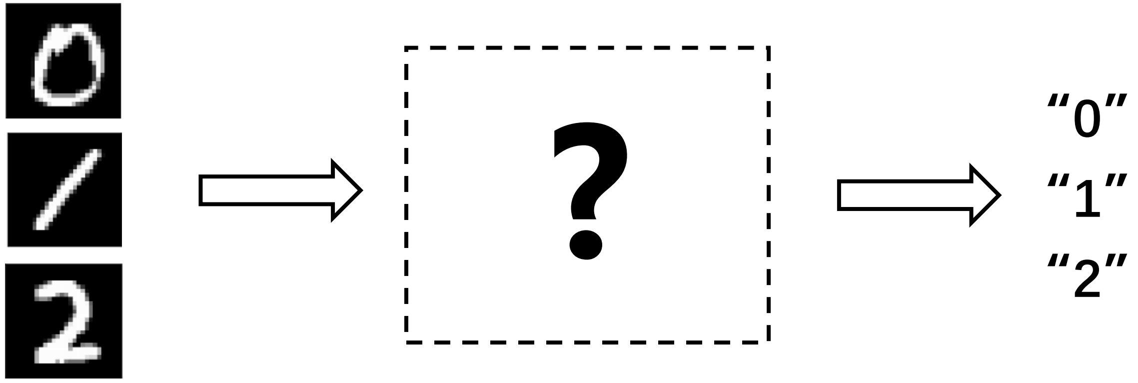
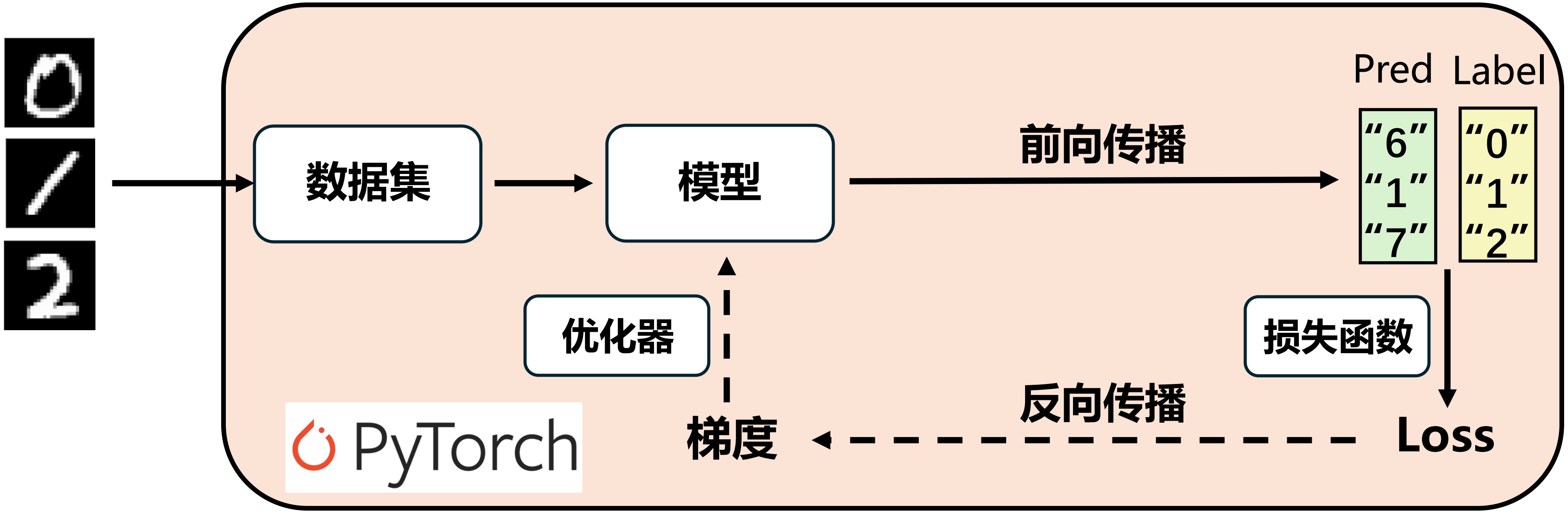
PyTorch Tutorial

Week 3

逐步构建训练流程



- 针对手写数字识别这一任务，如何建立一个分类模型，并通过模型的训练过程不断进行优化，让计算机能较为准确地识别手写数字图片呢?



基于PyTorch实现手写数字识别

导入PyTorch相关库

```python
import torch
from torch.utils.data import Dataset, DataLoader

import torch.nn as nn
import torch.nn.functional as F

import torch.optim as optim
```

模型

```python
class Mnist_CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=32,
             kernel_size=5, stride=1, padding=2)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2,
padding=0)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64,
             kernel_size=5, stride=1, padding=2)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2,
padding=0)
        self.fc1 = nn.Linear(7*7*64, 512)
        self.fc2 = nn.Linear(512, 10)

    def forward(self, inputs):
        tensor = F.relu(self.conv1(inputs))
        tensor = self.pool1(tensor)
        tensor = F.relu(self.conv2(tensor))
        tensor = self.pool2(tensor)
        tensor = tensor.view(-1, 7*7*64)
        tensor = F.relu(self.fc1(tensor))
        tensor = self.fc2(tensor)
        return tensor
```

数据集

```python
class Mnist(Dataset):
    def __init__(self, root, train=True, transform=torch.tensor):
        self.file_pre = 'train' if train == True else 't10k'
        self.transform = transform #定义变换函数
        self.label_path = os.path.join(root,
                                       '%s-labels-idx1-ubyte.gz' % self.file_pre)
        self.image_path = os.path.join(root,
                                       '%s-images-idx3-ubyte.gz' % self.file_pre)
        # 读取文件数据，返回图片和标签
        self.images, self.labels = self.__read_data__(
            self.image_path,
            self.label_path)

    def __read_data__(self, image_path, label_path):
        # 数据集读取
        with gzip.open(label_path, 'rb') as lbpath:
            labels = np.frombuffer(lbpath.read(), np.uint8, offset=8)#将data以流的形
              式读入转化成ndarray对象，ndarray对象是用于存放同类型元素的多维数组
        with gzip.open(image_path, 'rb') as imgpath:
            images = np.frombuffer(imgpath.read(), np.uint8,
            offset=16).reshape(len(labels), 1, 28, 28)#将图片以标签文件的元素个数读取，
              设置大小为28*28
        return images, labels

    def __getitem__(self, index):
        image, label = np.array(self.images[index], dtype=np.float32)/255,
             int(self.labels[index])

        if self.transform is not None:
            image = self.transform(image)  # 此处需要用 np.array(image)，转化为数组
        return image, label

    def __len__(self):
        return len(self.labels)
```

