EEE418: Advanced Pattern Recognition **Spring 2020**

Lab 5: Neural Network

Student: Zhiyu Xu *ID: 1926906*

5.1 Estimation of Classification Methods

1. Read the dataset and shuffle it

Table 5.1: Attribute Information: (class attribute has been moved to last column)

Attribute	Domain
Sample code number	1 - 10
Clump Thickness	1 - 10
Uniformity of Cell Size	1 - 10
Uniformity of Cell Shape	1 - 10
Marginal Adhesion	1 - 10
Single Epithelial Cell Size	1 - 10
Bare Nuclei	1 - 10
Bland Chromatin	1 - 10
Normal Nucleoli	1 - 10
Mitoses	1 - 10
Class	**2 for benign, 4 for malignant**

In order to make the operation and display beautiful, the Pandas library is used here, the data is read as a Dataframe file, and the column name is added.

Code as shown below:

```
columnNames = [
        'Sample code number',
        'Clump Thickness',
        'Uniformity of Cell Size',
        'Uniformity of Cell Shape',
        'Marginal Adhesion',
        'Single Epithelial Cell Size',
        'Bare Nuclei',
        'Bland Chromatin',
        'Normal Nucleoli',
        'Mitoses',
        'Class'
]
data = pd.read_csv('breast-cancer-wisconsin.data', names = columnNames)
print(data.shape)
data.head(10)
```

Result:

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	1000025	5	1	1	1	2	1	3	1	1	2
1	1002945	5	4	4	5	7	10	3	2	1	2
2	1015425	3	1	1	1	2	2	3	1	1	2
3	1016277	6	8	8	1	3	4	3	7	1	2
4	1017023	4	1	1	3	2	1	3	1	1	2
5	1017122	8	10	10	8	7	10	9	7	1	4
6	1018099	1	1	1	1	2	10	3	1	1	2
7	1018561	2	1	2	1	2	1	3	1	1	2
8	1033078	2	1	1	1	2	1	1	1	5	2
9	1033078	4	2	1	1	2	1	2	1	1	2

Figure 5.1: Top 10 of breast-cancer-wisconsin.data

2. Remove missing attribute values
   According to the documentation: There are 16 instances in Groups 1 to 6 that contain a single missing (i.e., unavailable) attribute value, now denoted by "?".

   Code as shown below:

```
#Replace ? with NaN
data = data.replace(to_replace='?',value = np.nan)
#Discard data with missing values (discard as long as one dimension is missing)
data = data.dropna(how='any')
data.shape
```

   Result:

```
(683, 11)
```

Figure 5.2: Size of new data

   Because there are 16 missing values, missing values where the row has been deleted, the data from the (699,11) into the (683,11)

3. Suffle dataset
   random.seed = 17

   Code as shown below:

```
data_shuffled = shuffle(data,random_state = 17).reset_index(drop=True)
data_shuffled.head(10)
```

Result:

	Sample code number	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Class
0	1231853	4	2	2	1	2	1	2	1	1	2
1	1304595	3	1	1	1	1	1	2	1	1	2
2	1083817	3	1	1	1	2	1	2	1	1	2
3	1175937	5	4	6	7	9	7	8	10	1	4
4	188336	5	3	2	8	5	10	8	1	2	4
5	1080233	7	6	6	3	2	10	7	1	1	4
6	1232225	10	4	5	5	5	10	4	1	1	4
7	1230175	10	10	10	3	10	10	9	10	1	4
8	1212422	3	1	1	1	2	1	3	1	1	2
9	1243256	10	4	3	2	3	10	5	3	2	4

Figure 5.3: Shuffled data

4. Split the dataset as five parts
   Each of 5 subsets was used as test set and the remaining data was used for training. Five subsets were used for testing rotationally to evaluate the classification accuracy

