
265                                                 X_test,
                                                    y_test)
266             train_accuracy_history.append(train_accuracy)
267             test_accuracy_history.append(test_accuracy)
268             loss_history.append(totalcost /
                    batch_size)
```

```
269             print(f"Iteration {i}: Cost {totalcost /
                    batch_size}")
270             print('Training Set Accuracy:
                    {:f}'.format(train_accuracy))
271             print('Test Set Accuracy:
                    {:f}'.format(test_accuracy))
272
273         plot_loss_and_accuracy(loss_history,
                train_accuracy_history,
                test_accuracy_history)
274         return nn_params
275
276
277     def Adamax(nn_params, input_layer_size,
            hidden_layer_size, num_labels, X, y,
            lambda_reg, iter_num, alpha_rate, X_test,
278                 y_test):
279         beta1 = 0.9
280         beta2 = 0.999
281         epsilon = 1e-8
282         # 初始化一阶矩和 max 累积
283         m = np.zeros(nn_params.shape)
284         u = np.zeros(nn_params.shape)
285
286         # 创建空列表来存储损失和准确度
287         loss_history = []
288         train_accuracy_history = []  #
                用于存储训练集准确度的历史数据
289         test_accuracy_history = []  #
                用于存储测试集准确度的历史数据
290         M = X.shape[0]  # 总样本数
291         batch_size = 32
292         for i in range(iter_num):
```

```
293             totalcost = 0
294             # 随机打乱数据和标签，以创建随机的小批次
295             indices = list(range(M))
296             random.shuffle(indices)
297             for j in range(0, M, batch_size):
298                 batch_indices = indices[j:j +
                        batch_size]
299                 X_batch = X[batch_indices]
300                 y_batch = y[batch_indices]
301                 cost, grad =
                        neural_network(nn_params,
                        input_layer_size,
                        hidden_layer_size, num_labels,
                        X_batch, y_batch,
302                                                lambda_reg)
303                 # 更新一阶矩和 max 累积
304                 m = beta1 * m + (1 - beta1) * grad
305                 u = np.maximum(beta2 * u,
                        np.abs(grad))
306                 # 计算适应的学习率
307                 alpha = alpha_rate / (u + epsilon)
308                 # 更新参数
309                 nn_params -= alpha * m
310                 # 记录损失和准确度
311                 totalcost += cost
312             # 计算并记录训练集和测试集准确度
313             train_accuracy, test_accuracy =
                    accuracy(nn_params, input_layer_size,
                    hidden_layer_size, num_labels, X, y,
314                                                    X_test,
                                                       y_test)
315             train_accuracy_history.append(train_accuracy)
316             test_accuracy_history.append(test_accuracy)
```

```
317             loss_history.append(totalcost /
                    batch_size)
318             print(f"Iteration {i}: Cost {totalcost /
                    batch_size}")
319             print('Training Set Accuracy:
                    {:f}'.format(train_accuracy))
320             print('Test Set Accuracy:
                    {:f}'.format(test_accuracy))
321
322         plot_loss_and_accuracy(loss_history,
                train_accuracy_history,
                test_accuracy_history)
323         return nn_params
```

6.2 双向 LSTM 实现命名实体识别

6.2.1 main.py

```
1   import tensorflow as tf
2   import numpy as np
3   import os, argparse, time, random
4   from model import BiLSTM_CRF
5   from utils import str2bool, get_logger, get_entity
6   from data import read_corpus, read_dictionary,
        tag2label, random_embedding
7
8
9   ## Session configuration
10  os.environ['CUDA_VISIBLE_DEVICES'] = '0'
11  os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'  #
        default: 0
12  config = tf.ConfigProto()
13  config.gpu_options.allow_growth = True
```

