人工基础实验报告

> PB19051035周佳豪

传统机器学习

预测结果为：

```
DecisionTree acc: 78.57%
SVM(Linear kernel) acc: 86.67%
SVM(Poly kernel) acc: 80.00%
SVM(Gauss kernel) acc: 80.00%
```

决策树

决策树的准确率为：

```
DecisionTree acc: 78.57%
```

1. 如何划分最优属性?

   在决策树的每个中间节点，通过计算A中每个剩余属性的信息增益值，选择最大的作为划分最优属性。

```python
def getOptimal(D,A):
    max_v = 0
    max_index = 0
    X_i= []
    P_i = []
    N_i = []
    p = np_count(D.train_labels,1)
    n = np_count(D.train_labels,0)
    I=getI(p/(p+n),n/(p+n))
    start = 0
    for index in range(len(A)):
        attr = A[index]
        v = I
        # a为attr属性的取值集合,取值范围为[0,10]
        a = []
        for i in range(len(D.train_features)):
            a.append(int(D.train_features[i][attr]))
        # x[i]表示取值为i的个数
        # Positives[i]表示在取值为i中正样本的个数
        # Negatives[i]表示在取值为i中负样本的个数
        x = [0]*11
        Positives = [0]*11
        Negatives = [0]*11
        for i in range(len(a)):
            x[a[i]] += 1
            if(D.train_labels[i]==1):
                Positives[a[i]] += 1
            else:
                Negatives[a[i]] += 1
        # 计算每个属性的信息增益
```

```
        # 用max_v和max_index储存信息增益的最大值以及对应的属性
        for i in range(len(x)):
            if x[i]!=0:
                v -=
x[i]/(p+n)*getI(Positives[i]/(Positives[i]+Negatives[i]),Negatives[i]/(Positive
s[i]+Negatives[i]))
        if start==0:
            max_v = v
            max_index = index
            start = 1
            X_i = x
            P_i = Positives
            N_i = Negatives
        else:
            if(max_v<v):
                max_v = v
                max_index = index
                X_i = x
                P_i = Positives
                N_i = Negatives
    return A[max_index],X_i,P_i,N_i
```

2. 如何生成叶节点与中间节点?

   对于一个节点，应先判断它是叶节点还是叶节点。若是叶节点（关于如何判断该点是叶节点，实验文档的算法已经进行详细说明，这里不再赘述），则停止生成子节点并设置该叶节点的预测值，若样本全是正样本或全是负样本，则返回1或0，否则少数服从多数返回样本最多的一个预测。若该节点为中间节点（不是叶节点则就是中间节点），则继续生成子节点。

   判断该点是中间节点后，选择最优化分属性，生成子节点，并按顺序储存在该节点的child列表中，即列表中的第i个子节点对应划分属性的值为i。由于每个属性的取值范围为0-10，且是离散的，故每个中间节点有11个子节点，分别对应属性值0,1,2，..，10。然后在A中剔除该最优属性，并选择对应属性值的样本，传递给子节点，递归对11个子节点进行相同的操作。

```
    # 判断是否为叶节点
    result1 = np.asarray([0]*len(D.train_labels))
    result2 = np.asarray([1]*len(D.train_labels))
    if (D.train_labels==result1).all():
        node.item="leaf"
        node.value = 0
        return node
    if (D.train_labels==result2).all():
        node.item="leaf"
        node.value = 1
        return node
    if len(A)==0 or is_equal(D,A)==True:
        node.item="leaf"
        numN=np_count(D.train_labels,0)
        numP=np_count(D.train_labels,1)
        if numN>numP:
            node.value = 0
        else:
            node.value = 1
        return node
```

```
    # 中间节点生成子节点
    node.branchIndex = attr
    for i in range(len(x)):
```

```python
        # value为该属性的取值
        value = i
        node.branchValue.append(value)

        newTrainFeatures = []
        newTrainLabels = []
        for j in range(len(D.train_labels)):
            if D.train_features[j][attr]==value:
                newTrainFeatures.append(D.train_features[j])
                newTrainLabels.append(D.train_labels[j])
        newTrainFeatures = np.asarray(newTrainFeatures)
        newTrainLabels = np.asarray(newTrainLabels)

        newTestFeatures = []
        newTestLabels = []
        for j in range(len(T.train_labels)):
            if T.train_features[j][attr]==value:
                newTestFeatures.append(T.train_features[j])
                newTestLabels.append(T.train_labels[j])
        newTestFeatures = np.asarray(newTestFeatures)
        newTestLabels = np.asarray(newTestLabels)

        newA = A.copy()
        newA.remove(attr)
        newD = TrainSets(newTrainFeatures,newTrainLabels)
        newT = TrainSets(newTestFeatures,newTestLabels)

        node.children.append(TreeGenerate(newD,newA,newT))
```