   Code as shown below:

```
from sklearn.model_selection import KFold
kf = KFold(n_splits=5)
i = 0
data_train_index,data_test_index = [],[]
for train_index, test_index in kf.split(data):
        i +=1
        data_train_index.append(train_index)
        data_test_index.append(test_index)
        print('No.%i train:%i test:%i'%(i,len(data_train_index[i-1]),len(
            data_test_index[i-1])))
print(data_train_index[0])
```

Result:

```
No. 1 train: 546 test: 137
No. 2 train: 546 test: 137
No. 3 train: 546 test: 137
No. 4 train: 547 test: 136
No. 5 train: 547 test: 136
[137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154
 155 156 157 158 159 160 161 162 163 164 165 166 167 168 169 170 171 172
 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190
 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208
 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226
 227 228 229 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244
 245 246 247 248 249 250 251 252 253 254 255 256 257 258 259 260 261 262
 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280
 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298
 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316
 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334
 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352
 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370
 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388
 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406
 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424
 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442
 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460
 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478
 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496
 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514
 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532
 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550
 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568
 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586
 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604
 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622
 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640
 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658
 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676
 677 678 679 680 681 682]
```

Figure 5.4: Splitted result and first train data index

5.2 MLP Algorithm

1. All input feature vectors are augmented with the 1 as follows:

$$\hat{X} = [X \quad 1_{N\times 1}]$$

Code as shown below:

```
data, label =data_shuffled[columnNames[1:10]], data_shuffled[columnNames[10]]
data['Extra features'] = 1
data.head(10)
```

Result:

	Clump Thickness	Uniformity of Cell Size	Uniformity of Cell Shape	Marginal Adhesion	Single Epithelial Cell Size	Bare Nuclei	Bland Chromatin	Normal Nucleoli	Mitoses	Extra features
0	4	2	2	1	2	1	2	1	1	1
1	3	1	1	1	1	1	2	1	1	1
2	3	1	1	1	2	1	2	1	1	1
3	5	4	6	7	9	7	8	10	1	1
4	5	3	2	8	5	10	8	1	2	1
5	7	6	6	3	2	10	7	1	1	1
6	10	4	5	5	5	10	4	1	1	1
7	10	10	10	3	10	10	9	10	1	1
8	3	1	1	1	2	1	3	1	1	1
9	10	4	3	2	3	10	5	3	2	1

Figure 5.5: New train data

In this step, I first extracted 9 original features as the data set, class as the label set, and then added an additional feature constant 1 at the end of the data set.

2. Scale linearly the attribute values x_{ij} of the data matrix $\hat{X}$ into $[-1, 1]$ for each dimensional feature as follows:

$$x_{ij} \leftarrow 2\frac{x_{ij} - min_i x_{ij} + 10^{-6}}{max_i - min_i x_{ij} + 10^{-6}} - 1$$

Function code as shown below:

```
def Scale(data):
        arr =  np.array(data)
        new_data = []
        for i in range(len(arr)):
                tmp = []
                for j in range(len(arr[i])):
                tmp.append(2*((arr[i][j]-np.min(arr[i])+1E-6)/(np.max(arr[i])-np.
                    min(arr[i])+1E-6))-1)
        new_data.append(tmp)
        return new_data
```

```
arr = Scale(data_list)
for i in range(5):
        print(data_list[i])
        print(arr[i])
```

Result:

```
[4, 2, 2, 1, 2, 1, 2, 1, 1, 1]
[1.0, -0.3333328888890371, -0.3333328888890371, -0.9999993333335555, -0.3333328888890371, -0.9999993333335555, -0.3333328888890371, -0.9999993333335555, -0.3333328888890371, -0.9999993333335555]
[3, 1, 1, 1, 1, 1, 2, 1, 1, 1]
[1.0, -0.9999990000005, -0.9999990000005, -0.9999990000005, -0.9999990000005, -0.9999990000005, 4.999997498256192e-07, -0.9999990000005, -0.9999990000005, -0.9999990000005]
[3, 1, 1, 1, 2, 1, 2, 1, 1, 1]
[1.0, -0.9999990000005, -0.9999990000005, -0.9999990000005, 4.999997498256192e-07, -0.9999990000005, 4.999997498256192e-07, -0.9999990000005, -0.9999990000005]
[5, 4, 6, 7, 9, 7, 8, 10, 1, 1]
[-0.11111098765433458, -0.3333331851852016, 0.11111120987653234, 0.33333340740739925, 0.7777778024691331, 0.33333340740739925, 0.5555556049382662, 1.0, -0.9999997777778025, -0.9999997777778025]
[5, 3, 2, 8, 5, 10, 8, 1, 2, 1]
[-0.11111098765433458, -0.5555553827160685, -0.7777775802469356, 0.5555556049382662, -0.11111098765433458, 1.0, 0.5555556049382662, -0.9999997777778025, -0.7777775802469356, -0.9999997777778025]
```