```
14      config.gpu_options.per_process_gpu_memory_fraction
            = 0.2  # need ~700MB GPU memory
15
16
17      ## hyperparameters
18      parser =
            argparse.ArgumentParser(description='BiLSTM-CRF
            for Chinese NER task')
19      parser.add_argument('--train_data', type=str,
            default='data_path', help='train data source')
20      parser.add_argument('--test_data', type=str,
            default='data_path', help='test data source')
21      parser.add_argument('--batch_size', type=int,
            default=64, help='#sample of each minibatch')
22      parser.add_argument('--epoch', type=int,
            default=40, help='#epoch of training')
23      parser.add_argument('--hidden_dim', type=int,
            default=300, help='#dim of hidden state')
24      parser.add_argument('--optimizer', type=str,
            default='Adam',
            help='Adam/Adadelta/Adagrad/RMSProp/Momentum/SGD')
25      parser.add_argument('--CRF', type=str2bool,
            default=True, help='use CRF at the top layer.
            if False, use Softmax')
26      parser.add_argument('--lr', type=float,
            default=0.001, help='learning rate')
27      parser.add_argument('--clip', type=float,
            default=5.0, help='gradient clipping')
28      parser.add_argument('--dropout', type=float,
            default=0.5, help='dropout keep_prob')
29      parser.add_argument('--update_embedding',
            type=str2bool, default=True, help='update
            embedding during training')
```

```
30      parser.add_argument('--pretrain_embedding',
            type=str, default='random', help='use
            pretrained char embedding or init it randomly')
31      parser.add_argument('--embedding_dim', type=int,
            default=300, help='random init char
            embedding_dim')
32      parser.add_argument('--shuffle', type=str2bool,
            default=True, help='shuffle training data
            before each epoch')
33      parser.add_argument('--mode', type=str,
            default='demo', help='train/test/demo')
34      parser.add_argument('--demo_model', type=str,
            default='1521112368', help='model for test and
            demo')
35      args = parser.parse_args()
36
37
38      ## get char embeddings
39      word2id = read_dictionary(os.path.join('.',
            args.train_data, 'word2id.pkl'))
40      if args.pretrain_embedding == 'random':
41          embeddings = random_embedding(word2id,
                args.embedding_dim)
42      else:
43          embedding_path = 'pretrain_embedding.npy'
44          embeddings =
                np.array(np.load(embedding_path),
                dtype='float32')
45
46
47      ## read corpus and get training data
48      if args.mode != 'demo':
```

```
49          train_path = os.path.join('.',
                args.train_data, 'train_data')
50          test_path = os.path.join('.', args.test_data,
                'test_data')
51          train_data = read_corpus(train_path)
52          test_data = read_corpus(test_path); test_size
                = len(test_data)
53
54
55      ## paths setting
56      paths = {}
57      timestamp = str(int(time.time())) if args.mode ==
            'train' else args.demo_model
58      output_path = os.path.join('.',
            args.train_data+"_save", timestamp)
59      if not os.path.exists(output_path):
            os.makedirs(output_path)
60      summary_path = os.path.join(output_path,
            "summaries")
61      paths['summary_path'] = summary_path
62      if not os.path.exists(summary_path):
            os.makedirs(summary_path)
63      model_path = os.path.join(output_path,
            "checkpoints/")
64      if not os.path.exists(model_path):
            os.makedirs(model_path)
65      ckpt_prefix = os.path.join(model_path, "model")
66      paths['model_path'] = ckpt_prefix
67      result_path = os.path.join(output_path, "results")
68      paths['result_path'] = result_path
69      if not os.path.exists(result_path):
            os.makedirs(result_path)
70      log_path = os.path.join(result_path, "log.txt")
```

```
71      paths['log_path'] = log_path
72      get_logger(log_path).info(str(args))
73
74
75      ## training model
76      if args.mode == 'train':
77          model = BiLSTM_CRF(args, embeddings,
                tag2label, word2id, paths, config=config)
78          model.build_graph()
79
80          ## hyperparameters-tuning, split train/dev
81          # dev_data = train_data[:5000]; dev_size =
                len(dev_data)
82          # train_data = train_data[5000:]; train_size
                = len(train_data)
83          # print("train data: {0}\ndev data:
                {1}".format(train_size, dev_size))
84          # model.train(train=train_data, dev=dev_data)
85
86          ## train model on the whole training data
87          print("train data:
                {}".format(len(train_data)))
88          model.train(train=train_data, dev=test_data)
                # use test_data as the dev_data to see
                overfitting phenomena
89
90      ## testing model
91      elif args.mode == 'test':
92          ckpt_file =
                tf.train.latest_checkpoint(model_path)
93          print(ckpt_file)
94          paths['model_path'] = ckpt_file
```

