人工智能EXP2实验报告

PB19071535徐昊天

1.传统机器学习

1.1 决策树

I.实验思路

①设计叶子节点类

代码如下：

```python
class Node:
    def __init__(self, leftchild, rightchild, divide, attribute, type, result):
        self.leftchild = leftchild
        self.rightchild = rightchild
        self.divide = divide
        self.attribute = attribute
        self.type = type
        self.result = result
```

其中包含六个参数，含义分别如下：

leftchild：左孩子结点

rightchild：右孩子结点

divide：用于决策的参数的分割值

attribute：用于决策的参数

type：结点类型，若为True，则为叶子节点；否则为中间节点

result：存储的结果（若为中间节点则为空，否则为0或1）

②设计信息增益函数

代码如下：

```python
def get_H(result):
    H = 0
    un_result = np.unique(result)
    for i in un_result:
        p = 0
        for j in result:
            if j == i:
                p += 1
        proportion = p / len(result)
        H -= proportion * math.log(proportion, 2)
    return H

```

参考人工智能教材决策树一章中的熵的概念，设计熵函数用于选择测试属性即其对应的分割值。以上`get_H`函数相对于课本中的H函数，用于计算熵。

③设计决策树生成算法

代码如下：

```python
def treegenerate(train_features, train_labels):
    train_labels = np.expand_dims(train_labels, axis=1)
    labels_features = np.concatenate((train_labels, train_features), axis=1)
    features_dim = np.shape(train_features)
    max_H = 0
    for feature in range(features_dim[1]):
        un_divide = np.unique(np.expand_dims(train_features[:, feature],
axis=1))
        for divide in un_divide:
            left = []
            right = []
            list = labels_features.tolist()
            for element in list:
                if element[feature + 1] > divide:
                    right.append(element)
                else:
                    left.append(element)
            leftgoal = np.array(left)
            rightgoal = np.array(right)
            if len(leftgoal) != 0 and len(rightgoal) != 0:
                pro1 = len(leftgoal[:, 0]) / len(train_labels)
                pro2 = len(rightgoal[:, 0]) / len(train_labels)
                if max_H < get_H(train_labels) - pro1 * get_H(leftgoal[:,
0]) - pro2 * get_H(rightgoal[:, 0]):
                    max_H = get_H(train_labels) - pro1 * get_H(leftgoal[:,
0]) - pro2 * get_H(rightgoal[:, 0])
                    max_feature = feature
                    max_divide = divide
                    max_left = leftgoal
                    max_right = rightgoal
      if max_H == 0:
          max_num = 0
          un_labels = np.unique(train_labels)
          for i in un_labels:
              for j in train_labels:
                  num = 0
                  if j[0] == i:
                      num += 1
              if max_num < num:
                  max_num = num
                  max_result = i
          return Node(None, None, None, None, True, max_result)
      else:
          leftchild = treegenerate(max_left[:, 1:], max_left[:, 0])
          rightchild = treegenerate(max_right[:, 1:], max_right[:, 0])
          return Node(leftchild, rightchild, max_divide, max_feature, False, None)
```

参考实验文档提供的treegenerate函数，首先遍历所有属性，然后对于每种属性遍历该属性的每一种可能的取值并将其作为分割点，利用信息增益选择两层循环得到的最大信息增益情况，并将其作为生成分支的依据。并考虑最大信息增益：若为0，则得到叶节点；若不为0，则得到中间节点。

④设计预测函数

代码如下：

```python
        predict_result = []
        for feature in test_features:
            node = self.root
            while node.type != True:
                if feature[node.attribute] > node.divide:
                    node = node.rightchild
                else:
                    node = node.leftchild
            predict_result.append(node.result)
        print(predict_result)
        return np.array(predict_result)
```

对于每一项测试数据，分别从根结点开始向下遍历直至叶节点，即可得到预测结果。

II.实验结果

```
[1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0,
 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0]
DecisionTree acc: 58.93%
```