3. 进行的一些优化

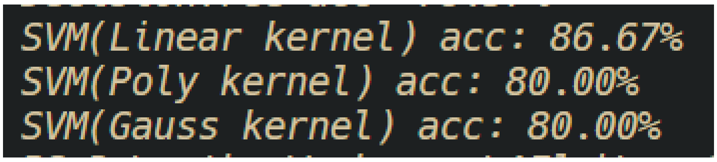
   本次实验进行了预剪枝的优化，最终对测试集的预测准确率达到了78.57%。把训练集按3:1分为训练集和验证集。当使用文档的算法判断某个节点是中间节点后，此时先不急着划分，使用验证集评估把该节点标记为叶节点的准确率。然后再一次使用验证集计算用最优属性划分后的准确率。若果当前结点的划分后的准确率低，则直接把该节点标记为叶节点。否则继续划分。要注意的一点是在从中间节点到子节点时，验证集的样本要选择与该属性值相同的样本进行传递。但是在实际剪枝的过程中，会发现决策树直接把根节点减掉了，即把所有样本均预测为0，这样准确率达到了70%左右，但决策树已不存在。故在实际剪枝过程中规定只有在节点达到一定的深度时(通过计算A中剩余属性的个数)才考虑剪枝，这样最终的测试集准确率为78.57%

```python
    # 计算划分前的准确率
    numN=np_count(D.train_labels,0)
    numP=np_count(D.train_labels,1)
    if numN>numP:
        resultPredict = 0
    else:
        resultPredict = 1
    numDivide = len(T.train_labels)
    correctDivide = 0
    for i in range(len(T.train_labels)):
        if(T.train_labels[i]==resultPredict):
            correctDivide += 1
    if numDivide!=0:
        Acc1 = correctDivide/numDivide
        #print('划分前的准确率:',Acc1)

    # 计算划分后的正确率
    # Predictions[i]为attr为i时的预测值
```

```python
    Predictions = []
    for i in range(len(x)):
        if(x[i]!=0):
            if(P[i]>N[i]):
                Predictions.append(1)
            else:
                Predictions.append(0)
        else:
            Predictions.append(resultPredict)
    correctNoDivide = 0
    for i in range(len(T.train_labels)):
        test_features = T.train_features[i]
        test_labels = T.train_labels[i]
        if(test_labels==Predictions[test_features[attr]]):
            correctNoDivide += 1
    if numDivide!=0:
        Acc2 = correctNoDivide/numDivide
        #print('划分前的准确率:',Acc2)

    # 若划分前正确率更高，则不划分
    # 第一层禁止剪枝
    if(numDivide!=0 and Acc1>Acc2 and len(A)<=8 ):
        #print('不进行划分,attr=',attr)
        node.item="leaf"
        node.value = resultPredict
        return node
```

支持向量机

SVM的准确率为：



通过求解此问题得到超平面 $w^T x + b$:

$$min_{\alpha}\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j K(x_i,x_j)-\sum_{i=1}^{N}\alpha_i$$

