深度学习方法与实践第九次作业

训练和优化方法

姓名：杨玉雷
学号：18023040

1.参数正则化

要求：训练 MNIST 分类模型，比较不同学习率情况下，loss 的收敛情况和实际精度 acc 的变化情况。比较添加参数正则化方法防止模型过拟合的效果。

模型结构要求：使用如下全连接网络：

```
def model(x):
    w1=tf.Variable(dtype=tf.float32,  initial_value=np.random.rand(784,1500))
    w2=tf.Variable(dtype=tf.float32,  initial_value=np.random.rand(1500,1000))
    w3=tf.Variable(dtype=tf.float32,  initial_value=np.random.rand(1000,500),name='w3')
    w4=tf.Variable(dtype=tf.float32,  initial_value=np.random.rand(500,10),name='w4')
    b1=tf.Variable(dtype=tf.float32,  initial_value=np.random.rand(1500))
    b2=tf.Variable(dtype=tf.float32,  initial_value=np.random.rand(1000))
    b3=tf.Variable(dtype=tf.float32, initial_value=np.random.rand(500))
    b4=tf.Variable(dtype=tf.float32, initial_value=np.random.rand(10))

    fc1=tf.nn.relu(tf.matmul(x,w1)+b1)
    fc2=tf.nn.relu(tf.matmul(fc1,w2)+b2)
    fc3=tf.nn.relu(tf.matmul(fc2,w3)+b3)
    fc4=tf.matmul(fc3,w4)+b4
    return fc4
```

参数设置：batchsize=64, 迭代 30000 次，使用 AdamOptimizer

实验内容：1. 比较学习率 lr=0.0001 和 lr=0.005 时的网络学习效果。画出 training loss, validation loss, validation acc 曲线（每 100 次迭代记一下。为了更容易看出后期起伏效果，可单独再绘制一条曲线，不包含前 20 个左右的记录点（loss 快速降落的区域）使得后期变化能够加显著的被可视化。）。给出最终网络的 test acc 以及 网络中 w1,w2,w3,w4 的参数矩阵可视化图。观察，是否某些 lr 会在训练中间，有精度先上升并保持一段时间（acc 甚至到 0.95 左右维持一段时间），后来 acc 又开始下降，但 loss 却一直保持下降的情况？其他规律？不同 lr 最后的收敛效果？（感兴趣的可以看一下 lr=0.05，lr=0.00005 的情况）

2. 比较 lr=0.005 时，使用参数正则化和不使用参数正则化（l2 正则化推荐 lamda 0.0005）的训练效果。画出 training loss, validation loss, validation acc 曲线（每 100 次迭代记一下）。给出最终网络的 test acc 以及 网络中 w1,w2,w3,w4 的参数矩阵可视化图。正则化是否提升了网络训练效果？参数矩阵比较，是否稀疏化了？

提交：1. 实验报告：包括如上所要求曲线图、参数可视化图以及其他要求数据；实验截图。2. 代码。

2.实验过程

本次实验采用全连接网络来对 mnist 数据集进行分类，比较在不同学习率，参数使用和不使用正则化等情况下的实验效果，并绘制参数矩阵。

2.1 不同学习率的情况

（1）在 fcnet.py 中使用类 fcNet 封装全连接网络结构：

```
class fcNet:

    def __init__(self, learning_rate):
        self.learning_rate = learning_rate
        # 在创建的时候运行画图
        self._build_graph()

    # 涉及网络的所有画图 build graph 过程,常用一个 build graph 封起来
    def _build_graph(self, network_name='fcNet'):
        self._setup_placeholders_graph()
        self._build_network_graph(network_name)
        self._compute_loss_graph()
        self._create_train_op_graph()
        self._compute_acc_graph()

    # 构建图
    def _setup_placeholders_graph(self):
        self.x = tf.placeholder(tf.float32, [None, 784], name='input_x')
        self.y = tf.placeholder(tf.int32, [None,10])

    def _build_network_graph(self, scope_name):
        with tf.variable_scope(scope_name):
            #生成一组服从“0~1”均匀分布的随机样本值。随机样本取值范围是[0,
1), 不包括 1
            w1=tf.Variable(dtype=tf.float32, initial_value=np.random.rand(7
84, 1500), name='w1') #维度(784, 1500)
            w2 = tf.Variable(dtype=tf.float32, initial_value=np.random.rand
(1500, 1000), name='w2')
```

```
            w3 = tf.Variable(dtype=tf.float32, initial_value=np.random.rand(1000, 500), name='w3')
            w4 = tf.Variable(dtype=tf.float32, initial_value=np.random.rand(500, 10), name='w4')
            b1 = tf.Variable(dtype=tf.float32, initial_value=np.random.rand(1500), name='b1') #返回一个值
            b2 = tf.Variable(dtype=tf.float32, initial_value=np.random.rand(1000), name='b2')
            b3 = tf.Variable(dtype=tf.float32, initial_value=np.random.rand(500), name='b3')
            b4 = tf.Variable(dtype=tf.float32, initial_value=np.random.rand(10), name='b4')

            fc1 = tf.nn.relu(tf.matmul(self.x, w1) + b1)
            fc2 = tf.nn.relu(tf.matmul(fc1, w2) + b2)
            fc3 = tf.nn.relu(tf.matmul(fc2, w3) + b3)
            fc4 = tf.matmul(fc3, w4) + b4

            self.digits = fc4
            return self.digits

    def _compute_loss_graph(self):
        cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=self.y, logits=self.digits)
        self.loss = tf.reduce_mean(cross_entropy)

    def _create_train_op_graph(self):
        self.train_op = tf.train.AdamOptimizer(self.learning_rate).minimize(self.loss)

    def _compute_acc_graph(self):
        self.prediction = tf.equal(tf.argmax(self.digits, 1), tf.argmax(self.y, 1))
        self.accuracy = tf.reduce_mean(tf.cast(self.prediction, tf.float32))
```