上面的列表是预测结果，下面为输出的预测准确率，为58.93%。

1.2 支持向量机

I.实验思路

①设计求解凸优化问题的函数

代码如下：

```python
        def get_value(dim):
            m1 = np.outer(train_label, train_label) * kernel
            m2 = (-1) * np.ones(dim).T
            m3 = np.vstack((np.diag(np.ones(dim) * -1), np.diag(np.ones(dim))))
            if self.C is not None:
                m4 = np.hstack((np.zeros(dim), self.C * np.ones(dim)))
            else:
                m4 = np.zeros(dim)
            m5 = train_label.reshape(1, dim)
            vari = Variable(dim)
            obj = Minimize(quad_form(vari, m1) / 2 + m2 @ vari)
            prob = Problem(obj, [m5 @ vari == np.array([0]), m3 @ vari <= m4])
            prob.solve(solver=cvxpy.ECOS)
            value = np.ravel(vari.value)
            return value
```

根据实验文档提供的示例，按照对应步骤创建变量，建立约束方程，建立目标函数，构造问题，求解问题，返回求解结果。

②计算SV

代码如下：

```
value = get_value(data_dim[0])
list_SV = []
list_alpha = []
list_label = []
for i in range(len(value)):
    if value[i] > self.epsilon:
        list_SV.append(train_data[i])
        list_alpha.append(value[i])
        list_label.append(train_label[i])
        self.b = list_label[0]
        for i in range(len(list_alpha)):
            self.b -= self.KERNEL(list_SV[i], list_SV[0]) * list_alpha[i] * list_label[i]
            self.SV = np.array(list_SV)
            self.SV_alpha = np.array(list_alpha)
            self.SV_label = np.array(list_label)
```

根据求解凸优化函数得到的结果，结合SVM计算公式计算出对应SV的参数。

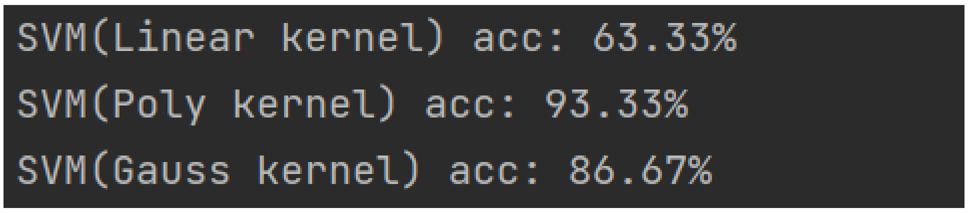
③设计预测函数

代码如下：

```
result = []
for data in test_data:
    num = self.b
    for i in range(len(self.SV_alpha)):
        num += self.KERNEL(self.SV[i], data) * self.SV_alpha[i] * self.SV_label[i]
        result.append(np.sign(num))
        return np.array(result)
```

对于每一项测试数据，分别根据SVM计算公式，即可得到预测结果。

II.实验结果



```
SVM(Linear kernel) acc: 63.33%
SVM(Poly kernel) acc: 93.33%
SVM(Gauss kernel) acc: 86.67%
```

根据上图，线性核准确率为63.33%，多项式核准确率为93.33%，高斯核准确率为86.67%。仅从此次结果分析，多项式核分类效果最佳，线性核分类效果最差。

2.深度学习

2.1 手写感知机模型并进行反向传播

I.实验思路

①定义层的类

代码如下：

```python
class Layer():
    def __init__(self, type, size_1 = 0, size_2 = 0):
        # 若为True，则存储每层之间执行激活函数之前的数据；若为False，则存储每层之间执行激活函数之后的数据
        self.type = type
        # 分别表示相邻层的size
        self.size_1 = size_1
        self.size_2 = size_2
        # 储存上一层的前向传播的值
        self.back_value = None
        # 定义w、b
        if type == True:
            scale = np.sqrt(size_1 / 2)
            self.w = np.random.randn(size_1, size_2) / scale
            self.b = np.random.randn(1, size_2) / scale
```