```
95          model = BiLSTM_CRF(args, embeddings,
                tag2label, word2id, paths, config=config)
96          model.build_graph()
97          print("test data: {}".format(test_size))
98          model.test(test_data)
99
100     ## demo
101     elif args.mode == 'demo':
102         ckpt_file =
                tf.train.latest_checkpoint(model_path)
103         print(ckpt_file)
104         paths['model_path'] = ckpt_file
105         model = BiLSTM_CRF(args, embeddings,
                tag2label, word2id, paths, config=config)
106         model.build_graph()
107         saver = tf.train.Saver()
108         with tf.Session(config=config) as sess:
109             print('============= demo =============')
110             saver.restore(sess, ckpt_file)
111             while(1):
112                 print('Please input your sentence:')
113                 demo_sent = input()
114                 if demo_sent == '' or
                        demo_sent.isspace():
115                     print('See you next time!')
116                     break
117                 else:
118                     demo_sent =
                            list(demo_sent.strip())
119                     demo_data = [(demo_sent, ['O'] *
                            len(demo_sent))]
120                     tag = model.demo_one(sess,
                            demo_data)
```

```
121                     PER, LOC, ORG = get_entity(tag,
                            demo_sent)
122                     print('PER: {}\nLOC: {}\nORG:
                            {}'.format(PER, LOC, ORG))
```

6.2.2 model.py

```
1   import numpy as np
2   import os, time, sys
3   import tensorflow as tf
4   from tensorflow.contrib.rnn import LSTMCell
5   from tensorflow.contrib.crf import
        crf_log_likelihood
6   from tensorflow.contrib.crf import viterbi_decode
7   from data import pad_sequences, batch_yield
8   from utils import get_logger
9   from eval import conlleval
10
11
12  class BiLSTM_CRF(object):
13      def __init__(self, args, embeddings,
            tag2label, vocab, paths, config):
14          self.batch_size = args.batch_size
15          self.epoch_num = args.epoch
16          self.hidden_dim = args.hidden_dim
17          self.embeddings = embeddings
18          self.CRF = args.CRF
19          self.update_embedding =
                args.update_embedding
20          self.dropout_keep_prob = args.dropout
21          self.optimizer = args.optimizer
22          self.lr = args.lr
23          self.clip_grad = args.clip
```

```
24              self.tag2label = tag2label
25              self.num_tags = len(tag2label)
26              self.vocab = vocab
27              self.shuffle = args.shuffle
28              self.model_path = paths['model_path']
29              self.summary_path = paths['summary_path']
30              self.logger =
                    get_logger(paths['log_path'])
31              self.result_path = paths['result_path']
32              self.config = config
33
34          def build_graph(self):
35              self.add_placeholders()
36              self.lookup_layer_op()
37              self.biLSTM_layer_op()
38              self.softmax_pred_op()
39              self.loss_op()
40              self.trainstep_op()
41              self.init_op()
42
43          def add_placeholders(self):
44              self.word_ids = tf.placeholder(tf.int32,
                    shape=[None, None], name="word_ids")
45              self.labels = tf.placeholder(tf.int32,
                    shape=[None, None], name="labels")
46              self.sequence_lengths =
                    tf.placeholder(tf.int32, shape=[None],
                    name="sequence_lengths")
47
48              self.dropout_pl =
                    tf.placeholder(dtype=tf.float32,
                    shape=[], name="dropout")
```