（2）在 train.py 中训练网络，绘制 training loss, validation loss, validation acc 曲线和参数矩阵可视化图

```
#加载数据
def load_data(signal, batchsize):
    '''
    导入数据
    :return:训练集、测试集、验证集
    '''
```

```
    X_train, y_train = mninst.train.next_batch(batchsize)
    X_valid = mninst.validation.images
    y_valid = mninst.validation.labels
    X_test = mninst.test.images
    y_test = mninst.test.labels

    assert (len(X_train) == len(y_train))
    assert (len(X_test) == len(y_test))

    if signal == "train":
        return X_train, y_train
    if signal == "validation":
        return X_valid, y_valid
    elif signal == "test":
        return X_test, y_test

#绘制 training loss, validation loss,  validation acc 曲线
def img_save(learning_rate,y1,y2,y3,signal1,signal2,signal3):
    x= [i for i in range(0, 30000)]
    p1,= plt.plot(x, y1, label=signal1)

    p2,=plt.plot(x, y2, label=signal2)

    name= str(learning_rate) +"loss_acc" + ".png"
    p3,= plt.plot(x, y3, label=signal3)

    plt.legend(loc=0, ncol=1)  # 参数：loc 设置显示的位置，0 是自适应；ncol 设
置显示的列数
    plt.xlabel("iteration")
    plt.legend([p1, p2,p3], [signal1,signal2,signal3], loc='upper left')
    plt.savefig(name)


#画出参数可视化矩阵并存储到本地
def save_w(learning_rate,w1,w2,w3,w4):
    w1_min = np.min(w1)
    w1_max = np.max(w1)
    w1_0_to_1 = (w1 - w1_min) / (w1_max - w1_min)
    image.imsave(str(learning_rate)+'w1.png', w1_0_to_1)

    w2_min = np.min(w2)
    w2_max = np.max(w2)
    w2_0_to_1 = (w2 - w2_min) / (w2_max - w2_min)
    image.imsave(str(learning_rate)+'w2.png', w2_0_to_1)
```

```
    w3_min = np.min(w3)
    w3_max = np.max(w3)
    w3_0_to_1 = (w3 - w3_min) / (w3_max - w3_min)
    image.imsave(str(learning_rate)+'w3.png', w3_0_to_1)

    w4_min = np.min(w4)
    w4_max = np.max(w4)
    w4_0_to_1 = (w4 - w4_min) / (w4_max - w4_min)
    image.imsave(str(learning_rate)+'w4.png', w4_0_to_1)

#显示所有变量
def show_all_variables():
    model_vars = tf.trainable_variables()
    slim.model_analyzer.analyze_vars(model_vars, print_info=True)

TRAINING_STEPS=30000
batchsize=64
def main():
    for learning_rate in [0.0001,0.005]:
        training_loss=[]
        validation_loss=[]
        vaildation_acc=[]
        test_acc=0
        fcnet=fcNet(learning_rate)
        # 显示所有变量
        show_all_variables()

        with tf.Session() as sess:
            sess.run(tf.global_variables_initializer())

            for step in range(TRAINING_STEPS):
               X_train,y_train=load_data("train",batchsize)
               _,train_loss,train_acc=sess.run([fcnet.train_op,fcnet.loss,fcnet.accuracy],feed_dict={fcnet.x:X_train,fcnet.y:y_train})

               X_valid,y_valid=load_data("validation",batchsize)
               valid_loss,valid_acc=sess.run([fcnet.loss,fcnet.accuracy],feed_dict={fcnet.x:X_valid,fcnet.y:y_valid})

               X_test,y_test=load_data("test",batchsize)
               training_loss.append(train_loss)
               validation_loss.append(valid_loss)
               vaildation_acc.append(valid_acc)
```

```
            if  step%100==99:
                print("step=",step,",train_loss=",train_loss,",valid_loss=",valid_loss,",valid_acc=",valid_acc)
        test_loss, test_acc = sess.run([fcnet.loss, fcnet.accuracy],feed_dict={fcnet.x: X_test, fcnet.y: y_test})
        #输出最终的 test_acc
        print("test acc:",test_acc)

        #获取图里的所有 tensor
        # graph = tf.get_default_graph()
        # for op in graph.get_operations():
        #     print(op.name)

        w1 = sess.run('fcNet/w1:0')
        w2= sess.run('fcNet/w2:0')
        w3 = sess.run('fcNet/w3:0')
        w4 = sess.run('fcNet/w4:0')
        #方法二
        # w1 = sess.run(graph.get_tensor_by_name("fcNet/w1:0"))
        #绘制参数矩阵
        save_w(learning_rate,w1,w2,w3,w4)
        #绘制 loss 和 acc 曲线
        img_save(learning_rate,training_loss,validation_loss,vaildation_acc,"train_loss","valid_loss","valid_acc")

if __name__ == '__main__':
    main()
```