Figure 5.6: Scale result

It can be seen from the above results that the function can change the maximum value in a set of numbers to 1, and the remaining values to values close to -1.But when the formula given by the teacher was used at the beginning, the result was not good, and the data spread to both ends (+1 and -1), so the formula was modified as follows:

$$x_{ij} \leftarrow 2\frac{x_{ij} - min_i + 10^{-6}}{max_i - min_i + 10^{-6}} - 1$$

3. The label l_n of the n-th example is converted into a K dimensional vector t_n as follows(K is the number of the classes)

$$t_n = \begin{cases} +1, & k = l_n \\ 0, & k \neq l_n \end{cases}$$

In this case, data havs two labels and 683 examples, so size of t_{nk} should be (683,2)
Code as shown below:

```
def Onehot(label,num):
        length = len(label)
        t_nk = np.zeros((length,num),dtype='int')
        for i in range(length):
                if label[i] == 2:
                t_nk[i][0] = 1
        elif label[i] == 4:
                t_nk[i][1] = 1
        return t_nk
```

```
t_nk = Onehot(label_list,2)
for i in range(10):
        print(f'Orignal label is {label[i]}, t_{i+1}k is {t_nk[i]}')
```

Result:

```
Orignal label is 2, t_1k is [1 0]
Orignal label is 2, t_2k is [1 0]
Orignal label is 2, t_3k is [1 0]
Orignal label is 4, t_4k is [0 1]
Orignal label is 4, t_5k is [0 1]
Orignal label is 4, t_6k is [0 1]
Orignal label is 4, t_7k is [0 1]
Orignal label is 4, t_8k is [0 1]
Orignal label is 2, t_9k is [1 0]
Orignal label is 4, t_10k is [0 1]
```

Figure 5.7: Onehot function test result

4. Initialize all weight w_{ij} of MLP network

$$\omega \leftarrow \left(X^T X\right)^{-1} X^T 1$$

Such as $w_{ij} \in \left[-\sqrt{\frac{6}{D+1+K}}, \sqrt{\frac{6}{D+1+K}}\right]$ where D and K is the number of the input nodes and the output nodes (each node is related to a class), respectively.
In this case, D = 10 , K = 2, so $w_{ij} \in \left[-\sqrt{\frac{6}{13}}, \sqrt{\frac{6}{13}}\right]$ Code as shown below:

```
def get_w(size, D, K, seed = 666):
        np.random.seed(seed)
        w = np.random.random(size)   # w in  [0,1]
        w = 2*w -1 # w in [-1,1]
        w = w * np.sqrt(6/(D+K+1)) # w in [-sqrt(6/(D+K+1)),sqrt(6/(D+K+1))]
        return w
print(get_w([10,2],10,2,seed = 10))
print(get_w([10,2],10,2,seed = 10).shape)
```

Result:

```
[[ 0.36865216 -0.65116987]
 [ 0.18159219  0.33805791]
 [-0.00202857 -0.37392773]
 [-0.41025178  0.35399153]
 [-0.44958984 -0.55933605]
 [ 0.2518544   0.61604025]
 [-0.67400158  0.01656602]
 [ 0.42476824  0.15289282]
 [ 0.30130614 -0.28278474]
 [ 0.56764325  0.29155108]]
(10, 2)
```

