# 实验报告

## 决策树

### 准确率

准确率如下

```
(py36) ➜  src1 git:(main) ✗ python main.py
DecisionTree acc: 66.07%
```

## 选择最优划分属性

选择最优属性通过信息增益计算最大值完成。

在函数 `optimalAttr` 中选择按照属性划分之后能够得到最大信息增益的属性作为最优属性。该次划分的数据集是上一次按照某属性的某个值划分之后得到的数据集。

```
def optimalAttr(self, train_features, train_labels, attr_set) -> Tuple[int, bool]:
    """按照信息增益进行划分

    Args:
        train_features : 上一次按照某个属性的某个值划分之后得到的数据集
        train_labels (_type_): 上一次按照某个属性的某个值划分之后得到的数据标签
        attr_set (_type_): 仍未划分的属性

    Returns:
        Tuple[int, bool]: 第一个返回值为进一步划分的最优属性，第二个返回值表示是否需要进一步划分，
    """
```

首先进行一些初始化工作，计算在没有划分前的熵 `origin_I`

```
    max_IG = 0  # 最大信息增益
    opt_attr = -1   # 最优属性，初始化为-1,如果在attr_set划分之后信息增益都为负数，即没有信息增益
    label_count_dict = self.countLabel(train_labels)
    p_count = label_count_dict[1]
    n_count = label_count_dict[0]
    origin_I = DecisionTree.I(p_count, n_count) # I(p/(p+n), n/(p+n))划分前的熵
    train_set_count = len(train_labels)
```

然后对属性集中的属性进行遍历，找出其中信息增益最大的属性

- 针对某个具体的属性 `attr` ，根据其值 `value` 进行划分，得到“值-样本列表”( `value2features` )，“值-标签列表”( `value2labels` )字典
- 然后针对不同的值 `value` ，计算条件熵 `I_after_divide`
- 最后计算信息增益 `origin_I-I_after_divide` ，如果大于当前最大熵，则找到更优属性与更优信息增益

```
for attr in attr_set:
    value2features, value2labels = self.divideByAttr(train_features, train_labels, a
    I_after_divide = 0  # 划分之后的熵
    total_count = 0
    for value in value2features.keys():
        label_count_dict = self.countLabel(value2labels[value])
        p_count = label_count_dict[1]
        n_count = label_count_dict[0]
        total_count += p_count + n_count
        I_after_divide += (p_count+n_count)*1.0/train_set_count*DecisionTree.I(p_cou
    if total_count != train_set_count:
        fatal("something is miss when divide by attr")
    # print("attr {} calc IG={}".format(attr, origin_I-I_after_divide))
    if origin_I-I_after_divide > max_IG:
        opt_attr = attr
        max_IG = origin_I-I_after_divide
```

如果所有信息增益全部为负，则不做进一步划分；否则返回最优属性

```
if opt_attr == -1:  # 按照attr_set中属性进行划分后，信息增益均为负数，不做进一步划分
    # print('no info gain')
    return None, False
else:
    # print('choose {}'.format(opt_attr))
    return opt_attr, True
```

# 生成叶子节点

生成叶子节点有几种情况，分别为

1. 所有标签全部相同，标签设置为相同的标签

```
same, label = self.allSameLabel(train_labels)
if same:
    node.set_label(label)
    node.set_value(-1)
    return node
```

2. 如果已经对全部属性都进行了划分，或者说在剩下属性集上，所有值都相等，则生成叶子节点。标签均为当前最多出现的标签

```
max_count_train_label = self.maxCountLabel(train_labels)
if len(attr_set) == 0 or self.allSameValueInAttrSet(train_features, attr_set):
    node.set_label(max_count_train_label)
    node.set_value(-1)
    return node
```

3. 按照剩余属性进行划分，信息增益全部为负，则生成叶子节点，标签同样设置为出现最多的标签

```
opt_attr, info_gain_ok = self.optimalAttr(train_features, train_labels, attr_set)
if not info_gain_ok:
    node.set_label(max_count_train_label)
    node.set_value(-1)
    return node
```

4. 在生成树之前，首先要通过 `self.getFeatureValueDict` 统计某个属性下所有可能的值，记在 `self.attr2value_set` 字典中。如果在按照某个属性进行划分的时候，由于前面划分所限定的条件导致此次按照最优属性进行划分后，某个值没有出现在划分之后的数据集中，则此时为该值节点创造叶子节点，其中label标记为未划分前数据集的最多出现的标签。

```
for value in self.attr2value_set[opt_attr]:
    features = value2features[value]
    labels = value2labels[value]
    if len(labels) == 0:    # 某个值在该次划分中没有出现
        branch = DecisionTree.Node(node)
        branch.set_label(max_count_train_label)
        branch.set_value(value)
        node.add_child(branch)
```