$$s.t.\quad \sum_{i=1}^{N}\alpha_i y_i = 0$$

$$0 <= \alpha_i <= C$$

通过求得的 α 可由以下公式得到 w, b

$$w = \sum_{i=1}^{n}\alpha_i x_i y_i$$

$$b = \frac{\sum_{i=1}^{N}(y_i - \sum_{j=1}^{n}\alpha_j y_j K(x_j,x_i))}{N}$$

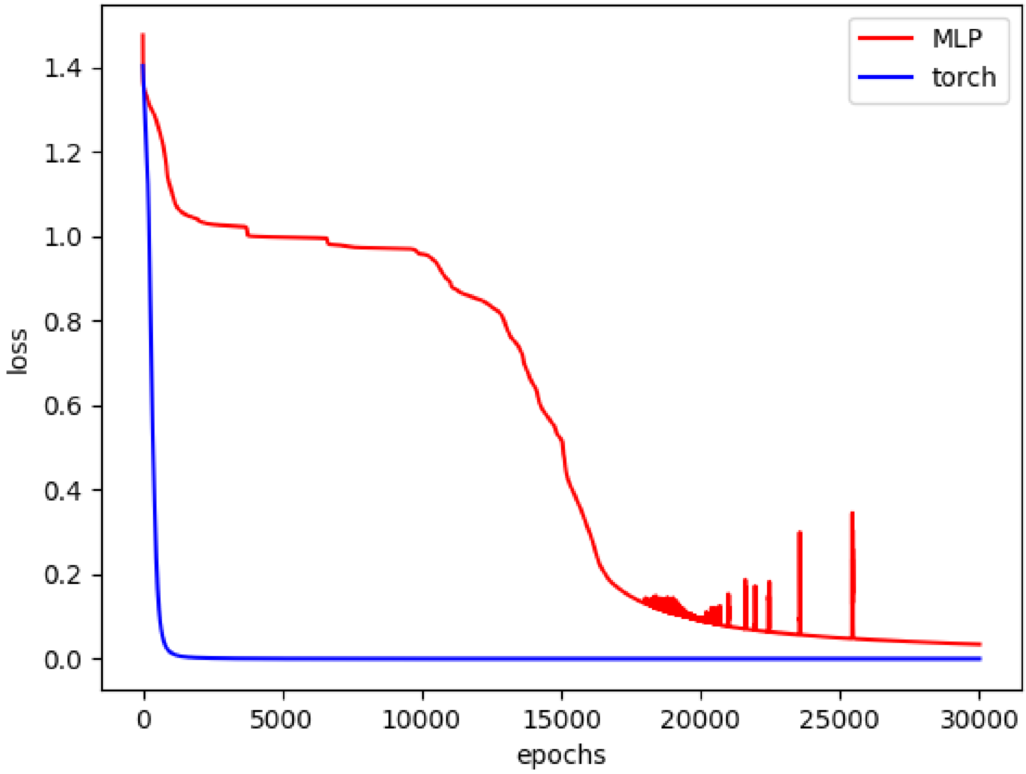
此问题使用cvxpy求解，其中核函数K有三种情况：Gauss，Linear，Poly。通过观察三种核函数的预测准确率，可发现Linear的最高，为86.6%，其他两种均为80%

深度学习

手写感知机模型并进行反向传播

> 设置训练轮数epochs=30000，学习率lr=0.001

loss训练曲线：



可以看出手写的MLP收敛速度远不如torch，并且在训练轮数达到20000左右后，loss会出现跳跃,这是因为设置的batch_size=1，随机性比较大，很容易出现跳跃。但从全局来看，loss是逐渐趋于0的

