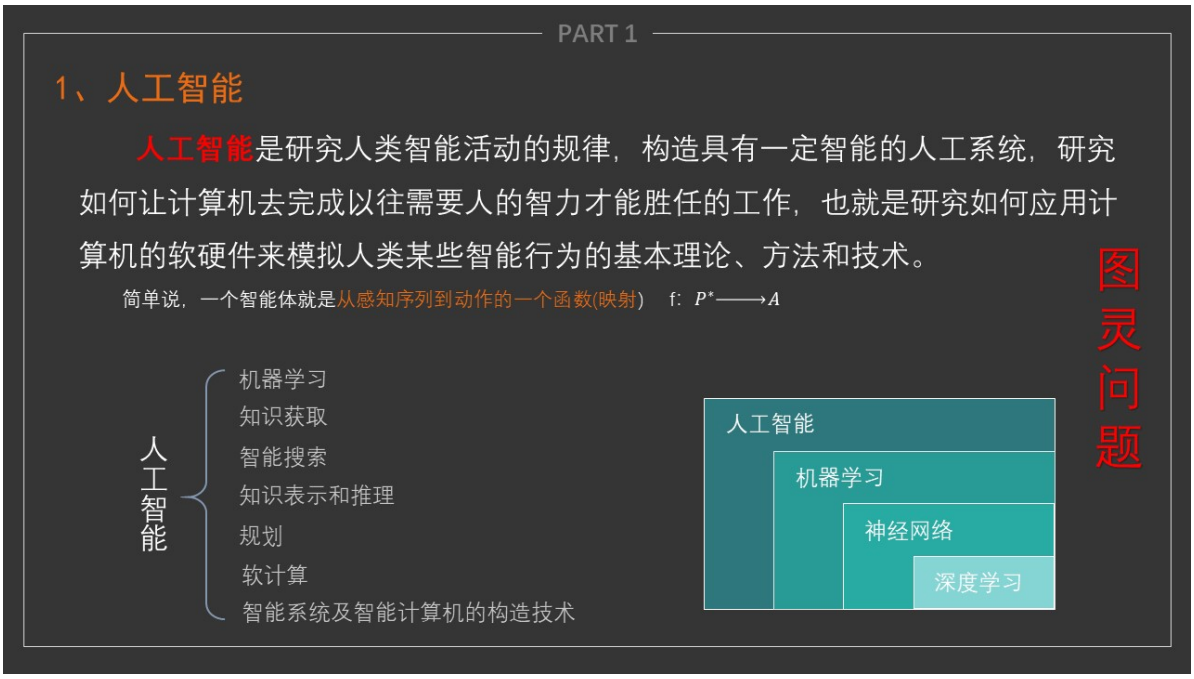
机器视觉精品课程-神经网络部分

授课内容

- **神经网络基础：基础概念：损失函数、神经元结构、多层感知机，手写数字识别**



PART 1

2、机器学习

官方定义：

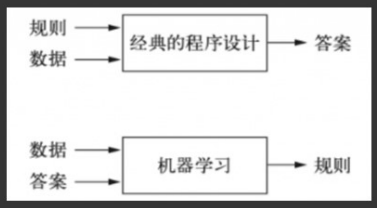
让计算机能够像人一样自动的获取新知识，并在实践中不断的完善自我和增强能力.

围绕三方面展开：

1）面向任务的研究：研究和分析改进一组预定任务执行性能的学习系统。

2）模型研究：研究人类学习过程并进行计算机模拟。

3）理论分析：从理论上探索各种可能的学习方法和独立于应用领域的算法。



最简单的例子：最小二乘法

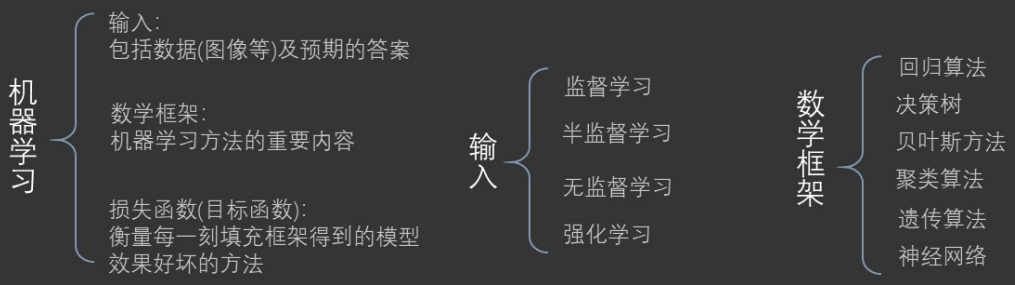
通过最小化误差的平方和寻找数据的最佳函数匹配。利用最小二乘法可以简便地求得未知的数据，并使得这些求得的数据与实际数据之间误差的平方和为最小。最小二乘法还可用于曲线拟合。其他一些优化问题也可通过最小化能量或最大化熵用最小二乘法来表达。

损失函数

PART 1

2、机器学习

数学框架：机器学习需要通过训练自己获得从输入得到输出的规则，但想要计算机彻底从无到有实现这一过程是不实际的，所以还需提供一个数学框架给计算机，因而机器学习中的“学习”更准确地描述应该是将这一框架填充完善为适用于所研究任务的可用模型。



最终可以得到一个精简的技术定义：

机器学习就是在给定数学框架的基础上，利用反馈信号的指引来得到从输入到输出的有用模型。

举例-线性回归

- **回归**（regression）是能为一个或多个自变量与因变量之间关系建模的一类方法。
- 在机器学习领域中的大多数任务通常都与**预测**（prediction）有关。
- 常见的例子包括：预测价格（房屋、股票等）、预测住院时间（针对住院病人等）、预测需求（零售销量等）。

线性回归的基本元素

为了解释**线性回归**，我们举一个实际的例子：
我们希望根据房屋的面积（平方英尺）和房龄（年）来估算房屋价格（美元）。
为了开发一个能预测房价的模型，我们需要收集一个真实的数据集。
这个数据集包括了房屋的销售价格、面积和房龄。

在机器学习的术语中

- 数据集会被分为为*训练数据集*（training data set）或*训练集*（training set）。

- 每行数据（比如一次房屋交易相对应的数据）称为*样本*（sample），也可以称为*数据点*（data point）或*数据样本*（data instance）。
- 把试图预测的目标（比如预测房屋价格）称为*标签*（label）或*目标*（target）。
- 预测所依据的自变量（面积和房龄）称为*特征*（feature）或*协变量*（covariate）。

通常，我们使用n来表示数据集中的样本数。
对索引为i的样本，其输入表示为$\mathbf{x}^{(i)} = [x_1^{(i)}, x_2^{(i)}]^\top$，
其对应的标签是$y^{(i)}$。

模型 = 框架 + 参数

线性模型

线性假设目标（房屋价格）可以表示为特征（面积和房龄）的加权和，如下面的式子：

$$\mathrm{price} = w_{\mathrm{area}} \cdot \mathrm{area} + w_{\mathrm{age}} \cdot \mathrm{age} + b.$$

其中：

- $area$和age 称为*特征*（feature）。
- w_{area}和w_{age} 称为*权重*（weight），权重决定了每个特征对我们预测值的影响。
- b称为*偏置*（bias）、*偏移量*（offset）或*截距*（intercept）。偏置是指当所有特征都取值为0时，预测值应该为多少。

目标：

- 使用给定数据集，寻找模型的权重$\mathbf{w}$和偏置b
- 使得根据模型做出的预测大体符合数据里的真实价格。

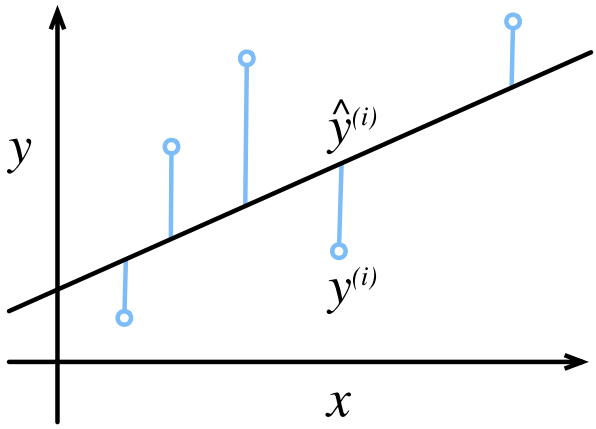
损失函数

均方误差 MSE（mean squared error）

在我们开始考虑如何用模型*拟合*（fit）数据之前，我们需要确定一个拟合程度的**度量**。

- *损失函数*（loss function）能够量化目标的*实际*值与*预测*值之间的差距。
- 通常我们会选择**非负数**作为损失，且数值越小表示损失越小，完美预测时的损失为0。
- 回归问题中最常用的损失函数是平方误差函数。
- 当样本i的预测值为$\hat{y}^{(i)}$，其相应的真实标签为$y^{(i)}$时，
  平方误差可以定义为以下公式：

$$l^{(i)}(\mathbf{w}, b) = \frac{1}{2}\left(\hat{y}^{(i)} - y^{(i)}\right)^2.$$



为了度量模型在整个数据集上的质量，我们需计算在训练集n个样本上的损失均值（也等价于求和）。

$$L(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} l^{(i)}(\mathbf{w}, b) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}\left(\mathbf{w}^\top \mathbf{x}^{(i)} + b - y^{(i)}\right)^2.$$