```
49              self.lr_pl =
                    tf.placeholder(dtype=tf.float32,
                    shape=[], name="lr")
50
51          def lookup_layer_op(self):
52              with tf.variable_scope("words"):
53                  _word_embeddings =
                        tf.Variable(self.embeddings,
54                                                  dtype=tf.float32,
55                                                  trainable=self.update_embeddi
56                                                  name="_word_embeddings")
57                  word_embeddings =
                        tf.nn.embedding_lookup(params=_word_embeddings,
58                                                          ids=self.word_ids,
59                                                          name="word_embeddin
60              self.word_embeddings =
                    tf.nn.dropout(word_embeddings,
                    self.dropout_pl)
61
62          def biLSTM_layer_op(self):
63              with tf.variable_scope("bi-lstm"):
64                  cell_fw = LSTMCell(self.hidden_dim)
65                  cell_bw = LSTMCell(self.hidden_dim)
66                  (output_fw_seq, output_bw_seq), _ =
                        tf.nn.bidirectional_dynamic_rnn(
67                      cell_fw=cell_fw,
68                      cell_bw=cell_bw,
69                      inputs=self.word_embeddings,
70                      sequence_length=self.sequence_lengths,
71                      dtype=tf.float32)
72                  output = tf.concat([output_fw_seq,
                        output_bw_seq], axis=-1)
```

```
73                  output = tf.nn.dropout(output,
                        self.dropout_pl)
74
75              with tf.variable_scope("proj"):
76                  W = tf.get_variable(name="W",
77                                      shape=[2 *
                                           self.hidden_dim,
                                           self.num_tags],
78                                      initializer=tf.contrib.layers.xavier_ini
79                                      dtype=tf.float32)
80
81                  b = tf.get_variable(name="b",
82                                      shape=[self.num_tags],
83                                      initializer=tf.zeros_initializer(),
84                                      dtype=tf.float32)
85
86                  s = tf.shape(output)
87                  output = tf.reshape(output, [-1,
                        2*self.hidden_dim])
88                  pred = tf.matmul(output, W) + b
89
90                  self.logits = tf.reshape(pred, [-1,
                        s[1], self.num_tags])
91
92          def loss_op(self):
93              if self.CRF:
94                  log_likelihood,
                        self.transition_params =
                        crf_log_likelihood(inputs=self.logits,
95                                                                      tag_i
96                                                                      seque
97                  self.loss =
                        -tf.reduce_mean(log_likelihood)
```

```
98
99              else:
100                 losses =
                       tf.nn.sparse_softmax_cross_entropy_with_logits(logits=sel
101                                                                    labe
102                 mask =
                       tf.sequence_mask(self.sequence_lengths)
103                 losses = tf.boolean_mask(losses, mask)
104                 self.loss = tf.reduce_mean(losses)
105
106             tf.summary.scalar("loss", self.loss)
107
108         def softmax_pred_op(self):
109             if not self.CRF:
110                 self.labels_softmax_ =
                       tf.argmax(self.logits, axis=-1)
111                 self.labels_softmax_ =
                       tf.cast(self.labels_softmax_,
                       tf.int32)
112
113         def trainstep_op(self):
114             with tf.variable_scope("train_step"):
115                 self.global_step = tf.Variable(0,
                       name="global_step",
                       trainable=False)
116                 if self.optimizer == 'Adam':
117                     optim =
                           tf.train.AdamOptimizer(learning_rate=self.lr_pl)
118                 elif self.optimizer == 'Adadelta':
119                     optim =
                           tf.train.AdadeltaOptimizer(learning_rate=self.lr_pl)
120                 elif self.optimizer == 'Adagrad':
```

```
121                     optim =
                           tf.train.AdagradOptimizer(learning_rate=self.lr_pl)
122                 elif self.optimizer == 'RMSProp':
123                     optim =
                           tf.train.RMSPropOptimizer(learning_rate=self.lr_pl)
124                 elif self.optimizer == 'Momentum':
125                     optim =
                           tf.train.MomentumOptimizer(learning_rate=self.lr_pl,
                           momentum=0.9)
126                 elif self.optimizer == 'SGD':
127                     optim =
                           tf.train.GradientDescentOptimizer(learning_rate=self.
128                 else:
129                     optim =
                           tf.train.GradientDescentOptimizer(learning_rate=self.
130
131                 grads_and_vars =
                       optim.compute_gradients(self.loss)
132                 grads_and_vars_clip =
                       [[tf.clip_by_value(g,
                       -self.clip_grad, self.clip_grad),
                       v] for g, v in grads_and_vars]
133                 self.train_op =
                       optim.apply_gradients(grads_and_vars_clip,
                       global_step=self.global_step)
134
135         def init_op(self):
136             self.init_op =
                   tf.global_variables_initializer()
137
138         def add_summary(self, sess):
139             """
140
```

