基于神经网络原理从零开始训练神经网络模型

朱海源 / 2024-07

CONTENTS

期望收获

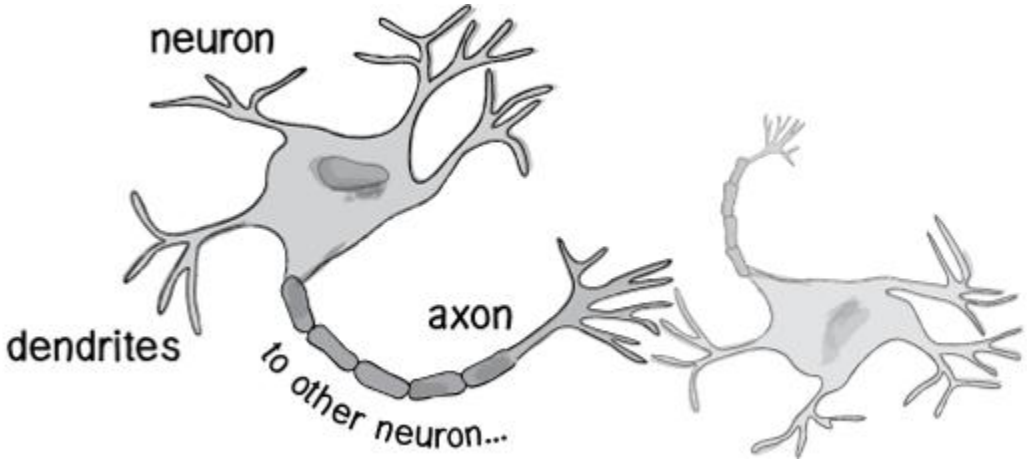
- ✓ 了解神经网络原理
- ✓ 了解神经网络能做什么
- ✓ 了解如何手写一个简单的基于神经网络的模型
- ✓ 利用神经网络训练手写体数字识别模型
- ✓ 利用神经网络训练猫狗图片识别模型
- ✓ ……

01

原理介绍

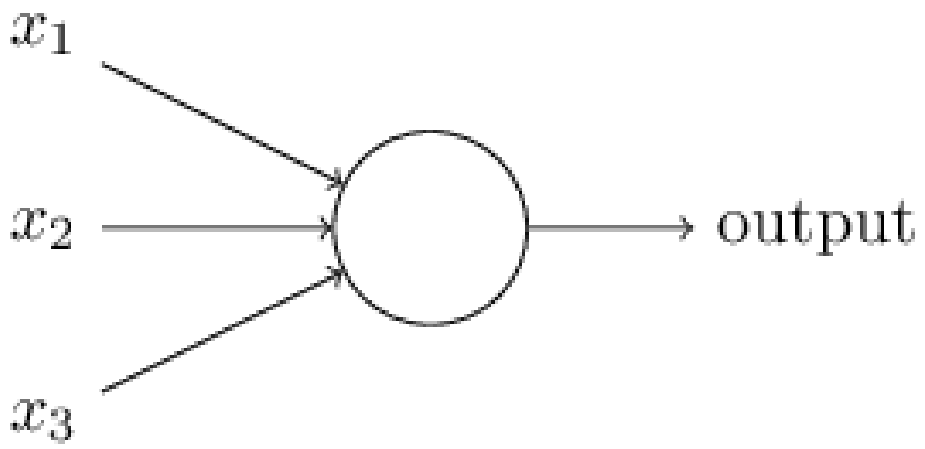
人为什么能够思考？

人体的神经网络



外部刺激通过神经末梢，转化为电信号，转导到神经细胞（又叫神经元）。
无数神经元构成神经中枢。神经中枢综合各种信号，做出判断。人体根据神经中枢的指令，对外部刺激做出反应。

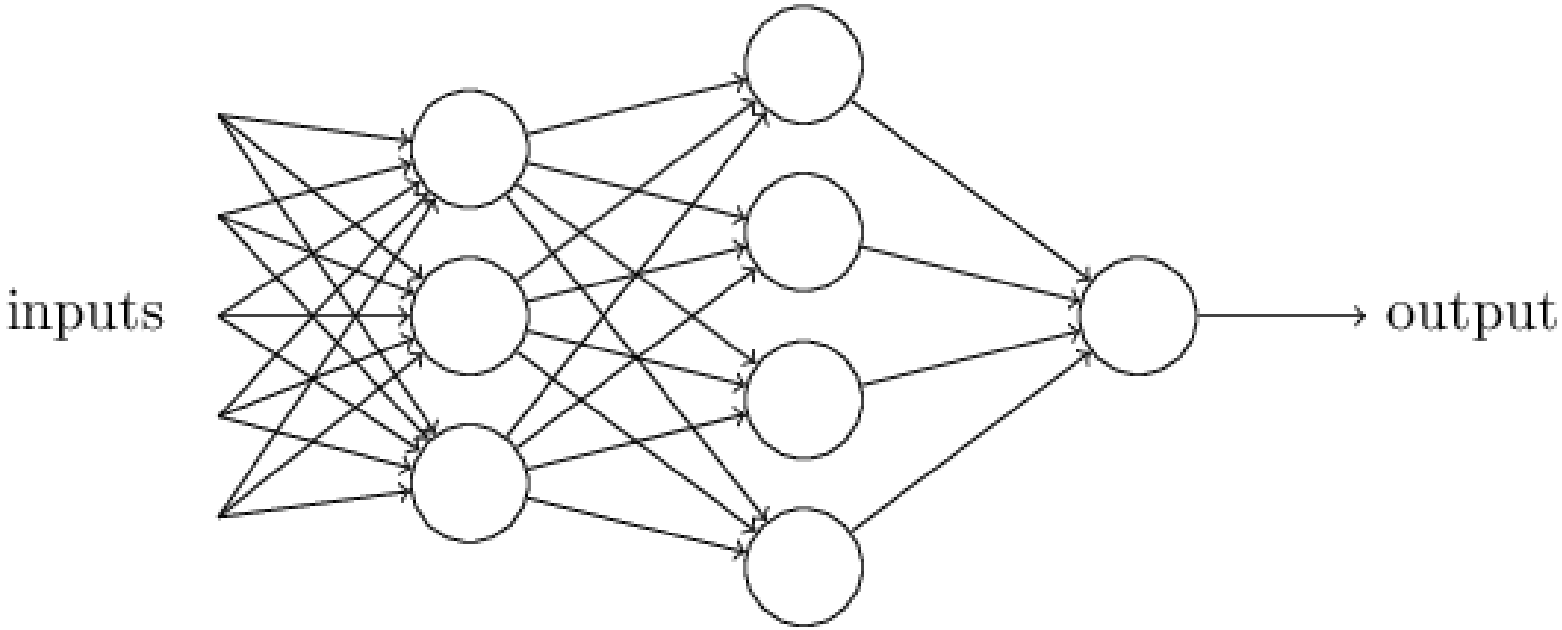
人造神经元



大脑决策“今天加班吗？”