- 在训练模型时，我们希望寻找一组参数（$\mathbf{w}^*, b^*$），这组参数能最小化在所有训练样本上的总损失。如下式：

$$\mathbf{w}^*, b^* = \operatorname{argmin}_{\mathbf{w},b}\ L(\mathbf{w}, b).$$

```
import torch
y = torch.tensor([1,2,3],dtype = torch.float)
y_hat = torch.tensor([5,10,4],dtype = torch.float)
loss = torch.nn.MSELoss(reduction='mean')
print('MSELoss = ',loss(y_hat,y).item())

MSELoss = 0
for i in range(y.shape[0]):
    MSELoss += (y[i] - y_hat[i])**2 /2
MSELoss /= y.shape[0]
print('MSELoss = ',loss(y_hat,y).item())
```

```
MSELoss =  27.0
MSELoss =  27.0
```

梯度下降

- 用到一种名为*梯度下降*（gradient descent）的方法，这种方法几乎可以优化所有深度学习模型。
- 它通过不断地在损失函数递减的方向上更新参数来降低误差。
- 梯度下降最简单的用法是计算损失函数（数据集中所有样本的损失均值）关于模型参数的导数（在这里也可以称为梯度）。
- 我们用下面的数学公式来表示这一更新过程（∂表示偏导数）：

$$\begin{aligned}(\mathbf{w}, b) &\leftarrow (\mathbf{w}, b) - lr\sum_{i\in\mathcal{B}} \partial_{(\mathbf{w},b)} l^{(i)}(\mathbf{w}, b).\\ y = f(w) &\approx f(w_0) + f'(w_0)\Delta w \rightarrow \Delta y = f'(w_0)\Delta w\\ if\ \ \Delta w &= -lr * f'(w_0)(lr > 0), \Delta y = -lr * f'^2(w_0) < 0\end{aligned} \tag{1}$$

总结一下，算法的步骤如下：

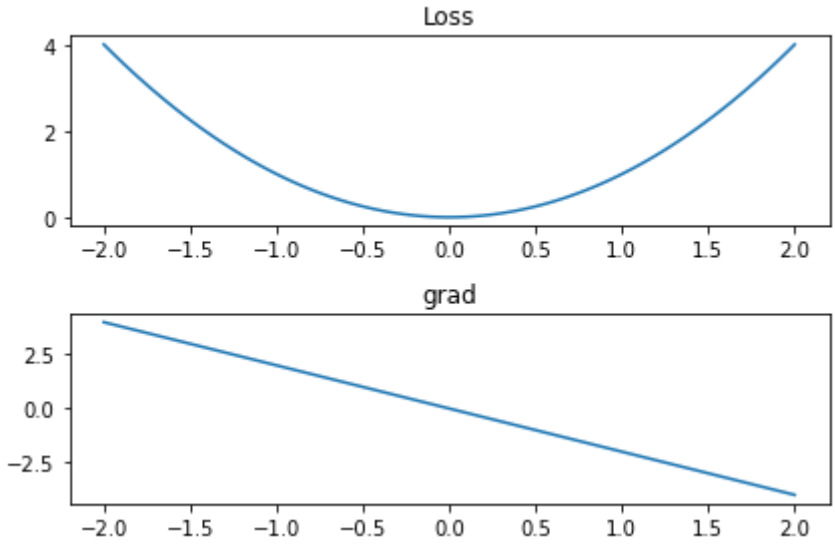
（1）初始化模型参数的值，如随机初始化；

（2）利用数据集在负梯度的方向上更新参数，并不断迭代这一步骤。

- 学习率 lr

```
import matplotlib.pyplot as plt
import torch

x = torch.linspace(-2, 2, steps =100,requires_grad=True)
y = x**2
y.backward(torch.ones_like(x))
```

```
z = x.grad
# 激活第一个 subplot
plt.subplot(2,  1,  1)
# 绘制第一个图像
plt.plot(x.detach().numpy(), y.detach().numpy())
plt.title('Loss')
# 将第二个 subplot 激活，并绘制第二个图像
plt.subplot(2,  1,  2)
plt.plot(x.detach().numpy(),-z)
plt.title('grad')
# 展示图像
plt.tight_layout()
plt.show()
```



模型 = 框架 + 参数

所以保存模型就是保存模型的框架与所有涉及到参数的值

用模型进行预测

- 给定“已学习”的线性回归模型$\hat{\mathbf{w}}^\top \mathbf{x} + \hat{b}$，我们就可以通过房屋面积$x_1$和房龄$x_2$来估计一个（未包含在训练数据中的）新房屋价格。
- 给定特征估计目标的过程通常称为*预测*（prediction）或*推断*（inference）。

•PART 2

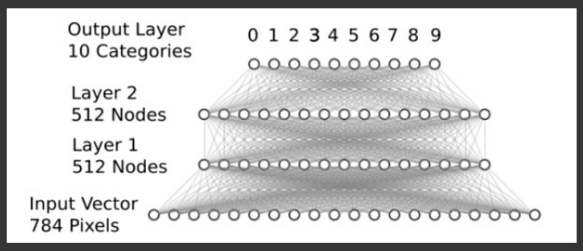
神经网络

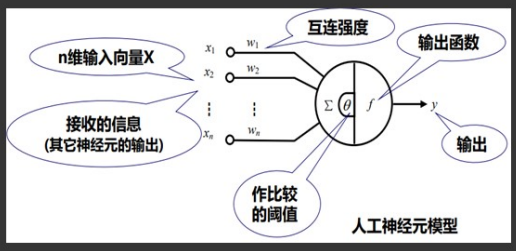
人造神经元、感知机、神经网络

1、人工神经元

关系：

人造神经元是神经网络的基础，神经网络中最基本的成分是神经元模型

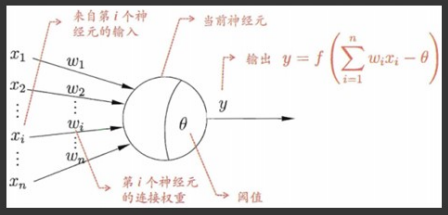




1、人工神经元——M-P神经元模型

M-P神经元模型定义：

神经元接收到来自n个其他神经元传递过来的输入信号，这些输入信号通过带权重的连接进行传递，神经元接收到的总输入值将于神经元的阈值进行比较，然后通过“激活函数”处理以产生神经元的输出



生物神经元：

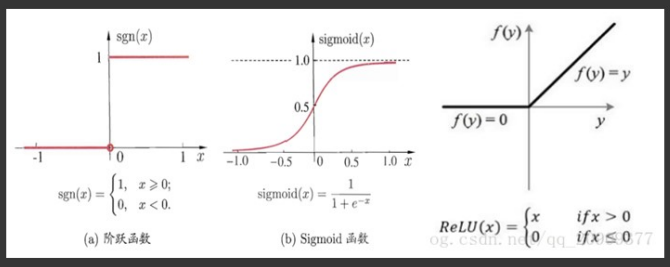
在生物神经网络中，每个神经元与其他神经元相连，当它“兴奋”时，就会向相连的神经元发送化学物质，从而改变这些神经元内的 电位。如果某神经元的电位超过了一个“阈值”，那么它就会被激活，即“兴奋”起来，向其他神经元发送化学物质。