为方便应用每一层经过激活函数前后的值，将层分为两类，若参数type为True，则表示为待执行前向传播计算的层，否则为待执行激活函数的层。

②设计前向传播

代码如下：

```python
def forward(self, x):
    # 前向传播
    for i in self.all_layer:
        if i.type == True:
            i.back_value = x
            x = np.add(np.dot(x, i.w), i.b)
        else:
            i.back_value = x
            x = np.tanh(x)
            return x
```

若type为True，则执行以下前向传播计算，否则执行激活函数。

前向传播

$$\begin{aligned}
\mathbf{h}_1 &= s_1(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) \\
\mathbf{h}_2 &= s_2(\mathbf{W}_2\mathbf{h}_1 + \mathbf{b}_2) \\
\mathbf{h}_3 &= s_3(\mathbf{W}_3\mathbf{h}_2 + \mathbf{b}_3) \\
\hat{\mathbf{y}} &= s_4(\mathbf{W}_4\mathbf{h}_3 + \mathbf{b}_4) \\
l(\hat{\mathbf{y}}, \mathbf{y}) &= CrossEntropyLoss(\hat{\mathbf{y}}, \mathbf{y}) = -\log(\hat{y}_t)
\end{aligned}$$

③设计反向传播

代码如下：

```python
    def backward(self, lr, inputs, labels):  # 自行确定参数表
        # 反向传播
        def get_loss(labels, forward_result):
            subtract_result = np.subtract(forward_result, 
np.max(forward_result, axis=1, keepdims=True))
            exp_result = np.exp(subtract_result)
            divide_result = np.divide(exp_result, np.sum(exp_result, axis=1, 
keepdims=True))
            list = []
            for i in range(train_batch_size):
                list.append(i)
            positive_pre = divide_result[np.array(list), labels]
            loss = -np.log(positive_pre)
            return np.mean(loss), divide_result

        def update_wb(list_1, list_2, list_3, list_4):
            for i in range(len(list_1)):
                list_1[i] -= lr * list_3[i]
                list_2[i] -= lr * list_4[i]

        forward_result = self.forward(inputs)
        loss, new_result = get_loss(labels, forward_result)
        forgrad = np.zeros_like(new_result)
        list = []
        for i in range(train_batch_size):
            list.append(i)
        forgrad[np.array(list), labels] = 1.0
        list_1 = []
        list_2 = []
        list_3 = []
        list_4 = []
        grad = (new_result - forgrad) / train_batch_size
        for i in reversed(self.all_layer):
            if i.type == True:
                list_1.append(i.w)
                list_2.append(i.b)
                list_3.append(np.matmul(i.back_value.T, grad))
                list_4.append(np.sum(grad, axis=0, keepdims=True))
                grad = np.matmul(grad, i.w.T)
            else:
                grad = (1 - np.power(np.tanh(i.back_value), 2)) * grad
        update_wb(list_1, list_2, list_3, list_4)
        return loss
```

反向传播

链式法则的展开形式：

$$
\begin{aligned}
\frac{\partial L}{\partial \mathbf{W_4}} &= (l' s_4')\mathbf{h}_3^{\mathbf{T}}, \frac{\partial L}{\partial \mathbf{b_4}} = l' s_4' \\
\frac{\partial L}{\partial \mathbf{W_3}} &= (\mathbf{W}_4^{\mathbf{T}}(l' s_4') \odot s_3')\mathbf{h}_2^{\mathbf{T}}, \frac{\partial L}{\partial \mathbf{b_3}} = \mathbf{W}_4^{\mathbf{T}}(l' s_4') \odot s_3' \\
\frac{\partial L}{\partial \mathbf{W_2}} &= (\mathbf{W}_3^{\mathbf{T}}(\mathbf{W}_4^{\mathbf{T}}(l' s_4') \odot s_3') \odot s_2')\mathbf{h}_1^{\mathbf{T}}, \frac{\partial L}{\partial \mathbf{b_2}} = \mathbf{W}_3^{\mathbf{T}}(\mathbf{W}_4^{\mathbf{T}}(l' s_4') \odot s_3') \odot s_2' \\
\frac{\partial L}{\partial \mathbf{W_1}} &= (\mathbf{W}_2^{\mathbf{T}}(\mathbf{W}_3^{\mathbf{T}}(\mathbf{W}_4^{\mathbf{T}}(l' s_4') \odot s_3') \odot s_2') \odot s_1')\mathbf{x}^{\mathbf{T}}, \frac{\partial L}{\partial \mathbf{b_1}} = \mathbf{W}_2^{\mathbf{T}}(\mathbf{W}_3^{\mathbf{T}}(\mathbf{W}_4^{\mathbf{T}}(l' s_4') \odot s_3') \odot s_2') \odot s_1'
\end{aligned}
$$