```
141             :param sess:
142             :return:
143             """
144             self.merged = tf.summary.merge_all()
145             self.file_writer =
                    tf.summary.FileWriter(self.summary_path,
                    sess.graph)
146
147         def train(self, train, dev):
148             """
149
150             :param train:
151             :param dev:
152             :return:
153             """
154             saver =
                    tf.train.Saver(tf.global_variables())
155
156             with tf.Session(config=self.config) as
                    sess:
157                 sess.run(self.init_op)
158                 self.add_summary(sess)
159
160                 for epoch in range(self.epoch_num):
161                     self.run_one_epoch(sess, train,
                            dev, self.tag2label, epoch,
                            saver)
162
163         def test(self, test):
164             saver = tf.train.Saver()
165             with tf.Session(config=self.config) as
                    sess:
```

```
166                 self.logger.info('========== testing
                        ==========')
167                 saver.restore(sess, self.model_path)
168                 label_list, seq_len_list =
                        self.dev_one_epoch(sess, test)
169                 self.evaluate(label_list,
                        seq_len_list, test)
170
171         def demo_one(self, sess, sent):
172             """
173
174             :param sess:
175             :param sent:
176             :return:
177             """
178             label_list = []
179             for seqs, labels in batch_yield(sent,
                    self.batch_size, self.vocab,
                    self.tag2label, shuffle=False):
180                 label_list_, _ =
                        self.predict_one_batch(sess, seqs)
181                 label_list.extend(label_list_)
182             label2tag = {}
183             for tag, label in self.tag2label.items():
184                 label2tag[label] = tag if label != 0
                        else label
185             tag = [label2tag[label] for label in
                    label_list[0]]
186             return tag
187
188         def run_one_epoch(self, sess, train, dev,
                tag2label, epoch, saver):
189             """
```

```
190
191             :param sess:
192             :param train:
193             :param dev:
194             :param tag2label:
195             :param epoch:
196             :param saver:
197             :return:
198             """
199             num_batches = (len(train) +
                    self.batch_size - 1) // self.batch_size
200
201             start_time = time.strftime("%Y-%m-%d
                    %H:%M:%S", time.localtime())
202             batches = batch_yield(train,
                    self.batch_size, self.vocab,
                    self.tag2label, shuffle=self.shuffle)
203             for step, (seqs, labels) in
                    enumerate(batches):
204
205                 sys.stdout.write(' processing: {}
                        batch / {} batches.'.format(step +
                        1, num_batches) + '\r')
206                 step_num = epoch * num_batches + step
                        + 1
207                 feed_dict, _ =
                        self.get_feed_dict(seqs, labels,
                        self.lr, self.dropout_keep_prob)
208                 _, loss_train, summary, step_num_ =
                        sess.run([self.train_op,
                        self.loss, self.merged,
                        self.global_step],
209                                                                 feed_dict=feed_
```

```
210                 if step + 1 == 1 or (step + 1) % 300
                        == 0 or step + 1 == num_batches:
211                     self.logger.info(
212                         '{} epoch {}, step {}, loss:
                                {:.4}, global_step:
                                {}'.format(start_time,
                                epoch + 1, step + 1,
213
214
215                 self.file_writer.add_summary(summary,
                        step_num)
216
217                 if step + 1 == num_batches:
218                     saver.save(sess, self.model_path,
                            global_step=step_num)
219
220             self.logger.info('===========validation /
                        test===========')
221             label_list_dev, seq_len_list_dev =
                        self.dev_one_epoch(sess, dev)
222             self.evaluate(label_list_dev,
                        seq_len_list_dev, dev, epoch)
223
224         def get_feed_dict(self, seqs, labels=None,
                    lr=None, dropout=None):
225             """
226
227             :param seqs:
228             :param labels:
229             :param lr:
230             :param dropout:
231             :return: feed_dict
```