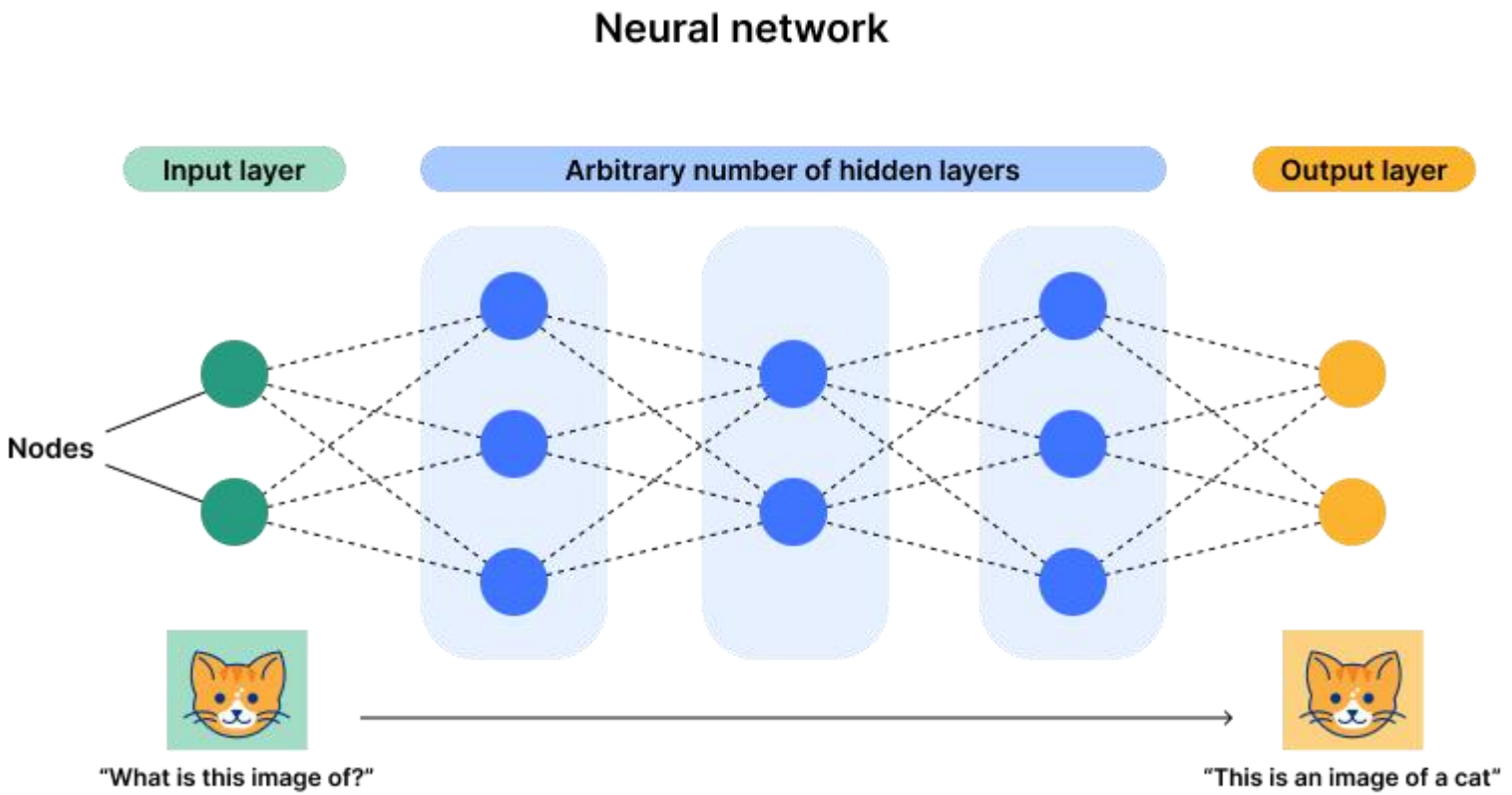
- 有点累，正常下班 0
- 活确实不少，不加班干不完 1
- 晚上有个饭局，六点下班 0

实际决策



神经网络是一种人工智能方法，用于教计算机以受人脑启发的方式处理数据。这是一种机器学习过程，称为深度学习，它使用类似于人脑的分层结构中的互连节点或神经元。

01 什么是神经网络



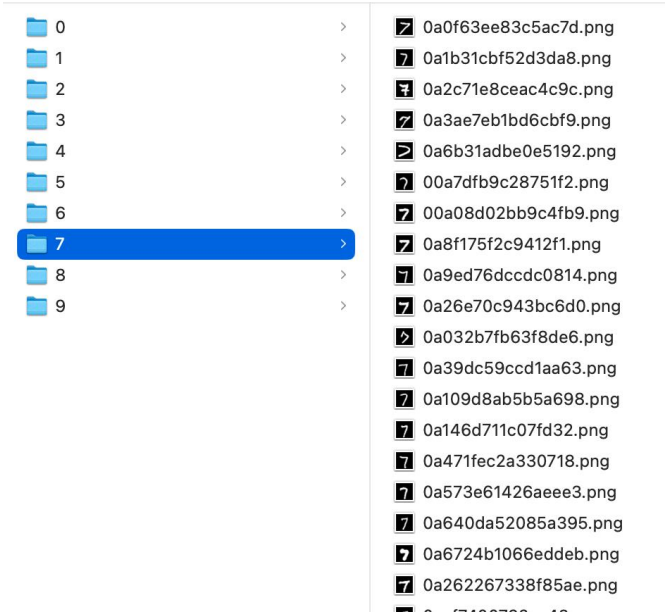
每个神经网络都由多个节点层或人工神经元组成 – 一个输入层、一个或多个隐藏层和一个输出层。每个节点都与其他节点相连，具有一个关联的权重和阈值。如果任何单个节点的输出高于指定的阈值，那么该节点将被激活，并将数据发送到网络的下一层。否则，不会将数据传递到网络的下一层。

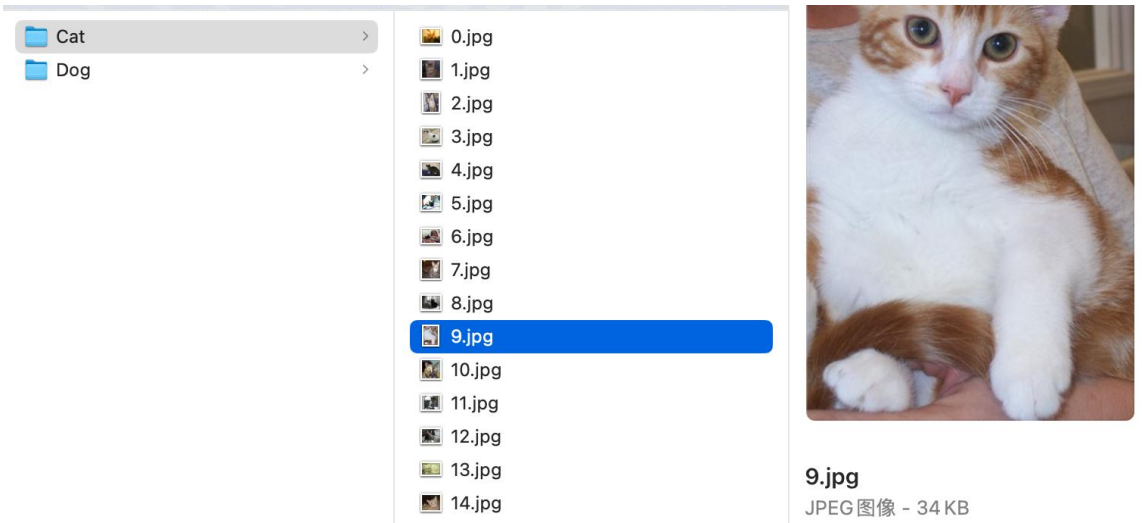
什么是神经网络

02

训练准备

案例数据集





在有监督学习中，为人工神经网络提供标记数据集，这些数据集提前提供正确答案。

Datasets

02 加载 Datasets

```python
from datasets import load_dataset, DatasetDict  # 导入所需的库和类

# 使用 load_dataset 加载训练集和验证集
ds_train = load_dataset("huggingface-course/codeparrot-ds-train", split="train")  # 加载训练集数据
ds_valid = load_dataset("huggingface-course/codeparrot-ds-valid", split="validation")  # 加载验证集数据

# 将加载的数据组织为 DatasetDict 结构
raw_datasets = DatasetDict(
    {
        "train": ds_train,  # 训练集数据
        "valid": ds_valid,  # 验证集数据
        # 你也可以在这里添加其他键值对，如测试集等
    }
)

raw_datasets  # 展示 DatasetDict 结构的数据集
```