Figure 5.8: weight vector ω_{ij} and its shape

5. Choose randomly an input vector x to network and forward propagate through the network (H is the

number of the hidden units)

$$a_j = \sum_{i=0}^{D} \omega_{ij}^{(1)} x_i$$
$$z_j = tanh(a_j)$$
$$y_k = \sum_{j=0}^{H} \omega_{ij}^{(2)} z_j$$

According to the above formulas, the following structure can be obtained:

input layer: (N, 10)
$\omega_{ij}^{(1)}$: (10, H)
hidden layer: (N, H)
$\omega_{kj}^{(2)}$: (H, 2)
output layer: (N, 2)

Where N is the number of input samples and H is the number of hidden layers. In order to save the calculation time, the numpy matrix operation is used here
Code as shown below:

```
def twolayer(Input, w1, w2):
        Input = np.array(Input)
        a1 = Input @ w1
        z1 = np.tanh(a1)
        yk = z1 @ w2
        return yk
```

```
w1 = get_w([10, 50],10,2)
w2 = get_w([50, 2 ],10,2)
ans = twolayer(data_list, w1, w2)
print(ans,'\n',ans.shape)
```

Result:

```
[[ 0.28205616 -2.68709007]
 [ 1.29892027 -1.68803515]
 [ 0.73945765 -2.11643316]
 ...
 [ 0.24719097 -3.06454142]
 [ 4.94216549  0.95316127]
 [ 2.75443411 -0.57151883]]
 (683, 2)
```

Figure 5.9: Predicted label

In this step, I set the hidden layer to 50 and used 683 data for frame testing. The result after one training is as above. Since the weights are set randomly, it can be seen that the value of the result is also random, but the size of the result is correct, which shows that the overall frame structure is correct

6. Evaluate the δ_k for all output units

$$\delta_k = y_k - t_k$$

Code as shown below:

```
yk = ans
delta_k  = yk - t_nk
print(delta_k,'\n',delta_k.shape)
```

Result:

```
[[-0.71794384 -2.68709007]
 [ 0.29892027 -1.68803515]
 [-0.26054235 -2.11643316]
 ...
 [-0.75280903 -3.06454142]
 [ 3.94216549  0.95316127]
 [ 1.75443411 -0.57151883]]
(683, 2)
```

Figure 5.10: Evaluation of all output units

7. Backpropagate the δ'_s to obtain δ_j for each hidden unit in the network

$$\begin{aligned}\delta_j &= tanh(a_j)' \sum_{k=1}^{K} w_{ij}\delta_k \\ &= (1 - z_j^2) \sum_{k=1}^{K} w_{ij}\delta_k\end{aligned}$$

Code as shown below:

```
z1 = np.tanh(Input @ w1)
delta_j = (1 - z1 ** 2)*(delta_k @ w2.T)
```

8. The derivative with respect to the first-layer and the second-layer weights are given by

$$\frac{\partial E}{\partial \omega_{ij}^{(1)}} = \delta_j x_i, \frac{\partial E}{\partial \omega_{kj}^{(2)}} = \delta_k z_j$$

And we shoule update the weights of the network:

$$w_{kj} = w_{kj} - \eta \frac{\partial E}{\partial \omega_{ij}^{(1)}}, w_{ji} = w_{ji} - \eta \frac{\partial E}{\partial \omega_{kj}^{(2)}}$$

Code as shown below:

```
w1 = w1 - Input.T  @ delta_j
print(w1.shape)
w2 = w2 - z1.T @ delta_k
print(w2.shape)
```

9. Training phase
   Integrate the above framework into the training function, the function code is as follows:

```
def SBA(data, label, w1, w2, lr, steps):
        for step in range(steps):
                data = shuffle(np.array(Scale(data)),random_state =step)
                label= shuffle(np.array(label),random_state =step)
                a1 = data @ w1
                z1 = np.tanh(a1)
                yk = z1 @ w2
                delta_k  = yk - label
                delta_j = (1 - (z1 ** 2))*(delta_k @ w2.T)
                w1 -= lr * (data.T  @ delta_j)
                w2 -= lr * (z1.T @ delta_k)
        return w1, w2
```