## 生成中间节点

排除以上情况，递归生成中间节点。

对于最优属性 `opt_attr` ，对于其所有在数据集中出现的值，生成 `node` 节点的子节点 `child` ，并且将子节点的 `value` 域设置为值，表示按照 `value` 进行的划分，并且将 `child` 通过 `add_child` 加入 `node` 节点的孩子列表中。每个node节点同时还需要将 `attr` 域设置为最优属性 `opt_attr` ，表示子节点是依据 `attr` 域进行划分的，在预测时可用。

如果某个值下的数据集为空，则按照上述第4种情况生成叶节点，并加入 `node` 节点的孩子中。否则，在属性级中移除最优属性 `opt_attr` ，然后生成子节点（同时也是分支的根节点），然后将子节点加入 `node` 节点的孩子中。

```
    node.set_attr(opt_attr) # 表示子节点依据opt_attr进行划分
    value2features, value2labels = self.divideByAttr(train_features, train_labels, opt_a
    for value in self.attr2value_set[opt_attr]:
        features = value2features[value]
        labels = value2labels[value]
        if len(labels) == 0:    # 某个值在该次划分中没有出现
            branch = DecisionTree.Node(node)
            branch.set_label(max_count_train_label)
            branch.set_value(value)
            node.add_child(branch)
        else:
            new_attr_set = attr_set.copy()
            new_attr_set.remove(opt_attr)
            branch = self.treeGenerate(features, labels, new_attr_set, node)
            branch.set_value(value)
            node.add_child(branch)
    node.set_label(max_count_train_label)   # 如果测试集中有一些特征值不在训练集中出现，则按照最
    return node
```

# 支持向量机

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

Gauss多项式二阶核函数有相对较好的效果

在求解完成后选择支持向量时，由于精度问题，需要选择$\alpha > \varepsilon$的那些作为支持向量

即

```
self.SV_alpha = [alpha for alpha in alpha_vec.value if alpha > self.epsilon]
self.SV = [train_data[i] for i, alpha in enumerate(alpha_vec.value) if alpha > self.epsi
self.SV_label = [train_label[i] for i, alpha in enumerate(alpha_vec.value) if alpha > se
```

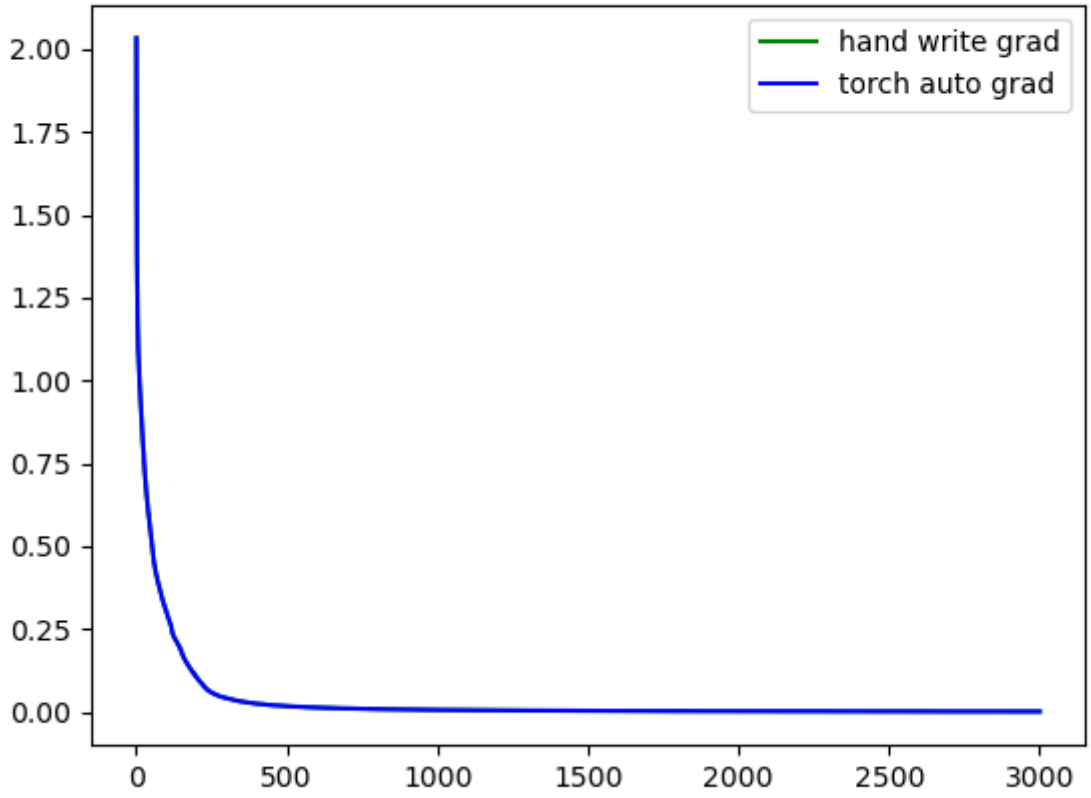
在计算b时，需要选择$0 < \alpha < C$的支持向量进行计算，也是由于精度问题，在选择小于C的alpha时C需要减去$\varepsilon$