基于PyTorch实现手写数字识别

生成数据集、模型对象&定义损失函数、优化器

```python
## 生成训练集
train_set = Mnist(
    root='data/MNIST/raw',
    train=True
)
train_loader = DataLoader(
    dataset=train_set, #输出的数据
    batch_size=32,
    shuffle=True #将元素随机排序
)
## 生成模型对象
net = Mnist_CNN()

## 选择数据和模型放置在 CPU/哪个GPU 上
device = torch.device("cuda", 0) #选择将程序放置到哪个GPU上
#device = torch.device('cpu')
net.to(device)

## 定义损失函数
loss_function = torch.nn.CrossEntropyLoss()
## 定义优化器
optimizer = optim.SGD(
    net.parameters(),#网络参数
    lr=0.001, #学习率
    momentum=0.9#Momentum 用于加速 SGD（随机梯度下降）在某一方向上的搜
        索以及抑制震荡的发生。
)
```

迭代训练（反向传播与模型优化）

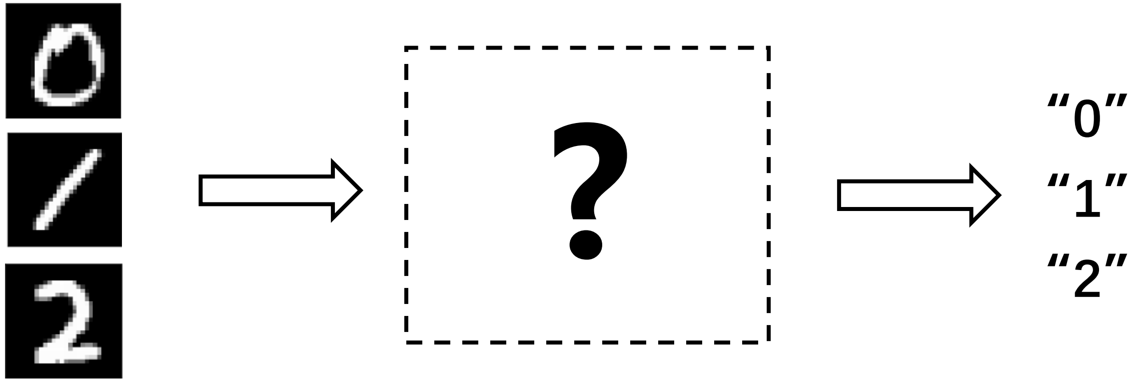
```python
loss_list,acc_list = [],[]
for epoch in range(10):#训练10次
    running_loss = 0.0
    total, correct=0,0
    for images, labels in tqdm(train_loader):#enumerate索引函数, start
            下标开始位置
        images=images.to(device)  #将images放进GPU
        labels=labels.to(device)  #将labels放进GPU

        optimizer.zero_grad()
            # 梯度清零, 初始化,如果不初始化, 则梯度会叠加
        outputs = net(images)                          # 前向传播
        loss = loss_function(outputs, labels)
            # 计算误差, label标准?
        loss.backward()                                # 反向传播
        optimizer.step()                               # 权重更新

        running_loss += loss.item()                    # 误差累计
        _, predict = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predict == labels).sum()


    print('epoch:{:d} loss:{:.3f} acc:{:.3f}'
          .format(epoch+1, running_loss/len(train_loader),
            correct/total), flush=True)
    loss_list.append(running_loss/len(train_loader))
    acc_list.append(correct/total)

print('Finished Training!')

torch.save(net.state_dict(), "Linear.pth")#保存训练模型
```

什么是PyTorch?

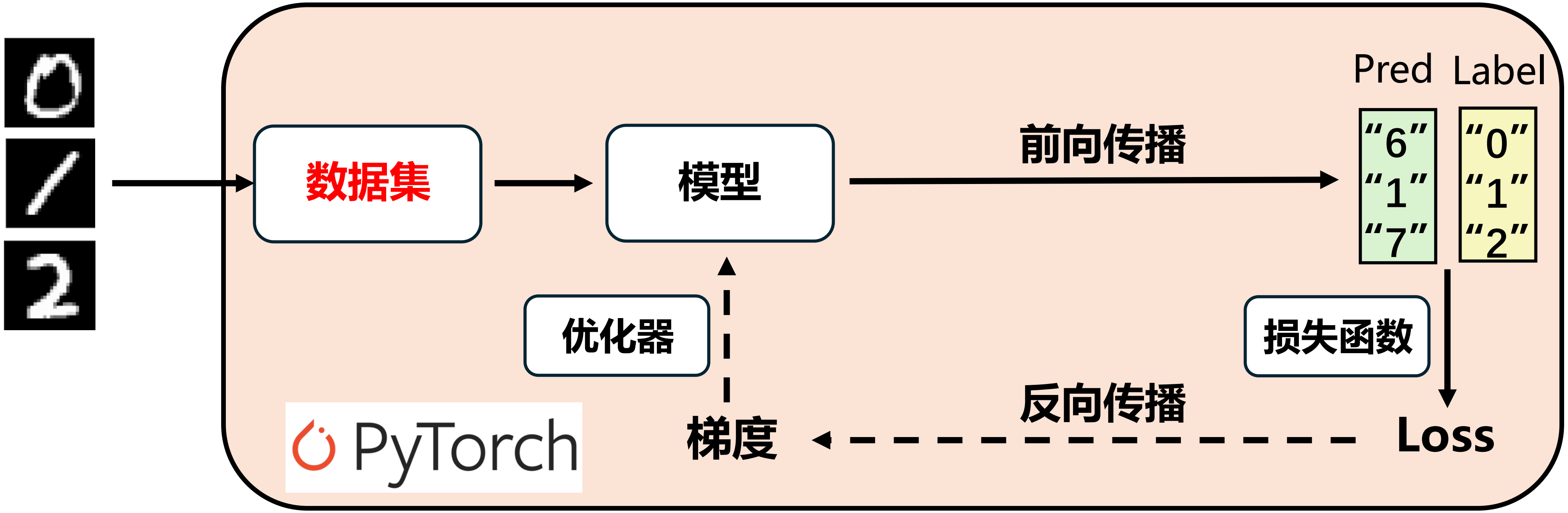
- Torch：一个开源的机器学习框架，早在2002年即发布初版，使用的编程语言为C和Lua。目前Torch7依然是热门的深度学习框架之一。
- **PyTorch**：由Facebook在2017年1月推出。PyTorch是基于Python语言构建的机器学习框架Torch的端口。PyTorch具有以下特点：
  - 拥有一套**完整、成熟的API**，代码简洁、符合人类思维、容易上手.
  - 可以高效、灵活地使用**GPU资源**，加速计算；可实现反向自动微分.
  - 拥有大量开源代码与开源社区资源.

```
import torch
from torch.utils.data import Dataset, DataLoader 数据集

import torch.nn as nn  神经网络(Nerual Network)

import torch.nn.functional as F  相关函数
import torch.optim as optim  优化器
```