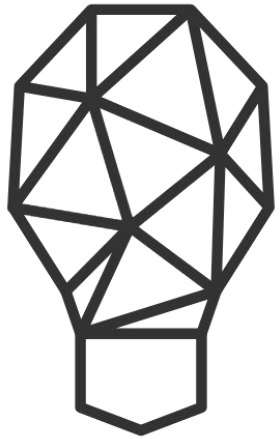
上节课内容

MNIST 数据集

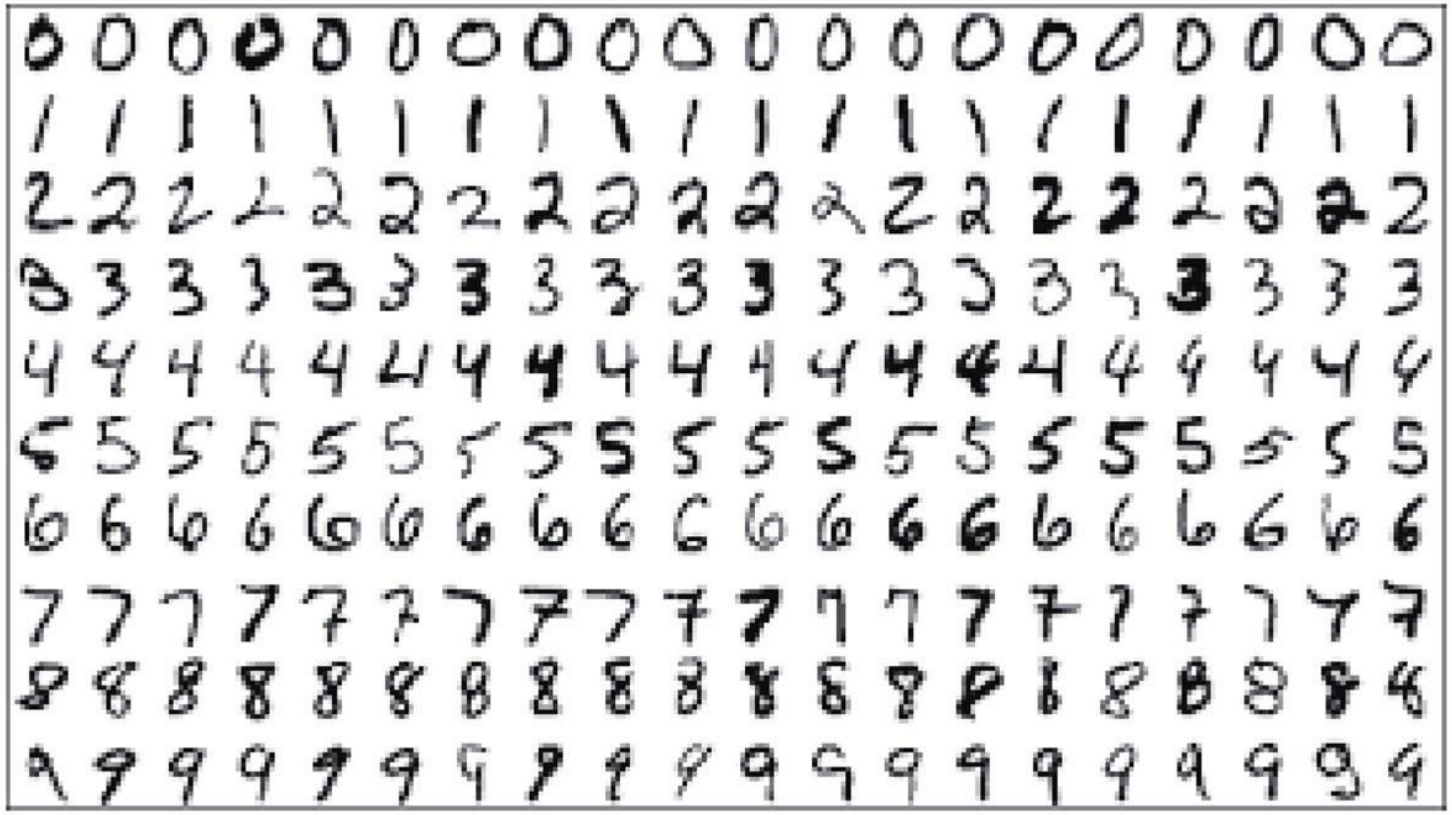
MNIST（修改后的美国国家标准与技术研究院）数据集是一个大型手写数字数据库，常用于训练各种图像处理系统和机器学习模型。它是通过对 NIST 原始数据集中的样本进行 "重新混合 "而创建的，已成为评估图像分类算法性能的基准。

主要功能
MNIST 包含 60,000 张手写数字训练图像和 10,000 张测试图像。
数据集由大小为 28x28 像素的灰度图像组成。
对图像进行归一化处理，使其适合 28x28 像素的边界框，并进行抗锯齿处理，引入灰度级。
MNIST 广泛用于机器学习领域的训练和测试，尤其是图像分类任务。
数据集结构
MNIST 数据集分为两个子集：

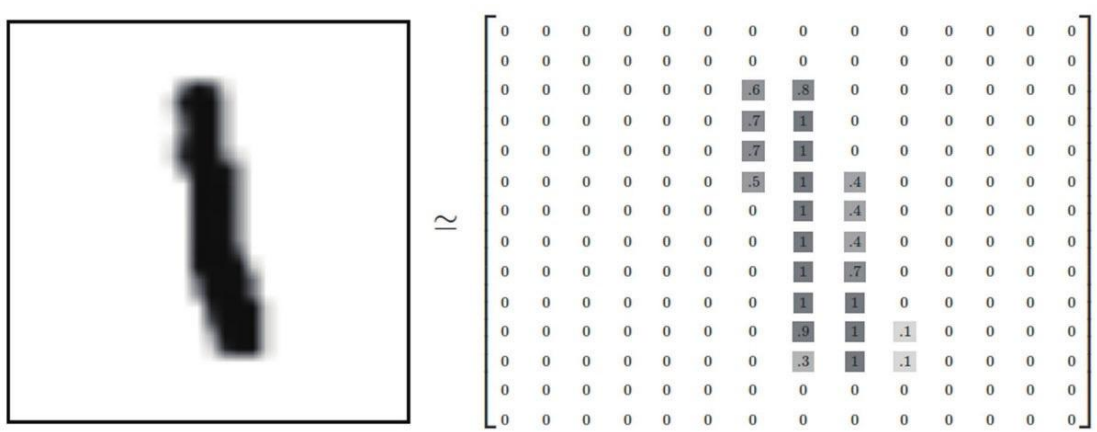
训练集：该子集包含 60,000 张手写数字图像，用于训练机器学习模型。
测试集：该子集由 10,000 张图像组成，用于测试和基准测试训练有素的模型