   The accuracy calculation function is as follows:

```
def Acc(data, label, w1, w2):
        a1 = np.array(Scale(data)) @ w1
        z1 = np.tanh(a1)
        yk = z1 @ w2
        return np.mean(np.equal(np.argmax(yk,1),np.argmax(Onehot(label,2),1)))
```

   First, we tested the accuracy of the hidden layer from 5 to 50 with a step size of 5 and training times of 100, and visualized the result. The code is as follows:

```
train_Acc_mean = []
test_Acc_mean = []
for h in range(10):
        H = 5 * (h + 1)
        train_Acc = []
        test_Acc = []
        for i in range(5):
                # Initialize w and learning rate
                w1 = get_w([10, H],10,2)
                w2 = get_w([H, 2 ],10,2)
                lr = 0.001
                #training
                w1, w2 = SBA(data_train[i], Onehot(label_train[i],1), w1, w2, lr,
                    100)
                train_Acc.append(Acc(data_train[i], label_train[i], w1, w2))
                test_Acc.append(Acc(data_test[i], label_test[i], w1, w2))
        print(f'The number of hidden layers is{H}')
        print(f'Training dataset accuracy: {np.mean(np.array(train_Acc))},Testing
            dataset accuracy: {np.mean(np.array(test_Acc))}')
        train_Acc_mean.append(np.mean(np.array(train_Acc)))
        test_Acc_mean.append(np.mean(np.array(test_Acc)))

import matplotlib.pyplot as plt
x = np.linspace(5,50,10)
plt.figure(figsize= (16,9))
plt.plot(x,train_Acc_mean,label='train')
plt.plot(x,test_Acc_mean,label='test')
plt.xlabel('Number of hidden layer')
```

```
27 plt.ylabel('Accuracy')
28 plt.legend()
```

```
The number of hidden layers is5
Training dataset accuracy: 0.9396026277196295,Testing dataset accuracy: 0.9355624731644483
The number of hidden layers is10
Training dataset accuracy: 0.9344791101646678,Testing dataset accuracy: 0.9282417346500644
The number of hidden layers is15
Training dataset accuracy: 0.9344791101646678,Testing dataset accuracy: 0.9209424645770717
The number of hidden layers is20
Training dataset accuracy: 0.5220021294975591,Testing dataset accuracy: 0.5327179046801203
The number of hidden layers is25
Training dataset accuracy: 0.5302864107251676,Testing dataset accuracy: 0.5143301846285959
The number of hidden layers is30
Training dataset accuracy: 0.27745344235289393,Testing dataset accuracy: 0.2841133533705453
The number of hidden layers is35
Training dataset accuracy: 0.4655603993812402,Testing dataset accuracy: 0.4613031343924431
The number of hidden layers is40
Training dataset accuracy: 0.27745746027281676,Testing dataset accuracy: 0.28262129669386005
The number of hidden layers is45
Training dataset accuracy: 0.29247309667784993,Testing dataset accuracy: 0.30015027908973807
The number of hidden layers is50
Training dataset accuracy: 0.2902712765601248,Testing dataset accuracy: 0.28262129669386005
```

Figure 5.11: Result after training(100 times)