2.1 学习率为 0.005 时使用参数正则化

添加正则化的代码只需将计算 loss 的方法改为：

```
    def _compute_loss_graph(self):
        cross_entropy = tf.nn.softmax_cross_entropy_with_logits(labels=self.y, logits=self.digits)
        self.loss = tf.reduce_mean(cross_entropy)
        #下面代码为参数使用正则化
        tv = tf.trainable_variables()
        lambda_l = 0.0005
        Regularization_term = lambda_l * tf.reduce_sum([tf.nn.l2_loss(v) for v in tv])
        self.loss = Regularization_term + self.loss
        tf.summary.scalar("loss", self.loss)
```

3.实验结果

（1）学习率为 0.005 时训练过程如图 1 所示。

```
step= 599 ,train_loss= 1068.13 ,valid_loss= 1647.64 ,valid_acc= 0.7798
step= 699 ,train_loss= 3954.06 ,valid_loss= 5783.49 ,valid_acc= 0.6822
step= 799 ,train_loss= 3879.53 ,valid_loss= 4412.99 ,valid_acc= 0.676
step= 899 ,train_loss= 2847.8 ,valid_loss= 2495.26 ,valid_acc= 0.753
step= 999 ,train_loss= 3578.0 ,valid_loss= 3158.9 ,valid_acc= 0.7268
step= 1099 ,train_loss= 1485.53 ,valid_loss= 2960.37 ,valid_acc= 0.7558
step= 1199 ,train_loss= 2529.56 ,valid_loss= 3810.82 ,valid_acc= 0.7314
step= 1299 ,train_loss= 2625.38 ,valid_loss= 2683.35 ,valid_acc= 0.7476
step= 1399 ,train_loss= 3699.56 ,valid_loss= 3475.79 ,valid_acc= 0.768
step= 1499 ,train_loss= 1265.22 ,valid_loss= 2096.8 ,valid_acc= 0.7948
step= 1599 ,train_loss= 1132.25 ,valid_loss= 1211.15 ,valid_acc= 0.8342
step= 1699 ,train_loss= 1748.98 ,valid_loss= 1108.52 ,valid_acc= 0.8438
step= 1799 ,train_loss= 359.219 ,valid_loss= 789.336 ,valid_acc= 0.8756
step= 1899 ,train_loss= 479.5 ,valid_loss= 1112.01 ,valid_acc= 0.8478
step= 1999 ,train_loss= 2135.92 ,valid_loss= 1972.45 ,valid_acc= 0.7464
step= 2099 ,train_loss= 648.813 ,valid_loss= 683.336 ,valid_acc= 0.8678
step= 2199 ,train_loss= 270.047 ,valid_loss= 719.737 ,valid_acc= 0.8672
step= 2299 ,train_loss= 320.664 ,valid_loss= 454.854 ,valid_acc= 0.906
step= 2399 ,train_loss= 534.078 ,valid_loss= 737.546 ,valid_acc= 0.8616
step= 2499 ,train_loss= 552.945 ,valid_loss= 549.166 ,valid_acc= 0.8856
step= 2599 ,train_loss= 434.141 ,valid_loss= 326.536 ,valid_acc= 0.9136
step= 2699 ,train_loss= 308.734 ,valid_loss= 302.875 ,valid_acc= 0.9136
step= 2799 ,train_loss= 82.5586 ,valid_loss= 503.185 ,valid_acc= 0.8812
step= 2899 ,train_loss= 307.941 ,valid_loss= 414.798 ,valid_acc= 0.889
step= 2999 ,train_loss= 486.773 ,valid_loss= 270.071 ,valid_acc= 0.9124
```

```
step= 6999 ,train_loss= 5.44458 ,valid_loss= 52.5477 ,valid_acc= 0.8918
step= 7099 ,train_loss= 4.47482 ,valid_loss= 21.1973 ,valid_acc= 0.9362
step= 7199 ,train_loss= 13.1388 ,valid_loss= 19.3678 ,valid_acc= 0.936
step= 7299 ,train_loss= 17.8204 ,valid_loss= 21.7649 ,valid_acc= 0.9332
step= 7399 ,train_loss= 5.13172 ,valid_loss= 24.0856 ,valid_acc= 0.9254
step= 7499 ,train_loss= 48.1541 ,valid_loss= 27.237 ,valid_acc= 0.9178
step= 7599 ,train_loss= 5.2448 ,valid_loss= 13.4338 ,valid_acc= 0.9436
step= 7699 ,train_loss= 6.75581 ,valid_loss= 16.3837 ,valid_acc= 0.9344
step= 7799 ,train_loss= 63.1542 ,valid_loss= 19.3721 ,valid_acc= 0.934