其中⊙表示按位乘，并且：

$$
\begin{aligned}
s_4(x_1, x_2, x_3, x_4) = Softmax(x_1, x_2, x_3, x_4) &= \frac{1}{e^{x_1} + e^{x_2} + e^{x_3} + e^{x_4}}(e^{x_1}, e^{x_2}, e^{x_3}, e^{x_4}), \\
s_1 = s_2 = s_3 &= \tanh(\cdot) \\
s_1' = s_2' = s_3' &= 1 - \tanh^2 \\
(l' s_4')_i &= \begin{cases} \hat{y}_i - 1 & i = t \\ \hat{y}_i & i \neq t \end{cases}
\end{aligned}
$$

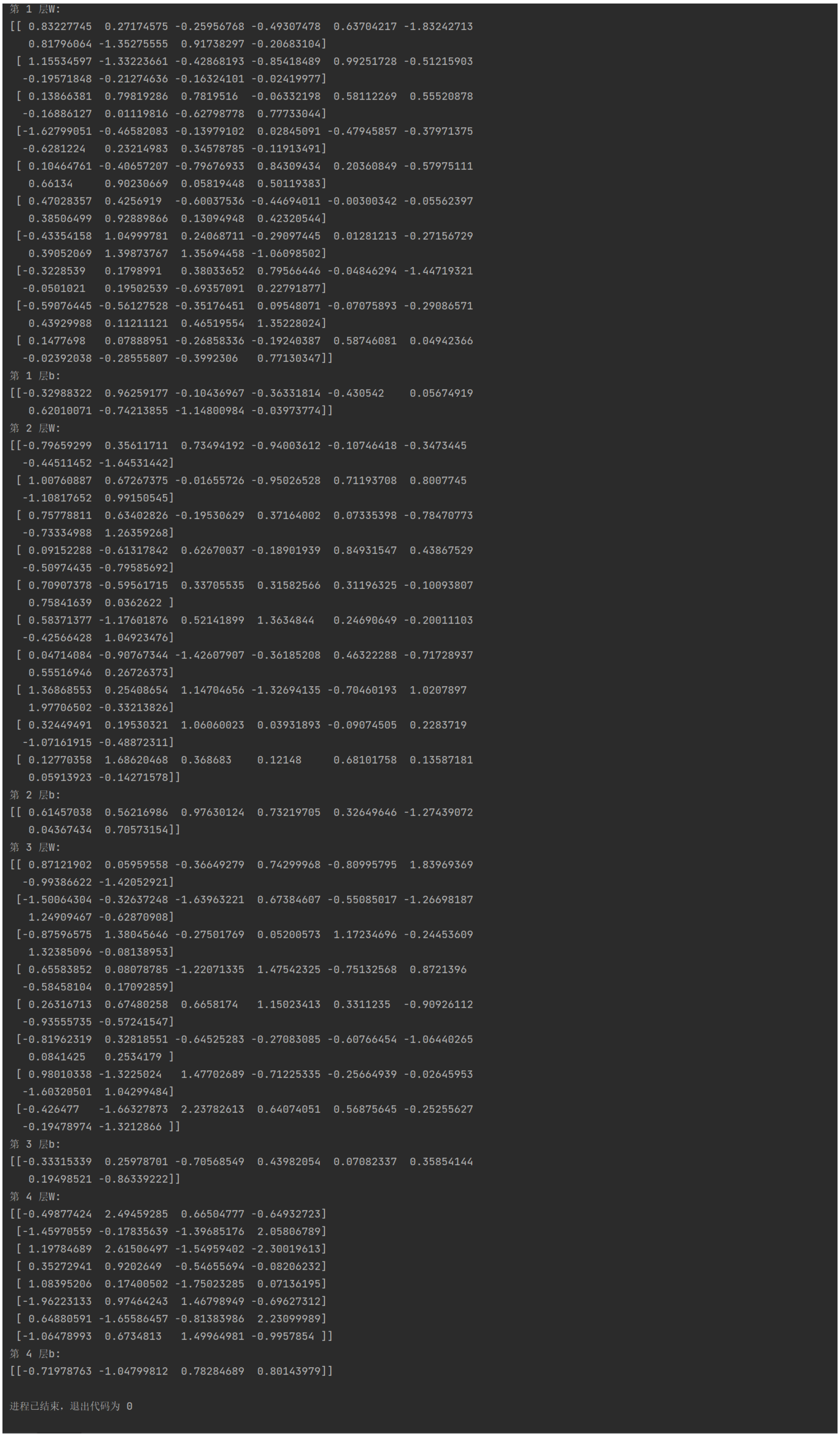
梯度下降：

$$
\begin{aligned}
\mathbf{W}_i &= \mathbf{W}_i - \eta \frac{\partial L}{\partial \mathbf{W}_i} \\
\mathbf{b}_i &= \mathbf{b}_i - \eta \frac{\partial L}{\partial \mathbf{b}_i}
\end{aligned}
$$