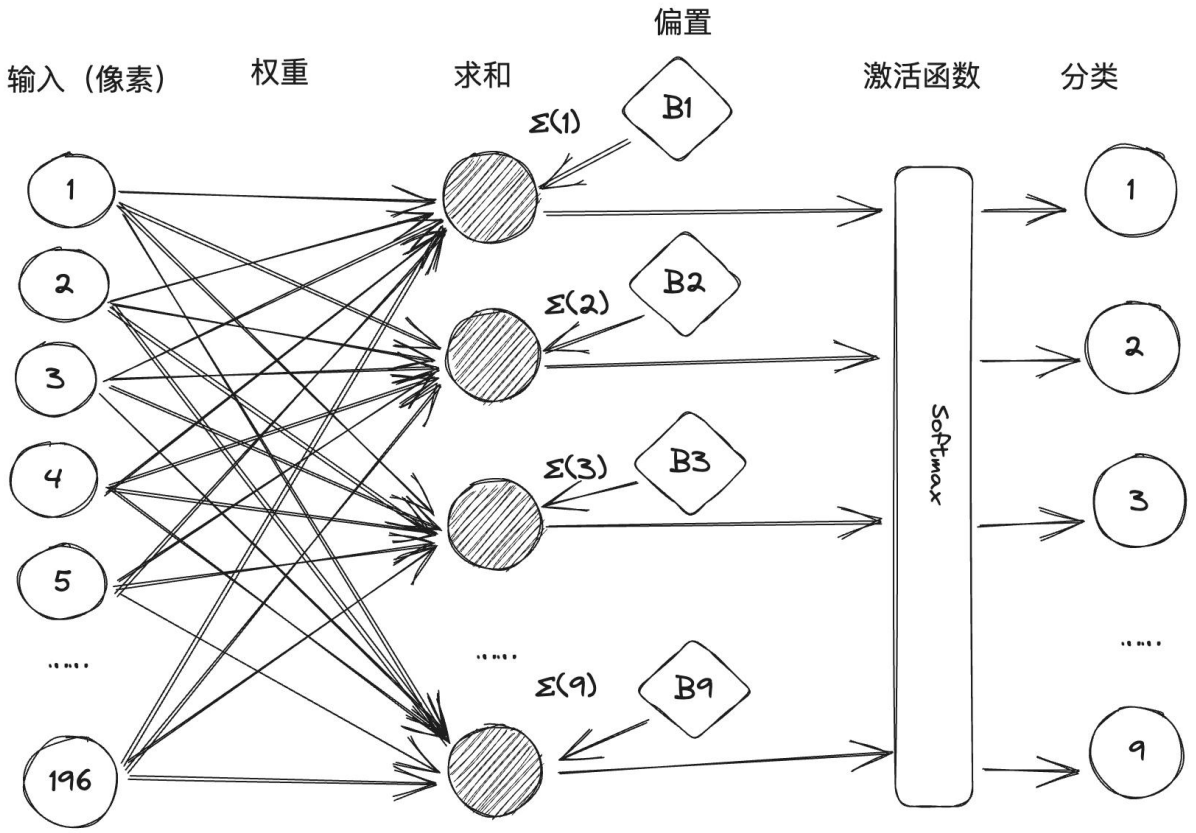
理论准备



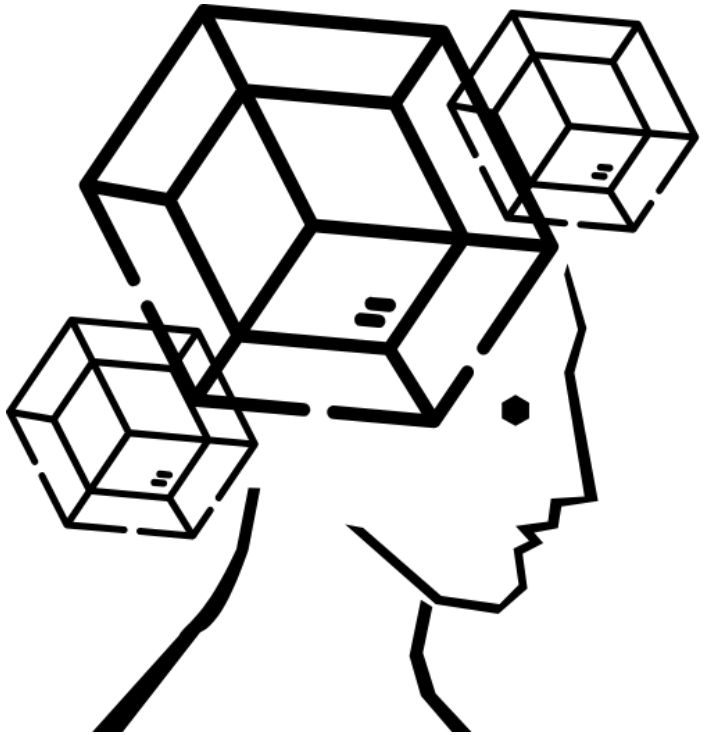
假设任务目标是自动识别阿拉伯数字，待识别的数字是将手写或印刷的各种形式的数字，将数字通过扫描后存储在14*14（28*28）像素大小的图片文件中



将黑色像素用1来表示，白色像素用0表示，介于黑白间的灰度像素根据其灰度强度用0-1间的浮点数表示。



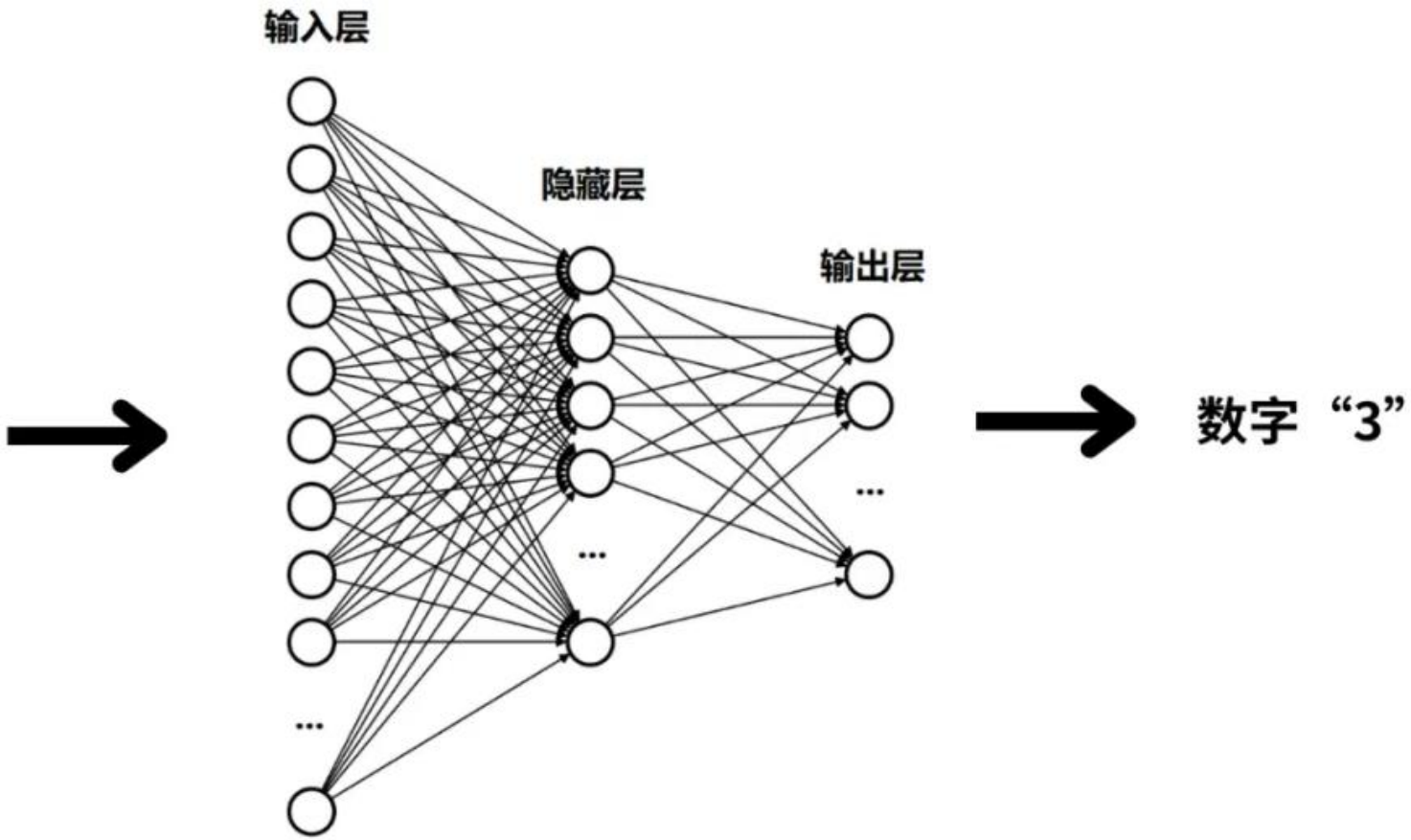
比如对于数字"0"的图片中间点的像素不应该有黑色（1）像素，如果出现了则表明该图片属于数字0为负面证据，就降低该图片是数字0的概率。这样，经过对数据集的训练和校准，就可以得到14*14（=196）每个像素对应0-9各数字的权重分布。



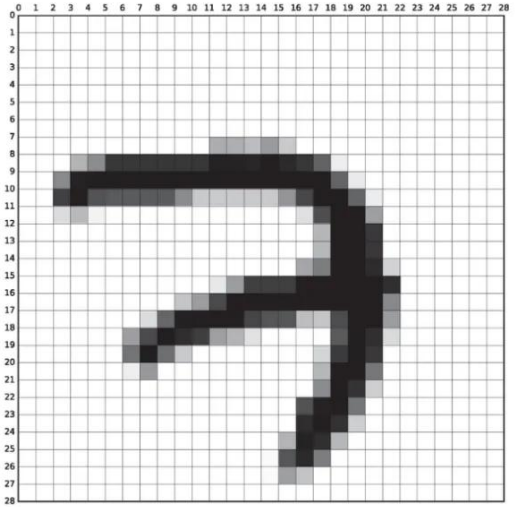
需求分析

整体需求

0a4d5bd4a6947c10.png
PNG图像 - 269字节
信息 更少
创建时间
修改时间
上次打开时间 --
内容创建时间
尺寸 28×28
色彩空间 Gray