step= 7899 ,train_loss= 23.1377 ,valid_loss= 17.8165 ,valid_acc= 0.9262
step= 7999 ,train_loss= 4.09186 ,valid_loss= 20.356 ,valid_acc= 0.9364
step= 8099 ,train_loss= 4.47979 ,valid_loss= 21.4847 ,valid_acc= 0.9324
step= 8199 ,train_loss= 17.3948 ,valid_loss= 16.9783 ,valid_acc= 0.938
step= 8299 ,train_loss= 0.666216 ,valid_loss= 14.0514 ,valid_acc= 0.9328
step= 8399 ,train_loss= 1.12296 ,valid_loss= 18.208 ,valid_acc= 0.9342
step= 8499 ,train_loss= 1.59439 ,valid_loss= 14.0056 ,valid_acc= 0.9356
step= 8599 ,train_loss= 2.88884 ,valid_loss= 13.5409 ,valid_acc= 0.9252
step= 8699 ,train_loss= 2.26465 ,valid_loss= 14.0503 ,valid_acc= 0.938
step= 8799 ,train_loss= 6.62843 ,valid_loss= 16.108 ,valid_acc= 0.9284
step= 8899 ,train_loss= 22.5433 ,valid_loss= 19.3633 ,valid_acc= 0.8978
step= 8999 ,train_loss= 1.3215 ,valid_loss= 13.3069 ,valid_acc= 0.9304
step= 9099 ,train_loss= 7.48608 ,valid_loss= 11.7564 ,valid_acc= 0.9404
step= 9199 ,train_loss= 2.00731 ,valid_loss= 10.4082 ,valid_acc= 0.9386
step= 9299 ,train_loss= 1.49652 ,valid_loss= 21.3025 ,valid_acc= 0.9154
step= 9399 ,train_loss= 8.00868 ,valid_loss= 12.2014 ,valid_acc= 0.9282
```

```
step= 27599 ,train_loss= 1.20618 ,valid_loss= 2.15169 ,valid_acc= 0.6898
step= 27699 ,train_loss= 1.06448 ,valid_loss= 1.89445 ,valid_acc= 0.7116
step= 27799 ,train_loss= 0.883849 ,valid_loss= 1.64624 ,valid_acc= 0.7286
step= 27899 ,train_loss= 0.626879 ,valid_loss= 1.84726 ,valid_acc= 0.7216
step= 27999 ,train_loss= 1.13995 ,valid_loss= 1.48367 ,valid_acc= 0.659
step= 28099 ,train_loss= 0.532167 ,valid_loss= 1.11666 ,valid_acc= 0.7326
step= 28199 ,train_loss= 0.846276 ,valid_loss= 0.942803 ,valid_acc= 0.7514
step= 28299 ,train_loss= 0.599921 ,valid_loss= 0.830306 ,valid_acc= 0.7544
step= 28399 ,train_loss= 0.57344 ,valid_loss= 0.862557 ,valid_acc= 0.7636
step= 28499 ,train_loss= 0.850175 ,valid_loss= 0.809355 ,valid_acc= 0.7798
step= 28599 ,train_loss= 0.521074 ,valid_loss= 0.715635 ,valid_acc= 0.7626
step= 28699 ,train_loss= 0.864485 ,valid_loss= 1.36967 ,valid_acc= 0.7872
step= 28799 ,train_loss= 0.46411 ,valid_loss= 0.995481 ,valid_acc= 0.8068
step= 28899 ,train_loss= 0.589566 ,valid_loss= 0.896304 ,valid_acc= 0.8112
step= 28999 ,train_loss= 0.794279 ,valid_loss= 0.853235 ,valid_acc= 0.8064
step= 29099 ,train_loss= 0.539185 ,valid_loss= 1.07967 ,valid_acc= 0.7922
step= 29199 ,train_loss= 0.364329 ,valid_loss= 1.64473 ,valid_acc= 0.823
step= 29299 ,train_loss= 0.667117 ,valid_loss= 1.18414 ,valid_acc= 0.7928
step= 29399 ,train_loss= 0.62477 ,valid_loss= 0.776572 ,valid_acc= 0.778
step= 29499 ,train_loss= 0.402362 ,valid_loss= 0.990716 ,valid_acc= 0.7696
step= 29599 ,train_loss= 0.660165 ,valid_loss= 0.981529 ,valid_acc= 0.804
step= 29699 ,train_loss= 0.867033 ,valid_loss= 0.894529 ,valid_acc= 0.8048
step= 29799 ,train_loss= 0.792181 ,valid_loss= 0.959722 ,valid_acc= 0.8198
step= 29899 ,train_loss= 0.588089 ,valid_loss= 1.28739 ,valid_acc= 0.7766
step= 29999 ,train_loss= 0.672515 ,valid_loss= 1.32856 ,valid_acc= 0.7932
Test: accuracy is 0.777300
```

