

机器视觉精品课程-卷积神经网络部分

授课内容

- 基础概念回顾与拓展、梯度反向传播、交叉损失熵、Softmax回归、卷积神经网络
- 对上节课内容有问题的先提出进行讲解或者为本节课内容

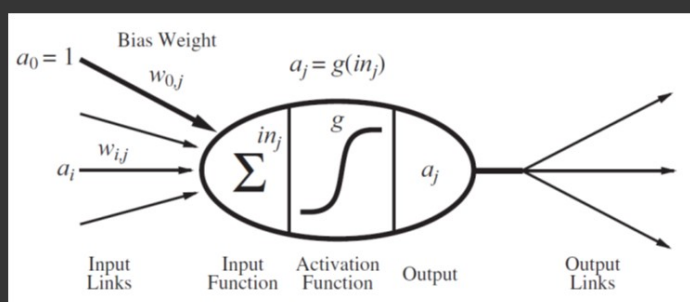
PART 1

1、内容回顾-人工神经元

MP神经元模型是对生物神经元的一种可能的数学表达:

$$a_j = g(\sum_{i=1}^n w_{ij} a_i - \theta_j)$$

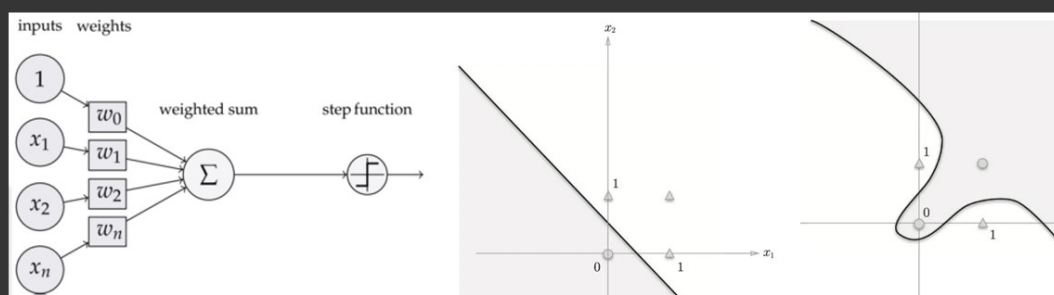
即输入n个量 $a_1, a_2, \dots, a_i, \dots, a_n$, 经过给定的权重 w 加权求和并与偏置量相加之后, 经过激活函数 g 进行运算, 得到输出量 a_j .



PART 1

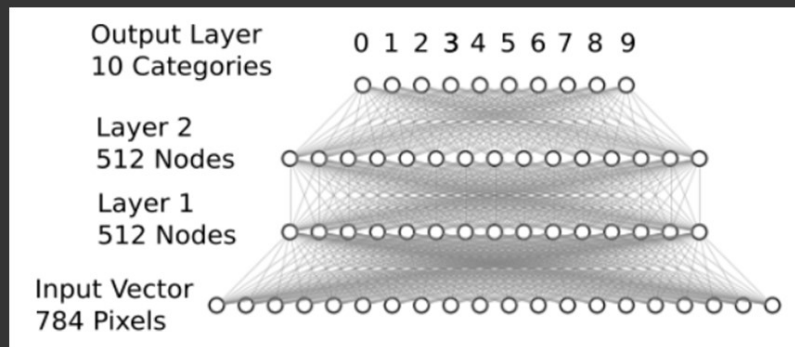
1、内容回顾-感知机

- 单层感知机: $y = \text{step}(w \cdot x + b)$, 即完成了一个二分类问题; 可以解决或的问题
- 多层感知机: 通过加深层数可以解决非线性问题, 比如异或



1、内容回顾-神经网络

神经网络是基于感知机的扩展，可以理解为有很多隐藏层的神经网络。对于全连接神经网络，按不同层的位置划分，内部的神经网络层可以分为三类，输入层，隐藏层和输出层，一般来说第一层是输入层，最后一层是输出层，而中间的层数都是隐藏层。



- nn:neural network
- Sequential:顺序容器
- Linear(m,n):m个输入，n个输出的1层神经网络，其中n也可以代表这一层有多少个神经元

```

1 import numpy as np
2 import torch
3
4 def step_func(x): # 参数x可以接受numpy数组
5     y = x >= 0 # y是布尔型数组
6     return y.type(torch.int)
7
8 # 定义单个神经元 2输入
9 net = torch.nn.Sequential(torch.nn.Linear(2,1))
10
11 # 定义权重
12 w = torch.tensor([1,1]).type(torch.float32)
13 b = -2
14
15 net[0].weight.data = w.reshape(net[0].weight.shape)
16 net[0].bias.data.fill_(b)
17
18 # 验证输出
19 x = torch.tensor([1.0, 1.0]).type(torch.float32)
20 print('The result of 1 and 1 is', step_func(net.forward(x)).item())
21 x = torch.tensor([0, 1.0]).type(torch.float32)
22 print('The result of 1 and 1 is', step_func(net.forward(x)).item())
23 x = torch.tensor([0, 0]).type(torch.float32)
24 print('The result of 1 and 1 is', step_func(net.forward(x)).item())
25
26 torch.matmul(x, w) + b

```

```

1 The result of 1 and 1 is 1
2 The result of 1 and 1 is 0
3 The result of 1 and 1 is 0

```

1 | tensor(-2.)