即

```
for i, alpha in enumerate(self.SV_alpha):
    if alpha < self.C-self.epsilon:
        self.b = self.SV_label[i] - self.wx(self.SV[i])
        break
```

# 感知机模型

loss训练曲线(手动编写求导(hand write grad)与自动求导(torch auto grad))，由于两者各层初始参数均采用相同初始值，因此两者loss曲线重合




W,b梯度比较见附录1，或者见 `src2/mlp_grad.out`

W,b最终结果见附录2，或者见 `src2/mlp_w_b.out`

# 卷积神经网络

(8+2)%6+1=5

前两个2d卷积层padding设置为(kernel size-1)/2，使得输出图像和输入图像大小相同，同时防止边缘信息丢失

最大池化根据参数会导致宽度和高度减半，第5层结束后，宽度和高度为$7*7$,所以线性输入为$32*7*7$

计算过程为

layer1 32-5+2*2+1=32
layer2 32/2=16
layer3 16-3+2*1+1=16
layer4 16/2=8
layer5 8-2+1=7
所以layer5之后为32*7*7的特征

除了池化层之外都使用relu激活函数

在传入线性层前，需要将除了batch size之外的维度扁平化 `x = x.view(-1, self.num_flat_features(x))` ，即变成$32 * 7 * 7$的一维向量（加上batch size就是二维张量）

计算其他维度的规模

```
def num_flat_features(self, x):
    size = x.size()[1:]
    num_features = 1
    for s in size:
        num_features *= s
    return num_features
```

```
(py36) ➜  src2 git:(main) ✗ python MyNet.py
Train Epoch: 0/5 [0/50000]       Loss: 2.302881
Train Epoch: 0/5 [12800/50000]   Loss: 2.248610
Train Epoch: 0/5 [25600/50000]   Loss: 1.911257
Train Epoch: 0/5 [38400/50000]   Loss: 1.736851
Train Epoch: 1/5 [0/50000]       Loss: 1.622783
Train Epoch: 1/5 [12800/50000]   Loss: 1.650238
Train Epoch: 1/5 [25600/50000]   Loss: 1.470649
Train Epoch: 1/5 [38400/50000]   Loss: 1.334364
Train Epoch: 2/5 [0/50000]       Loss: 1.325757
Train Epoch: 2/5 [12800/50000]   Loss: 1.199710
Train Epoch: 2/5 [25600/50000]   Loss: 1.367777
Train Epoch: 2/5 [38400/50000]   Loss: 1.315201
Train Epoch: 3/5 [0/50000]       Loss: 1.272510
Train Epoch: 3/5 [12800/50000]   Loss: 1.177636
Train Epoch: 3/5 [25600/50000]   Loss: 1.303796
Train Epoch: 3/5 [38400/50000]   Loss: 1.240439
Train Epoch: 4/5 [0/50000]       Loss: 0.959395
Train Epoch: 4/5 [12800/50000]   Loss: 0.926257
Train Epoch: 4/5 [25600/50000]   Loss: 1.138729
Train Epoch: 4/5 [38400/50000]   Loss: 1.087468
Finished Training
Test set: Average loss: 1.0987   Acc 0.61
```