图 1 训练过程

（2）学习率为 0.0001 时的参数矩阵 w1,w2,w3,w4 分别如图 2 到图 5 所示。

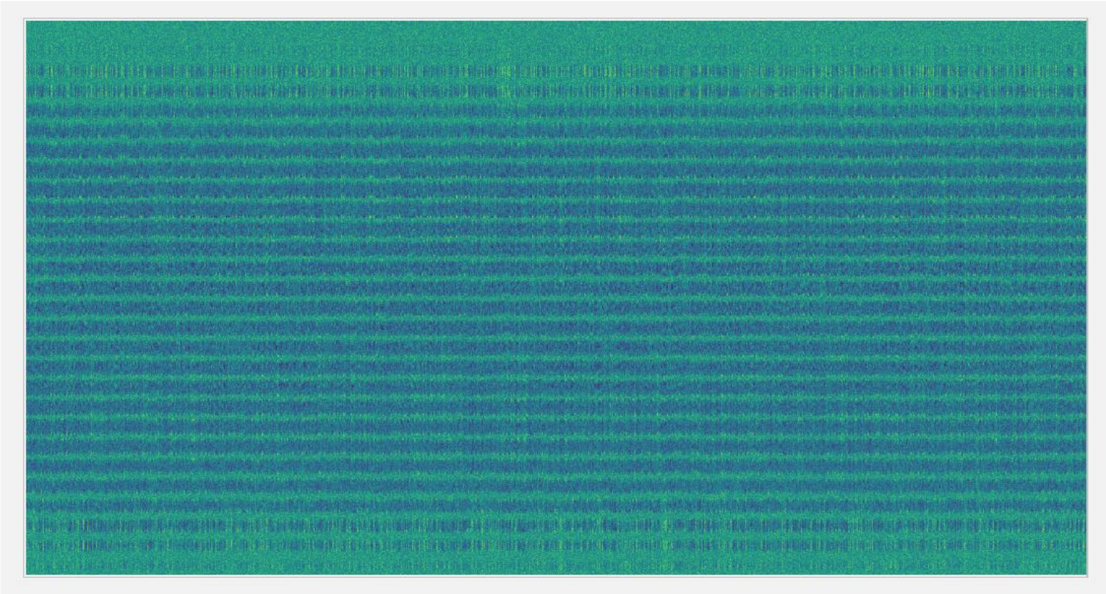


图 2 w1 的参数矩阵可视化图

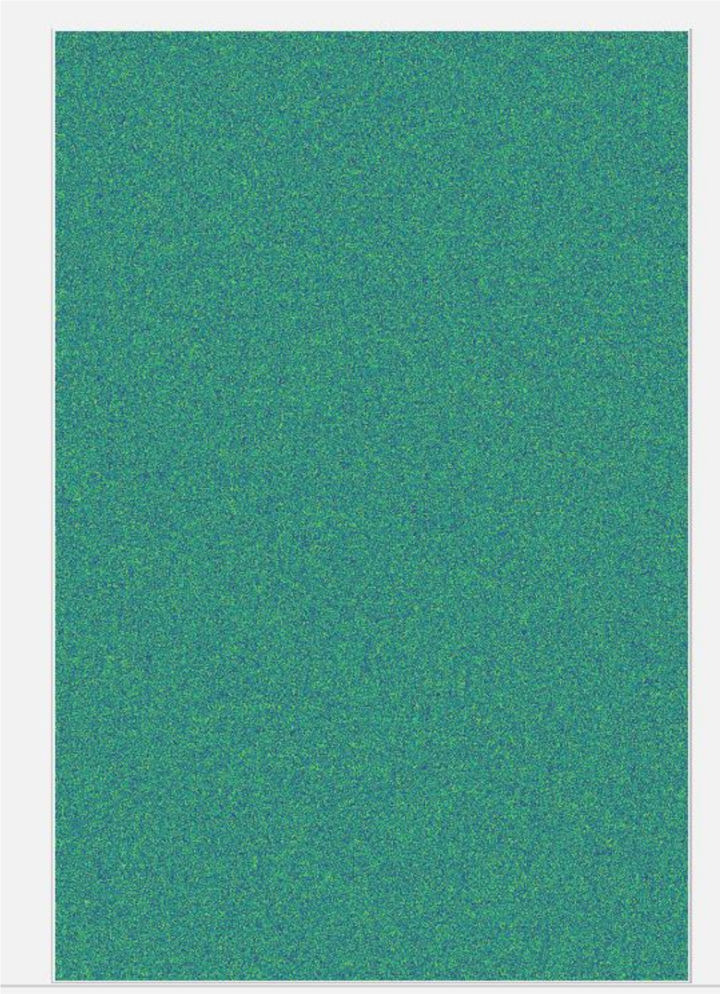


图 3 w2 的参数矩阵可视化图

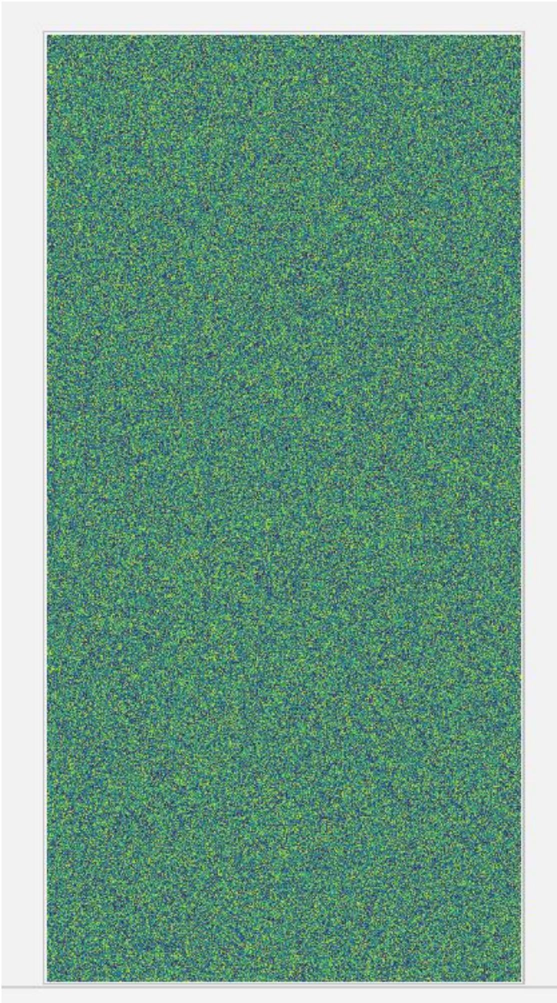


图 4 w3 的参数矩阵可视化图

图 5 w4 的参数矩阵可视化图

（2）学习率为 0.0001 时的参数矩阵 w1,w2,w3,w4 分别如图 6 到图 9 所示。