根据以上实验文档中提供的反向传播和梯度下降公式，对loss和梯度进行计算。

II.实验结果

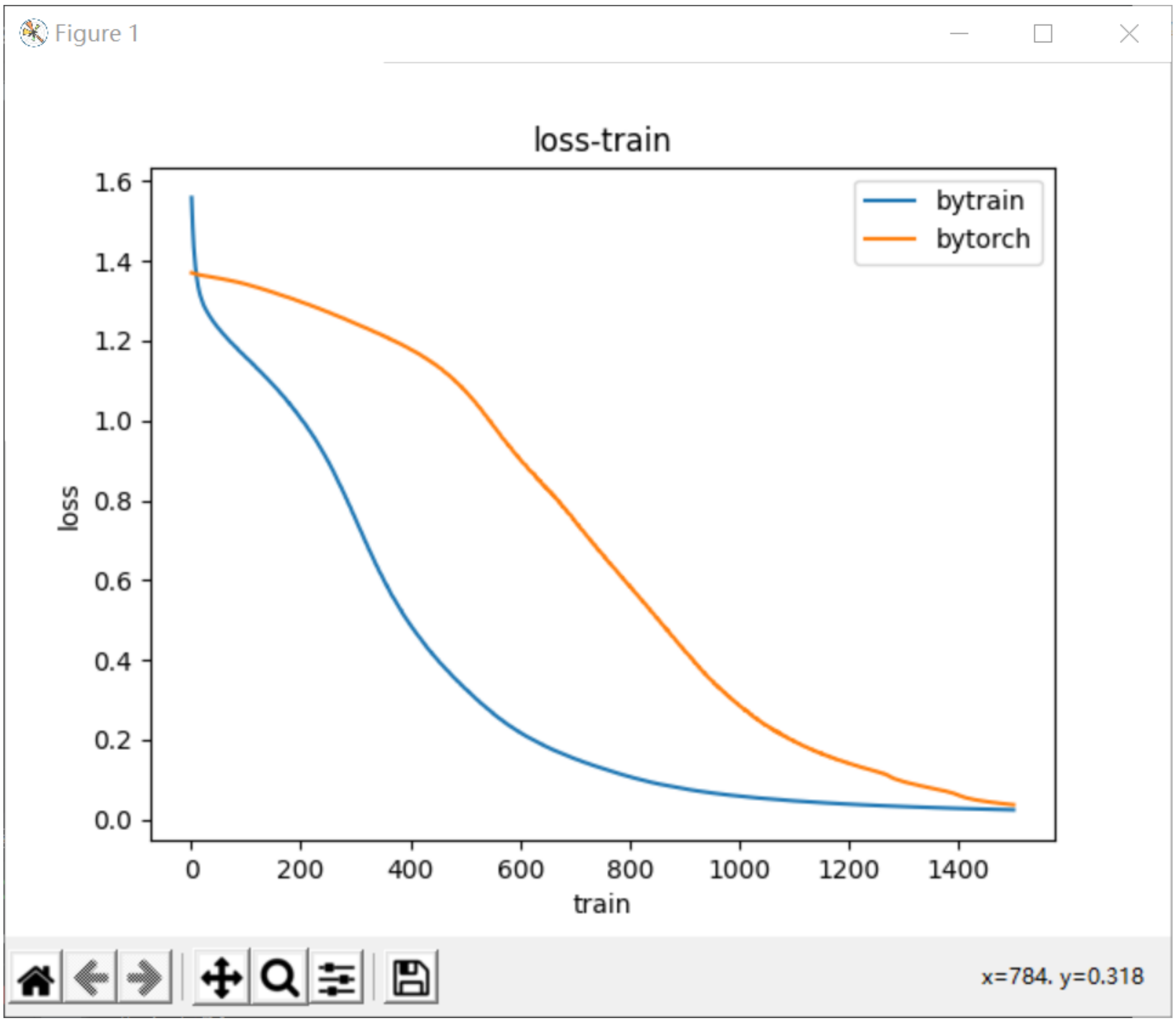
W，b的最终结果如下图：



```
第 1 层W:
[[ 0.83227745  0.27174575 -0.25956768 -0.49307478  0.63704217 -1.83242713
   0.81796064 -1.35275555  0.91738297 -0.20683104]
 [ 1.15534597 -1.33223661 -0.42868193 -0.85418489  0.99251728 -0.51215903
  -0.19571848 -0.21274636 -0.16324101 -0.02419977]
 [ 0.13866381  0.79819286  0.7819516  -0.06332198  0.58112269  0.55520878
  -0.16886127  0.01119816 -0.62798778  0.77733044]
 [-1.62799051 -0.46582083 -0.13979102  0.02845091 -0.47945857 -0.37971375
  -0.6281224   0.23214983  0.34578785 -0.11913491]
 [ 0.10464761 -0.40657207 -0.79676933  0.84309434  0.20360849 -0.57975111
   0.66134     0.90230669  0.05819448  0.50119383]
 [ 0.47028357  0.4256919  -0.60037536 -0.44694011 -0.00300342 -0.05562397
   0.38506499  0.92889866  0.13094948  0.42320544]
 [-0.43354158  1.04999781  0.24068711 -0.29097445  0.01281213 -0.27156729
   0.39052069  1.39873767  1.35694458 -1.06098502]
 [-0.3228539   0.1798991   0.38033652  0.79566446 -0.04846294 -1.44719321
  -0.0501021   0.19502539 -0.69357091  0.22791877]
 [-0.59076445 -0.56127528 -0.35176451  0.09548071 -0.07075893 -0.29086571
   0.43929988  0.11211121  0.46519554  1.35228024]
 [ 0.1477698   0.07888951 -0.26858336 -0.19240387  0.58746081  0.04942366
  -0.02392038 -0.28555807 -0.3992306   0.77130347]]