最后平均loss和训练时loss接近,说明符合的较好

# 附录1

手动求导与pytorch自动求导梯度结果对比

参数为W, b

仅比较最后一次epoch结束后的W梯度和b梯度

共3个隐层，4组参数，分别在layer1, 2, 3, 4层

其中 `mlp grad for w` 为W的手动求导, `torch auto grad for w` 为用pytorch的autograd自动求导得到的W偏导。

b参数的类似

手动求导结果和autograd结果基本相同

```
layer 1
mlp grad for w
tensor([[-1.1156e-09,  8.3482e-10, -2.6792e-09, -4.9908e-09, -8.9714e-09,
          8.0907e-10,  1.5823e-08,  4.8376e-10, -2.4587e-09,  1.2990e-09],
        [-1.9988e-04,  1.4957e-04, -4.8003e-04, -8.9419e-04, -1.6074e-03,
          1.4496e-04,  2.8350e-03,  8.6674e-05, -4.4053e-04,  2.3274e-04],
        [-2.3042e-05,  1.7242e-05, -5.5336e-05, -1.0308e-04, -1.8529e-04,
          1.6710e-05,  3.2680e-04,  9.9914e-06, -5.0782e-05,  2.6829e-05],
        [-1.8747e-09,  1.4029e-09, -4.5023e-09, -8.3867e-09, -1.5076e-08,
          1.3596e-09,  2.6590e-08,  8.1293e-10, -4.1318e-09,  2.1829e-09],
        [ 1.2842e-06, -9.6093e-07,  3.0840e-06,  5.7447e-06,  1.0327e-05,
         -9.3129e-07, -1.8213e-05, -5.5684e-07,  2.8302e-06, -1.4952e-06],
        [ 9.2641e-05, -6.9323e-05,  2.2248e-04,  4.1443e-04,  7.4498e-04,
         -6.7185e-05, -1.3139e-03, -4.0171e-05,  2.0417e-04, -1.0787e-04],
        [-2.4274e-04,  1.8164e-04, -5.8294e-04, -1.0859e-03, -1.9520e-03,
          1.7604e-04,  3.4427e-03,  1.0526e-04, -5.3497e-04,  2.8263e-04],
        [-9.5575e-07,  7.1518e-07, -2.2953e-06, -4.2756e-06, -7.6857e-06,
          6.9312e-07,  1.3555e-05,  4.1443e-07, -2.1064e-06,  1.1128e-06],
        [ 7.8280e-05, -5.8576e-05,  1.8799e-04,  3.5019e-04,  6.2949e-04,
         -5.6770e-05, -1.1102e-03, -3.3944e-05,  1.7252e-04, -9.1146e-05],
        [ 1.2234e-06, -9.1546e-07,  2.9380e-06,  5.4729e-06,  9.8380e-06,
         -8.8723e-07, -1.7352e-05, -5.3049e-07,  2.6963e-06, -1.4245e-06]],
       dtype=torch.float64)
torch auto grad for w
tensor([[-1.1156e-09,  8.3482e-10, -2.6792e-09, -4.9908e-09, -8.9714e-09,
          8.0907e-10,  1.5823e-08,  4.8376e-10, -2.4587e-09,  1.2990e-09],
        [-1.9988e-04,  1.4957e-04, -4.8003e-04, -8.9419e-04, -1.6074e-03,
          1.4496e-04,  2.8350e-03,  8.6674e-05, -4.4053e-04,  2.3274e-04],
        [-2.3042e-05,  1.7242e-05, -5.5336e-05, -1.0308e-04, -1.8529e-04,
          1.6710e-05,  3.2680e-04,  9.9914e-06, -5.0782e-05,  2.6829e-05],
        [-1.8747e-09,  1.4029e-09, -4.5023e-09, -8.3867e-09, -1.5076e-08,
          1.3596e-09,  2.6590e-08,  8.1293e-10, -4.1318e-09,  2.1829e-09],
        [ 1.2842e-06, -9.6093e-07,  3.0840e-06,  5.7447e-06,  1.0327e-05,
         -9.3129e-07, -1.8213e-05, -5.5684e-07,  2.8302e-06, -1.4952e-06],
        [ 9.2641e-05, -6.9323e-05,  2.2248e-04,  4.1443e-04,  7.4498e-04,
         -6.7185e-05, -1.3139e-03, -4.0171e-05,  2.0417e-04, -1.0787e-04],