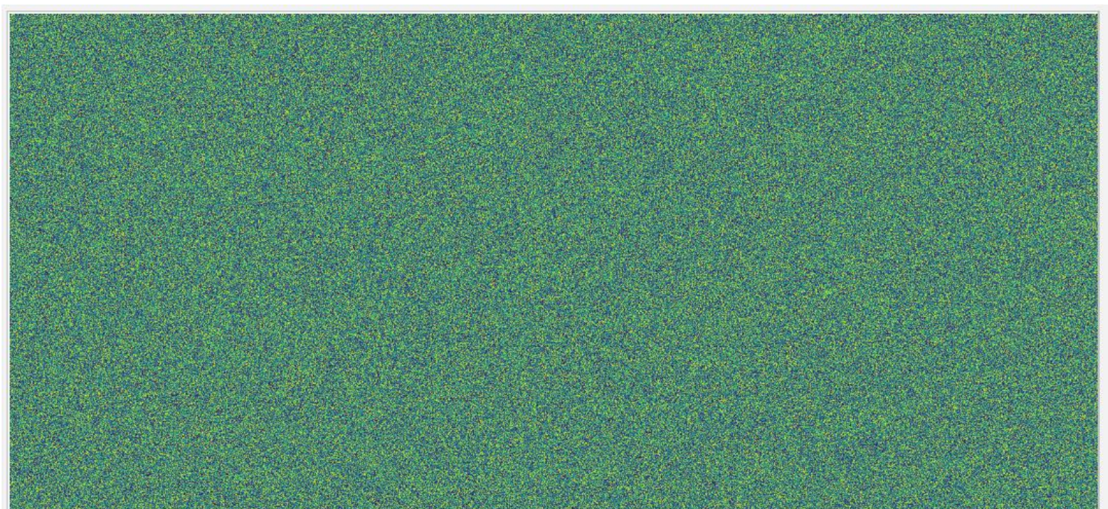


图 6 w1 的参数矩阵可视化图

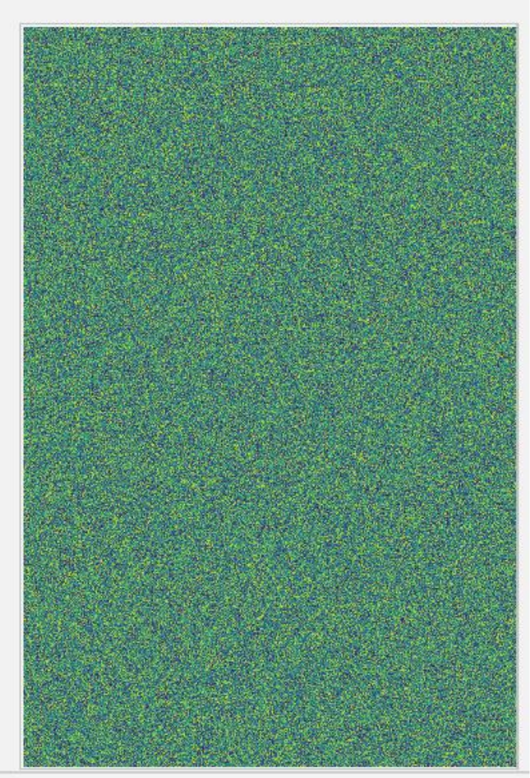


图 7 w2 的参数矩阵可视化图

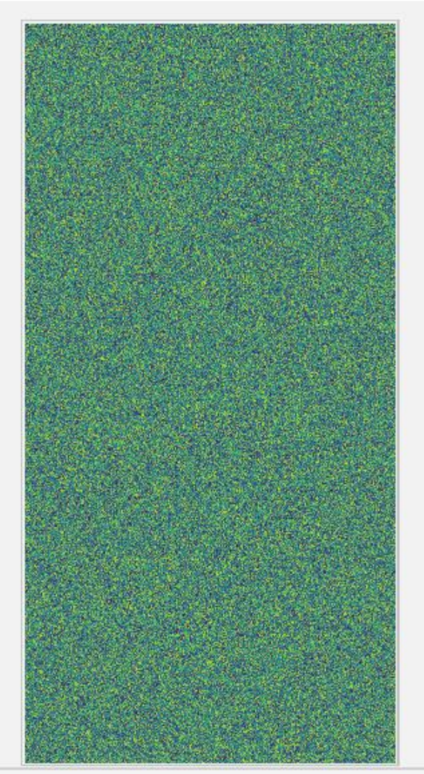


图 8 w3 的参数矩阵可视化图

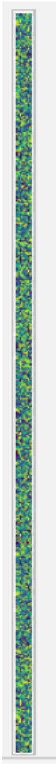
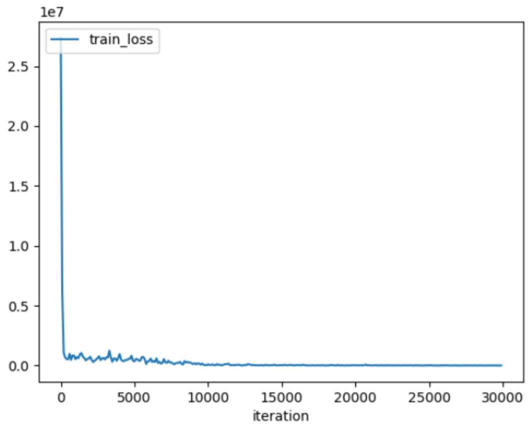
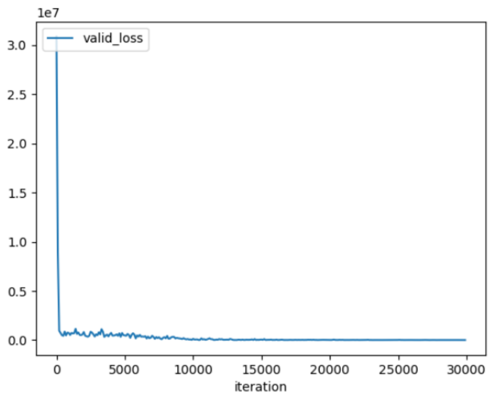