神经网络是基于感知机的扩展，而DNN（deep-neural-networks）可以理解为有很多隐藏层的神经网络。多层神经网络和深度神经网络DNN其实也是指的一个东西，DNN有时也叫做多层感知机（Multi-Layer perceptron,MLP）。

PART 1

2、深度学习定义

深度学习的深度就是从“输入层”到“输出层”所经历层次的数目，即“隐藏层”的数目；学习指机器学习方法。简而言之，只要一个神经网络层数足够多，就可以称之为深度神经网络。目前典型的神经网络可以有成百上千层，具有数十万个可训练参数。

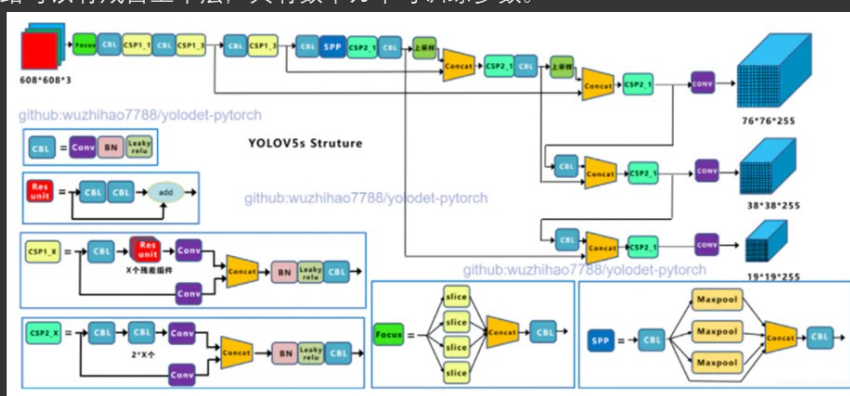


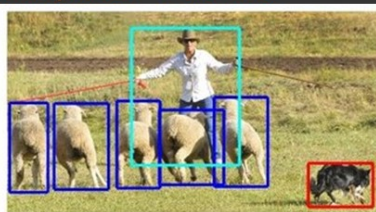
image.png

PART 1

3、深度学习典型应用-计算机视觉（Computer Vision）



(a) Image classification



<https://arxiv.org/pdf/1405.0312.pdf>

2、神经网络训练中的基本概念

基本中的基本：

神经元、层、激活函数、损失函数、epoch、batch size

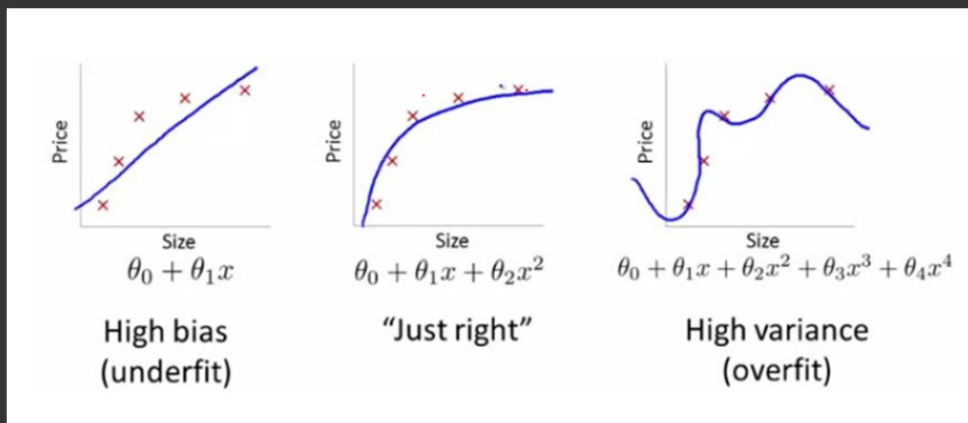
神经网络中的超参数（相对概念：可训练参数）：

1. 学习率 η ，2. 正则化参数 λ ，3. 神经网络的层数 L ，4. 每一个隐层中神经元的个数 j ，5. 学习的回合数 Epoch，6. 批量数据 batch 的大小，7. 输出神经元的编码方式，8. 代价函数的选择，9. 权重初始化的方法，10. 神经元激活函数的种类，11. 参加训练模型数据的规模

说明：对于训练一个神经网络，一般只需要关注学习的回合数 Epoch，批量数据 batch 的大小，参加训练模型数据的规模即可

2、神经网络训练中的基本概念

欠拟合与过拟合：“泛化能力”



2、神经网络训练中的基本概念

优化（Optimization）：

求极值对应的参数的过程

优化器（Optimizer）：

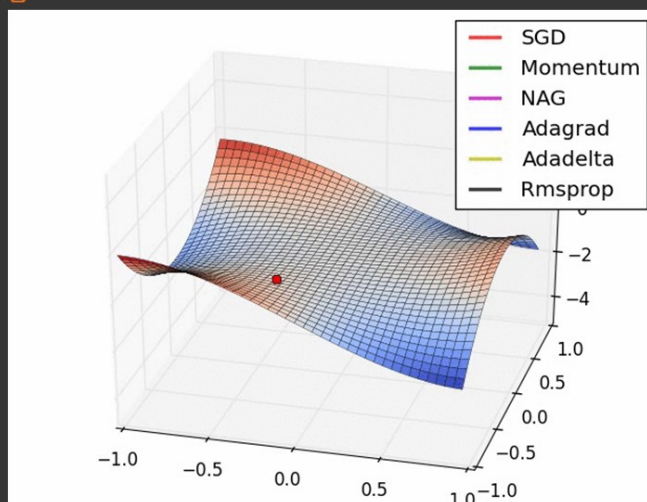
优化过程遵循的规则

标准梯度下降法

$$W_{t+1} = W_t - \eta_t \Delta J(W_t)$$

Adam：

$$\begin{cases} m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \\ v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \\ \hat{m}_t = \frac{m_t}{1 - \beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t} \\ W_{t+1} = W_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \end{cases}$$



参考：https://blog.csdn.net/weixin_40170902/article/details/80092628

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）：

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

总结一下，算法的步骤如下：

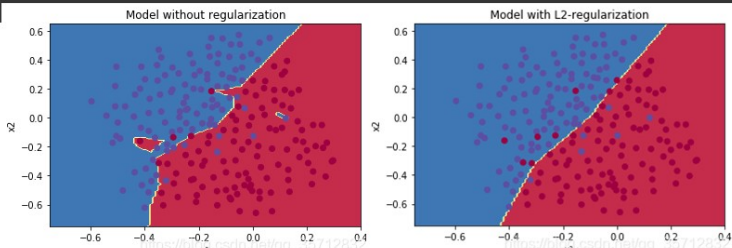
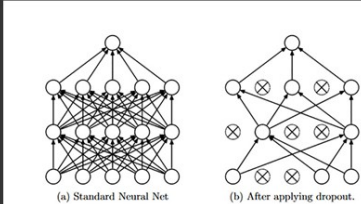
- (1) 初始化模型参数的值，如随机初始化；
- (2) 利用数据集在负梯度的方向上更新参数，并不断迭代这一步骤。