        [-2.4274e-04,  1.8164e-04, -5.8294e-04, -1.0859e-03, -1.9520e-03,
          1.7604e-04,  3.4427e-03,  1.0526e-04, -5.3497e-04,  2.8263e-04],
        [-9.5575e-07,  7.1518e-07, -2.2953e-06, -4.2756e-06, -7.6857e-06,
          6.9312e-07,  1.3555e-05,  4.1443e-07, -2.1064e-06,  1.1128e-06],
        [ 7.8280e-05, -5.8576e-05,  1.8799e-04,  3.5019e-04,  6.2949e-04,
         -5.6770e-05, -1.1102e-03, -3.3944e-05,  1.7252e-04, -9.1146e-05],
        [ 1.2234e-06, -9.1546e-07,  2.9380e-06,  5.4729e-06,  9.8380e-06,
         -8.8723e-07, -1.7352e-05, -5.3049e-07,  2.6963e-06, -1.4245e-06]],
       dtype=torch.float64)
mlp grad for b
tensor([[-6.9481e-09],
        [-1.2449e-03],
        [-1.4350e-04],
        [-1.1676e-08],
```

```
        [ 7.9978e-06],
        [ 5.7697e-04],
        [-1.5118e-03],
        [-5.9524e-06],
        [ 4.8753e-04],
        [ 7.6193e-06]], dtype=torch.float64)
torch auto grad for b
tensor([[-6.9481e-09],
        [-1.2449e-03],
        [-1.4350e-04],
        [-1.1676e-08],
        [ 7.9978e-06],
        [ 5.7697e-04],
        [-1.5118e-03],
        [-5.9524e-06],
        [ 4.8753e-04],
        [ 7.6193e-06]], dtype=torch.float64)
layer 2
mlp grad for w
tensor([[ 1.9457e-05, -8.7202e-06,  1.7473e-05, -1.9457e-05, -1.9386e-05,
          1.5794e-05,  5.5139e-06,  1.9209e-05,  1.7268e-05, -1.8773e-05],
        [ 8.6881e-04, -3.8938e-04,  7.8022e-04, -8.6880e-04, -8.6562e-04,
          7.0523e-04,  2.4621e-04,  8.5771e-04,  7.7104e-04, -8.3827e-04],
        [-8.3514e-05,  3.7429e-05, -7.4998e-05,  8.3513e-05,  8.3208e-05,
         -6.7790e-05, -2.3666e-05, -8.2447e-05, -7.4116e-05,  8.0578e-05],
        [-1.6830e-08,  7.5426e-09, -1.5114e-08,  1.6830e-08,  1.6768e-08,
         -1.3661e-08, -4.7693e-09, -1.6615e-08, -1.4936e-08,  1.6238e-08],
        [ 1.1451e-08, -5.1321e-09,  1.0283e-08, -1.1451e-08, -1.1409e-08,
          9.2951e-09,  3.2451e-09,  1.1305e-08,  1.0163e-08, -1.1049e-08],
        [ 4.5335e-04, -2.0318e-04,  4.0712e-04, -4.5334e-04, -4.5168e-04,
          3.6799e-04,  1.2847e-04,  4.4755e-04,  4.0233e-04, -4.3741e-04],
        [-4.3089e-05,  1.9311e-05, -3.8695e-05,  4.3088e-05,  4.2931e-05,
         -3.4976e-05, -1.2211e-05, -4.2538e-05, -3.8240e-05,  4.1574e-05],
        [-2.7834e-06,  1.2475e-06, -2.4996e-06,  2.7834e-06,  2.7732e-06,
         -2.2594e-06, -7.8877e-07, -2.7478e-06, -2.4702e-06,  2.6856e-06]],
       dtype=torch.float64)
torch auto grad for w
tensor([[ 1.9457e-05, -8.7202e-06,  1.7473e-05, -1.9457e-05, -1.9386e-05,
          1.5794e-05,  5.5139e-06,  1.9209e-05,  1.7268e-05, -1.8773e-05],
        [ 8.6881e-04, -3.8938e-04,  7.8022e-04, -8.6880e-04, -8.6562e-04,