手写MLP的W和b最终结果：

```
w= [array([[-9.99972296e-01,  9.22126722e-01, -3.54329791e-01,
          1.04280648e+00,  9.88523418e-01, -8.11034558e-01,
          1.30339897e+00,  1.22054599e+00,  5.31803794e-02,
         -1.82780672e-02],
        [-1.01401502e+00,  3.11311453e-01,  8.01327691e-01,
          1.03832876e+00,  5.38644030e-01,  8.98816841e-01,
          8.55783605e-01, -2.05434495e-01, -1.70961405e-02,
         -3.85887561e-02],
        [ 1.61793779e+00,  5.94218843e-01,  4.24050234e-01,
          2.23640773e-01,  1.25719187e+00, -5.41172936e-03,
         -6.60937960e-01, -7.46377188e-01, -4.89946428e-01,
          6.15625851e-01],
        [ 7.87352310e-01,  8.78885824e-01,  1.35641528e+00,
          4.41318130e-01,  1.97946271e+00,  8.08481740e-01,
          9.99978463e-01,  1.03138280e+00,  9.67015783e-01,
         -6.94425887e-03],
        [ 3.58600018e-01,  2.63384720e-01,  1.62391781e-01,
          1.33387025e+00,  1.41856815e-01,  1.38483867e+00,
          3.64520725e-01,  3.33498871e-01, -7.32114229e-01,
         -1.02337898e-01],
```

```
        [-7.10776789e-01,  1.00528946e+00,  3.99931791e-01,
          8.41732464e-01,  8.84323533e-01,  6.27005010e-02,
         -1.40351101e-01, -3.91379022e-01,  1.34404239e-01,
          1.93908373e+00],
        [-7.49320641e-02,  1.83414369e+00,  1.24361988e+00,
          3.88311171e-02, -1.07057070e-03,  9.98956646e-02,
          1.16810159e+00,  4.09322306e-01,  1.43229390e+00,
          1.05499377e+00],
        [ 1.03842562e+00, -6.09571238e-01,  1.06911220e+00,
          4.60853883e-01, -9.66956343e-02,  4.50712490e-01,
          1.26266670e+00,  4.23134242e-01,  1.77496096e+00,
          1.69948761e+00],
        [ 9.94407708e-01,  6.72158522e-01,  1.35950352e+00,
         -6.75162136e-01,  4.60768637e-01,  1.36272867e+00,
         -1.20204112e-01,  1.41332624e+00,  4.54470613e-01,
         -6.34712415e-01],
        [ 3.58871224e-01,  3.96030704e-01,  4.07469863e-01,
          9.59168401e-01,  1.96763455e+00, -2.29694600e-01,
          1.25254312e+00,  6.36821877e-01, -3.49786226e-01,
          5.99741865e-02]]), array([[ 0.04133732,  0.26235191,  1.20359401,
0.15816841,  0.2014189 ,
          0.8832176 ,  0.94179833,  0.60824863,  0.95702762,  0.20414293],
        [-0.78414543, -0.98144258, -1.00576978,  1.1003628 ,  1.06886039,
         -0.59874987,  0.91779365,  0.50445397, -0.48701052, -0.05530193],
        [ 0.19371016,  1.34792424,  0.89692878,  0.80849536,  0.9519987 ,
         -0.09196814,  0.78381082, -0.47061069,  1.07737403,  1.34282459],
        [ 2.6621843 ,  0.14987681, -0.24709221, -0.59659977, -0.05044174,
         -0.4941338 ,  0.40458471,  2.44708163,  0.14840516, -2.00996268],
        [ 0.00660812,  1.51855137,  0.75007795,  0.24851945,  0.34037129,
          0.54632601,  0.60703923,  0.20855284,  1.83029758,  0.79104244],
        [ 0.50351081,  1.10183568,  1.38430628,  1.91561017,  1.11582421,
         -0.28438964, -0.09239147, -0.27073864,  0.76696107,  1.09509568],
        [ 0.47779185,  1.17537024,  0.49443049,  1.21492243,  1.04770991,
          1.42811344,  0.5691272 ,  0.06827378, -0.71093346,  0.80761146],
        [ 0.15661806,  0.32513986,  0.64070257,  0.26527559,  0.2233512 ,
          2.40779716, -0.31487537,  0.65603433, -0.52851755,  0.91486132]]), array([[
0.84337207,  0.71184583,  1.17438049,  0.05534304,  1.54852091,
          1.11991208, -0.20755805, -0.06514575],
        [-0.29546146, -3.91601852,  1.14161077,  1.79086917,  0.02368126,
          2.41315488, -1.66800099, -2.9197601 ],
        [ 0.36178874,  0.75764029, -0.12003543,  4.08003081,  1.07627096,
         -0.42183308,  0.03729542,  0.43111418],
        [ 0.10999841,  1.07416943,  0.43932684,  0.96918334,  0.33956177,
          0.99951819,  1.84702767,  1.52769075],
        [ 0.97762635,  0.40310218,  1.62922104, -0.04762848,  1.71221509,
          0.72607699, -0.69884135,  0.4796053 ],
        [ 0.89958363,  0.98412796,  0.52161205,  0.91275047,  0.97661725,
          0.56039433,  0.92641737,  0.42629405],
        [ 0.0591667 ,  0.79405981,  1.22356081,  0.47527233, -0.03321753,
          1.41650241,  2.51958086,  1.85130573],
        [ 0.68283496,  1.08503147,  0.80007698,  0.78911094, -0.00942464,
          0.70234232,  1.14819764,  1.60436768]]), array([[  3.07611392,  -4.1246045
,   0.5398679 ,  -2.16162588,
           3.95298432,   0.29289907,  -0.69189632,  -1.38395254],
        [ -1.57670439,   4.38721229,   6.22188281,   0.97894911,
          -0.67458864,  -0.65600393,  -2.08383457,   1.17311899],
        [ -0.20471148,  14.36212038,  -2.89875181,   2.55840916,
          -0.92027299,   1.67489516,   2.36446305,   0.91838591],
```

```
        [  0.32053133, -12.63933498,  -1.41066855,   0.77112545,
           0.78022794,   0.58871562,   2.24292112,   1.91471812]])]
b= [array([[ 0.01370861],
       [ 0.11091874],
       [ 0.29006856],
       [ 0.114527  ],
       [ 1.6735113 ],
       [ 0.47127391],
       [-0.05015931],
       [ 1.52611757],
       [ 1.07199713],
       [ 1.20255323]]), array([[ 0.5066008 ],
       [-0.53020795],
       [ 0.34216388],
       [ 0.4267884 ],
       [ 0.78090652],
       [ 0.79474943],
       [ 0.59000713],
       [ 0.19893868]]), array([[ 1.20831821],
       [-0.31516574],
       [ 0.2483067 ],
       [ 0.50686937],
       [ 0.73536755],
       [ 0.37881852],
       [ 0.36647878],
       [ 0.36117497]]), array([[ 3.82002699],
       [ 4.18475279],
       [-2.49830368],
       [-3.62071698]])]
```