PART 2

2、神经网络训练中的基本概念

防止过拟合的方法：

- Dropout：随机性地在训练中冻结某些神经元的权重
- 正则化：权重较小的模型比权重较大的模型更简单。因此，通过对损失函数中权重的平方值进行惩罚，可以将所有权重驱动为较小的值，从而使模型更平滑，输出随着输入的变化而变化得更慢。有了正则化就不需要Dropout



Dropout

- 过拟合相当于一道走到黑，加入dropout，它强迫一个神经单元，和随机挑选出来的其他神经单元共同工作，达到好的效果。消除减弱了神经元节点间的联合适应性，增强了泛化能力。
- 过拟合中无限逼近某个噪声点，dropout将它从这个过程中拉出来

正则化

- $J(\theta) + \lambda R(w)$

2、神经网络训练中的基本概念

预训练与分段训练：

- “预训练”即先期使用大量普适性的数据先训练神经网络，训练出的权重一般会随代码公开；下载该模型的码农不需要重新对网络初始化参数结构从零训练，而只需在这个已有权重的基础上进行训练即可
- “预训练”（pre-training）的过程可以方便的让神经网络中的权值找到一个接近最优解的值，之后再使用“微调”（fine-tuning）来对整个网络进行优化训练。这两个技术的运用大幅度减少了训练多层神经网络的时间。此外，它也有有效减小有序训练数据过少带来的过拟合。
- 分段训练即并不训练所有网络参数，而只训练神经网络的部分层，如卷积神经网络输出前的数层全连接层。相较而言这些层完成的是综合分析判断而非特征提取的工作。

2、神经网络训练中的基本概念

数据增强：

有一批在有限场景中拍摄的数据集，但是目标应用可能存在于不同的条件，比如在不同的方向、位置、缩放比例、亮度等；所以可以通过额外合成的数据来训练神经网络来解释这些情况。

批标准化：

使输入具有相同的分布，而且还使每个输入都白化（白化是对原始数据 x 实现一种变换，变换成 x_{Whitened} ，使 x_{Whitened} 的协方差矩阵的为单位阵。）。该方法是由一些研究提出的，这些研究表明，如果对网络的输入进行白化，则网络训练收敛得更快，因此，增强各层输入的白化是网络的一个理想特性。（类比直方图均衡）



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

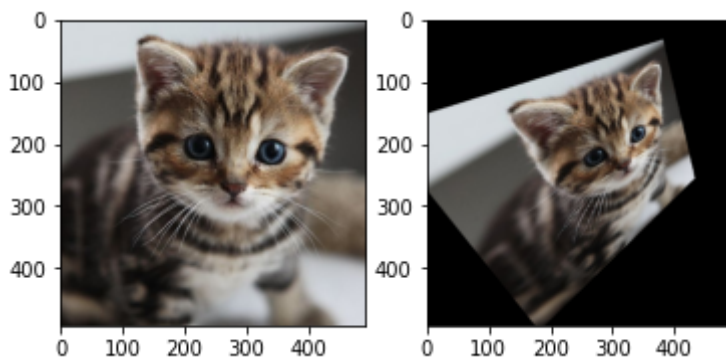
- [ILLUSTRATION OF TRANSFORMS](#)



```
1 from torchvision import transforms # 图像变换、数据预处理
2 import cv2
3 import matplotlib.pyplot as plt
4 #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
5 # 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
6 # transforms.RandomRotation(degrees=(0, 180))
7 transform =
transform.Compose([transforms.ToTensor(),transforms.RandomPerspective(distortion_scale=0.6, p=1.0),transforms.RandomRotation(degrees=(0, 180))])
```

```
1 cat_bgr = cv2.imread(r'./images/cat.jpg')
2 cat_rgb = cv2.cvtColor(cat_bgr, cv2.COLOR_BGR2RGB)
3 plt.subplot(1,2,1)
4 plt.imshow(cat_rgb)
5 cat_trans = transform(cat_rgb)
6 plt.subplot(1,2,2)
7 plt.imshow(cat_trans.numpy().transpose((1,2,0)))
```

```
1 <matplotlib.image.AxesImage at 0x1c310343c40>
```



梯度反向传播 (Back Propagation, BP)

- 参考链接: [Back Propagation \(梯度反向传播\) 实例讲解](#)
- 为什么要求梯度?
- 求关于谁的梯度?

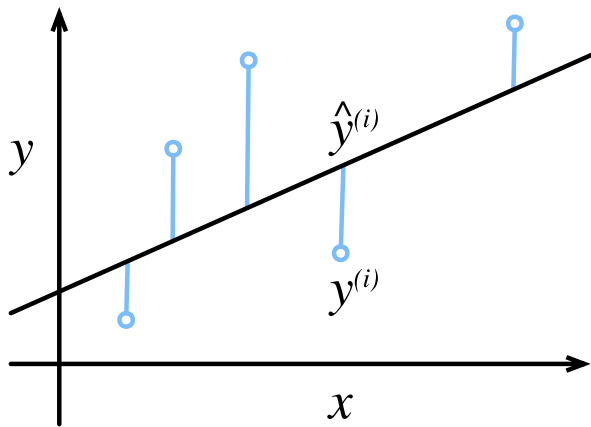
损失函数

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:

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



- 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）。

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

- 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：

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

梯度下降法 (SGD, Stochastic Gradient Descent)

$$w^+ = w - \eta \cdot \frac{\partial L}{\partial w} \quad (2)$$

理论依据

自动求导/数值求导

$$\frac{\partial f(x)}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} \quad (3)$$

链式法则

$$y = f(u), u = g(x) \quad \frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \frac{\partial u}{\partial x} \quad (4)$$

举例



- 误差

$$E = \frac{1}{2} (y - t)^2 \quad (5)$$

- 更新 w_5

$$\begin{aligned} \frac{\partial E}{\partial w_5} &= \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial w_5} \\ w_5^+ &= w_5 - \eta \cdot \frac{\partial E}{\partial w_5} \end{aligned} \quad (6)$$