图像数据预处理



28×28=784

$$\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_{783} \end{bmatrix}$$

1×784

```python
# 图像的预处理
# transforms.Compose()将多个转换操作组合在一起
transform = transforms.Compose([
    transforms.Grayscale(num_output_channels=1),  # 转换为单通道灰度图
    transforms.ToTensor()  # 转换为张量
])
```

构建数据集对象

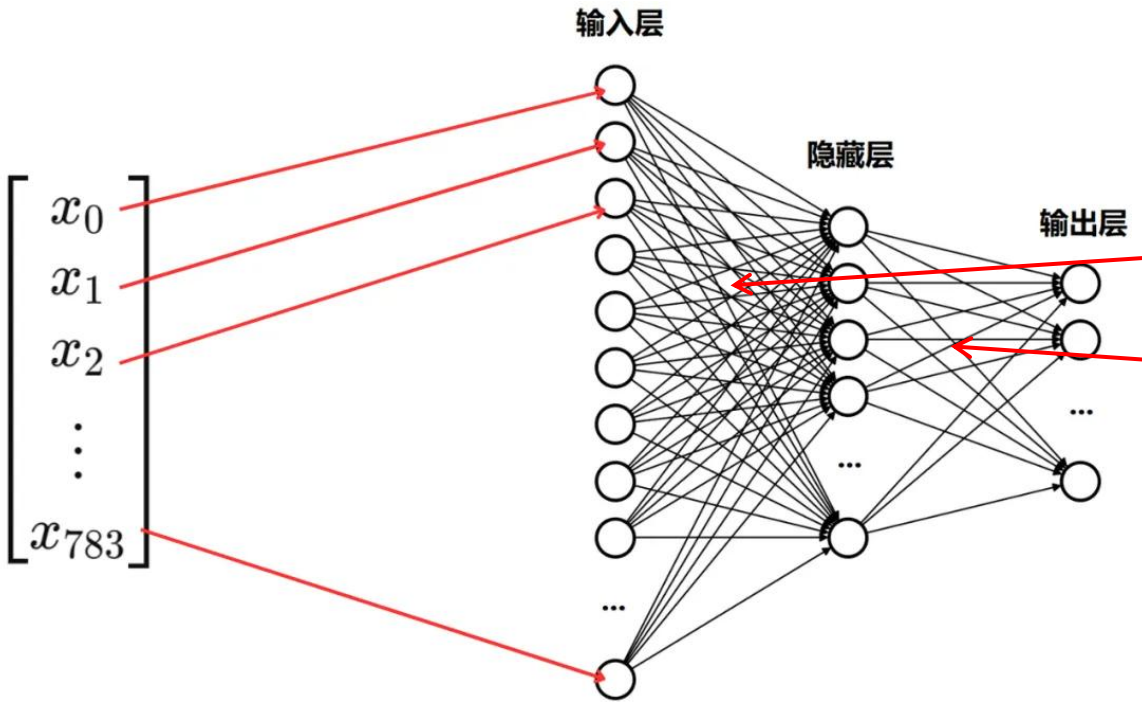
对于名字为 0 的文件夹，就会将 0 作为文件夹中的图像数据的标签

```
# 读取并构造数据集
train_dataset = datasets.ImageFolder(root='./mnist_images/train',
transform=transform)
print("train_dataset length: ", len(train_dataset))
```

观察数据集对象

```
train_dataset length:  60000
Dataset ImageFolder
    Number of datapoints: 60000
    Root location: ./mnist_images/train
    StandardTransform
Transform: Compose(
               Grayscale(num_output_channels=1)
               ToTensor()
           )
```

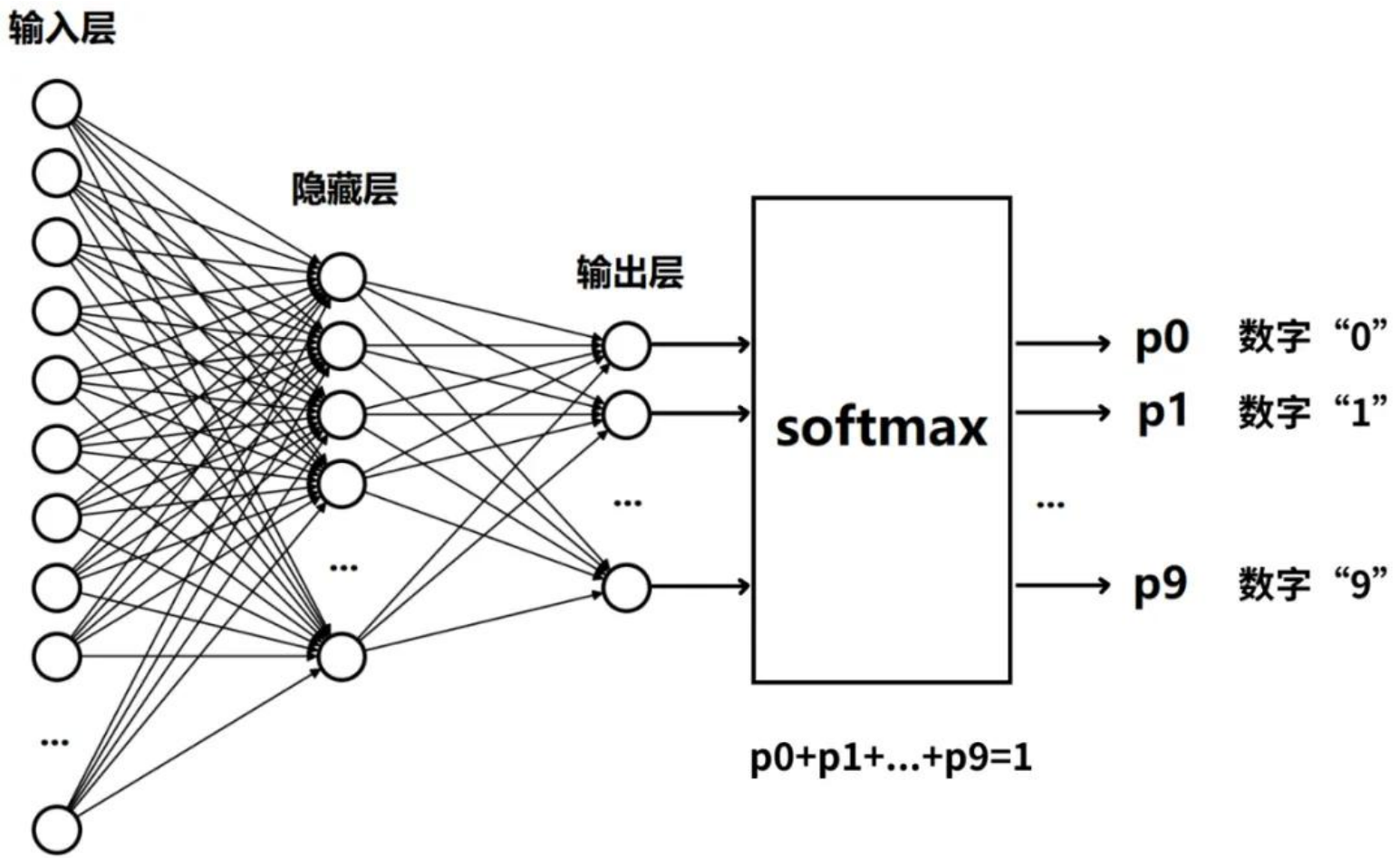
定义神经网络




```python
# 定义神经网络Network
class Network(nn.Module):
    def __init__(self):
        super().__init__()
        # 线性层1，输入层和隐藏层之间的线性层
        self.layer1 = nn.Linear(784, 256)
        # 线性层2，隐藏层和输出层之间的线性层
        self.layer2 = nn.Linear(256, 10)

    # 在前向传播，forward函数中，输入为图像x
    def forward(self, x):
        x = x.view(-1, 28 * 28) # 使用view函数，将x展平
        x = self.layer1(x)  # 将x输入至layer1
        x = torch.relu(x)  # 使用relu激活
        return self.layer2(x) # 输入至layer2计算结果
```