```
232             """
233             word_ids, seq_len_list =
                    pad_sequences(seqs, pad_mark=0)
234
235             feed_dict = {self.word_ids: word_ids,
236                          self.sequence_lengths:
                                 seq_len_list}
237             if labels is not None:
238                 labels_, _ = pad_sequences(labels,
                        pad_mark=0)
239                 feed_dict[self.labels] = labels_
240             if lr is not None:
241                 feed_dict[self.lr_pl] = lr
242             if dropout is not None:
243                 feed_dict[self.dropout_pl] = dropout
244
245             return feed_dict, seq_len_list
246
247         def dev_one_epoch(self, sess, dev):
248             """
249
250             :param sess:
251             :param dev:
252             :return:
253             """
254             label_list, seq_len_list = [], []
255             for seqs, labels in batch_yield(dev,
                    self.batch_size, self.vocab,
                    self.tag2label, shuffle=False):
256                 label_list_, seq_len_list_ =
                        self.predict_one_batch(sess, seqs)
257                 label_list.extend(label_list_)
258                 seq_len_list.extend(seq_len_list_)
```

```
259             return label_list, seq_len_list
260
261         def predict_one_batch(self, sess, seqs):
262             """
263
264             :param sess:
265             :param seqs:
266             :return: label_list
267                      seq_len_list
268             """
269             feed_dict, seq_len_list =
                   self.get_feed_dict(seqs, dropout=1.0)
270
271             if self.CRF:
272                 logits, transition_params =
                       sess.run([self.logits,
                       self.transition_params],
273                                                    feed_dict=feed_dict)
274                 label_list = []
275                 for logit, seq_len in zip(logits,
                       seq_len_list):
276                     viterbi_seq, _ =
                           viterbi_decode(logit[:seq_len],
                           transition_params)
277                     label_list.append(viterbi_seq)
278                 return label_list, seq_len_list
279
280             else:
281                 label_list =
                       sess.run(self.labels_softmax_,
                       feed_dict=feed_dict)
282                 return label_list, seq_len_list
283
```

```
284         def evaluate(self, label_list, seq_len_list,
                data, epoch=None):
285             """
286
287             :param label_list:
288             :param seq_len_list:
289             :param data:
290             :param epoch:
291             :return:
292             """
293             label2tag = {}
294             for tag, label in self.tag2label.items():
295                 label2tag[label] = tag if label != 0
                        else label
296
297             model_predict = []
298             for label_, (sent, tag) in
                    zip(label_list, data):
299                 tag_ = [label2tag[label__] for
                        label__ in label_]
300                 sent_res = []
301                 if  len(label_) != len(sent):
302                     print(sent)
303                     print(len(label_))
304                     print(tag)
305                 for i in range(len(sent)):
306                     sent_res.append([sent[i], tag[i],
                            tag_[i]])
307                 model_predict.append(sent_res)
308             epoch_num = str(epoch+1) if epoch != None
                    else 'test'
309             label_path =
                    os.path.join(self.result_path,
```

```
                'label_' + epoch_num)
310             metric_path =
                os.path.join(self.result_path,
                'result_metric_' + epoch_num)
311             for _ in conlleval(model_predict,
                label_path, metric_path):
312                 self.logger.info(_)
```

6.2.3 eval.py

```
1   import os
2
3
4   def conlleval(label_predict, label_path,
        metric_path):
5       """
6
7       :param label_predict:
8       :param label_path:
9       :param metric_path:
10      :return:
11      """
12      eval_perl = "./conlleval_rev.pl"
13      with open(label_path, "w") as fw:
14          line = []
15          for sent_result in label_predict:
16              for char, tag, tag_ in sent_result:
17                  tag = 'O' if tag == 'O' else tag
18                  char = char.encode("utf-8")
19                  line.append("{} {}
                        {}\n".format(char, tag, tag_))
20              line.append("\n")
21          fw.writelines(line)
```

```
22          os.system("perl {} < {} >
               {}".format(eval_perl, label_path,
               metric_path))
23          with open(metric_path) as fr:
24              metrics = [line.strip() for line in fr]
25          return metrics
```

6.2.4 data.py