- 更新 w_1

$$\frac{\partial E}{\partial w_1} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial h_1} \cdot \frac{\partial h_1}{\partial w_1} \quad (7)$$

$$w_1^+ = w_1 - \eta \cdot \frac{\partial E}{\partial w_1}$$

```

1 import numpy as np
2 import torch
3 # 定义单个神经元 2输入
4 net = torch.nn.Sequential(torch.nn.Linear(2,2),torch.nn.Linear(2,1))
5
6 # 定义权重
7 net[0].weight.data =
  torch.tensor([1,2,3,4]).type(torch.float32).reshape(net[0].weight.shape)
8 net[1].weight.data =
  torch.tensor([0.5,0.6]).type(torch.float32).reshape(net[1].weight.shape)
9
10 net[0].bias.data.fill_(0)
11 net[1].bias.data.fill_(0)
12
13 x = torch.tensor([1.0, 0.5]).type(torch.float32)
14 net(x)

```

```

1 | tensor([4.], grad_fn=<AddBackward0>)

```

```

1 # 数值求导举例
2 # 定义权重
3 net[0].weight.data =
  torch.tensor([0.5,1.5,2.3,3]).type(torch.float32).reshape(net[0].weight.shap
  e)
4 net[1].weight.data =
  torch.tensor([1,1]).type(torch.float32).reshape(net[1].weight.shape)
5 # y1 = 0.5*(4-net(x).item())**2
6 y1 = net(x).item()
7 dx = 0.001
8 net[0].weight.data =
  torch.tensor([0.5,1.5,2.3,3]).type(torch.float32).reshape(net[0].weight.shap
  e)
9 net[1].weight.data =
  torch.tensor([1+dx,1]).type(torch.float32).reshape(net[1].weight.shape)
10 # y2 = 0.5*(4-net(x).item())**2
11 y2 = net(x).item()
12 (y2-y1)/dx

```

```

1 | 1.2497901916503906

```

```

1 # 数值求导举例 y = x^T * x
2 import torch
3 x = torch.arange(4.0)
4 x.requires_grad_(True)
5 y = 2 * torch.dot(x, x)
6
7 y.backward()
8 x.grad, x.grad == 4 * x

```

```

1 (tensor([ 0.,  4.,  8., 12.]), tensor([True, True, True, True]))

```

Softmax激活函数



- 现在我们将优化参数以最大化观测数据的概率。
- 为了得到预测结果，我们将设置一个阈值，如选择具有最大概率的标签。
- 我们希望模型的输出 \hat{y}_j 可以视为属于类 j 的概率，然后选择具有最大输出值的类别 $\operatorname{argmax}_j y_j$ 作为我们的预测。
- 例如，如果 \hat{y}_1 、 \hat{y}_2 和 \hat{y}_3 分别为0.1、0.8和0.1，那么我们预测的类别是2，在我们的例子中代表“鸡”。
- 社会科学家邓肯·卢斯于1959年在*选择模型*（choice model）的理论基础上发明的softmax函数如下式：

$$\hat{y} = \operatorname{softmax}(\mathbf{o}) \quad \text{其中} \quad \hat{y}_j = \frac{\exp(o_j)}{\sum_k \exp(o_k)}$$

交叉熵损失函数

- 一般用于分类问题
- $$H(p, q) = - \sum_x p(x) \log q(x)$$
- 交叉熵刻画的是两个概率分布之间的距离，p代表正确答案，q代表的是预测值，交叉熵越小，两个概率的分布约接近

举例

- 假设有一个3分类问题，某个样例的正确答案是 (1, 0, 0)
- 甲模型经过softmax回归之后的预测答案是 (0.5, 0.2, 0.3)
- 乙模型经过softmax回归之后的预测答案是 (0.7, 0.1, 0.2)

$$l_n = -\log \frac{\exp(x_{n,y_n})}{\sum_{c=1}^C \exp(x_{n,c})}, y_n = y_{\text{target}} \quad (8)$$

```

1 import torch
2 def softmax(X):
3     x_exp = torch.exp(X)
4     partition = x_exp.sum(1, keepdim=True)
5     return x_exp / partition # 这里应用了广播机制
6 x = torch.normal(0, 1, (2, 5))
7 x_prob = softmax(x)

```

```

8 def cross_entropy(y_hat, y):
9     return - torch.log(y_hat[range(len(y_hat))], y])
10
11 y = torch.tensor([0, 2])
12 y_hat = torch.tensor([[0.1, 0.3, 0.6], [0.3, 0.2, 0.5]])
13
14
15 x_prob, x_prob.sum(1), y_hat[[0, 1], y], cross_entropy(y_hat, y)

```

```

1 (tensor([[0.0369, 0.0550, 0.0587, 0.5402, 0.3092],
2         [0.2373, 0.6351, 0.0845, 0.0074, 0.0357]]),
3  tensor([1.0000, 1.0000]),
4  tensor([0.1000, 0.5000]),
5  tensor([2.3026, 0.6931]))

```

卷积神经网络

- 深度学习是指多层神经网络上运用各种机器学习算法解决图像，文本等各种问题的算法集合。
- 深度学习的核心是特征学习，旨在通过分层网络获取分层次的特征信息，从而解决以往需要人工设计特征的重要难题。
- 人类的视觉原理如下：
 - 从原始信号摄入开始（瞳孔摄入像素 Pixels）
 - 接着做初步处理（大脑皮层某些细胞发现边缘和方向）
 - 然后抽象（大脑判定，眼前的物体的形状，是圆形的）
 - 然后进一步抽象（大脑进一步判定该物体是只气球）。



概念

- 卷积神经网络是一种多层神经网络，擅长处理图像特别是大图像的相关机器学习问题。CNN通过卷积来模拟特征区分，并且通过卷积的权值共享及池化，来降低网络参数的数量级，最后通过传统神经网络完成分类等任务。
- 卷积神经网络是一种自动化特征提取的机器学习模型，主要用于解决图像识别问题。
- 从直观上讲，是一个从细节到抽象的过程。这里的关键是如何抽象，抽象就是把图像中的各种零散的特征通过某种方式汇总起来，形成新的特征。深度学习网络最上层的特征是最抽象的。