观察神经网络

```
Network(
  (layer1): Linear(in_features=784, out_features=256, bias=True)
  (layer2): Linear(in_features=256, out_features=10, bias=True)
)

layer(layer1) parameters:
         torch.Size([256, 784]) has 200704 parameters
         torch.Size([256]) has 256 parameters
layer(layer2) parameters:
         torch.Size([10, 256]) has 2560 parameters
         torch.Size([10]) has 10 parameters
The model has 203530 trainable parameters
```


输入层
隐藏层
输出层
softmax
p0
数字“0”
p1
数字“1”
p9
数字“9”
p0+p1+...+p9=1

训练模型

```
model = Network()  # 模型本身，它就是我们设计的神经网络
    # 优化器，用于更新模型中的参数
optimizer = optim.Adam(model.parameters())  # 优化模型中的参数
    # 交叉熵损失函数，用于计算模型的损失
criterion = nn.CrossEntropyLoss()  # 分类问题，使用交叉熵损失误差
```

小批量加载数据训练

```
# 小批量的数据读入
# DataLoader是一个PyTorch的类，用于将数据集拆分为小批量，并支持多线程读入
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
# 60000个训练数据，如果每个小批量，读入64个样本；那么60000个数据会被分成938组
print("train_loader length: ", len(train_loader))
```

小批量加载数据训练

```python
# 内存循环使用train_loader，进行小批量的数据读取
# 每一次循环，都会取出64个图像数据，作为一个小批量 batch
for batch_idx, (data, label) in enumerate(train_loader):
    if batch_idx ==3: # 打印前3个batch观察
        break
    print ("batch idx: ", batch_idx)
    print("data. shape : ", data.shape)# 数据的尺寸
    print ("label: ", label.shape)#图像中的数字
    print (label)
```

```
train_dataset length:  60000
train_loader length:  938
batch idx:  0
data. shape :  torch.Size([64, 1, 28, 28])
label:  torch.Size([64])
tensor([9, 9, 3, 5, 6, 3, 1, 7, 7, 4, 9, 3, 2, 2, 4, 9, 7, 2, 4, 9, 1, 3, 7, 1,
        6, 7, 0, 9, 9, 6, 3, 5, 0, 7, 8, 2, 6, 4, 1, 2, 2, 2, 7, 1, 0, 6, 2, 1,
        9, 0, 1, 1, 6, 5, 8, 1, 7, 9, 9, 0, 8, 1, 3, 2])
```

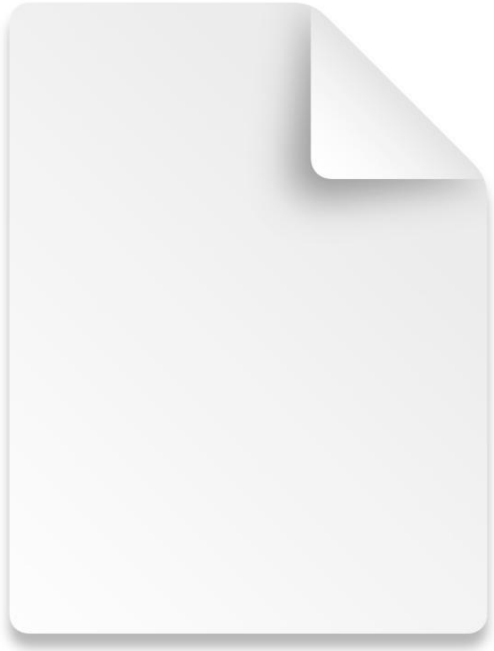
训练过程

```python
# 进入模型的迭代循环
for epoch in range(10):  # 外层循环，代表了整个训练数据集的遍历次数
    # 整个训练集要循环多少轮，是10次、20次或者100次都是可能的，
    # 内存循环使用train_loader，进行小批量的数据读取
    # 每一次循环，都会取出64个图像数据，作为一个小批量 batch
    for batch_idx, (data, label) in enumerate(train_loader):
        # 内层每循环一次，就会进行一次梯度下降算法
        # 包括了5个步骤：
        output = model(data) # 1.计算神经网络的前向传播结果
        loss = criterion(output, label) # 2.计算output和标签label之间的损失loss
        loss.backward()  # 3.使用backward计算梯度
        optimizer.step()  # 4.使用optimizer.step更新参数
        optimizer.zero_grad()  # 5.将梯度清零
        # 这5个步骤，是使用pytorch框架训练模型的定式，初学的时候，先记住就可以了

        # 每迭代100个小批量，就打印一次模型的损失，观察训练的过程
        if batch_idx % 100 == 0:
            print(f"Epoch {epoch + 1}/10 "
                  f"| Batch {batch_idx}/{len(train_loader)} "
                  f"| Loss: {loss.item():.4f}")
```

损失函数 = loss(预测值，实际值)

保存模型

```
torch.save(model.state_dict(), 'mnist.pth') # 保存模型
```



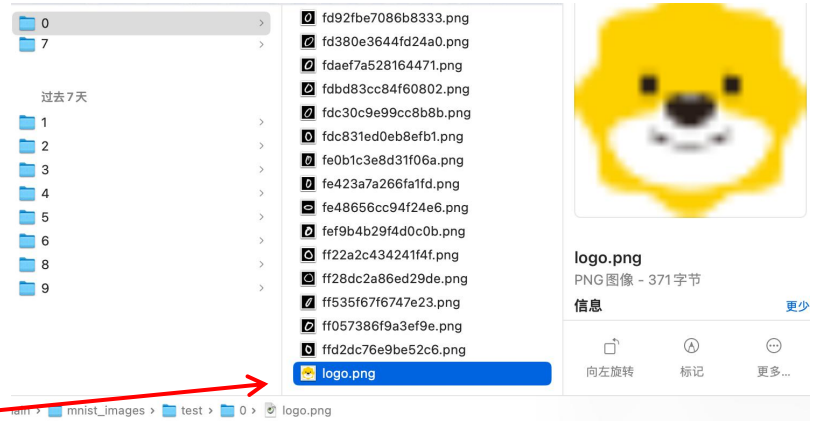
mnist.pth
文稿 - 816 KB

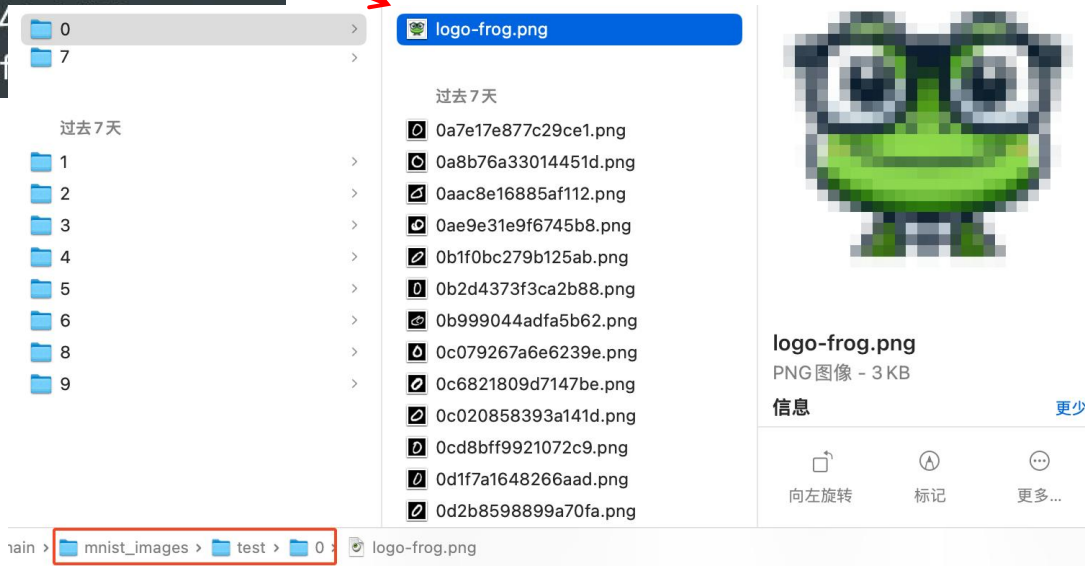
测试模型

```python
# 读取测试数据集
test_dataset = datasets.ImageFolder(root='./mnist_images/test',
transform=transform)
print("test_dataset length: ", len(test_dataset))

model = Network()  # 定义神经网络模型
model.load_state_dict(torch.load('mnist.pth')) # 加载刚刚训练好的
模型文件

right = 0 # 保存正确识别的数量
for i, (x, y) in enumerate(test_dataset):
    output = model(x)  # 将其中的数据x输入到模型
    predict = output.argmax(1).item() # 选择概率最大标签的作为预测
    结果
    # 对比预测值predict和真实标签y
    if predict == y:
        right += 1
    else:
        # 将识别错误的样例打印了出来
        img_path = test_dataset.samples[i][0]
        print(f"❌错误识别: 预测 = {predict} 实际 = {y} 图片路径 =
        {img_path}")
```