第 1 层b:
[[-0.32988322  0.96259177 -0.10436967 -0.36331814 -0.430542    0.05674919
   0.62010071 -0.74213855 -1.14800984 -0.03973774]]
第 2 层W:
[[-0.79659299  0.35611711  0.73494192 -0.94003612 -0.10746418 -0.3473445
  -0.44511452 -1.64531442]
 [ 1.00760887  0.67267375 -0.01655726 -0.95026528  0.71193708  0.8007745
  -1.10817652  0.99150545]
 [ 0.75778811  0.63402826 -0.19530629  0.37164002  0.07335398 -0.78470773
  -0.73334988  1.26359268]
 [ 0.09152288 -0.61317842  0.62670037 -0.18901939  0.84931547  0.43867529
  -0.50974435 -0.79585692]
 [ 0.70907378 -0.59561715  0.33705535  0.31582566  0.31196325 -0.10093807
   0.75841639  0.0362622 ]
 [ 0.58371377 -1.17601876  0.52141899  1.3634844   0.24690649 -0.20011103
  -0.42566428  1.04923476]
 [ 0.04714084 -0.90767344 -1.42607907 -0.36185208  0.46322288 -0.71728937
   0.55516946  0.26726373]
 [ 1.36868553  0.25408654  1.14704656 -1.32694135 -0.70460193  1.0207897
   1.97706502 -0.33213826]
 [ 0.32449491  0.19530321  1.06060023  0.03931893 -0.09074505  0.2283719
  -1.07161915 -0.48872311]
 [ 0.12770358  1.68620468  0.368683    0.12148     0.68101758  0.13587181
   0.05913923 -0.14271578]]
第 2 层b:
[[ 0.61457038  0.56216986  0.97630124  0.73219705  0.32649646 -1.27439072
   0.04367434  0.70573154]]
第 3 层W:
[[ 0.87121902  0.05959558 -0.36649279  0.74299968 -0.80995795  1.83969369