降低网络参数数量级

- 神经网络结构：在该结构中，输入是一个向量，然后在一系列的隐层中对它做变换，如果一个尺寸为200x200x3的图像，采用全连接方式会让每个神经元包含200x200x3=120,000个权重值，这种全连接方式效率低下，大量的参数也很快会导致网络过拟合。

卷积层：整合输入数据并从中提取有效数据，得到特征图。

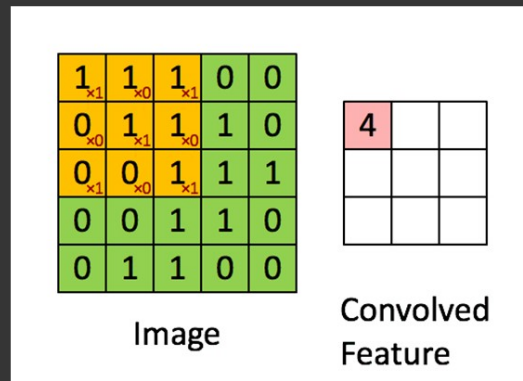
- 在这个卷积层，有两个关键操作：局部关联与窗口滑动。局部关联将每个神经元看作一个滤波器，窗口滑动则是使用滤波器对局部数据计算。
- 局部感知：人的大脑识别图片的过程中，并不是一下子整张图同时识别，而是对于图片中的每一个特征首先局部感知，然后更高层次对局部进行综合操作，从而得到全局信息。
- 卷积运算一个重要的特点就是，通过卷积运算，可以使原信号特征增强，并且降低噪音

- 这个过程我们可以理解为我们使用一个过滤器（卷积核）来过滤图像的各个小区域，从而得到这些小区域的特征值。
- 在实际训练过程中，卷积核的值是在学习过程中学到的。

PART 3

1、卷积神经网络基础-卷积核

- 卷积核就是图像处理时，给定输入图像，输入图像中一个小区域中像素加权平均后成为输出图像中的每个对应像素，其中权值由一个函数定义，这个函数称为卷积核。
- 卷积核具有大小和步长
- 使用卷积完成的神经网络层称为卷积层



- 使用多个滤波器，可以通过卷积将一维的输入变为四维，每一维代表一个特征。
- 每个卷积核的大小都是3维 ($x * y * n$)，代表长、宽和输入的深度，而其本身的深度是指卷积核个数，设为 k ，则该层卷积核参数个数为 ($x * y * n * k$)

image.png

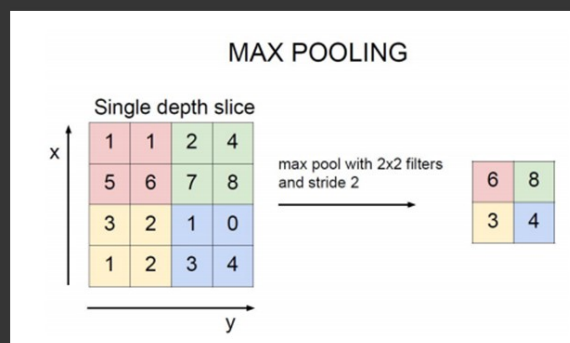
池化层（降采样层）：池化层能够压缩数据，提取明显特征。

PART 3

1、卷积神经网络基础-池化层

池化（Pooling）：也称为欠采样或下采样（Down-Sampling）。主要用于特征降维，压缩数据和参数的数量，减小过拟合，同时提高模型的容错性。主要有：

- Max Pooling：最大池化
- Average Pooling：平均池化



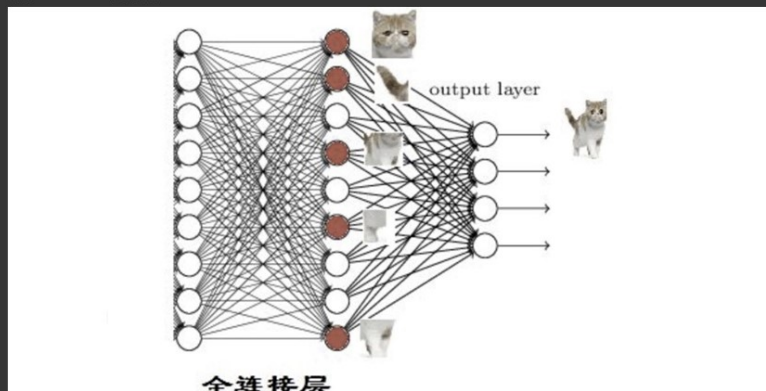
- 池化层位于连续的卷积层中间，本身没有可训练的参数。池化层的目的在于对卷积层提取的特征进行降维，减少整个神经网络的参数。
- 感受野：一个像素对应回原图的区域大小，假如没有pooling，一个 $3 * 3$ ，步长为1的卷积，那么输出的一个像素的感受野就是 $3 * 3$ 的区域，再加一个stride=1的 $3 * 3$ 卷积，则感受野为 $5 * 5$ 。

全连接层：将两层之间所有神经元都有权重连接。

PART 3

1、卷积神经网络基础-全连接层

全连接层的作用主要是分析综合并输出，而之前的卷积层的作用主要是特征提取。最后一层卷积或池化得到的矩阵信息拉直成一维后即可输入全连接层。



- 全连接层将两层之间所有神经元都有权重连接，通常全连接层在卷积神经网络尾部，与传统的神经网络神经元的连接方式是一样的。
- 经过几轮卷积和池化操作，可以认为图像中的信息已经被**抽象成了信息含量更高的特征**。我们可以将卷积和池化看成自动图像提取的过程，在特征提取完成后，仍然需要使用全连接层来完成分类任务

实用举例

```
1 | !pip install torchsummary
```

```
1 | Requirement already satisfied: torchsummary in d:\miniconda3\lib\site-packages (1.5.1)
```