使用torch计算的W和b:

```
linear1.weight [[-4.71015126e-01  2.65191734e-01 -4.18528169e-02  5.90032101e-01
   3.81774902e-01 -4.61170003e-02 -2.36282513e-01 -1.05075993e-01
  -7.86260366e-01  4.83463667e-02]
 [ 9.65878606e-01  3.28795403e-01  6.74928963e-01 -7.88597241e-02
   3.12125944e-02  6.77228153e-01 -4.33786988e-01  6.92761958e-01
  -7.30766207e-02 -1.75722279e-02]
 [-3.04631084e-01 -3.78132910e-01 -3.24442834e-02  8.10060084e-01
   8.76509845e-01 -2.02478051e+00 -1.06442153e+00 -1.79129988e-01
  -8.34368825e-01  4.94334847e-03]
 [-1.43723559e+00  7.08891213e-01  1.54629993e+00  4.99210268e-01
   1.38933337e+00 -6.58914149e-01  1.48112476e-01 -4.40060794e-01
  -2.00878590e-01 -1.34488776e-01]
 [ 3.40323001e-01 -6.78263009e-01 -5.36242366e-01 -5.05218565e-01
  -1.29067910e+00 -6.39362574e-01 -1.48254299e+00 -6.49205625e-01
   7.14957267e-02 -5.48044682e-01]
 [ 7.54054785e-02  9.18482006e-01  2.54570663e-01  2.21643090e-01
  -1.10328233e+00 -8.95163059e-01 -1.67743444e+00  4.15707201e-01
  -1.46862388e-01  1.34367794e-01]
 [ 1.31967580e+00 -2.20287666e-01 -4.47369486e-01 -2.55241543e-01
   3.68642420e-01 -6.52916580e-02 -1.36631715e+00  3.67226064e-01
  -1.25444460e+00 -1.10952151e+00]
 [-3.35666150e-01 -3.72012436e-01 -1.26509106e+00 -7.96916306e-01
   5.02844865e-04 -8.99484038e-01  1.94330290e-01  3.86587262e-01
  -3.28512043e-01  3.84798795e-01]
 [-1.36727107e+00  1.01906610e+00 -1.92551374e-01  3.59988809e-01
```

```
  -9.94516134e-01  7.81070709e-01  1.17583975e-01 -3.76541942e-01
   3.40462893e-01  8.48311722e-01]
 [ 1.01452208e+00  1.22737491e+00  1.19023478e+00  2.83167720e-01
   4.07629430e-01 -1.03637047e-01 -1.32965237e-01 -4.60893996e-02
  -9.00648475e-01 -9.19488251e-01]]
linear1.bias [ 0.9749167  1.2441475 -0.2166186 -0.7646365 -0.9060197  1.0948697
 -1.5677439  1.2046119  0.4200823  0.5289687]
linear2.weight [[-6.0943508e-01 -1.2230036e+00  8.8500774e-01 -1.2580427e+00
  -1.5776237e+00 -9.2446697e-01  1.0532753e+00 -6.5842062e-01
   8.3207738e-01  7.9200077e-01]
 [-9.0538579e-01  1.2400746e+00  1.4476140e-01  1.2164176e+00
   1.2552477e+00 -1.4602627e+00  3.9060894e-01  1.1234578e+00
   6.9544268e-01  9.2228556e-01]
 [-1.3645147e+00 -1.0772686e+00  1.0839182e+00 -1.2595177e+00
  -1.1268102e+00 -6.3848102e-01 -5.6008697e-01 -1.9121303e-01
   1.0933526e+00  9.4027120e-01]
 [ 1.1415831e+00  8.7381637e-01  9.2624813e-01 -4.5570457e-01
   4.9065530e-01 -2.3129840e-01 -8.3169931e-01  1.8311458e-03
   8.8822746e-01  6.6555060e-02]
 [-1.3807741e+00  1.0996660e+00  1.1575360e+00  3.8319147e-01
   2.9000494e-01 -1.6350745e+00  1.6251901e+00  1.0610802e+00
   2.5430856e+00  1.4570408e+00]
 [-4.7261801e-01 -5.7868290e-01  1.1505367e+00 -1.2629045e+00
  -1.4550540e+00 -9.4418007e-01  4.7250748e-01 -8.0539834e-01
   5.8276790e-01  5.1983070e-01]
 [ 1.9520959e-01  4.4508851e-01  8.6172682e-01  3.5854289e-01
   1.0513830e+00 -9.4249606e-01  1.4327582e+00  8.8830042e-01
   1.6688975e+00  1.6677628e+00]
 [-8.0421185e-01 -8.0613673e-01 -9.7434527e-01 -4.3396458e-01
  -1.2374791e+00  4.3478016e-02  1.2074839e+00 -6.6033119e-01
  -3.1253067e-01  1.0842165e+00]]
linear2.bias [ 1.0557315   0.74418414 -0.16324405 -0.27153307  0.21204545  0.7913995
 -0.61448735 -0.27657905]
linear3.weight [[-1.3302358  -0.778524   -1.2366936   0.85264236 -2.4871182
-1.3375276
  -1.6101077  -0.53660357]
 [-0.8417221  -1.6917431  -0.3299065  -0.68429846 -2.2865124  -0.6032592
  -1.7019215   1.2277589 ]
 [ 1.8769251  -1.1795936   1.0650213  -0.48570085  0.5232472   1.5097286
   0.47733638  0.9358809 ]
 -1.0040358   0.07231607]
linear4.weight [[ 1.7953438  -2.6233175  -2.14499     1.4496497  -1.5456789
-1.2853931
  -3.7178345  -3.3545709 ]
 [-0.86470586  2.6493018  -2.5856836  -1.2085027  -3.2651029  -1.5189937
   2.7812893   3.2141685 ]
 [-2.5542252  -2.1165297   2.4800398  -3.029333    1.1623169   2.6404927
  -1.0268372  -2.1703873 ]
 [ 2.2230625   2.299754    3.0733795   2.1297674   4.386307    2.6943917
  -0.67796797  1.8776635 ]]
linear4.bias [-4.2257752 -1.8943214 -0.6699339 -3.28821   ]
```