PART 2

1、人工神经元——激活函数

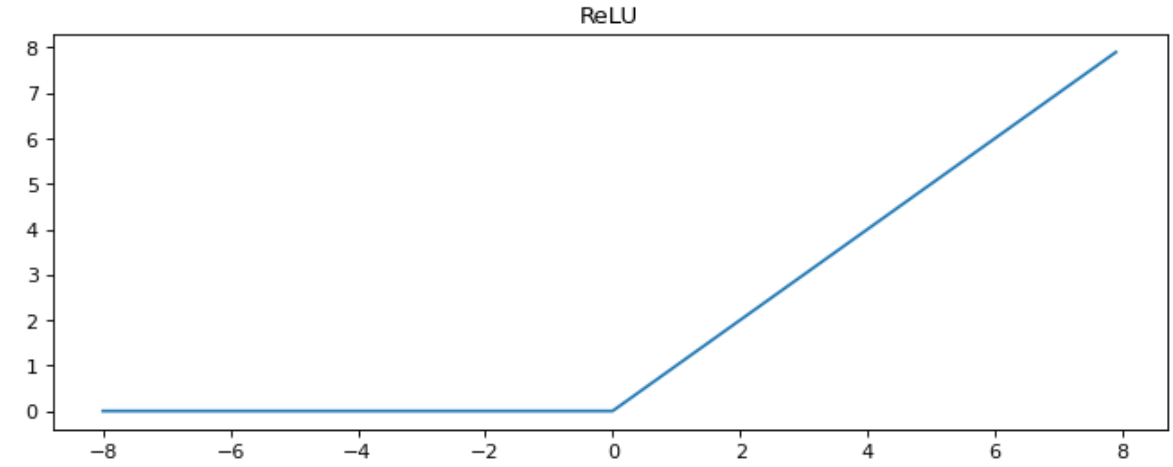
理想中的激活函数是图（a）所示阶跃函数，它将输入值映射为输出 值"0"或"1"，显然"1"对应于神经元兴奋，"0"对应于神经 元抑制。然而阶跃函数具有不连续、不光滑等不太好的性质，因此实 际常用Sigmoid函数作为激活函数，如图(b)所示。

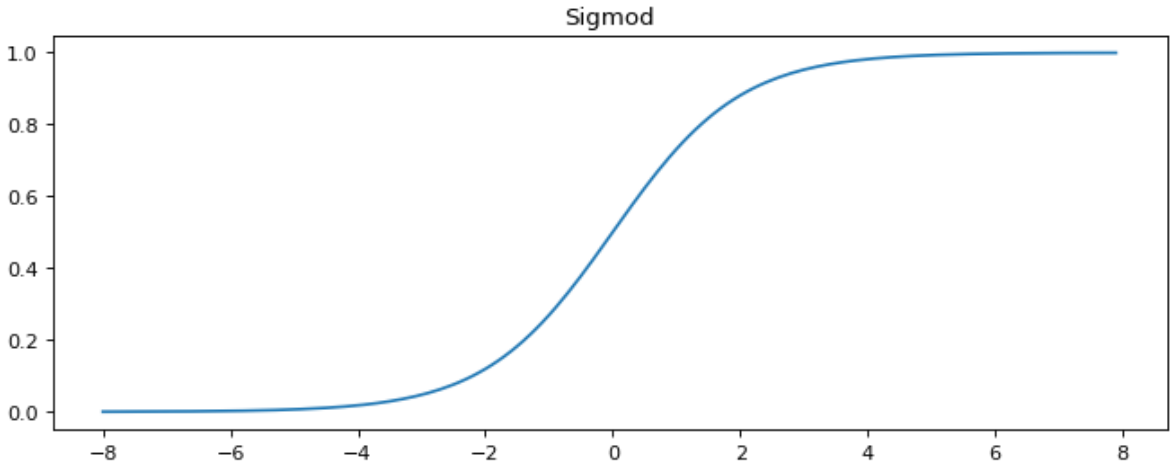
猜猜这是几?



```
fig = plt.figure(figsize=(10,8), dpi= 80)
plt.subplot(2,  1,  1)
x = torch.arange(-8.0, 8.0, 0.1, requires_grad=True)
y = torch.relu(x)
plt.plot(x.detach(), y.detach())
plt.title('ReLU')

plt.subplot(2,  1, 2)
y = torch.sigmoid(x)
plt.plot(x.detach(), y.detach())
plt.title('Sigmod')
```

```
Text(0.5, 1.0, 'Sigmod')
```





PART 2

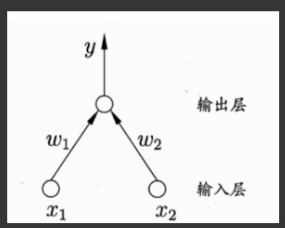
2、感知机

感知机（Perceptron）由两层神经元组成，如图所示，输入层接收外界输入信号后传递给输出层，输出层是M-P神经元，亦称“阈值逻辑单元”。

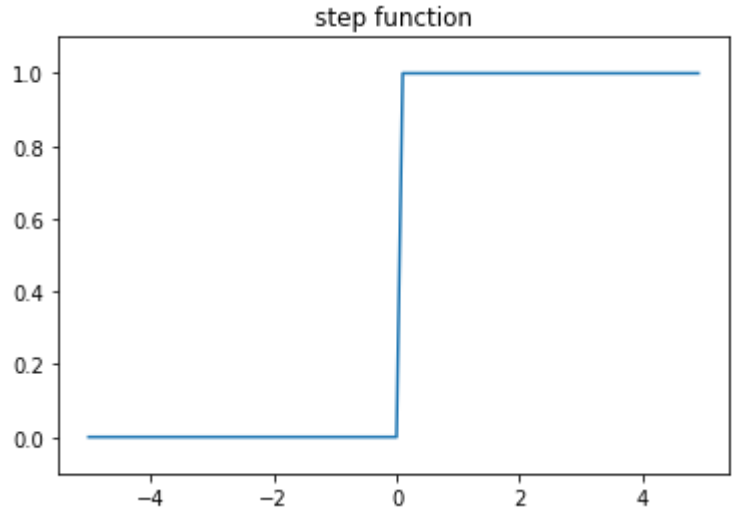
一个感知机就是一个最简单的学习模型，包含输入、输出、互联强度（权值）及比较阈值，将输入加权求和后与阈值做差，送入输出函数得到输出结果。适当选择权值、阈值即输出函数，通过感知机可以实现任意布尔函数。

感知机能容易的实现逻辑与、或、非的运算，例如假设 f 是阶跃函数:

- “与”$(x_1 \wedge x_2)$: 令 $w_1 = w_2 = 1, \theta = 2$, 则 $y = f(1 \cdot x_1 + 1 \cdot x_2 - 2)$, 仅在 $x_1 = x_2 = 1$ 时, $y = 1$;
- “或”$(x_1 \vee x_2)$: 令 $w_1 = w_2 = 1, \theta = 0.5$, 则 $y = f(1 \cdot x_1 + 1 \cdot x_2 - 0.5)$, 当 $x_1 = 1$ 或 $x_2 = 1$ 时, $y = 1$;
- “非”$(\neg x_1)$: 令 $w_1 = -0.6, w_2 = 0, \theta = -0.5$, 则 $y = f(-0.6 \cdot x_1 + 0 \cdot x_2 + 0.5)$, 当 $x_1 = 1$ 时, $y = 0$; 当 $x_1 = 0$ 时, $y = 1$.



```python
import matplotlib.pyplot as plt
def step_func(x): # 参数x可以接受numpy数组
    y = x >= 0 # y是布尔型数组
    return y.type(torch.int)

x = torch.arange(-5., 5., .1)
y = step_func(x)
plt.plot(x,y)
plt.ylim(-0.1, 1.1) # y轴范围
plt.title('step function')
plt.show()
```




```python
import numpy as np
import torch

# 定义单个神经元 2输入
net =
torch.nn.Sequential(torch.nn.Linear(2,1))#torch.nn.Hardtanh(min_val=0,max_val
=3,inplace=True)) #

# 定义权重
w = torch.tensor([1,1]).type(torch.float32)
b = -2

net[0].weight.data = w.reshape(net[0].weight.shape)
net[0].bias.data.fill_(b)

# 验证输出
x = torch.tensor([1.0, 1.0]).type(torch.float32)
print('The result of 1 and 1 is', step_func(net.forward(x)).item())
x = torch.tensor([0, 1.0]).type(torch.float32)
print('The result of 1 and 1 is', step_func(net.forward(x)).item())
x = torch.tensor([0, 0]).type(torch.float32)
print('The result of 1 and 1 is', step_func(net.forward(x)).item())

torch.matmul(x, w) + b
```

```
23
```

```
The result of 1 and 1 is 1
The result of 1 and 1 is 0
The result of 1 and 1 is 0
```

```
tensor(-2.)
```