逐步构建训练流程

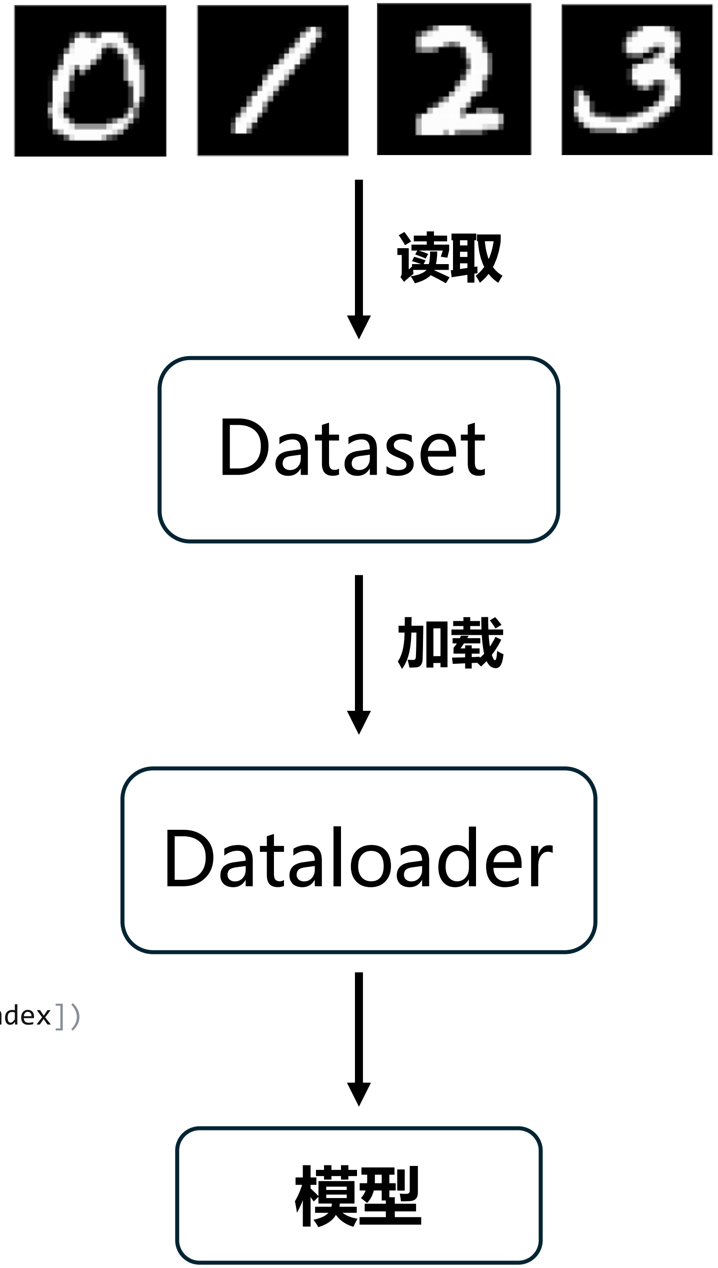


- 针对手写数字识别这一任务，如何建立一个分类模型，并通过模型的训练过程不断进行优化，让计算机能较为准确地识别手写数字图片呢?



1. 数据集

```python
class Mnist(Dataset):
    def __init__(self, root, train=True, transform=torch.tensor):
        self.file_pre = 'train' if train == True else 't10k'
        self.transform = transform  #定义变换函数
        self.label_path = os.path.join(root,
                                       '%s-labels-idx1-ubyte.gz' % self.file_pre)
        self.image_path = os.path.join(root,
                                       '%s-images-idx3-ubyte.gz' % self.file_pre)
        # 读取文件数据, 返回图片和标签
        self.images, self.labels = self.__read_data__(
            self.image_path,
            self.label_path)

    def __read_data__(self, image_path, label_path):
        # 数据集读取
        with gzip.open(label_path, 'rb') as lbpath:
            labels = np.frombuffer(lbpath.read(), np.uint8, offset=8)
            #将data以流的形式读入转化成ndarray对象, ndarray对象是用于存放同类型元素的多维数组
        with gzip.open(image_path, 'rb') as imgpath:
            images = np.frombuffer(imgpath.read(), np.uint8, offset=16)
            .reshape(len(labels), 1, 28, 28)
            #将图片以标签文件的元素个数读取.设置大小为28*28
        return images, labels

    def __getitem__(self, index):
        image, label = np.array(self.images[index], dtype=np.float32)/255, int(self.labels[index])

        if self.transform is not None:
            image = self.transform(image)  # 此处需要用 np.array(image), 转化为数组
        return image, label

    def __len__(self):
        return len(self.labels)
```



Dataset & DataLoader

- Dataset:

torch.utils.data.Dataset 是 PyTorch 提供的用于自定义数据集方法的**抽象类**，用户可以通过继承该类来自定义自己的数据集类，在继承时要求用户定义构造函数__init()__并重载__getitem__() 和__len__()这两个魔法方法。