通过比较手写MLP和torch实现的MLP，可发现W和b最终结果有很大差别，故可得知神经网络的W和b并不唯一

实现前向传播：

```
def forward(self, x):
        # 前向传播
        # 每一层的layer均为列向量
        self.layers[0] = x.reshape(-1,1)

        for i in range(1,len(self.layers)-1):
            self.layers[i] = self.activation(np.dot(self.weights[i-1],self.layers[i-
1]) + self.biases[i-1])

        k = len(self.layers)-1
        self.layers[k] = softmax(np.dot(self.weights[k-1],self.layers[k-1]) +
self.biases[k-1])
```

实现反向传播和梯度下降：

```
    def backward(self,y): # 自行确定参数表
        # 反向传播
        y_predict = self.layers[-1]
        # pre_gradient_neurons为列向量
        pre_gradient_neurons = y_predict - y.reshape(-1,1)
        gradientWeightList = []
        gradientBiasList = []
        # 从最后一层到第一层进行遍历
        for i in range(len(self.layers)-1,0,-1):

            # gradient_bias为列向量
            gradient_bias = pre_gradient_neurons
            # (-1,1)为列向量，(1,-1)为行向量
            gradient_weight = np.dot(gradient_bias,self.layers[i-1].reshape(1,-1))
            gradientWeightList.append(gradient_weight)
            gradientBiasList.append(gradient_bias)

            if(i>1):
                s_ = 1 - np.power(self.layers[i-1],2)
                pre_gradient_neurons = np.dot(self.weights[i-
1].T,pre_gradient_neurons ) * s_
        j=0
        for i in range(len(self.layers)-1,0,-1):
            self.weights[i-1] -= self.lr * gradientWeightList[j]
            self.biases[i-1] -= self.lr * gradientBiasList[j]
            j += 1
```