PART 2

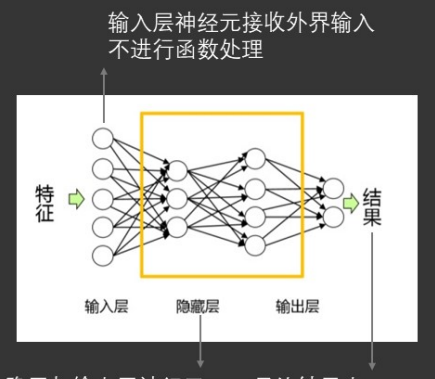
3、神经网络

神经网络以一个个人造神经元为基本单元，进一步以层为单位，层与层之间连接形成了神经网络，每一个神经元接收若干个收入，通过权值的作用得到输出，这些权值就是数学框架中需要填充修改的关键因素。

神经网络中每层对输入数据所做的操作其实就通过其保存的若干个权重来实现，所以这些权重也可以称为每一层的参数，学习的目的正是为神经网络的所有层找到一组合适的权值，使其能够得到预期输出，即使得模型的损失最小。

神经网络模型模拟人类的神经网络搭建，故而相邻层之间的神经元相互连接，而跨层之间不存在连接。

神经网络的学习过程，就是根据训练数据来调整神经元之间的“连接权”以及每个功能神经元的阈值。



$$y = w^T X + b \tag{2}$$

```
import torch
from torch import nn
import matplotlib.pyplot as plt # 绘图与图像显示
import torch.utils.data as Data # 数据操作
from torchvision.datasets import MNIST,FashionMNIST # 数据集
from torchvision import transforms # 图像变换、数据预处理
```

加载数据集

- torchvision 中的 transforms 模块可以针对每张图像进行预处理操作

```
#Compose是一个容器，传入的参数是列表，ToTensor()，类型变换，Normalize是数据标准化，去均值，除标准差
# transforms.ToTensor() 把取值[0, 255]的PIL图像形状为[H,W,C]转换为形状[C,H,W] 取值范围[0,1.0]的张量
transform = transforms.Compose([transforms.ToTensor()])
```

```
data_train = MNIST(root = "data/", ## 数据的路径，如果存在数据则加载，否则下载至此路径
                                transform = transform, ## 图像变换操作
                                train =True, ## 决定使用训练集还是测试集
                                download = True) ## 选择是否需要下载数据
data_test = MNIST(root = "data/",
                                transform = transform,
                                train =False)
```

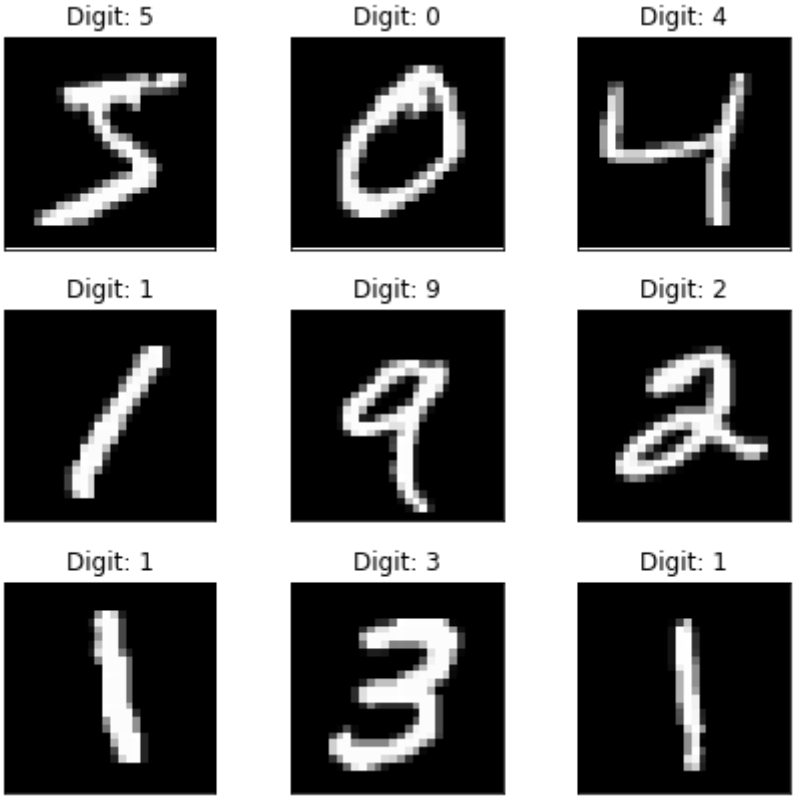
显示图像

```
for x,y in data_train:
    break
```

```
data_train[2][0].shape, len(data_train), len(data_test)
```

```
(torch.Size([1, 28, 28]), 60000, 10000)
```

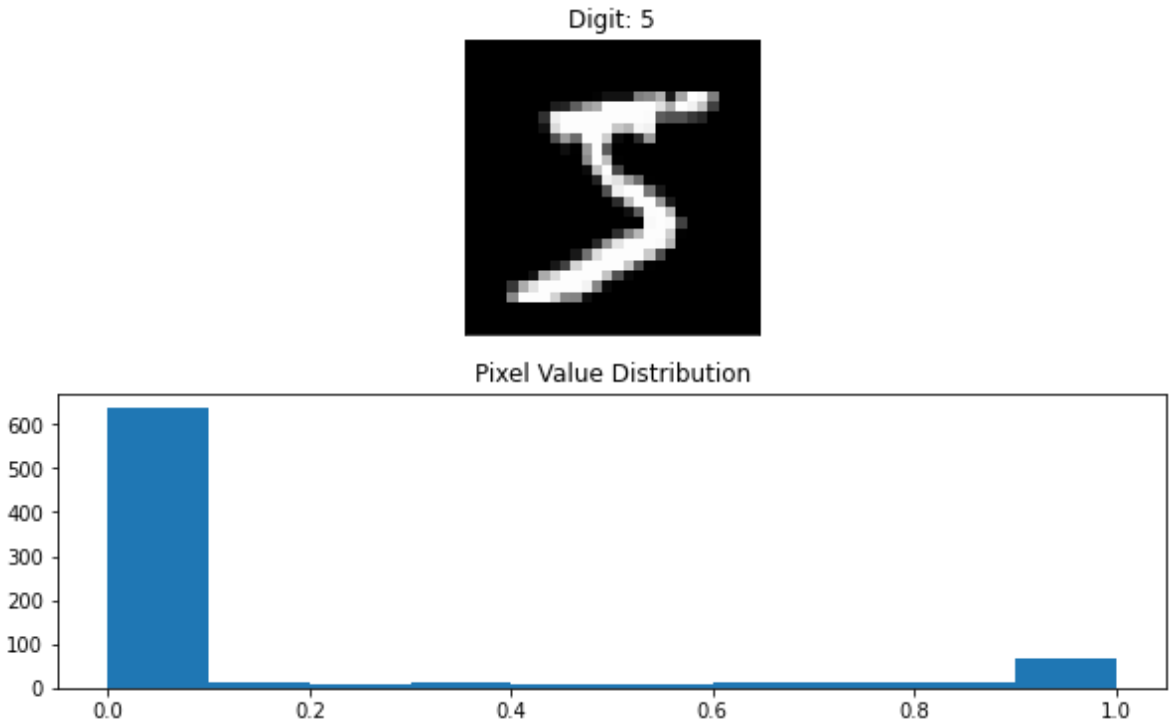
```
# plt.imshow(x[0].numpy().reshape(28,28))
# plt.show()
fig = plt.figure(figsize=(6,6))
for i in range(9):
  plt.subplot(3,3,i+1)
  plt.tight_layout()
  plt.imshow(data_train[i][0].numpy().reshape(28,28), cmap='gray',
interpolation='none')
  plt.title("Digit: {}".format(data_train[i][1]))
  plt.xticks([])
  plt.yticks([])
```



显示图像像素分布

```
fig = plt.figure(figsize = (10,6))
plt.subplot(2,1,1)
plt.imshow(data_train[0][0].numpy().reshape(28,28), cmap='gray',
interpolation='none')
plt.title("Digit: {}".format(data_train[0][1]))
plt.xticks([])
plt.yticks([])
plt.subplot(2,1,2)
plt.hist(data_train[0][0].numpy().reshape(784))
plt.title("Pixel Value Distribution")
```

```
Text(0.5, 1.0, 'Pixel Value Distribution')
```