➢ **__getitem__**：索引数据集中的任意一个数据，并可进行数据预处理（transform）。

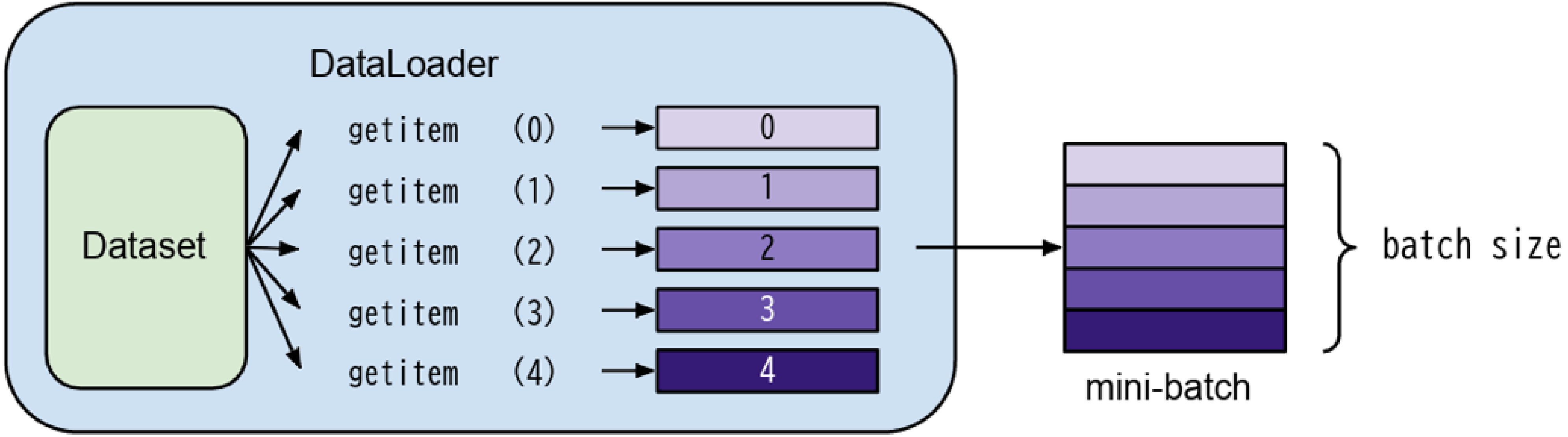
➢ **__len__**：返回数据集的大小

```python
from torch.utils.data import Dataset, DataLoader

class Mnist(Dataset):
  def __init__(self, …):
    self.images, self.labels=…

  def __getitem__(self, index):
    return image, label

  def __len__(self):
    return len(self.labels)
```

Dataset & DataLoader

- Dataloader：**torch.utils.data.DataLoader** 也是PyTorch提供的一个类，其可以将Dataset对象或自定义数据类的对象封装成一个**迭代器**，可以迭代输出Dataset的内容。
- 同时可以实现<u>shuffle</u>、<u>多进程</u>、<u>不同采样策略</u> 等处理过程。

```
train_set = Mnist(root='data/MNIST/raw’, train=True)
train_loader = DataLoader(dataset=train_set, batch_size=32, shuffle=True)
```



Dataset & DataLoader

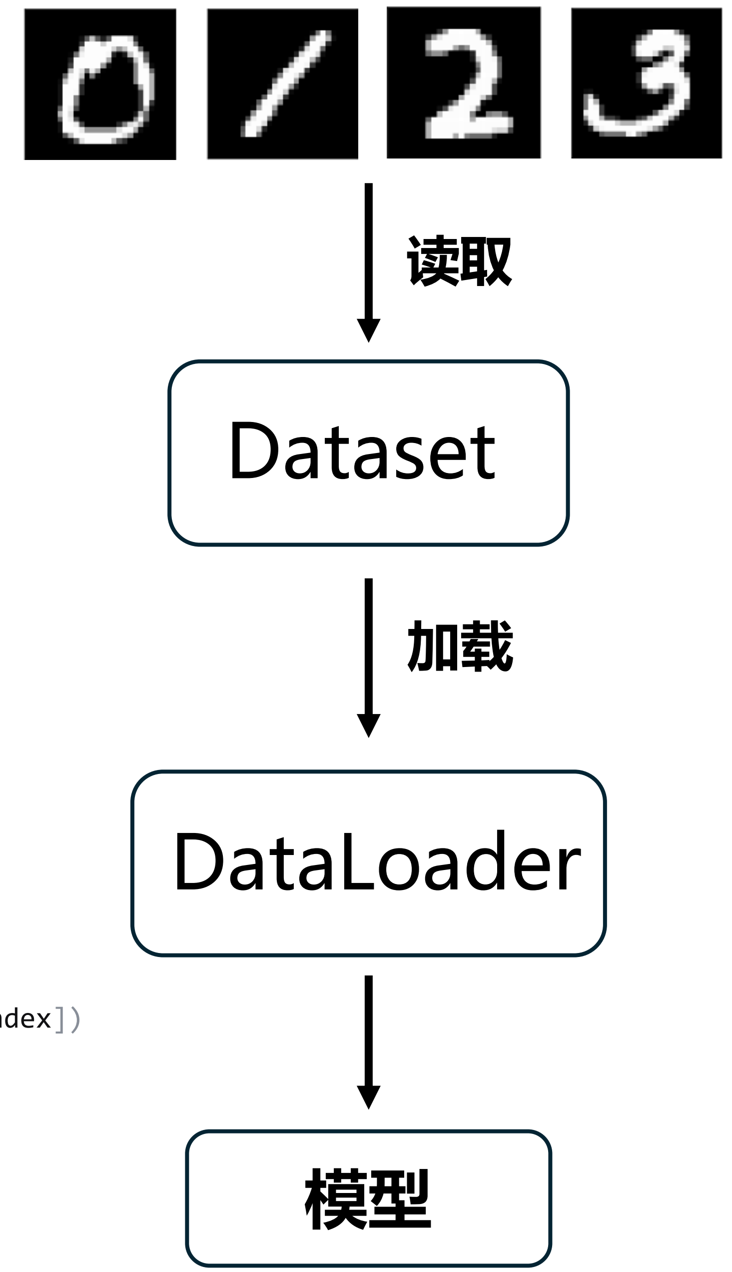
- train_loader = **DataLoader**(dataset, batch_size, shuffle, sampler, num_worker)

  - **dataset (Dataset)**：指定DataLoader从哪里读取数据
  - **batch_size (int)**：训练时每个batch加载的样本数量
  - **shuffle (bool)**：若为True，则每个训练过程中加载数据的顺序随机
  - **sampler (Sampler)**：定义每个batch的采样策略
  - **num_worker(int)**：使用多少个子进程来加载数据

※ 注意：当sampler有输入(不为None)时，**shuffle必须设置为False**，否则将直接报错。因为shuffle本质上是将sampler定义为 SequentialSampler(顺序采样) 或 RandomSampler(随机采样) .