图 9 w4 的参数矩阵可视化图

（3）学习率为 0.0001 时的 training loss，validation loss，validation acc 曲线图如图 10 所示。





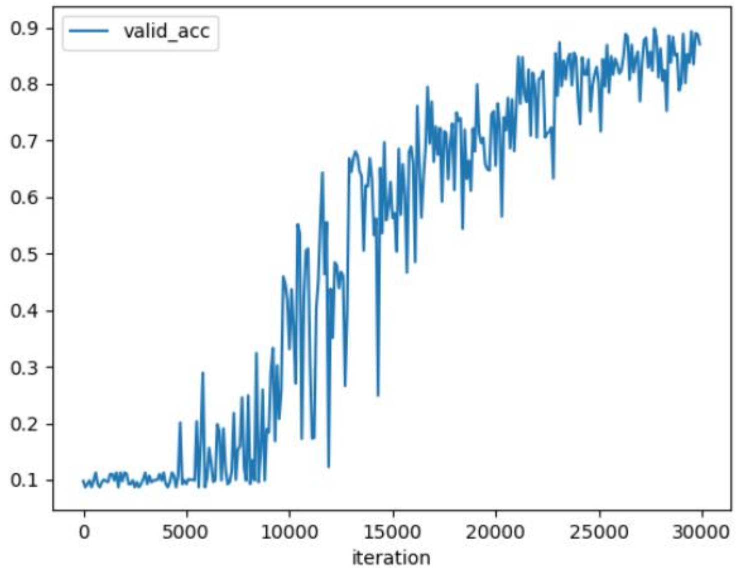
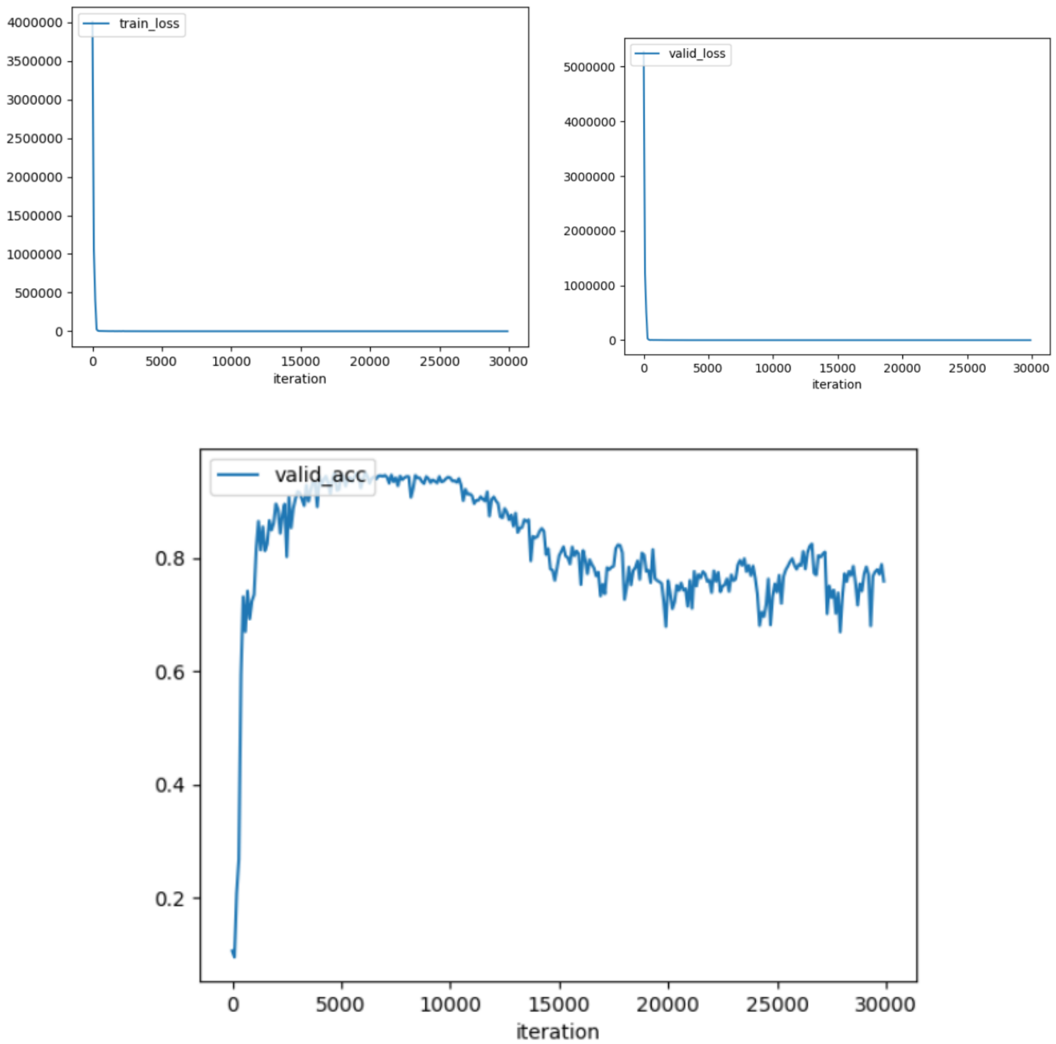


图 10 学习率为 0.0001 时的曲线图

（4）学习率为 0.005 时的 training loss, validation loss, validation acc 曲线图如图 11 所示。在 0.005 时，有 validation acc 先升高后降低的情况。