定义模型

```
## 定义数据加载器
batch_size = 256

train_loader = Data.DataLoader(dataset = data_train, ## 使用的数据集
                               batch_size = batch_size, ## 批处理样本大小
                               shuffle = True ## 是否打乱数据顺序
                               )

test_loader = Data.DataLoader(dataset = data_test,
                              batch_size = batch_size,
                              shuffle = True  #将顺序随机打乱
                              )
```

```
## 定义网络模型
net = nn.Sequential(nn.Flatten(), ## 将二维图像展平位一维数组，输入层
                    nn.Linear(in_features = 784, ## 隐藏层的输入，数据的特征数
                              out_features = 512, ## 隐藏层的输出，对应神经元的数量
                             bias = True
                             ),  ##
                    nn.ReLU(), ## 激活函数
                    nn.Dropout(p=0.2),
                    nn.Linear(512,512),
nn.ReLU(),nn.Dropout(p=0.2),nn.Linear(512,10)) # 深度网络模型

# net = nn.Sequential(nn.Flatten(), nn.Linear(784, 256), nn.ReLU(),
nn.Linear(256, 10)) # MLP模型

 # net = nn.Sequential(nn.Flatten(), nn.Linear(784, 10)) # softmax回归模型
```

```
## 初始化模型参数
def init_weights(m):
    if type(m) == nn.Linear:
        nn.init.normal_(m.weight, std=0.01)

net.apply(init_weights);
```

训练模型

```
len(train_loader),60000/256
```

```
(235, 234.375)
```

```
for X, y in train_loader:
    break
output = net.forward(X)
X.shape,output.shape
```

```
(torch.Size([256, 1, 28, 28]), torch.Size([256, 10]))
```

```
lr, num_epochs = 0.1, 20 ## 定义学习率及训练次数
optimizer = torch.optim.SGD(net.parameters(), lr = lr) ## 定义优化器
loss_func = nn.CrossEntropyLoss(reduction='mean') ## 定义损失函数，分类问题一般使用交叉熵损失

train_loss_all = []
```

```
train_acc_all = []
test_acc_all = []
test_loss_all = []

for epoch in range(num_epochs):
    print("Epoch {}/{}".format(epoch+1,num_epochs))
    print("-"*10)

    running_loss = 0.0
    running_correct = 0.0

    for X, y in train_loader:
        net.train() ## 表明模型在训练
        output = net.forward(X) ## 模型在 X 上的输出: N * num_class
        train_loss = loss_func(output, y ) ## 交叉熵误差
        _, pred = torch.max(output.data, 1) ## 获得预测结果
        optimizer.zero_grad() ## 每次迭代将梯度初始化为0
        train_loss.backward() ## 损失的后向传播， 计算梯度
        optimizer.step() ## 使用梯度进行优化
        running_loss += train_loss.item() ## 统计模型预测损失
        running_correct += torch.sum(pred == y.data) ## 统计模型预测准确个数

    test_correct = 0
    val_loss = 0
    for data in test_loader:
        X_test, y_test = data
        output = net(X_test)
        test_loss = loss_func(output, y_test )
        _, pred = torch.max(output.data, 1)
        test_correct += torch.sum(pred == y_test.data)
        val_loss += test_loss.item()
    print("Loss is:{:.4f}, Train_accuracy is {:.4f}%, Test_accuracy is
{:.4f}%"

.format(running_loss/len(train_loader),100*running_correct/len(data_train),
100*test_correct/len(data_test)))
    train_loss_all.append(running_loss/len(train_loader))
    train_acc_all.append(running_correct/len(data_train))
    test_loss_all.append(val_loss/len(test_loader))
    test_acc_all.append(test_correct/len(data_test))
```

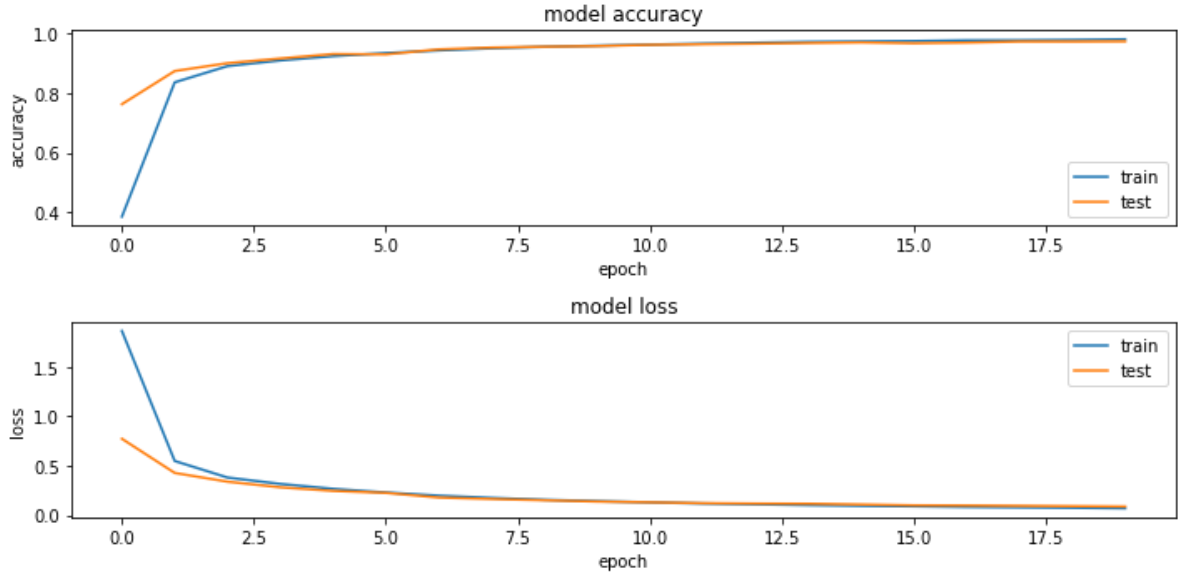
```
Epoch 1/20
----------
Loss is:1.8631, Train_accuracy is 38.5900%, Test_accuracy is 76.2600%
Epoch 2/20
----------
Loss is:0.5476, Train_accuracy is 83.5817%, Test_accuracy is 87.3700%
Epoch 3/20
----------
Loss is:0.3801, Train_accuracy is 89.0133%, Test_accuracy is 90.0000%
Epoch 4/20
----------
Loss is:0.3154, Train_accuracy is 90.9083%, Test_accuracy is 91.5700%
Epoch 5/20
----------
```

```
Loss is:0.2659, Train_accuracy is 92.3467%, Test_accuracy is 93.0600%
Epoch 6/20
----------
Loss is:0.2286, Train_accuracy is 93.4000%, Test_accuracy is 92.9500%
Epoch 7/20
----------
Loss is:0.1971, Train_accuracy is 94.2567%, Test_accuracy is 94.6300%
Epoch 8/20
----------
Loss is:0.1742, Train_accuracy is 94.9683%, Test_accuracy is 95.2300%
Epoch 9/20
----------
Loss is:0.1552, Train_accuracy is 95.4767%, Test_accuracy is 95.5200%
Epoch 10/20
----------
Loss is:0.1416, Train_accuracy is 95.8583%, Test_accuracy is 95.7300%
Epoch 11/20
----------
Loss is:0.1290, Train_accuracy is 96.2133%, Test_accuracy is 96.1500%
Epoch 12/20
----------
Loss is:0.1169, Train_accuracy is 96.5767%, Test_accuracy is 96.3500%
Epoch 13/20
----------
Loss is:0.1090, Train_accuracy is 96.8767%, Test_accuracy is 96.5300%
Epoch 14/20
----------
Loss is:0.1005, Train_accuracy is 97.1350%, Test_accuracy is 96.7500%
Epoch 15/20
----------
Loss is:0.0937, Train_accuracy is 97.2750%, Test_accuracy is 96.9500%
Epoch 16/20
----------
Loss is:0.0883, Train_accuracy is 97.4000%, Test_accuracy is 96.6700%
Epoch 17/20
----------
Loss is:0.0820, Train_accuracy is 97.6433%, Test_accuracy is 96.8900%
Epoch 18/20
----------
Loss is:0.0779, Train_accuracy is 97.6917%, Test_accuracy is 97.2500%
Epoch 19/20
----------
Loss is:0.0740, Train_accuracy is 97.8167%, Test_accuracy is 97.2400%
Epoch 20/20
----------
Loss is:0.0687, Train_accuracy is 98.0000%, Test_accuracy is 97.3000%
```

训练效果可视化

```
# plotting the metrics
fig = plt.figure(figsize=(10,5))
plt.subplot(2,1,1)
plt.plot(train_acc_all)
plt.plot(test_acc_all)
plt.title('model accuracy')
```

```
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='lower right')

plt.subplot(2,1,2)
plt.plot(train_loss_all)
plt.plot(test_loss_all)
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
# plt.legend(['train'], loc='upper right')
plt.legend(['train', 'test'], loc='upper right')

plt.tight_layout()

# fig
```