1. 数据集

```python
class Mnist(Dataset):
    def __init__(self, root, train=True, transform=torch.tensor):
        self.file_pre = 'train' if train == True else 't10k'
        self.transform = transform  #定义变换函数
        self.label_path = os.path.join(root,
                                       '%s-labels-idx1-ubyte.gz' % self.file_pre)
        self.image_path = os.path.join(root,
                                       '%s-images-idx3-ubyte.gz' % self.file_pre)
        # 读取文件数据, 返回图片和标签
        self.images, self.labels = self.__read_data__(
            self.image_path,
            self.label_path)

    def __read_data__(self, image_path, label_path):
        # 数据集读取
        with gzip.open(label_path, 'rb') as lbpath:
            labels = np.frombuffer(lbpath.read(), np.uint8, offset=8)
            #将data以流的形式读入转化成ndarray对象, ndarray对象是用于存放同类型元素的多维数组
        with gzip.open(image_path, 'rb') as imgpath:
            images = np.frombuffer(imgpath.read(), np.uint8, offset=16)
            .reshape(len(labels), 1, 28, 28)
            #将图片以标签文件的元素个数读取.设置大小为28*28
        return images, labels

    def __getitem__(self, index):
        image, label = np.array(self.images[index], dtype=np.float32)/255, int(self.labels[index])

        if self.transform is not None:
            image = self.transform(image)  # 此处需要用 np.array(image), 转化为数组
        return image, label

    def __len__(self):
        return len(self.labels)
```



基于PyTorch实现手写数字识别

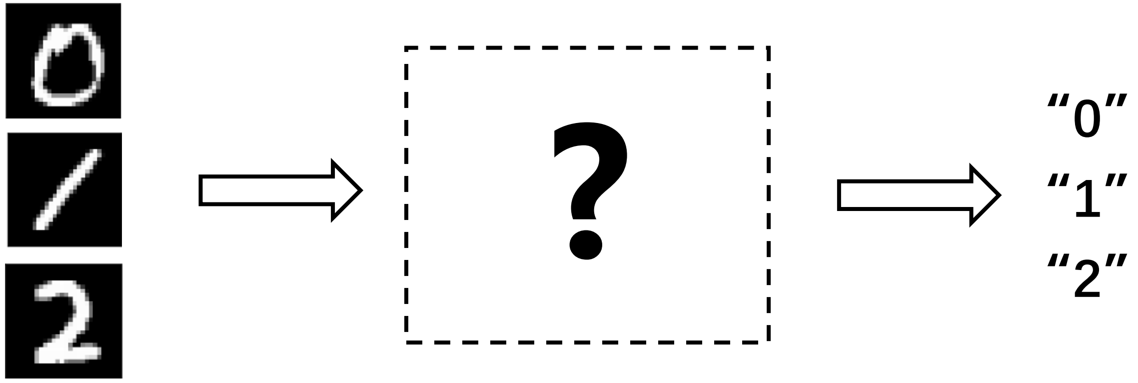
生成数据集、模型对象&定义损失函数、优化器

```
## 生成训练集
train_set = Mnist(
    root='data/MNIST/raw',
    train=True
)
train_loader = DataLoader(
    dataset=train_set, #输出的数据
    batch_size=32,
    shuffle=True #将元素随机排序
)
## 生成模型对象
net = Mnist_CNN()

## 选择数据和模型放置在 CPU/哪个GPU 上
device = torch.device("cuda", 0) #选择将程序放置到哪个GPU上
#device = torch.device('cpu')
net.to(device)

## 定义损失函数
loss_function = torch.nn.CrossEntropyLoss()
## 定义优化器
optimizer = optim.SGD(
    net.parameters(),#网络参数
    lr=0.001, #学习率
    momentum=0.9#Momentum 用于加速 SGD（随机梯度下降）在某一方向上的搜
            索以及抑制震荡的发生。
)
```