          7.0523e-04,  2.4621e-04,  8.5771e-04,  7.7104e-04, -8.3827e-04],
        [-8.3514e-05,  3.7429e-05, -7.4998e-05,  8.3513e-05,  8.3208e-05,
         -6.7790e-05, -2.3666e-05, -8.2447e-05, -7.4116e-05,  8.0578e-05],
        [-1.6830e-08,  7.5426e-09, -1.5114e-08,  1.6830e-08,  1.6768e-08,
         -1.3661e-08, -4.7693e-09, -1.6615e-08, -1.4936e-08,  1.6238e-08],
        [ 1.1451e-08, -5.1321e-09,  1.0283e-08, -1.1451e-08, -1.1409e-08,
          9.2951e-09,  3.2451e-09,  1.1305e-08,  1.0163e-08, -1.1049e-08],
        [ 4.5335e-04, -2.0318e-04,  4.0712e-04, -4.5334e-04, -4.5168e-04,
          3.6799e-04,  1.2847e-04,  4.4755e-04,  4.0233e-04, -4.3741e-04],
        [-4.3089e-05,  1.9311e-05, -3.8695e-05,  4.3088e-05,  4.2931e-05,
         -3.4976e-05, -1.2211e-05, -4.2538e-05, -3.8240e-05,  4.1574e-05],
```

```
        [-2.7834e-06,  1.2475e-06, -2.4996e-06,  2.7834e-06,  2.7732e-06,
         -2.2594e-06, -7.8877e-07, -2.7478e-06, -2.4702e-06,  2.6856e-06]],
       dtype=torch.float64)
mlp grad for b
tensor([[-1.9457e-05],
        [-8.6881e-04],
        [ 8.3514e-05],
        [ 1.6830e-08],
        [-1.1451e-08],
        [-4.5335e-04],
        [ 4.3089e-05],
        [ 2.7834e-06]], dtype=torch.float64)
torch auto grad for b
tensor([[-1.9457e-05],
        [-8.6881e-04],
        [ 8.3514e-05],
        [ 1.6830e-08],
        [-1.1451e-08],
        [-4.5335e-04],
        [ 4.3089e-05],
        [ 2.7834e-06]], dtype=torch.float64)
layer 3
mlp grad for w
tensor([[-2.3585e-09,  7.3857e-10, -1.8014e-09, -2.3896e-09, -2.3897e-09,
          1.9788e-10,  2.3076e-09, -2.3726e-09],
        [ 9.2211e-09, -2.8876e-09,  7.0431e-09,  9.3427e-09,  9.3429e-09,
         -7.7365e-10, -9.0220e-09,  9.2761e-09],
        [ 5.2871e-05, -1.6557e-05,  4.0383e-05,  5.3569e-05,  5.3570e-05,
         -4.4359e-06, -5.1730e-05,  5.3187e-05],
        [ 3.5748e-10, -1.1194e-10,  2.7304e-10,  3.6219e-10,  3.6220e-10,
         -2.9992e-11, -3.4976e-10,  3.5961e-10],
        [ 1.4881e-04, -4.6599e-05,  1.1366e-04,  1.5077e-04,  1.5077e-04,
         -1.2485e-05, -1.4559e-04,  1.4969e-04],
        [-9.1517e-07,  2.8659e-07, -6.9901e-07, -9.2724e-07, -9.2726e-07,
          7.6783e-08,  8.9542e-07, -9.2063e-07],
        [ 2.1927e-04, -6.8665e-05,  1.6748e-04,  2.2216e-04,  2.2217e-04,
         -1.8397e-05, -2.1454e-04,  2.2058e-04],
        [ 3.7081e-08, -1.1612e-08,  2.8322e-08,  3.7570e-08,  3.7571e-08,
         -3.1111e-09, -3.6280e-08,  3.7302e-08]], dtype=torch.float64)
torch auto grad for w
tensor([[-2.3585e-09,  7.3857e-10, -1.8014e-09, -2.3896e-09, -2.3897e-09,
          1.9788e-10,  2.3076e-09, -2.3726e-09],