保存模型

```
torch.save(net, "result/class_5/mlp.pkl")
```

加载并验证模型

```
net_load = torch.load("result/class_5/mlp.pkl")
net_load.eval()
```

```
Sequential(
  (0): Flatten(start_dim=1, end_dim=-1)
  (1): Linear(in_features=784, out_features=512, bias=True)
  (2): ReLU()
  (3): Dropout(p=0.2, inplace=False)
  (4): Linear(in_features=512, out_features=512, bias=True)
  (5): ReLU()
  (6): Dropout(p=0.2, inplace=False)
  (7): Linear(in_features=512, out_features=10, bias=True)
)
```

```
for X,y in test_loader:
    break
y_hat = net_load(X)
```

```
y_hat[0].reshape(-1,10).shape
```

```
torch.Size([1, 10])
```

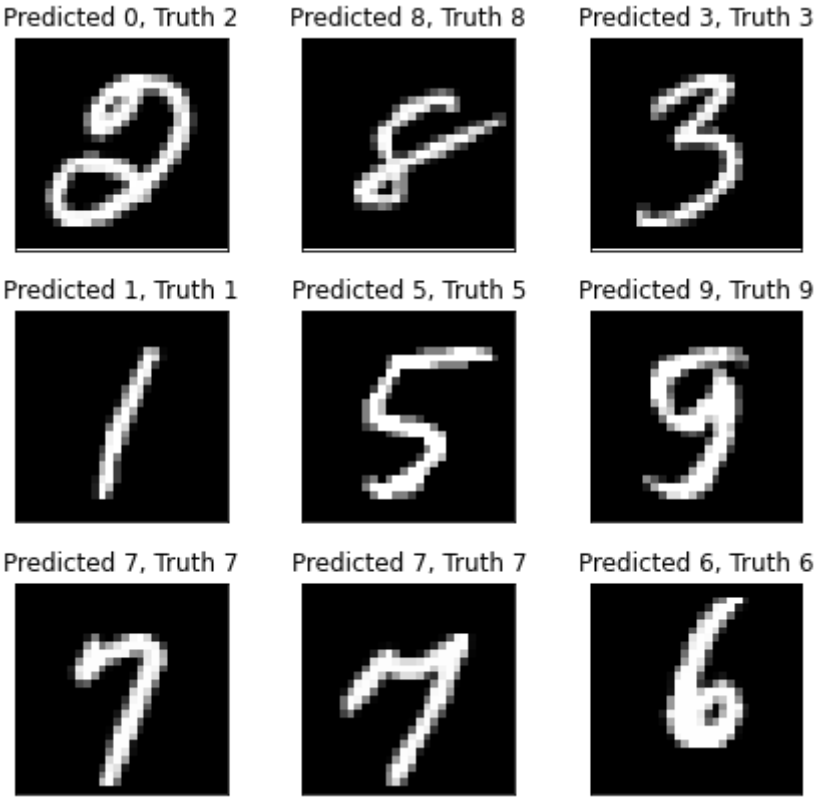
```
y_hat.shape,y_hat[4],torch.softmax(y_hat[4].reshape(-1,10),dim=1)
```

```
(torch.Size([256, 10]),
 tensor([-0.6811, -5.0644, -8.4763,  3.2538, -2.8183, 16.7534, -1.6986,
-7.9701,
          2.8291,  4.3758], grad_fn=<SelectBackward0>),
 tensor([[2.6811e-08, 3.3470e-10, 1.1037e-11, 1.3715e-06, 3.1631e-09, 9.9999e-
01,
          9.6917e-09, 1.8311e-11, 8.9694e-07, 4.2121e-06]],
        grad_fn=<SoftmaxBackward0>))
```

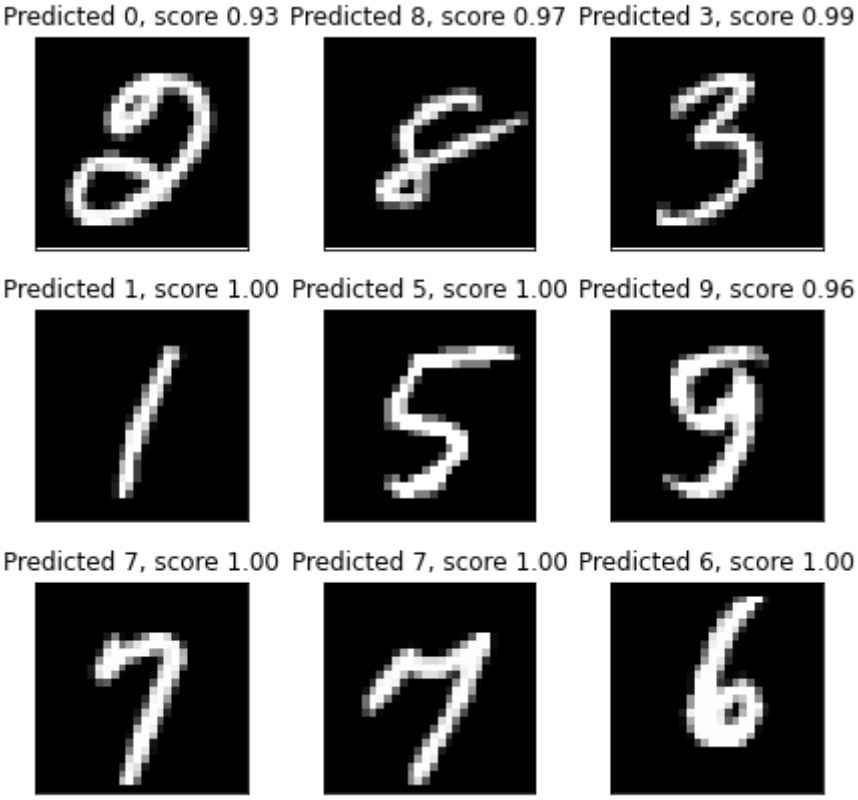
```
score,pred = torch.max(torch.softmax(y_hat[:10],dim = 1), 1)
score,pred,y[:10]
```

```
(tensor([0.9319, 0.9722, 0.9943, 0.9976, 1.0000, 0.9556, 0.9997, 0.9977,
0.9999,
         0.9986], grad_fn=<MaxBackward0>),
 tensor([0, 8, 3, 1, 5, 9, 7, 7, 6, 8]),
 tensor([2, 8, 3, 1, 5, 9, 7, 7, 6, 8]))
```

```
fig = plt.figure(figsize=(6,6))
for i in range(9):
  plt.subplot(3,3,i+1)
  plt.tight_layout()
  plt.imshow(X[i].reshape(28,28), cmap='gray', interpolation='none')
  plt.title("Predicted {}, Truth {}".format(pred[i],y[i]))
  plt.xticks([])
  plt.yticks([])
```




```
fig = plt.figure(figsize=(6,6))
for i in range(9):
  plt.subplot(3,3,i+1)
  plt.tight_layout()
  plt.imshow(X[i].reshape(28,28), cmap='gray', interpolation='none')
  plt.title("Predicted {}, score {:.2f}".format(pred[i],score[i]))
  plt.xticks([])
  plt.yticks([])
```



作用举例

- 电梯按键识别

```
import torch
import cv2
import numpy as np
# import matplotlib.colors as mat_color
from matplotlib import pyplot as plt
```

- 读入图片

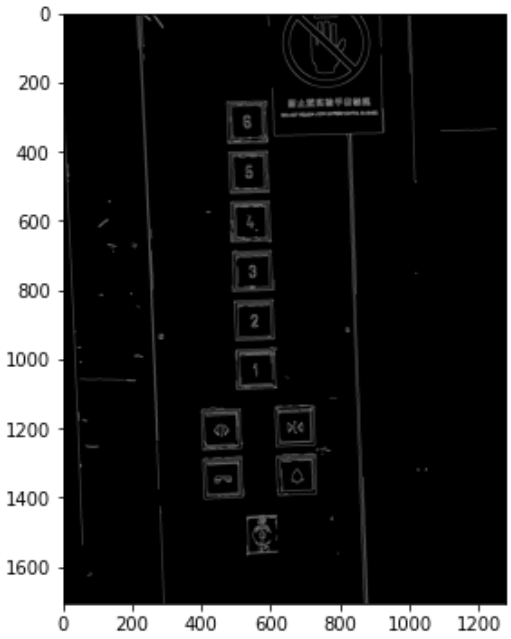
```
path = r"./images/loft.jpg"
img_bgr = cv2.imread(path)
print(np.shape(img_bgr))

# img_bgr = img_bgr.astype(np.float32)/255
plt.figure(figsize=(6,6))
img_rgb = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB)
plt.imshow(img_rgb)
```

```
(1706, 1279, 3)
```

```
<matplotlib.image.AxesImage at 0x1c09e065c10>
```