02 训练准备

```
test_dataset length:  10002
❌错误识别: 预测 = 7 实际 = 0 图片路径 = ./mnist_images/test/0/0e647b60c0ac28ed.png
❌错误识别: 预测 = 3 实际 = 0 图片路径 = ./mnist_images/test/0/1954d2ea1ca73812.png
❌错误识别: 预测 = 3 实际 = 0 图片路径 = ./mnist_images/test/0/1e126bfa2e2c695b.png
❌错误识别: 预测 = 1 实际 = 0 图片路径 = ./mnist_images/test/0/247640cc5ef741fd.png
❌错误识别: 预测 = 8 实际 = 0 图片路径 = ./mnist_images/test/0/a0741cc9a4ce4606.png
❌错误识别: 预测 = 4 实际 = 0 图片路径 = ./mnist_images/test/0/a433a474a8ac686c.png
❌错误识别: 预测 = 6 实际 = 0 图片路径 = ./mnist_images/test/0/b7fcbd22fb3e1d54.png
❌错误识别: 预测 = 8 实际 = 0 图片路径 = ./mnist_images/test/0/bb66e4114f4a9d33.png
❌错误识别: 预测 = 3 实际 = 0 图片路径 = ./mnist_images/test/0/c4b0a9e7d67c0bd1.png
❌错误识别: 预测 = 7 实际 = 0 图片路径 = ./mnist_images/test/0/logo-frog.png
❌错误识别: 预测 = 7 实际 = 0 图片路径 = ./mnist_images/test/0/logo.png
❌错误识别: 预测 = 7 实际 = 1 图片路径 = ./mnist_images/test/1/081184738c6fe0f3.png
❌错误识别: 预测 = 6 实际 = 1 图片路径 = ./mnist_images/test/1/2c293084
❌错误识别: 预测 = 2 实际 = 1 图片路径 = ./mnist_images/test/1/a36fbc5f
```