这里是将所有W和b的梯度计算出之后，再统一进行梯度下降

训练模型

```
def train(mlp: MLP, epochs,  inputs, labels):
    '''
        mlp: 传入实例化的MLP模型
        epochs: 训练轮数
        lr: 学习率，在mlp中已规定
        inputs: 生成的随机数据
        labels: 生成的one-hot标签
    '''
```

```python
    MyMlpLoss = []
    Index = []
    for j in range(epochs):
        print('{}th training in progress'.format(j+1))
        loss = 0
        for i in range(len(inputs)):
            #print('i=',i)
            mlp.forward(inputs[i])
            mlp.backward(labels[i])

        for i in range(len(inputs)):
            mlp.forward(inputs[i])
            t = labels[i].tolist().index(1)
            loss += -math.log((mlp.layers[-1].flatten())[t])
        loss /= len(inputs)
        Index.append(j+1)
        MyMlpLoss.append(loss)


    return MyMlpLoss,Index
```

实现一个卷积网络

该实验主要是参考**官方实验文档**的代码，本人学号为PB19051035，选择了第列表中第三个模型实现。最终实现结果为：

```
Train Epoch: 0/5 [0/50000]      Loss: 2.304314
Train Epoch: 0/5 [12800/50000]  Loss: 2.017579
Train Epoch: 0/5 [25600/50000]  Loss: 1.638553
Train Epoch: 0/5 [38400/50000]  Loss: 1.831659
Train Epoch: 1/5 [0/50000]      Loss: 1.585045
Train Epoch: 1/5 [12800/50000]  Loss: 1.490350
Train Epoch: 1/5 [25600/50000]  Loss: 1.551887
Train Epoch: 1/5 [38400/50000]  Loss: 1.536392
Train Epoch: 2/5 [0/50000]      Loss: 1.588516
Train Epoch: 2/5 [12800/50000]  Loss: 1.564009
Train Epoch: 2/5 [25600/50000]  Loss: 1.613194
Train Epoch: 2/5 [38400/50000]  Loss: 1.355198
Train Epoch: 3/5 [0/50000]      Loss: 1.491654
Train Epoch: 3/5 [12800/50000]  Loss: 1.550323
Train Epoch: 3/5 [25600/50000]  Loss: 1.489787
Train Epoch: 3/5 [38400/50000]  Loss: 1.493528
Train Epoch: 4/5 [0/50000]      Loss: 1.387696
Train Epoch: 4/5 [12800/50000]  Loss: 1.637470
Train Epoch: 4/5 [25600/50000]  Loss: 1.379541
Train Epoch: 4/5 [38400/50000]  Loss: 1.612634
Finished Training
Test set: Average loss: 1.4727   Acc 0.50
```

- MyNet的卷积层、池化层和全连接层

```python
        # 输入为3*32*32
        self.conv1 = nn.Conv2d(kernel_size=5,in_channels=3,out_channels=16)
```

```python
        # 输入为16*28*28
        self.pool = nn.AvgPool2d(kernel_size=2)
        # 输入为16*14*14
        self.conv2 = nn.Conv2d(kernel_size=5,in_channels=16,out_channels=32)
        # 输入为32*10*10,然后再次进入池化层

        # 输入为32*5*5
        self.linear1=nn.Linear(in_features=32*5*5,out_features=120)
        # 输入为120
        self.linear2=nn.Linear(in_features=120,out_features=84)
        # 输入为84
        self.linear3 = nn.Linear(in_features=84,out_features=10)
```

- 前向传播

```python
def forward(self, x):
        x=self.pool(F.relu(self.conv1(x)))
        x=self.pool(F.relu(self.conv2(x)))
        x=x.view(-1,32*5*5)
        x=F.relu(self.linear1(x))
        x=F.relu(self.linear2(x))
        x=F.relu(self.linear3(x))
        return x
```

- 训练

```python
        #计算loss并进行反向传播
        predict = net(inputs)
        optimizer.zero_grad()
        loss = loss_function(predict,labels)
        loss.backward()
        optimizer.step()
```

- 测试

```python
        #需要计算测试集的loss和accuracy
        outputs=net(inputs)
        test_loss += loss_function(outputs,labels)
        accuracy=torch.sum(torch.argmax(outputs, dim=1) == labels)/len(labels)
```

- 优化器和损失函数

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