- 使用opencv内置函数提取轮廓

```
img_gray = cv2.cvtColor(img_bgr, cv2.COLOR_BGR2GRAY)
plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
plt.imshow(img_gray, cmap='gray')
img_h, img_w = img_gray.shape

plt.subplot(1,2,2)

img_canny = cv2.Canny(img_gray, threshold1=70, threshold2=140,
                      apertureSize=3, L2gradient=False)
plt.imshow(img_canny,'gray')

# 获得模型输入
img_tensor = torch.from_numpy(img_gray.reshape((1,1,img_h,img_w)))
```

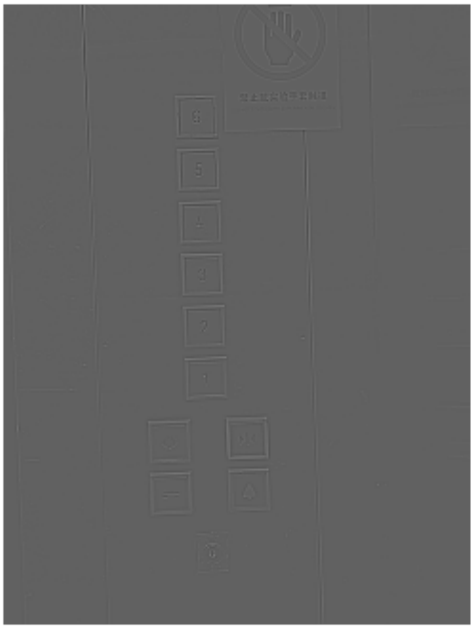


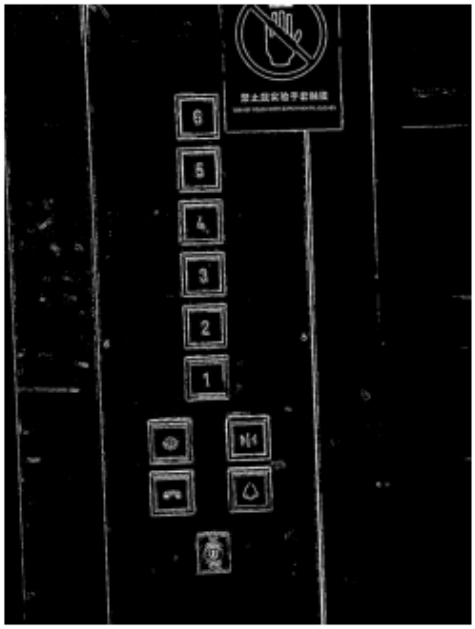
- 使用卷积方法提取轮廓
- torch 中处理数据需要归一化 （0-255 变为 0-1）

```
## 边缘提取
kersize = 7
ker = torch.ones(kersize,kersize,dtype=torch.float32)*-1
ker[3,3] = 48.0
ker = ker.reshape((1,1,kersize,kersize))
cov2d = torch.nn.Conv2d(1,1,(kersize,kersize),bias=False)
cov2d.weight.data[0] = ker
cov_out = cov2d(img_tensor/255)
cov_out = cov_out.data.squeeze()
```

```
plt.figure(figsize=(12,6))
plt.subplot(1,2,1)
plt.imshow(cov_out,cmap='gray')
plt.axis('off')

_, img_edge = cv2.threshold(cov_out.byte().numpy(),
200,255,cv2.THRESH_BINARY)
plt.subplot(1,2,2)
plt.imshow(img_edge,cmap='gray')
plt.axis('off')
```

```
(-0.5, 1272.5, 1699.5, -0.5)
```



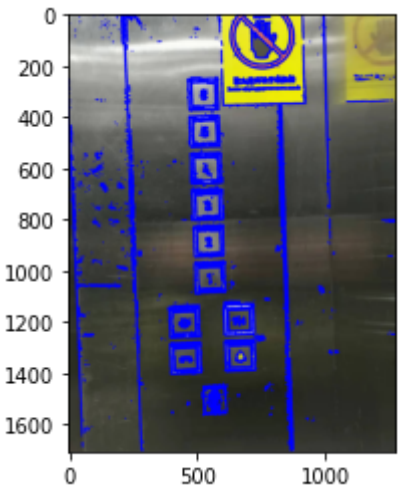


- 提取轮廓点

```
contours, hierarchy = cv2.findContours(img_edge, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
print("number of contours: " + str(len(contours)))
img_rgb_copy = np.copy(img_rgb)
img_con = cv2.drawContours(img_rgb_copy, contours, -1, (0,0,255), 5)
plt.imshow(img_con)
```

```
number of contours: 1885
```

```
<matplotlib.image.AxesImage at 0x1c0a14172e0>
```



```python
img_rgb_copy = np.copy(img_rgb)
i = 0
for obj in contours:
    area = cv2.contourArea(obj)  #计算轮廓内区域的面积
#     cv2.drawContours(img_rgb_copy, obj, -1, (255, 0, 0), 4)  #绘制轮廓线
    perimeter = cv2.arcLength(obj,True)  #计算轮廓周长
    approx = cv2.approxPolyDP(obj,0.02*perimeter,True)  #获取轮廓角点坐标
    CornerNum = len(approx)   #轮廓角点的数量
    x, y, w, h = cv2.boundingRect(approx)  #获取坐标值和宽度、高度

    #轮廓对象分类
    if CornerNum ==3: objType ="triangle"
    elif CornerNum == 4:
        if area >= 1000:
            i += 1
            if i >=5:
                print(area)
                break

cv2.rectangle(img_rgb_copy,(x,y),(x+w,y+h),(0,255,0),2)  #绘制边界框
cv2.putText(img_rgb_copy,objType,(x+(w//2),y+
(h//2)),cv2.FONT_HERSHEY_COMPLEX,0.6,(0,0,0),1)  #绘制文字
print('end')
```

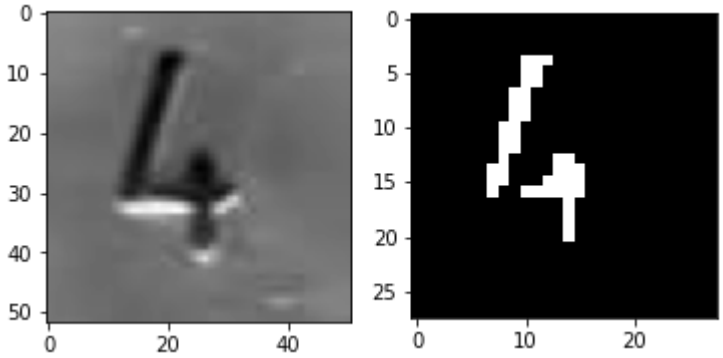
```
9284.5
end
```

```python
x, y, w, h
```

```
(487, 551, 101, 102)
```

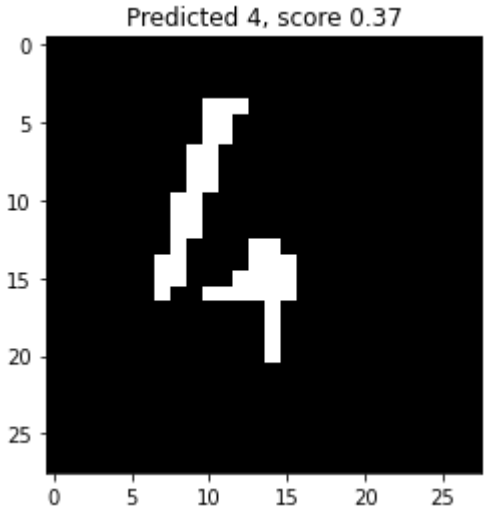
```python
plt.subplot(1,2,1)
num = img_gray[y+30:y+h-20,x+30:x+w-20] # img_bin
plt.imshow(num,'gray')

plt.subplot(1,2,2)
num = cv2.resize(num, (28, 28))
_, img_bin = cv2.threshold(num/255, 0.3,1,cv2.THRESH_BINARY_INV)
plt.imshow(img_bin,'gray')
```

```
<matplotlib.image.AxesImage at 0x1c09f75e5b0>
```