        [ 9.2211e-09, -2.8876e-09,  7.0431e-09,  9.3427e-09,  9.3429e-09,
         -7.7365e-10, -9.0220e-09,  9.2761e-09],
        [ 5.2871e-05, -1.6557e-05,  4.0383e-05,  5.3569e-05,  5.3570e-05,
         -4.4359e-06, -5.1730e-05,  5.3187e-05],
        [ 3.5748e-10, -1.1194e-10,  2.7304e-10,  3.6219e-10,  3.6220e-10,
         -2.9992e-11, -3.4976e-10,  3.5961e-10],
        [ 1.4881e-04, -4.6599e-05,  1.1366e-04,  1.5077e-04,  1.5077e-04,
         -1.2485e-05, -1.4559e-04,  1.4969e-04],
        [-9.1517e-07,  2.8659e-07, -6.9901e-07, -9.2724e-07, -9.2726e-07,
```

```
          7.6783e-08,  8.9542e-07, -9.2063e-07],
        [ 2.1927e-04, -6.8665e-05,  1.6748e-04,  2.2216e-04,  2.2217e-04,
         -1.8397e-05, -2.1454e-04,  2.2058e-04],
        [ 3.7081e-08, -1.1612e-08,  2.8322e-08,  3.7570e-08,  3.7571e-08,
         -3.1111e-09, -3.6280e-08,  3.7302e-08]], dtype=torch.float64)
mlp grad for b
tensor([[-2.3897e-09],
        [ 9.3430e-09],
        [ 5.3570e-05],
        [ 3.6220e-10],
        [ 1.5077e-04],
        [-9.2727e-07],
        [ 2.2217e-04],
        [ 3.7571e-08]], dtype=torch.float64)
torch auto grad for b
tensor([[-2.3897e-09],
        [ 9.3430e-09],
        [ 5.3570e-05],
        [ 3.6220e-10],
        [ 1.5077e-04],
        [-9.2727e-07],
        [ 2.2217e-04],
        [ 3.7571e-08]], dtype=torch.float64)
layer 4
mlp grad for w
tensor([[ 3.1321e-05,  3.1321e-05, -3.0664e-05, -3.1321e-05, -3.0063e-05,
          3.1284e-05, -3.0119e-05,  3.1320e-05],
        [ 4.6671e-06,  4.6671e-06, -4.5693e-06, -4.6671e-06, -4.4796e-06,
          4.6616e-06, -4.4881e-06,  4.6670e-06],
        [-5.1627e-04, -5.1627e-04,  5.0545e-04,  5.1627e-04,  4.9554e-04,
         -5.1567e-04,  4.9647e-04, -5.1627e-04],
        [ 4.8029e-04,  4.8029e-04, -4.7022e-04, -4.8029e-04, -4.6099e-04,
          4.7973e-04, -4.6186e-04,  4.8028e-04]], dtype=torch.float64)
torch auto grad for w
tensor([[ 3.1321e-05,  3.1321e-05, -3.0664e-05, -3.1321e-05, -3.0063e-05,
          3.1284e-05, -3.0119e-05,  3.1320e-05],
        [ 4.6671e-06,  4.6671e-06, -4.5693e-06, -4.6671e-06, -4.4796e-06,
          4.6616e-06, -4.4881e-06,  4.6670e-06],
        [-5.1627e-04, -5.1627e-04,  5.0545e-04,  5.1627e-04,  4.9554e-04,
         -5.1567e-04,  4.9647e-04, -5.1627e-04],
        [ 4.8029e-04,  4.8029e-04, -4.7022e-04, -4.8029e-04, -4.6099e-04,
          4.7973e-04, -4.6186e-04,  4.8028e-04]], dtype=torch.float64)
mlp grad for b
tensor([[ 3.1321e-05],
        [ 4.6671e-06],
        [-5.1627e-04],
        [ 4.8029e-04]], dtype=torch.float64)
torch auto grad for b
tensor([[ 3.1321e-05],
        [ 4.6671e-06],
        [-5.1627e-04],
```

```
        [ 4.8029e-04]], dtype=torch.float64)
```