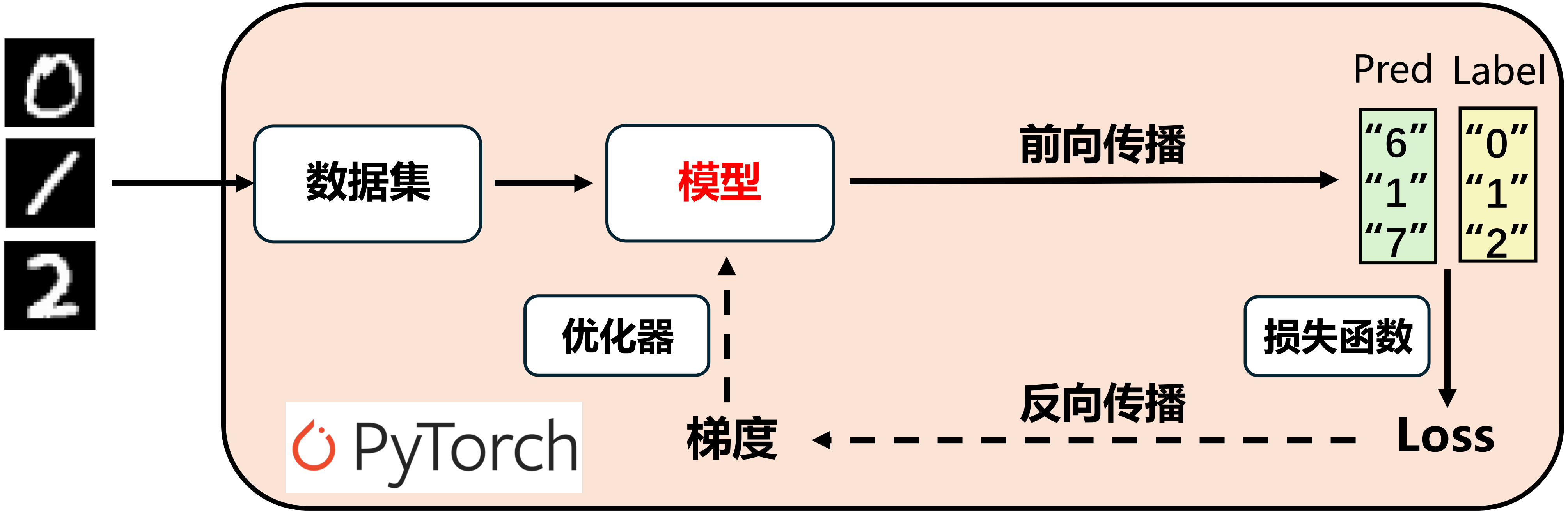
迭代训练（反向传播与模型优化）

```
loss_list,acc_list = [],[]
for epoch in range(10):#训练10次
    running_loss = 0.0
    total, correct=0,0
    for images, labels in tqdm(train_loader):#enumerate索引函数, start
            下标开始位置
        images=images.to(device)  #将images放进GPU
        labels=labels.to(device)  #将labels放进GPU

        optimizer.zero_grad()
            # 梯度清零, 初始化,如果不初始化, 则梯度会叠加
        outputs = net(images)                          # 前向传播
        loss = loss_function(outputs, labels)
            # 计算误差, label标准?
        loss.backward()                                # 反向传播
        optimizer.step()                               # 权重更新

        running_loss += loss.item()                    # 误差累计
        _, predict = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predict == labels).sum()


    print('epoch:{:d} loss:{:.3f} acc:{:.3f}'
          .format(epoch+1, running_loss/len(train_loader),
            correct/total), flush=True)
    loss_list.append(running_loss/len(train_loader))
    acc_list.append(correct/total)

print('Finished Training!')

torch.save(net.state_dict(), "Linear.pth")#保存训练模型
```

逐步构建训练流程

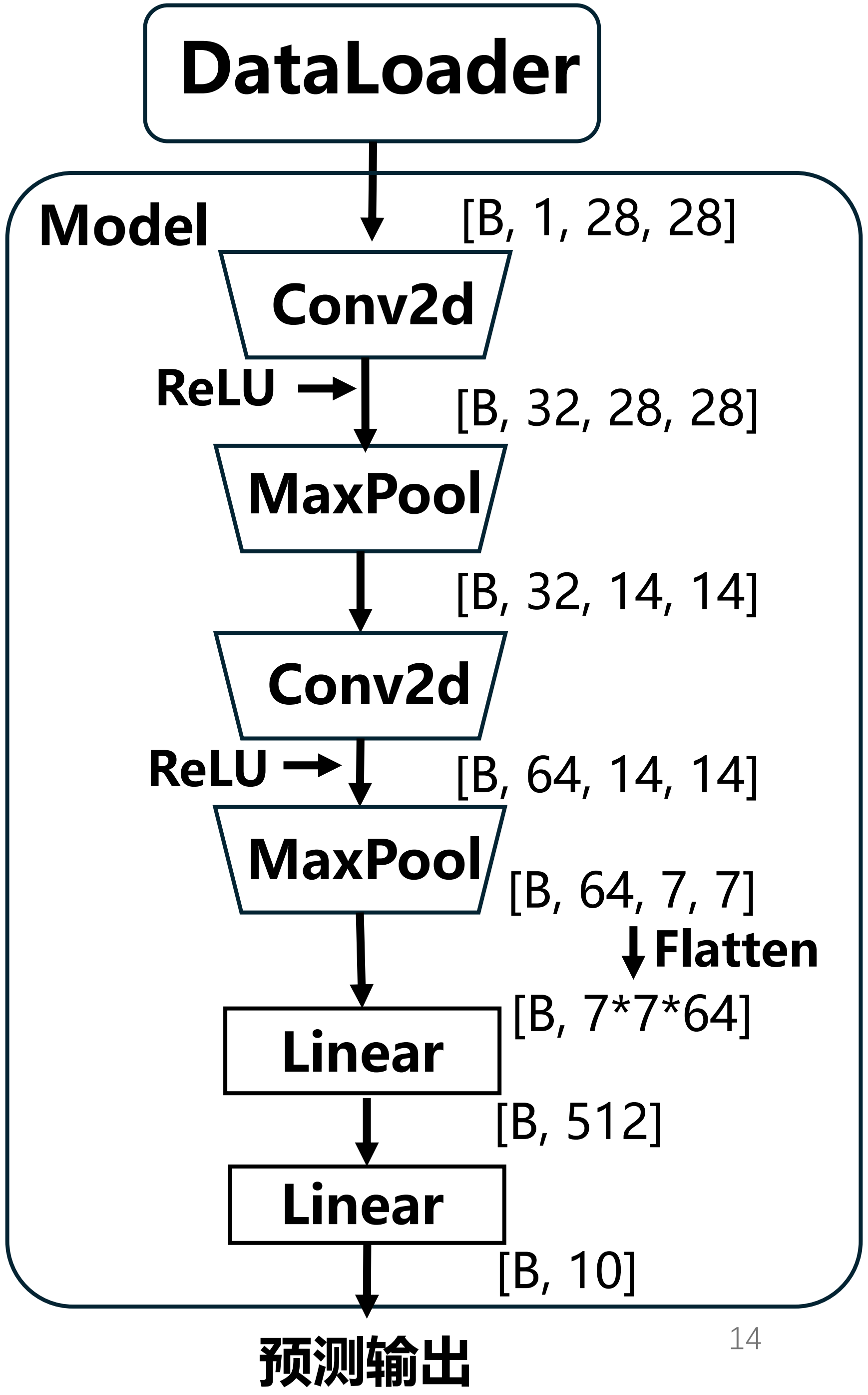


- 针对手写数字识别这一任务，如何建立一个分类模型，并通过模型的训练过程不断进行优化，让计算机能较为准确地识别手写数字图片呢?