```
1       import sys, pickle, os, random
2       import numpy as np
3
4       ## tags, BIO
5       tag2label = {"O": 0,
6                    "B-PER": 1, "I-PER": 2,
7                    "B-LOC": 3, "I-LOC": 4,
8                    "B-ORG": 5, "I-ORG": 6
9                    }
10
11
12      def read_corpus(corpus_path):
13          """
14          read corpus and return the list of samples
15          :param corpus_path:
16          :return: data
17          """
18          data = []
19          with open(corpus_path, encoding='utf-8') as
               fr:
20              lines = fr.readlines()
21          sent_, tag_ = [], []
22          for line in lines:
23              if line != '\n':
```

```
24                  [char, label] = line.strip().split()
25                  sent_.append(char)
26                  tag_.append(label)
27              else:
28                  data.append((sent_, tag_))
29                  sent_, tag_ = [], []
30
31          return data
32
33
34      def vocab_build(vocab_path, corpus_path,
            min_count):
35          """
36
37          :param vocab_path:
38          :param corpus_path:
39          :param min_count:
40          :return:
41          """
42          data = read_corpus(corpus_path)
43          word2id = {}
44          for sent_, tag_ in data:
45              for word in sent_:
46                  if word.isdigit():
47                      word = '<NUM>'
48                  elif ('\u0041' <= word <='\u005a') or
                        ('\u0061' <= word <='\u007a'):
49                      word = '<ENG>'
50                  if word not in word2id:
51                      word2id[word] = [len(word2id)+1,
                            1]
52                  else:
53                      word2id[word][1] += 1
```

```
54          low_freq_words = []
55          for word, [word_id, word_freq] in
               word2id.items():
56              if word_freq < min_count and word !=
                    '<NUM>' and word != '<ENG>':
57                  low_freq_words.append(word)
58          for word in low_freq_words:
59              del word2id[word]
60
61          new_id = 1
62          for word in word2id.keys():
63              word2id[word] = new_id
64              new_id += 1
65          word2id['<UNK>'] = new_id
66          word2id['<PAD>'] = 0
67
68          print(len(word2id))
69          with open(vocab_path, 'wb') as fw:
70              pickle.dump(word2id, fw)
71
72
73      def sentence2id(sent, word2id):
74          """
75
76          :param sent:
77          :param word2id:
78          :return:
79          """
80          sentence_id = []
81          for word in sent:
82              if word.isdigit():
83                  word = '<NUM>'
```

```
84              elif ('\u0041' <= word <= '\u005a') or
                    ('\u0061' <= word <= '\u007a'):
85                  word = '<ENG>'
86              if word not in word2id:
87                  word = '<UNK>'
88              sentence_id.append(word2id[word])
89          return sentence_id
90
91
92      def read_dictionary(vocab_path):
93          """
94
95          :param vocab_path:
96          :return:
97          """
98          vocab_path = os.path.join(vocab_path)
99          with open(vocab_path, 'rb') as fr:
100             word2id = pickle.load(fr)
101         print('vocab_size:', len(word2id))
102         return word2id
103
104
105     def random_embedding(vocab, embedding_dim):
106         """
107
108         :param vocab:
109         :param embedding_dim:
110         :return:
111         """
112         embedding_mat = np.random.uniform(-0.25,
                0.25, (len(vocab), embedding_dim))
113         embedding_mat = np.float32(embedding_mat)
114         return embedding_mat
```

```
115
116
117     def pad_sequences(sequences, pad_mark=0):
118         """
119
120         :param sequences:
121         :param pad_mark:
122         :return:
123         """
124         max_len = max(map(lambda x : len(x),
                sequences))
125         seq_list, seq_len_list = [], []
126         for seq in sequences:
127             seq = list(seq)
128             seq_ = seq[:max_len] + [pad_mark] *
                    max(max_len - len(seq), 0)
129             seq_list.append(seq_)
130             seq_len_list.append(min(len(seq),
                    max_len))
131         return seq_list, seq_len_list
132
133
134     def batch_yield(data, batch_size, vocab,
            tag2label, shuffle=False):
135         """
136
137         :param data:
138         :param batch_size:
139         :param vocab:
140         :param tag2label:
141         :param shuffle:
142         :return:
143         """
```