  -0.99386622 -1.42052921]
 [-1.50064304 -0.32637248 -1.63963221  0.67384607 -0.55085017 -1.26698187
   1.24909467 -0.62870908]
 [-0.87596575  1.38045646 -0.27501769  0.05200573  1.17234696 -0.24453609
   1.32385096 -0.08138953]
 [ 0.65583852  0.08078785 -1.22071335  1.47542325 -0.75132568  0.8721396
  -0.58458104  0.17092859]
 [ 0.26316713  0.67480258  0.6658174   1.15023413  0.3311235  -0.90926112
  -0.93555735 -0.57241547]
 [-0.81962319  0.32818551 -0.64525283 -0.27083085 -0.60766454 -1.06440265
   0.0841425   0.2534179 ]
 [ 0.98010338 -1.3225024   1.47702689 -0.71225335 -0.25664939 -0.02645953
  -1.60320501  1.04299484]
 [-0.426477   -1.66327873  2.23782613  0.64074051  0.56875645 -0.25255627
  -0.19478974 -1.3212866 ]]
第 3 层b:
[[-0.33315339  0.25978701 -0.70568549  0.43982054  0.07082337  0.35854144
   0.19498521 -0.86339222]]
第 4 层W:
[[-0.49877424  2.49459285  0.66504777 -0.64932723]
 [-1.45970559 -0.17835639 -1.39685176  2.05806789]
 [ 1.19784689  2.61506497 -1.54959402 -2.30019613]
 [ 0.35272941  0.9202649  -0.54655694 -0.08206232]
 [ 1.08395206  0.17400502 -1.75023285  0.07136195]
 [-1.96223133  0.97464243  1.46798949 -0.69627312]
 [ 0.64880591 -1.65586457 -0.81383986  2.23099989]
 [-1.06478993  0.6734813   1.49964981 -0.9957854 ]]
第 4 层b:
[[-0.71978763 -1.04799812  0.78284689  0.80143979]]

进程已结束，退出代码为 0
```