2. 模型

```python
class Mnist_CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=32,
        kernel_size=5, stride=1, padding=2)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2,
        padding=0)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64,
        kernel_size=5, stride=1, padding=2)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2,
        padding=0)
        self.fc1 = nn.Linear(7*7*64, 512)
        self.fc2 = nn.Linear(512, 10)

    def forward(self, inputs):
        tensor = F.relu(self.conv1(inputs))
        tensor = self.pool1(tensor)
        tensor = F.relu(self.conv2(tensor))
        tensor = self.pool2(tensor)
        tensor = tensor.view(-1, 7*7*64)
        tensor = F.relu(self.fc1(tensor))
        tensor = self.fc2(tensor)
        return tensor
```



nn.Module

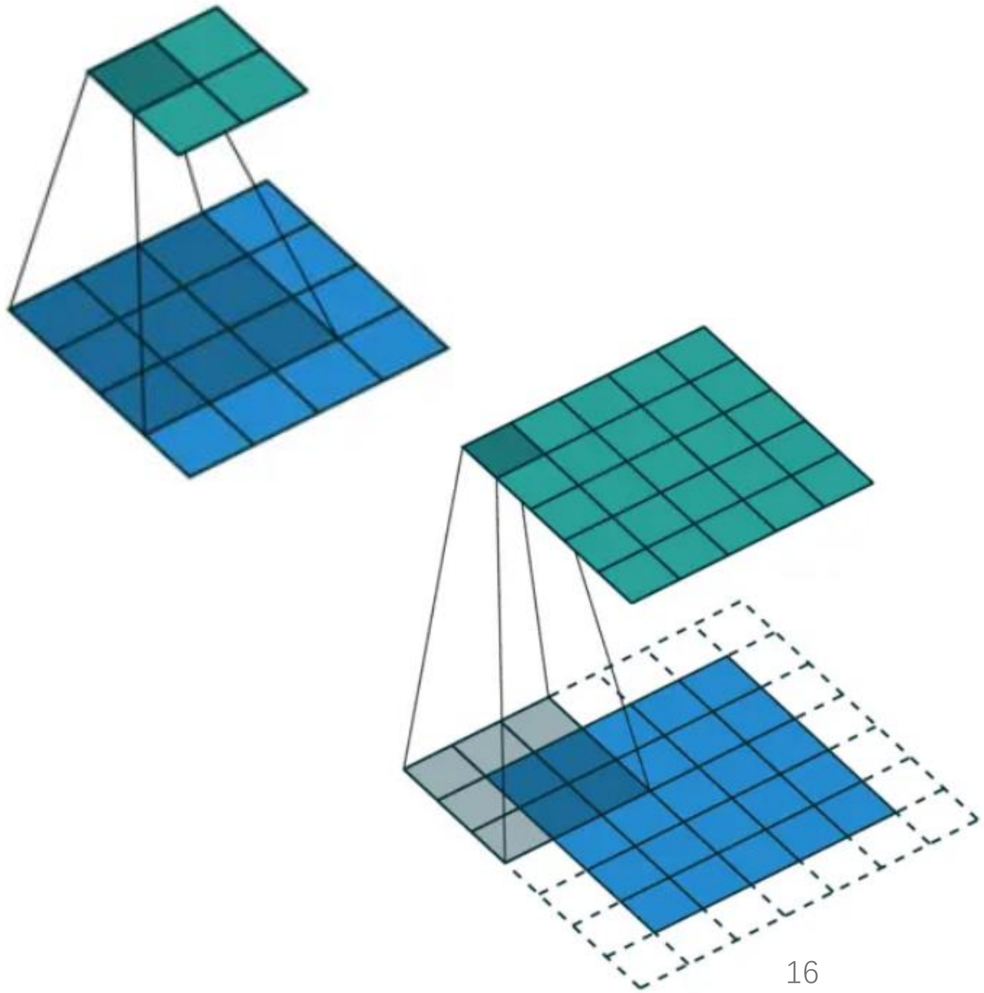
- nn.Module是PyTorch定义的所有神经网络的**基类**。
- 我们在定义自已的神经网络时，需要继承nn.Module类，并重新实现构造函数__init__和forward方法。

> - **__init__**：模型的构造函数，在定义自己的神经网络时，需要在其中定义各个网络层。
> - **forward**：嵌套在nn.Module的__call__()方法中，当模型被调用时，会返回forward的返回结果。因此，forward中需要调用模型的各个网络层，即完成前向传播的整个过程。

```python
class Mnist_CNN(nn.Module):
    def __init__(self):
        super().__init__()
        ```
        定义各个网络层
        ```
    def forward(self, inputs):
        ```
        调用各个网络层
        ```
        return （模型输出）
```

卷积层构建

- nn.conv2d(in_channels, out_channels, kernel_size, stride, padding)

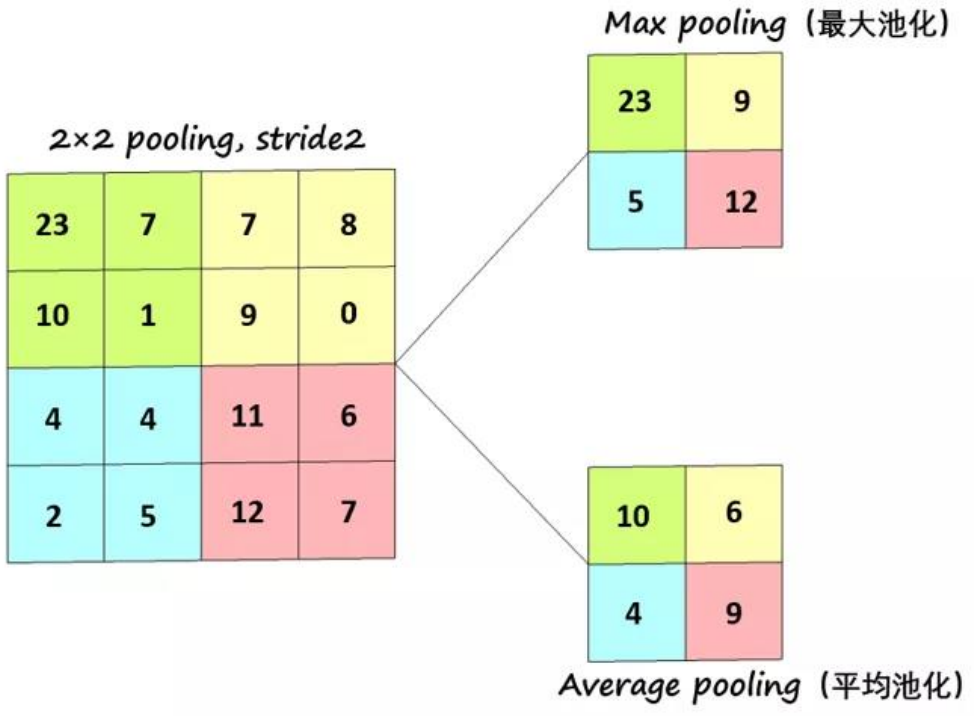


➢ **in_channels**：从上一层输入的通道数
➢ **out_channels**：输出的通道数
➢ **kernel_size**：卷积核的尺寸，例如：3x3的卷积核 –> kernel_size=(3, 3)
➢ **stride**：卷积核在图像窗口上每次平移的间隔，即步长
➢ **padding**：向图像四周填充的像素数量(默认为0填充，可用padding_mode指定)