引入需要的库

```
1 | import torch
2 | import torch.nn as nn
3 | import torch.nn.functional as F
4 | import numpy as np
5 | import time
6 | import torch.optim as optim
7 |
8 | import torchvision
9 | import torch.utils.data as Data
10 | import torchvision.transforms as transforms
11 | from torchvision.datasets import MNIST, FashionMNIST
12 | print('GPU useable state: ', torch.cuda.is_available())
```

```
1 | GPU useable state: False
```


加载数据

```
1 transform = transforms.Compose(  
2     [transforms.ToTensor()])  
3     # transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))]])  
4  
5  
6  
7 # MNIST  
8 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  
9  
10         transform = transform, ## 图像变换操作  
11         train = True, ## 决定使用训练集还是测试集  
12         download = True) ## 选择是否需要下载数据  
13  
14 data_test = MNIST(root = "data/",  
15                   transform = transform,  
16                   train = False)  
17  
18 ## 定义数据加载器  
19 batch_size = 256  
20 train_loader = Data.DataLoader(dataset = data_train, ## 使用的数据集  
21                               batch_size = batch_size, ## 批处理  
   样本大小  
22                               shuffle = True ## 是否打乱数据顺序  
23                               )  
24 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

卷积神经网络模型

```
1 class ConvNet(nn.Module):  
2     def __init__(self, ch = 3, h = 32, w = 32):  
3         super(ConvNet, self).__init__()  
4         # 3 input image channel, 6 output channels, 5x5 square convolution  
5         # kernel  
6         self.conv1 = nn.Sequential(  
7             nn.Conv2d(  
8                 in_channels = ch,  
9                 out_channels = 6,  
10                kernel_size = 5), ## (N*1* h * w ) ==> (N*6* (h-4) * (w-4))  
11            nn.ReLU(),  
12            nn.AvgPool2d(  
13                kernel_size = 2,  
14                stride = 2) ## (N*6* (h-4) * (w-4)) ==> (N*6* (w-4)/2 *  
   (w-4)/2)  
15        )  
16  
17        self.conv2 = nn.Sequential(  
18            nn.Conv2d(6, 16, 5), ## (N*6* (h-4)/2 * (w-4)/2) ==> (N*16* (h-  
   4)/2-4 * (w-4)/2-4)  
19            nn.ReLU(),
```

```

20         nn.AvgPool2d(2,2)      ## (N*16* (h-4)/2-4 * (w-4)/2-4) ==> (N*16*
h/4 -3 * w/4 -3)
21     )
22
23     self.classifier = nn.Sequential(
24         nn.Linear(int(16*(h/4-3)*(w/4-3)),120), # 5
25         nn.ReLU(),
26         nn.Linear(120,84),
27         nn.ReLU(),
28         nn.Linear(84,10))
29
30     def forward(self, x):
31         # Max pooling over a (2, 2) window
32         x = self.conv1(x)
33         # If the size is a square, you can specify with a single number
34         x = self.conv2(x)
35         x = torch.flatten(x, 1) # flatten all dimensions except the batch
dimension
36         x = self.classifier(x)
37         return x

```

MLP (多层感知机、全连接网络)

```

1 class dnn(nn.Module):
2     def __init__(self):
3         super(dnn,self).__init__()
4         ## 定义层
5         self.flatten = nn.Flatten()
6         self.hidden1 = nn.Linear(in_features = 784, out_features = 512, bias
= True)
7         self.active1 = nn.ReLU() ## 激活函数
8         self.drop1 = nn.Dropout(p=0.2)
9         self.hidden2 = nn.Linear(512,512)
10        self.active2 = nn.ReLU()
11        self.drop2 = nn.Dropout(p=0.2)
12        self.fc = nn.Linear(512,10)
13    def forward(self,x):
14        x = self.flatten(x)
15        x = self.hidden1(x)
16        x = self.active1(x)
17        x = self.drop1(x)
18        x = self.hidden2(x)
19        x = self.active2(x)
20        x = self.drop2(x)
21        x = self.fc(x)
22        return x

```

实例化网络模型

```

1 net = dnn()
2 device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
3 net.to(device)
4 print(device)
5 from torchsummary import summary
6 summary(net,(1,28,28))

```

```

1 cpu
2 -----
3           Layer (type)           Output Shape         Param #
4 =====
5           Flatten-1              [-1, 784]              0
6           Linear-2                [-1, 512]            401,920
7           ReLU-3                  [-1, 512]              0
8           Dropout-4               [-1, 512]              0
9           Linear-5                [-1, 512]            262,656
10          ReLU-6                  [-1, 512]              0
11          Dropout-7               [-1, 512]              0
12          Linear-8                [-1, 10]              5,130
13 =====
14 Total params: 669,706
15 Trainable params: 669,706
16 Non-trainable params: 0
17 -----
18 Input size (MB): 0.00
19 Forward/backward pass size (MB): 0.03
20 Params size (MB): 2.55
21 Estimated Total Size (MB): 2.59
22 -----

```

```

1 net = ConvNet(1,28,28)
2 device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
3 net.to(device)
4 print(device)
5 from torchsummary import summary
6 summary(net,(1,28,28))

```

```

1 cpu
2 -----
3           Layer (type)           Output Shape         Param #
4 =====
5           Conv2d-1                [-1, 6, 24, 24]       156
6           ReLU-2                  [-1, 6, 24, 24]        0
7           AvgPool2d-3              [-1, 6, 12, 12]        0
8           Conv2d-4                [-1, 16, 8, 8]        2,416
9           ReLU-5                  [-1, 16, 8, 8]         0
10          AvgPool2d-6              [-1, 16, 4, 4]         0
11          Linear-7                 [-1, 120]             30,840
12          ReLU-8                   [-1, 120]              0
13          Linear-9                 [-1, 84]              10,164
14          ReLU-10                  [-1, 84]               0

```

```

15      Linear-11                [-1, 10]                850
16      =====
17      Total params: 44,426
18      Trainable params: 44,426
19      Non-trainable params: 0
20      -----
21      Input size (MB): 0.00
22      Forward/backward pass size (MB): 0.08
23      Params size (MB): 0.17
24      Estimated Total Size (MB): 0.25
25      -----

```