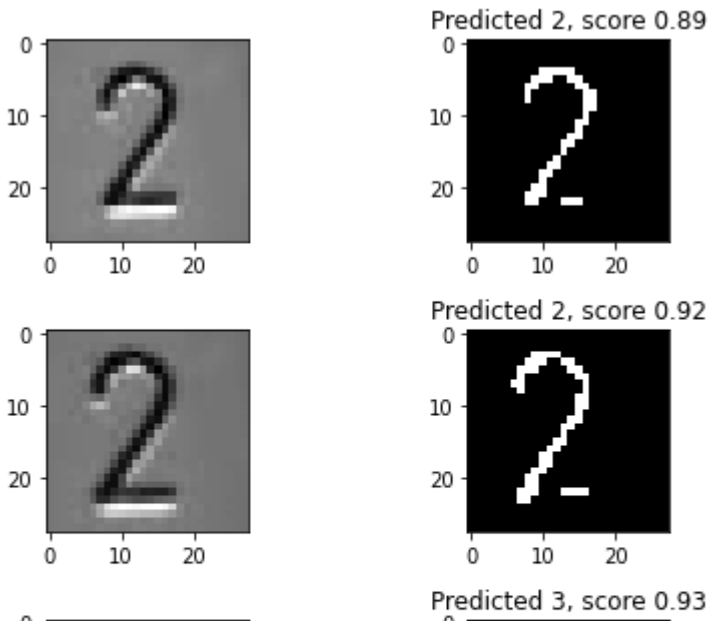
```
net_load = torch.load("result/class_5/mlp.pkl")
net_load.eval()

sample = torch.tensor(img_bin,dtype = torch.float).reshape(-1,28,28)

y_hat = net_load(sample)
score,pred = torch.max(torch.softmax(y_hat[:10],dim = 1), 1)

plt.title("Predicted {}, score {:.2f}".format(pred.item(),score.item()))
plt.imshow(img_bin,'gray')
```

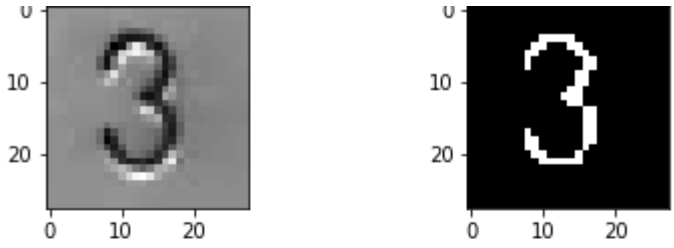
```
<matplotlib.image.AxesImage at 0x1c0943a2d30>
```



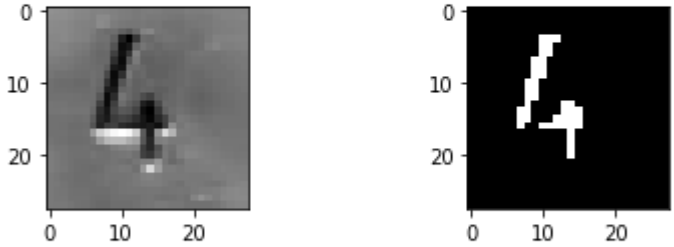

```
fig = plt.figure(figsize=(6,18))
for k in range(9):
```

```
    img_rgb_copy = np.copy(img_rgb)
    i = 0
    for obj in contours:
        area = cv2.contourArea(obj)  #计算轮廓内区域的面积
    #     cv2.drawContours(img_rgb_copy, obj, -1, (255, 0, 0), 4)  #绘制轮廓线
        perimeter = cv2.arcLength(obj,True)  #计算轮廓周长
        approx = cv2.approxPolyDP(obj,0.02*perimeter,True)  #获取轮廓角点坐标
        CornerNum = len(approx)   #轮廓角点的数量
        x, y, w, h = cv2.boundingRect(approx)  #获取坐标值和宽度、高度

        #轮廓对象分类
        if CornerNum ==3: objType ="triangle"
        elif CornerNum == 4:

            if area >= 1000:
                i += 1
                if i >=k+2:
                    break

    cv2.rectangle(img_rgb_copy,(x,y),(x+w,y+h),(0,255,0),2)  #绘制边界框
    cv2.putText(img_rgb_copy,objType,(x+(w//2),y+
(h//2)),cv2.FONT_HERSHEY_COMPLEX,0.6,(0,0,0),1)  #绘制文字

    num = img_gray[y+30:y+h-20,x+30:x+w-20] # img_bin
    num = cv2.resize(num, (28, 28))
    _, img_bin = cv2.threshold(num/255, 0.3,1,cv2.THRESH_BINARY_INV)
    sample = torch.tensor(img_bin,dtype = torch.float).reshape(-1,28,28)
    y_hat = net_load(sample)
    score,pred = torch.max(torch.softmax(y_hat[:10],dim = 1), 1)

    plt.tight_layout()
    plt.subplot(9,2,k*2+1)
    plt.imshow(num,'gray')

    plt.subplot(9,2,k*2+2)
    plt.imshow(img_bin,'gray')
    plt.title("Predicted {}, score {:.2f}".format(pred.item(),score.item()))
```

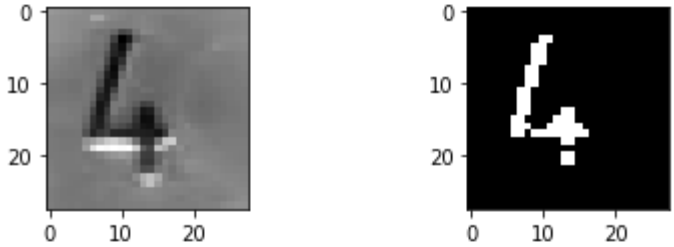




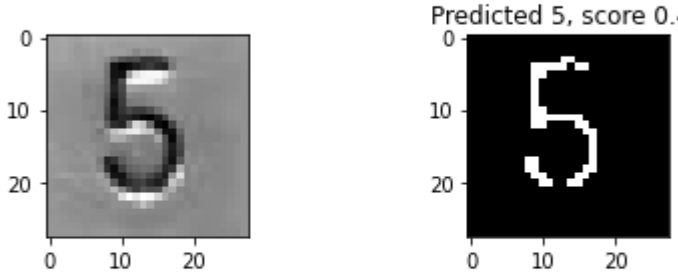
Predicted 4, score 0.37



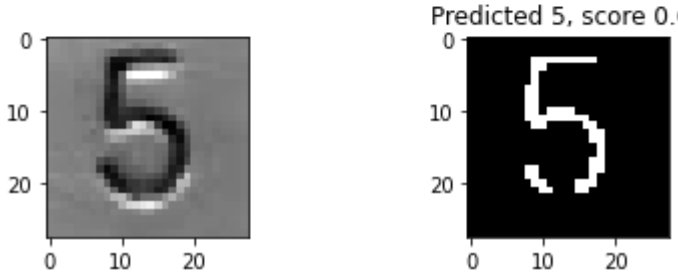
Predicted 4, score 0.45



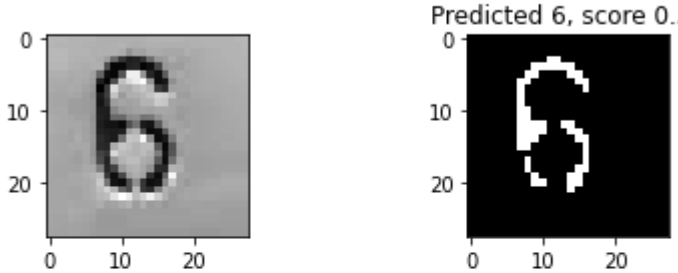
Predicted 5, score 0.49



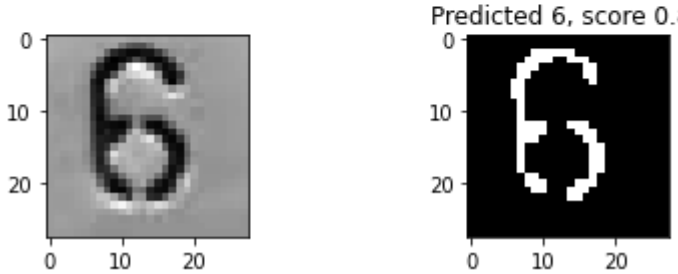
Predicted 5, score 0.62



Predicted 6, score 0.55



Predicted 6, score 0.83



6、模型改进

可以改进模型的方法很多，可以自己尝试修改每一层的节点数，找到最合适的损失函数、优化器等。

为了得到更好的结果，对机器学习而言，最好的办法其实是扩大数据集的样本量。