# 附录2

w[i],b[i]表示第i层参数，i与实验文档感知机模型图中的下标对应

```
w[1]=
[[-0.27763758 -0.13177916 -1.99016838 -2.10773093 -0.0134519   0.96730575
   2.00933427 -0.65945053 -0.9702496  -0.55966388]
 [-2.69364196 -0.45528174  0.76758551  1.92840008 -0.17426736  1.7810065
  -0.25486737 -0.72498822 -0.59385112  0.28620328]
 [-0.17057625  0.51982925 -0.49185795  0.9928277  -0.39651946 -1.02571681
   0.60166646 -0.9731959  -2.11131021  1.3735617 ]
 [ 1.70451661  2.49321449  1.3536435   0.07877109  1.59396654  0.733802
  -2.28369864  0.89369945 -0.00710366  0.07120367]
 [ 0.86373317  0.19832009  1.7825328   0.27114375 -0.09153661  0.59291285
  -0.21245541 -1.95254075  1.26355691  0.56388619]
 [ 1.80425821 -0.13324681 -0.17459183 -0.39799123 -0.13855407 -0.38245164
   0.80052427  0.25143499 -0.85309224  1.91913159]
 [-1.41050618 -0.98819567  0.61621885  0.36673261 -2.00690947 -0.26987126
  -0.52644397  0.94462977 -0.42658124  1.55491767]
 [ 0.36365618 -0.5048735  -0.48867086  0.97236518 -2.74182143 -0.78204412
  -0.44084383  1.15910757 -0.36312181  2.18681028]
 [-3.36053748  0.33849701  0.53511648 -0.52142237  0.21644464 -0.30703216
   0.34812944 -0.2100311  -1.72500562 -0.40521775]
 [ 2.42618389  0.03140831  0.72503958 -0.83218529 -0.40968496  0.74721248
  -1.04788744  0.80985516 -1.47056602 -0.0931549 ]]
b[1]=
[[ 0.24372719]
 [-0.75564603]
 [ 0.80333067]
 [-1.5420253 ]
 [ 1.37792348]
 [ 1.55097046]
 [ 1.18391454]
 [-0.0890484 ]
 [ 0.33616098]
 [ 0.73306434]]
w[2]=
[[-1.88365317e-02 -3.54801077e-01 -1.54950243e+00  5.18289676e-01
  -1.32371282e+00 -1.28349487e+00 -5.11275930e-01 -3.72175040e-01
  -1.34393843e-01 -7.77701079e-01]
 [ 1.97483919e+00  2.13262345e+00 -4.26270417e-01  8.98784253e-01
  -7.68290157e-01 -7.49190529e-01  1.51647739e+00  1.12579881e+00
  -2.27211586e+00 -3.94155779e-01]
 [-4.62337323e-01  1.58181607e-01  7.42486279e-01 -1.25863757e+00
  -1.24286723e+00 -1.45510129e+00 -2.00691973e+00  1.23714670e+00
  -2.16877311e-01  2.52087209e+00]
 [-2.22366711e+00  9.02246380e-01 -1.28132724e+00 -2.61702443e-02
  -1.76808468e+00 -1.04487024e+00 -1.01061518e+00 -4.62442224e-01
  -1.84492893e+00 -1.34348964e-01]
 [-7.69015387e-01 -2.24886715e-01  5.32179167e-01  4.89596993e-01
   5.25252165e-01 -2.43210776e-01  1.10346636e+00 -7.27635707e-01
  -1.01450084e+00  2.60232050e+00]
 [ 5.28551066e-01 -6.18523545e-01  2.63323124e+00 -6.30829846e-01
  -9.16436129e-01 -2.57356189e+00  4.47613113e-01 -1.19191882e+00
```

```
  -6.08221729e-01  1.02096221e+00]
 [ 1.13031720e+00 -8.34035043e-02 -1.73443605e-01  6.11122561e-01
   2.04146372e+00 -6.10731224e-01  7.26668252e-01  2.14331820e+00
   1.40599188e+00  1.10092045e-03]
 [-1.07159705e+00 -6.74328215e-01 -9.60233615e-01 -1.23503215e+00
  -6.77676160e-01 -1.18918241e+00  1.10301502e+00  1.80025441e-01
   1.52296160e-01  1.80393470e+00]]
b[2]=
[[ 1.13907242]
 [-0.51959541]
 [ 0.45900093]
 [ 0.43031676]
 [ 0.77711038]
 [-0.03228182]
 [-0.58377049]
 [ 1.01261355]]
w[3]=
[[ 0.89950061  0.57014557  1.75517382  1.1052431   0.86254548  0.90270622
   0.895782    3.67542032]
 [ 2.40349304  1.8226212   2.4630251   0.06834409  0.58449133  0.00403376
  -1.81144454  1.28841913]
 [ 1.48056942 -2.69376365  0.25583422 -1.82422441 -0.57146017 -1.72716478
   2.13223286  0.22098761]
 [-1.50720542  2.82804951 -1.2198619  -0.40148648 -1.60678359  0.23273307
   1.89902783 -1.06429592]
 [-1.26367966 -2.79045068  2.18915405  0.82723988 -1.84813471  1.28946848
   0.42848861 -1.91090671]
 [ 0.38586122 -0.2181766   1.43158754 -2.54901937  1.74035112  0.80061969
  -0.34039572  1.44431676]
 [-2.87622998 -1.78190413 -0.67097735  1.04819442 -0.67527254 -2.5095816
   2.06618083  2.12740289]
 [ 1.0658855   3.5667683   1.21620007 -1.0489208   0.118288   -1.9857766
  -1.90194888  1.46654223]]
b[3]=
[[ 0.8827086 ]
 [-0.15804035]
 [-0.66842402]
 [-0.1770155 ]
 [ 0.22936263]
 [ 1.30724564]
 [ 0.12879683]
 [ 2.68609022]]
w[4]=
[[ 2.03570412 -1.72806079 -1.30638525  3.62344399  4.23777328 -3.03103055
  -3.34860952  0.50834019]
 [ 1.45969524 -1.1036118   5.15656284  0.47939033  1.47599567 -2.74822728
  -0.64534089  5.13502722]
 [ 0.17965705 -3.59892963 -2.0036839  -3.28999829 -4.44721937  2.94889336
  -1.29494511 -2.84872358]
 [-4.86571834  4.77974608  0.57056088 -0.22255717 -1.08319787  2.56733156
   4.98045406  0.58256288]]
```

```
b[4]=
[[ 1.08253476]
 [-2.64657636]
 [-1.39308155]
 [-0.5658774 ]]
```