```

1  ## 初始化模型参数
2  def init_weights(m):
3      if type(m) == nn.Conv2d:
4          nn.init.normal_(m.weight, mean=0, std=0.5)
5      if type(m) == nn.Linear:
6          nn.init.normal_(m.weight, std=0.01)
7
8  net.apply(init_weights);

```

定义模型训练函数

```

1  import pandas as pd
2
3  def train_model(model, train_loader, test_loader, loss_func, optimizer,
4                  device, num_epochs = 5):
5      """
6      model: 网络模型;          train_loader: 训练数据集; test_loader: 测试数据集
7      loss_func: 损失函数; optimizer: 优化方法;          num_epochs: 训练的轮数
8      device: 控制是否使用GPU
9      """
10     train_loss_all = []
11     train_acc_all = []
12     val_acc_all = []
13     val_loss_all = []
14
15     gap = len(train_loader)//30
16     start_time = time.time()
17
18     for epoch in range(num_epochs):
19         if epoch:
20             print('-'*10)
21             print("Epoch {}/{}".format(epoch + 1, num_epochs))
22
23         ##
24         train_loss = 0.0
25         train_corrects = 0
26         train_num = 0
27
28         val_loss = 0.0
29         val_corrects = 0

```

```

29     val_num = 0
30
31     for step,data in enumerate(train_loader):
32         model.train()
33         x,y = data[0].to(device), data[1].to(device)
34         output = model(x) ## 模型在 X 上的输出: N * num_class
35         pre_lab = torch.argmax(output, 1) ## 获得预测结果
36         loss = loss_func(output, y) ## 损失
37         optimizer.zero_grad() ## 每次迭代将梯度初始化为0
38         loss.backward() ## 损失的后向传播, 计算梯度
39         optimizer.step() ## 使用梯度进行优化
40         train_loss += loss.item() * x.size(0) ## 统计模型预测损失
41         train_corrects += torch.sum(pre_lab == y.data)
42         train_num += x.size(0)
43
44         if step % gap == gap-1:
45             cont = step//gap
46             if cont > 30:
47                 cont = 30
48             print(str(cont)+'/', '30', '['+'='*cont+'>'+'-'*(30-cont)+''],
49                   'loss: {:.4f} - accuracy:
{:.4f}'.format(train_loss/train_num,train_corrects.double().item()/train_num)
50                   ,end="\r")
51
52     for data in test_loader:
53         model.eval()
54         X_test, y_test = data[0].to(device), data[1].to(device)
55         X_test.requires_grad=True
56         with torch.no_grad():
57             output = model(X_test)
58             test_loss = loss_func(output, y_test )
59             _, pred = torch.max(output.data, 1)
60             val_corrects += torch.sum(pred == y_test.data)
61             val_loss += test_loss.item()*X_test.size(0)
62             val_num += X_test.size(0)
63
64     ##
65     train_loss_all.append(train_loss/train_num)
66     train_acc_all.append(train_corrects.double().item()/train_num)
67     val_loss_all.append(val_loss/val_num)
68     val_acc_all.append(val_corrects.double().item()/val_num)
69
70     print('')
71     print("No.{ } Train Loss is:{:.4f}, Train_accuracy is {:.4f}%"
72           .format(epoch+1, train_loss_all[-1],train_acc_all[-1] * 100))
73     print("No.{ } Val Loss is:{:.4f}, Val_accuracy is {:.4f}%"
74           .format(epoch+1, val_loss_all[-1], val_acc_all[-1] * 100))
75
76     time_use = time.time() - start_time
77     print("Train and val complete in {:.0f}m
{:.0f}s".format(time_use//60, time_use%60))
78
79     train_process = pd.DataFrame(
80         data = {"epoch":range(num_epochs),
81               "train_loss":train_loss_all,

```



```

82         "train_acc":train_acc_all,
83         "val_loss":val_loss_all,
84         "val_acc":val_acc_all})
85     return model, train_process

```

```

1 criterion = nn.CrossEntropyLoss()
2 optimizer = optim.SGD(net.parameters(), lr=0.1, momentum=0.9)

```

```

1 net, train_process = train_model(net, train_loader, test_loader, criterion,
  optimizer, device, num_epochs = 10)

```

```

1 Epoch 1/10
2 30/ 30 [=====>] loss: 2.3930 - accuracy: 0.1195
3 No.1 Train Loss is:2.3918, Train_accuracy is 11.9200%
4 No.1 Val Loss is:2.3016, val_accuracy is 11.3500%
5 Train and val complete in 0m 7s
6 -----
7 Epoch 2/10
8 30/ 30 [=====>] loss: 2.3022 - accuracy: 0.1095loss:
  2.3022 - accuracy: 0.108730 [=====>] loss: 2.3022 -
  accuracy: 0.1093
9 No.2 Train Loss is:2.3022, Train_accuracy is 10.9667%
10 No.2 Val Loss is:2.3020, val_accuracy is 11.3500%
11 Train and val complete in 0m 14s
12 -----
13 Epoch 3/10
14 30/ 30 [=====>] loss: 2.3022 - accuracy: 0.1111loss:
  2.3022 - accuracy: 0.1100
15 No.3 Train Loss is:2.3022, Train_accuracy is 11.1017%
16 No.3 Val Loss is:2.3018, val_accuracy is 11.3500%
17 Train and val complete in 0m 21s
18 -----
19 Epoch 4/10
20 30/ 30 [=====>] loss: 2.3022 - accuracy: 0.1104
21 No.4 Train Loss is:2.3022, Train_accuracy is 11.0283%
22 No.4 Val Loss is:2.3019, val_accuracy is 11.3500%
23 Train and val complete in 0m 28s
24 -----
25 Epoch 5/10
26 30/ 30 [=====>] loss: 2.3021 - accuracy: 0.1113
27 No.5 Train Loss is:2.3022, Train_accuracy is 11.1300%
28 No.5 Val Loss is:2.3013, val_accuracy is 11.3500%
29 Train and val complete in 0m 36s
30 -----
31 Epoch 6/10
32 30/ 30 [=====>] loss: 2.3022 - accuracy: 0.11140
  [=====>-----] loss: 2.3017 - accuracy: 0.1146
33 No.6 Train Loss is:2.3022, Train_accuracy is 11.1350%
34 No.6 Val Loss is:2.3014, val_accuracy is 11.3500%
35 Train and val complete in 0m 43s
36 -----
37 Epoch 7/10
38 30/ 30 [=====>] loss: 2.3020 - accuracy: 0.1113

```