```
144         if shuffle:
145             random.shuffle(data)
146 
147         seqs, labels = [], []
148         for (sent_, tag_) in data:
149             sent_ = sentence2id(sent_, vocab)
150             label_ = [tag2label[tag] for tag in tag_]
151 
152             if len(seqs) == batch_size:
153                 yield seqs, labels
154                 seqs, labels = [], []
155 
156             seqs.append(sent_)
157             labels.append(label_)
158 
159         if len(seqs) != 0:
160             yield seqs, labels
```

6.2.5 utils.py

```
1   import logging, sys, argparse
2 
3 
4   def str2bool(v):
5       # copy from StackOverflow
6       if v.lower() in ('yes', 'true', 't', 'y',
            '1'):
7           return True
8       elif v.lower() in ('no', 'false', 'f', 'n',
            '0'):
9           return False
10      else:
```

```
11              raise argparse.ArgumentTypeError('Boolean
                    value expected.')
12
13
14      def get_entity(tag_seq, char_seq):
15          PER = get_PER_entity(tag_seq, char_seq)
16          LOC = get_LOC_entity(tag_seq, char_seq)
17          ORG = get_ORG_entity(tag_seq, char_seq)
18          return PER, LOC, ORG
19
20
21      def get_PER_entity(tag_seq, char_seq):
22          length = len(char_seq)
23          PER = []
24          for i, (char, tag) in enumerate(zip(char_seq,
                tag_seq)):
25              if tag == 'B-PER':
26                  if 'per' in locals().keys():
27                      PER.append(per)
28                      del per
29                  per = char
30                  if i+1 == length:
31                      PER.append(per)
32              if tag == 'I-PER':
33                  per += char
34                  if i+1 == length:
35                      PER.append(per)
36              if tag not in ['I-PER', 'B-PER']:
37                  if 'per' in locals().keys():
38                      PER.append(per)
39                      del per
40                  continue
41          return PER
```

```
42
43
44      def get_LOC_entity(tag_seq, char_seq):
45          length = len(char_seq)
46          LOC = []
47          for i, (char, tag) in enumerate(zip(char_seq,
                tag_seq)):
48              if tag == 'B-LOC':
49                  if 'loc' in locals().keys():
50                      LOC.append(loc)
51                      del loc
52                  loc = char
53                  if i+1 == length:
54                      LOC.append(loc)
55              if tag == 'I-LOC':
56                  loc += char
57                  if i+1 == length:
58                      LOC.append(loc)
59              if tag not in ['I-LOC', 'B-LOC']:
60                  if 'loc' in locals().keys():
61                      LOC.append(loc)
62                      del loc
63                  continue
64          return LOC
65
66
67      def get_ORG_entity(tag_seq, char_seq):
68          length = len(char_seq)
69          ORG = []
70          for i, (char, tag) in enumerate(zip(char_seq,
                tag_seq)):
71              if tag == 'B-ORG':
72                  if 'org' in locals().keys():
```

```
73                      ORG.append(org)
74                      del org
75                  org = char
76                  if i+1 == length:
77                      ORG.append(org)
78              if tag == 'I-ORG':
79                  org += char
80                  if i+1 == length:
81                      ORG.append(org)
82              if tag not in ['I-ORG', 'B-ORG']:
83                  if 'org' in locals().keys():
84                      ORG.append(org)
85                      del org
86                  continue
87          return ORG
88
89
90      def get_logger(filename):
91          logger = logging.getLogger('logger')
92          logger.setLevel(logging.DEBUG)
93          logging.basicConfig(format='%(message)s',
                level=logging.DEBUG)
94          handler = logging.FileHandler(filename)
95          handler.setLevel(logging.DEBUG)
96          handler.setFormatter(logging.Formatter('%(asctime)s:%(levelname)s:
                %(message)s'))
97          logging.getLogger().addHandler(handler)
98          return logger
```