（5）学习率为 0.005 时，不使用正则化和使用正则化的 validation accuracy 曲线如图 12 和 13 所示。从图中可以看出，使用正则化的方法效果很好。

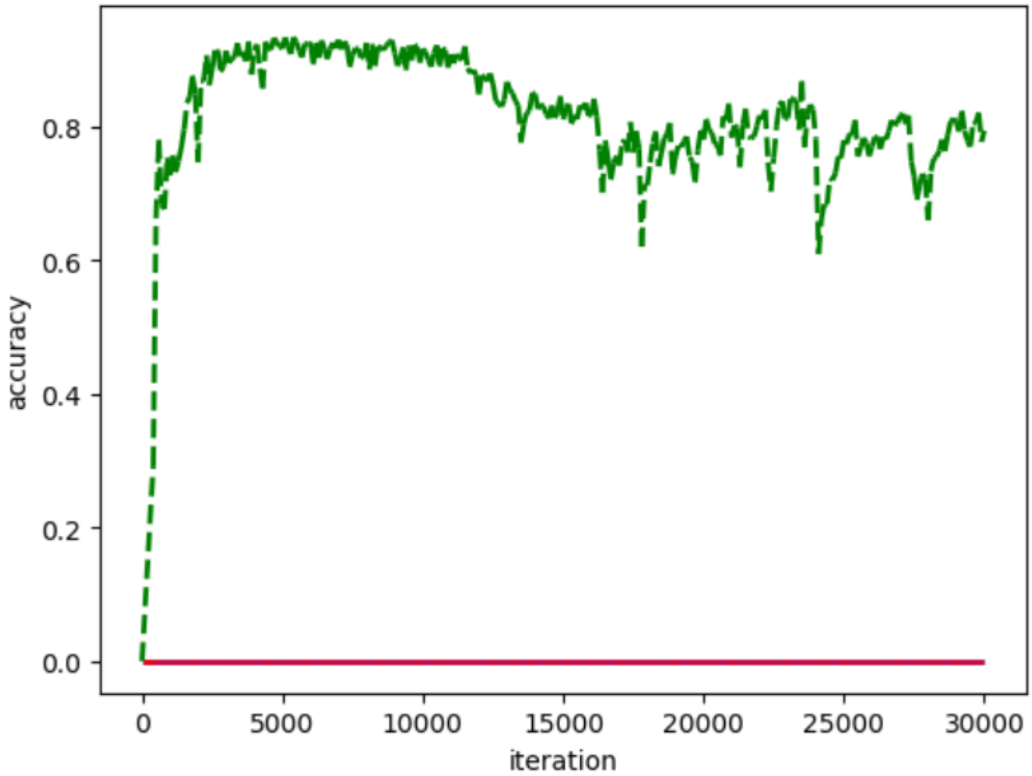


图 12 学习率为 0.005 时使用正则化的 validation accuracy 曲线图

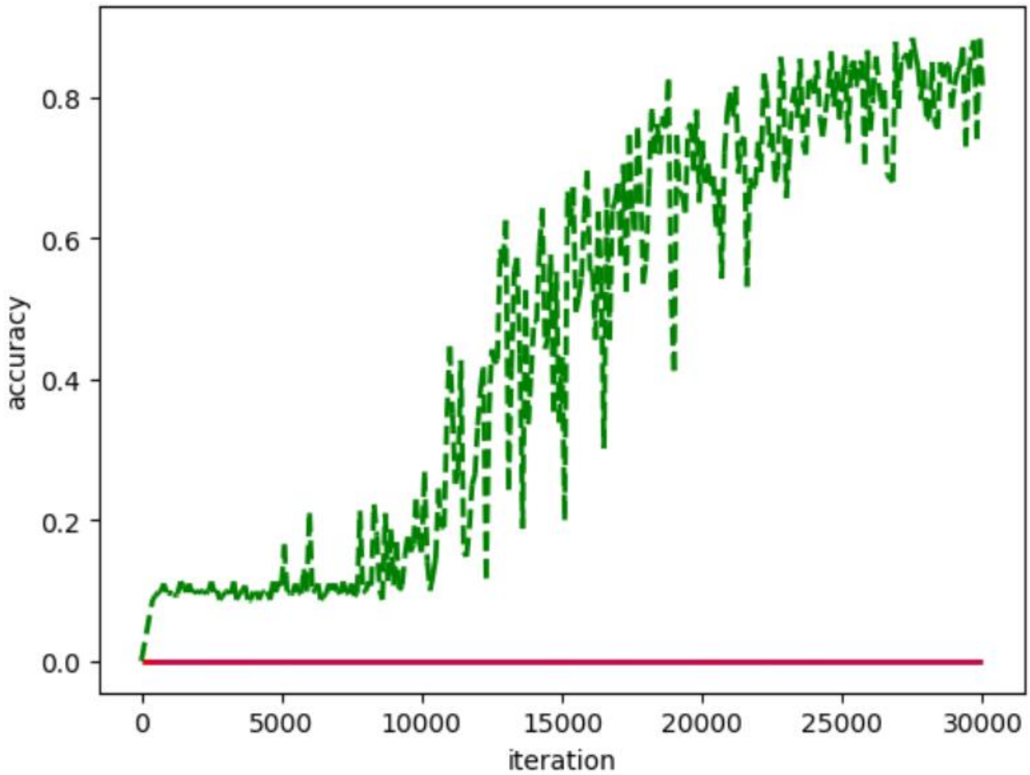


图 13 学习率为 0.0001 时不使用正则化的 validation accuracy 曲线图