```

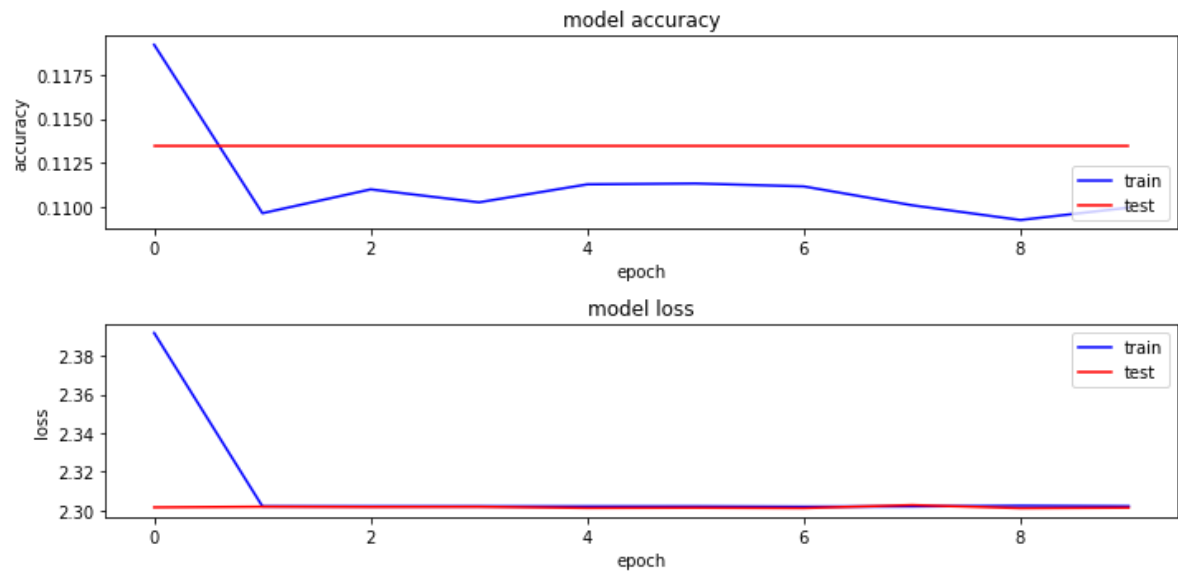
39 No.7 Train Loss is:2.3020, Train_accuracy is 11.1183%
40 No.7 Val Loss is:2.3012, val_accuracy is 11.3500%
41 Train and val complete in 0m 50s
42 -----
43 Epoch 8/10
44 30/ 30 [=====>] loss: 2.3021 - accuracy: 0.1100
   [=====>-----] loss: 2.3019 - accuracy: 0.1109
45 No.8 Train Loss is:2.3021, Train_accuracy is 11.0117%
46 No.8 Val Loss is:2.3028, val_accuracy is 11.3500%
47 Train and val complete in 0m 57s
48 -----
49 Epoch 9/10
50 30/ 30 [=====>] loss: 2.3025 - accuracy: 0.1090
51 No.9 Train Loss is:2.3025, Train_accuracy is 10.9283%
52 No.9 Val Loss is:2.3012, val_accuracy is 11.3500%
53 Train and val complete in 1m 5s
54 -----
55 Epoch 10/10
56 30/ 30 [=====>] loss: 2.3022 - accuracy: 0.1100
57 No.10 Train Loss is:2.3022, Train_accuracy is 10.9983%
58 No.10 Val Loss is:2.3014, val_accuracy is 11.3500%
59 Train and val complete in 1m 14s

```

```

1  # plotting the metrics
2  fig = plt.figure(figsize=(10,5))
3  plt.subplot(2,1,1)
4  plt.plot(train_process.train_acc, "b-")
5  plt.plot(train_process.val_acc, "r-")
6  plt.title('model accuracy')
7  plt.ylabel('accuracy')
8  plt.xlabel('epoch')
9  plt.legend(['train', 'test'], loc='lower right')
10
11 plt.subplot(2,1,2)
12 plt.plot(train_process.train_loss, "b-")
13 plt.plot(train_process.val_loss, "r-")
14 plt.title('model loss')
15 plt.ylabel('loss')
16 plt.xlabel('epoch')
17 # plt.legend(['train'], loc='upper right')
18 plt.legend(['train', 'test'], loc='upper right')
19
20 plt.tight_layout()

```



```

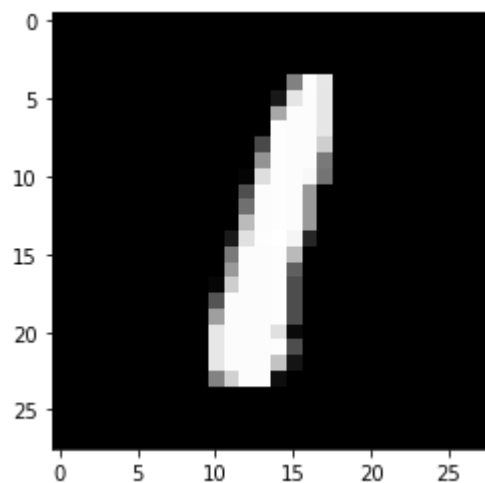
1 for x,y in test_loader:
2     break
3 plt.imshow(x[0].squeeze().numpy(), 'gray')

```

```

1 <matplotlib.image.AxesImage at 0x1c3128927f0>

```



```

1 output = net.conv1(x[0].unsqueeze(dim=0).to(device))
2 for i in range(6):
3     plt.subplot(2,3,i+1)
4     plt.imshow(output[0][i].squeeze().detach().cpu().numpy(), 'gray')

```