loss训练曲线和与pytorch自动求导的对比如下图：



2.2 实现一个卷积神经网络

I.实验思路

①设计MyNet的卷积层、池化层和全连接层

代码如下：

```
self.conv_1 = nn.Conv2d(3, 16, (5, 5))
self.conv_2 = nn.Conv2d(16, 32, (5, 5))
self.relu = nn.ReLU()
self.avg_pool = nn.AvgPool2d(2)
self.linear_1 = nn.Linear(800, 120)
self.linear_2 = nn.Linear(120, 84)
self.linear_3 = nn.Linear(84, 10)
```

我的学号为PB19071535，故(3+5)%6+1=3，应当选择列表中第三个模型。

根据第三组提供的参数可推知input channel = 32*5*5 = 800

故定义以上参数

②设计MyNet的前向传播

代码如下：

```
x = self.conv_1(x)
```

```python
x = self.relu(x)
x = self.avg_pool(x)
x = self.conv_2(x)
x = self.relu(x)
x = self.avg_pool(x)
x = x.view(x.size(0), -1)
x = self.linear_1(x)
x = self.relu(x)
x = self.linear_2(x)
x = self.relu(x)
x = self.linear_3(x)
x = self.relu(x)
return x
```

根据卷积层、池化层和全连接层的信息进行前向传播。

③计算loss并进行反向传播

代码如下：

```python
optimizer.zero_grad()

loss = loss_function(net(inputs), labels)
loss.backward()

optimizer.step()
```

④计算测试集的loss和accuracy

代码如下：

```python
test_loss = loss_function(net(inputs), labels)
predict = net(inputs).max(1)[1]
accuracy = torch.eq(predict, labels).float().mean()
```

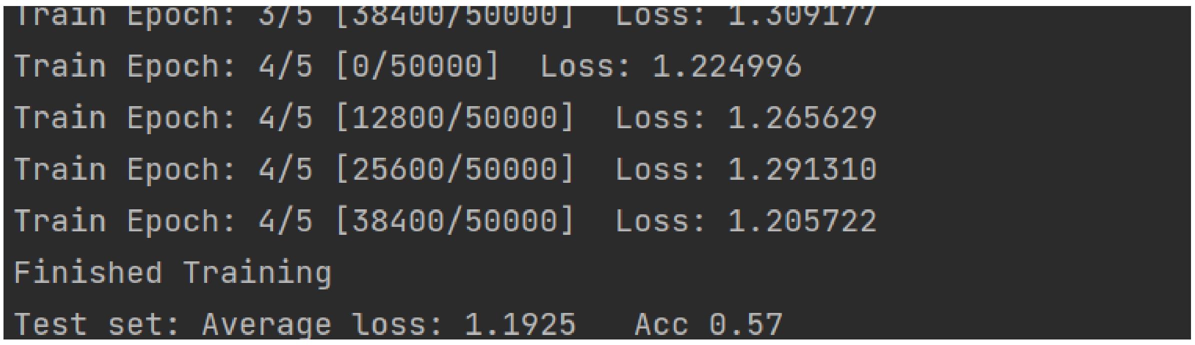
⑤设定优化器和损失函数

代码如下：

```python
optimizer = torch.optim.Adam(net.parameters(), lr=learning_rate)
loss_function = torch.nn.CrossEntropyLoss()
```

设定优化器为Adam类，损失函数为交叉熵损失。

II.实验结果



```
Train Epoch: 3/5 [38400/50000]  Loss: 1.309177
Train Epoch: 4/5 [0/50000]  Loss: 1.224996
Train Epoch: 4/5 [12800/50000]  Loss: 1.265629
Train Epoch: 4/5 [25600/50000]  Loss: 1.291310
Train Epoch: 4/5 [38400/50000]  Loss: 1.205722
Finished Training
Test set: Average loss: 1.1925  Acc 0.57
```

3.实验收获与感想

- 首次使用python语言完成较大量的代码任务，对于python语言的使用有了更加深入的了解。
- 首次实现经典机器学习及深度学习算法，对于课程中学习的理论知识有了更加深刻的理解。
- 首次调用numpy、cvxpy、torch等python库，对于其中许多函数接口的使用技巧有了深刻的认识。