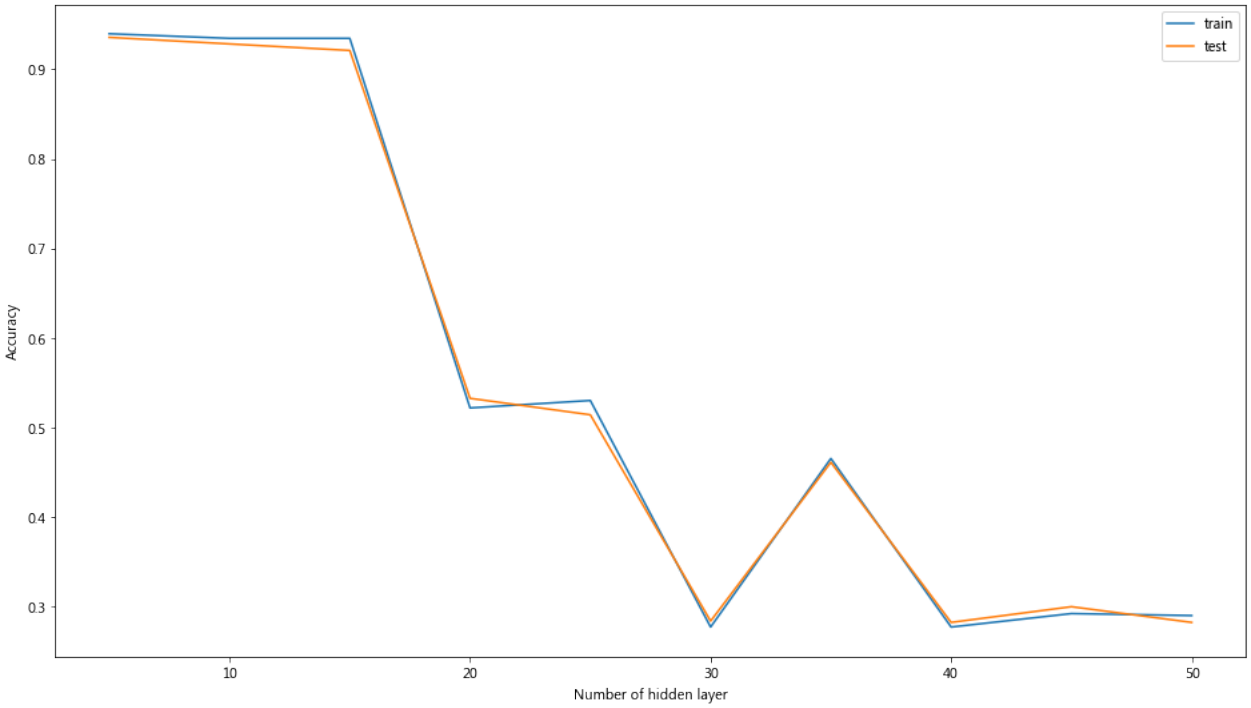

Figure 5.12: Result after training(100 times)

5.3 Conclusion

- It can be seen from Figure 5.12 that when the number of hidden layers is low, the accuracy of the training set and the test set is relatively high. And there is no overfitting or underfitting. The actual meaning of the hidden layer is to expand or compress the characteristics of the original data. For the data set selected in this experiment, the compressed feature number performance is better. But this is only the result of training 100 times. What happens if the number of training increases?
  Adjust the training times to 200, the results are as follows:

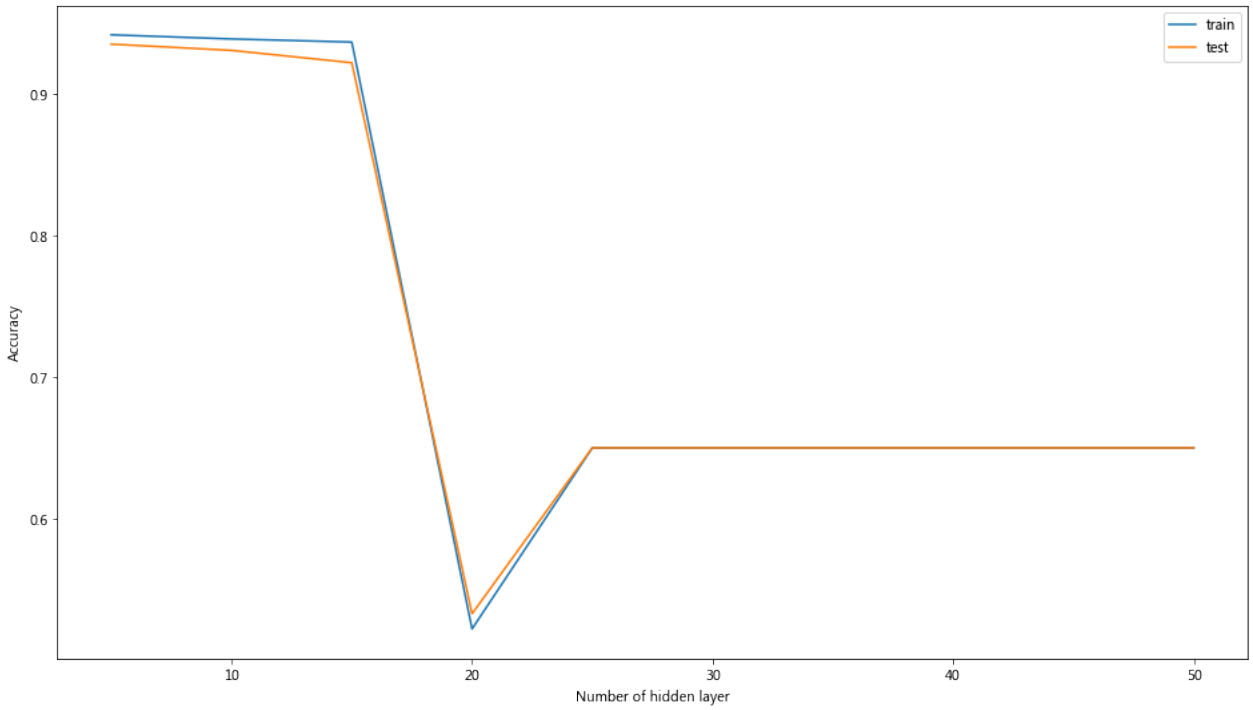

Figure 5.13: Result after training(200 times)

It turns out that the results did not improve very much, which is also determined by the data set itself. It may be that when the classification is performed, individual features account for a large amount, so expanding the features does not play a good effect, but adds a lot of calculation.

- In order to solve the above-mentioned problem of excessive calculation, we can use the GPU for calculation, which can save a lot of time. Here I used Tensorflow-GPU==13.1.0 for training(GPU:GTX 960m, cuda = 9.2.0), the code is as follows:

```
x = tf.placeholder(tf.float64, [None, 10], name="input_x")
label = tf.placeholder(tf.float64, [None, 2], name="input_y")

W1 = tf.Variable((np.random.random([10, 50])*2 - 1)*np.sqrt(6/(13)))
W2 = tf.Variable((np.random.random([50, 2])*2 - 1)*np.sqrt(6/(13)))

h = tf.matmul(x, W1)
h = tf.nn.tanh(h)
y = tf.matmul(h, W2)
```

```
10
11 loss = tf.reduce_mean(tf.square(y-label))
12 train_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)
13 correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(label, 1))
14 accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
15
16 for i in range(5):
17         sess = tf.Session()
18         sess.run(tf.global_variables_initializer())
19         Acc = []
20         for itr in range(2001):
21                 sess.run(train_step, feed_dict={x: data_train[i], label:
                       label_train[i]})
22                 if itr % 200 == 0:
23                         print("group%1d"%(i+1),"step:%6d  train accuracy:"%itr,
                               sess.run(accuracy, feed_dict={x: data_train[i],
24                             label: label_train[i]}),
25                             "test accuracy:",sess.run(accuracy, feed_dict={x:
                               data_test[i], label: label_test[i]}))
26         Acc.append(sess.run(accuracy, feed_dict={x: data_test[i],
27         label: label_test[i]}))
```

A total of 2001 trainings were performed. The average accuracy of the final training set was 97.03% and the accuracy of the test set was 94.35%. The overall accuracy rate is relatively high, this is due to the increase in training times, but there is an overfitting situation, this is because the number of hidden layers is relatively high, you can use the Dropout function to eliminate overfitting