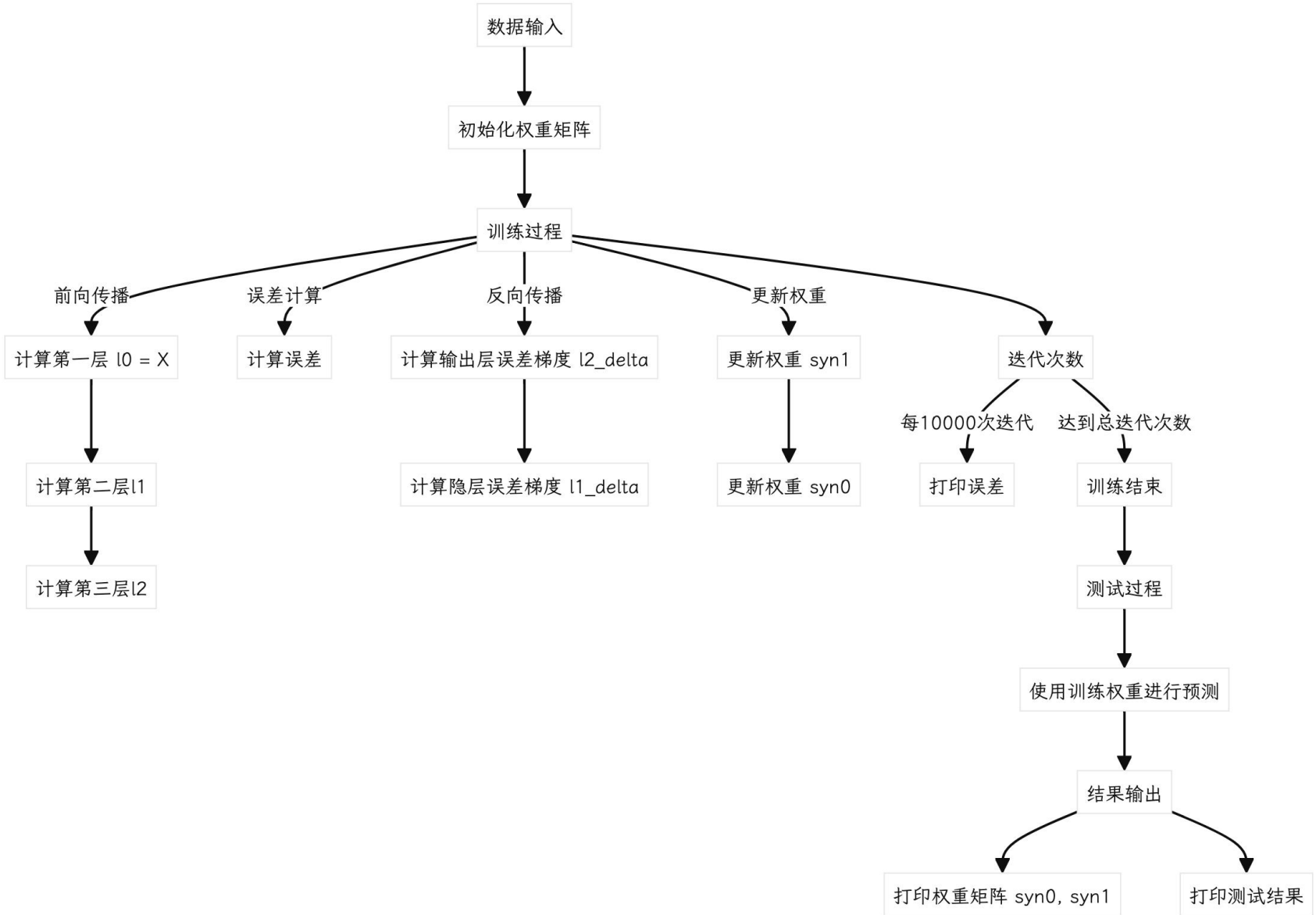


03

应用实践

三个案例代表

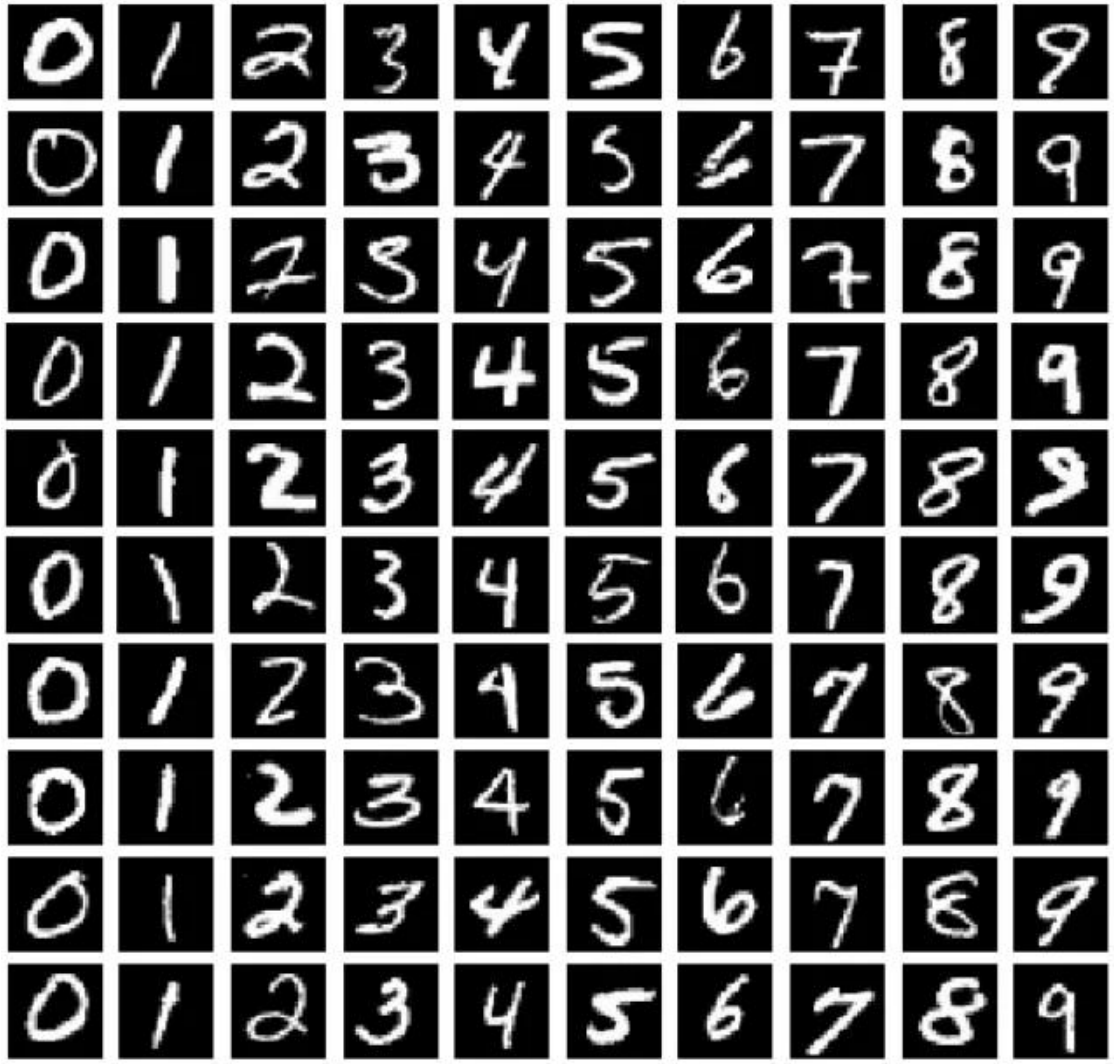
训练工具


数据输入
初始化权重矩阵
训练过程
前向传播
误差计算
反向传播
更新权重
计算第一层 l0 = X
计算误差
计算输出层误差梯度 l2_delta
更新权重 syn1
迭代次数
每10000次迭代
达到总迭代次数
计算第二层l1
计算隐层误差梯度 l1_delta
更新权重 syn0
打印误差
训练结束
计算第三层l2
测试过程
使用训练权重进行预测
结果输出
打印权重矩阵 syn0，syn1
打印测试结果

```
Error in 0 iteration for training: 0.11261954162061182
Error in 10000 iteration for training: 0.006851552154427041
Error in 20000 iteration for training: 0.001918471175058625
Error in 30000 iteration for training: 0.0010068413350842365
Error in 40000 iteration for training: 0.0006594791347571346
Error in 50000 iteration for training: 0.000482350813828031
Error in 60000 iteration for training: 0.000376688537952602
Error in 70000 iteration for training: 0.0003071670953030512
Error in 80000 iteration for training: 0.00025825440978620304
Error in 90000 iteration for training: 0.00022212438893800236
Training finish!
syn0:
[[-1.86350688  2.56106277 -3.1727484   1.0936684 ]
 [-2.21353485  2.21834879 -3.16753516  1.25690493]
 [ 0.60945164 -0.84350619  1.32478546  0.137065  ]]
syn1:
[[-3.03983008]
 [ 4.19761229]
 [-5.11288038]
 [ 2.17979564]]
test res:
[[0.02708799]
 [0.97994289]
 [0.97966914]
 [0.99753748]]
```

```python
# input data
# 每一列代表一个神经元
# 故总共有 3 个神经元，各自接受 4 个输入数据
X = np.array([[0, 0, 1],
              [0, 1, 1],
              [1, 0, 1],
              [1, 1, 1]])

# output data
Y = np.array([[0],
              [1],
              [1],
              [1]])

syn0, syn1 = train(X, Y)
res = test(X, syn0, syn1)
print("test res:")
print(res)
```

案例一（分类）



案例二（数字识别）

数据文档

背景描述

此数据集为kaggle竞赛Dogs vs. Cats Redux: Kernels Edition的数据集

数据说明

train文件夹包含25,000张狗和猫的图像。此文件夹中的每个图像都将标签作为文件名的一部分。

test文件夹包含12,500个图像，根据数字ID命名。对于测试集中的每个图像，您应该预测图像是狗的概率（1 =狗，0 =猫）。

问题描述

对于测试集中的每个图像，你能预测图像是狗的概率吗？ （1 =狗，0 =猫）

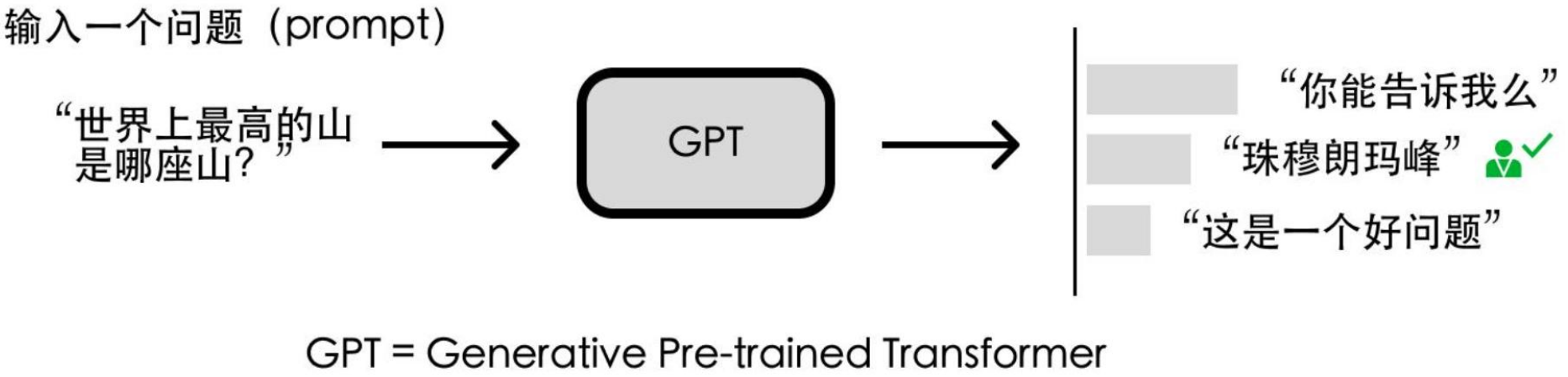
案例三（狗脸识别）

val_Loss

Run	Smoothed	Value	Step	Relative
runs/LeNet1_1	0.0194	0.0195	38	27.23 min
runs/LeNet1_2	0.0227	0.0225	3	23.83 hr
runs/LeNet_3	0.0201	0.0192	11	15.22 min
runs/ResNet34_2	0.0228	0.0215	2	40.8 min

Run	Smoothed	Value	Step	Relative
runs/LeNet1_1	69.6249	70	38	27.23 min
runs/LeNet1_2	59.439	61.75	3	23.83 hr
runs/LeNet_3	68.2249	70.38	11	15.22 min
runs/ResNet34_2	62.2235	64.88	2	40.8 min

01 第一步



第一步：人类引导接龙方向-有监督训练初始模型

研究人员让人类就一些问题写出人工答案，再把这些问题和答案丢给 GPT 学习。这便是有监督训练，即对于特定问题告诉 AI 人类认可的答案。通过这种有监督训练的方法，我们可以得到一个简易版的 ChatGPT 模型。

参考资料

- [PyTorch documentation — PyTorch 2.3 documentation](https://pytorch.org/docs/stable/index.html)
- [Dogs vs. Cats Redux: Kernels Edition | Kaggle](https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition/data)
- [猫狗图像分类数据集](https://www.heywhale.com/mw/dataset/5d11bb1a38dc33002bd6f1f1/content)
- [MNIST Dataset Download and Visualization | Red Fish 2th](https://combofish.github.io/2021/09/19/mnist-download-and-view.html)
- [从神经网络到 Hugging Face](https://hutusi.com/articles/the-history-of-neural-networks)

THANK YOU!