池化层构建

- nn.MaxPool2d(kernel_size, stride, padding)
- nn.AvgPool2d(kernel_size, stride, padding)

➢ **kernel_size**：池化窗的尺寸，例如：3x3 的池化窗 –> kernel_size=(3, 3)
➢ **stride**：池化窗在图像窗口上每次平移的间隔，即步长
➢ **padding**：向图像四周填充的像素数量



激活函数

- nn.functional.relu(input)

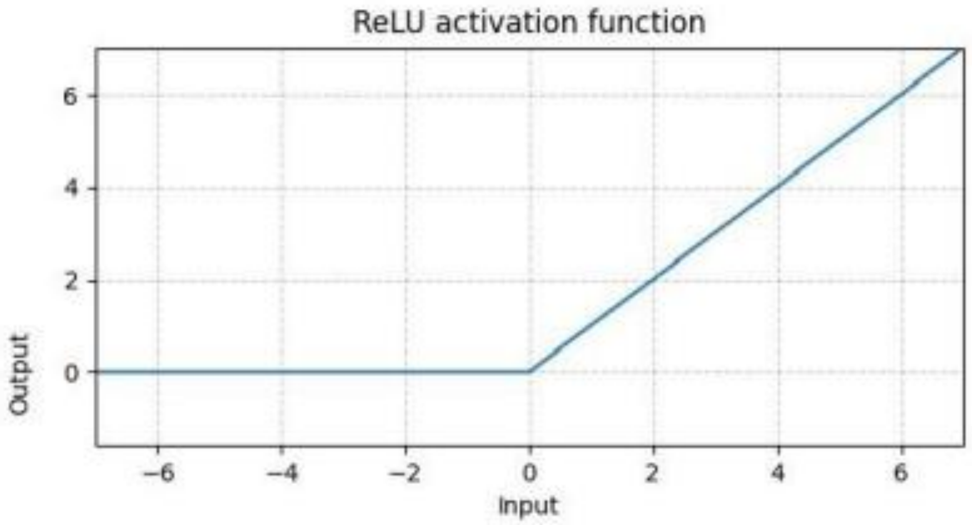
将卷积得到的特征图输入该函数，函数会将ReLU激活函数应用到特征图：

$$f(x) = \begin{cases} 0, & x \leq 0 \\ x, & x > 0 \end{cases}$$

5	-8
12	-1

$\Rightarrow$

5	0
12	0



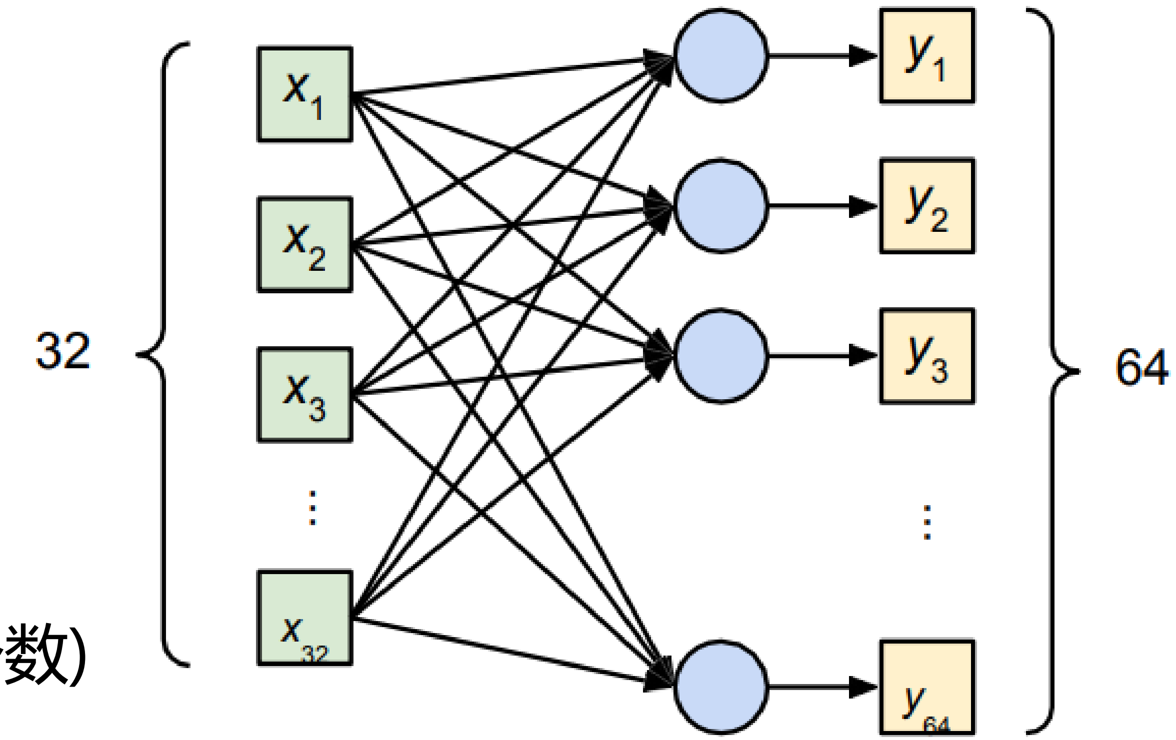
线性层构建

- nn.Linear(in_feature, out_feature)

➢ **in_feature:** 输入的二维张量大小，即[batch_size, size] 中的size

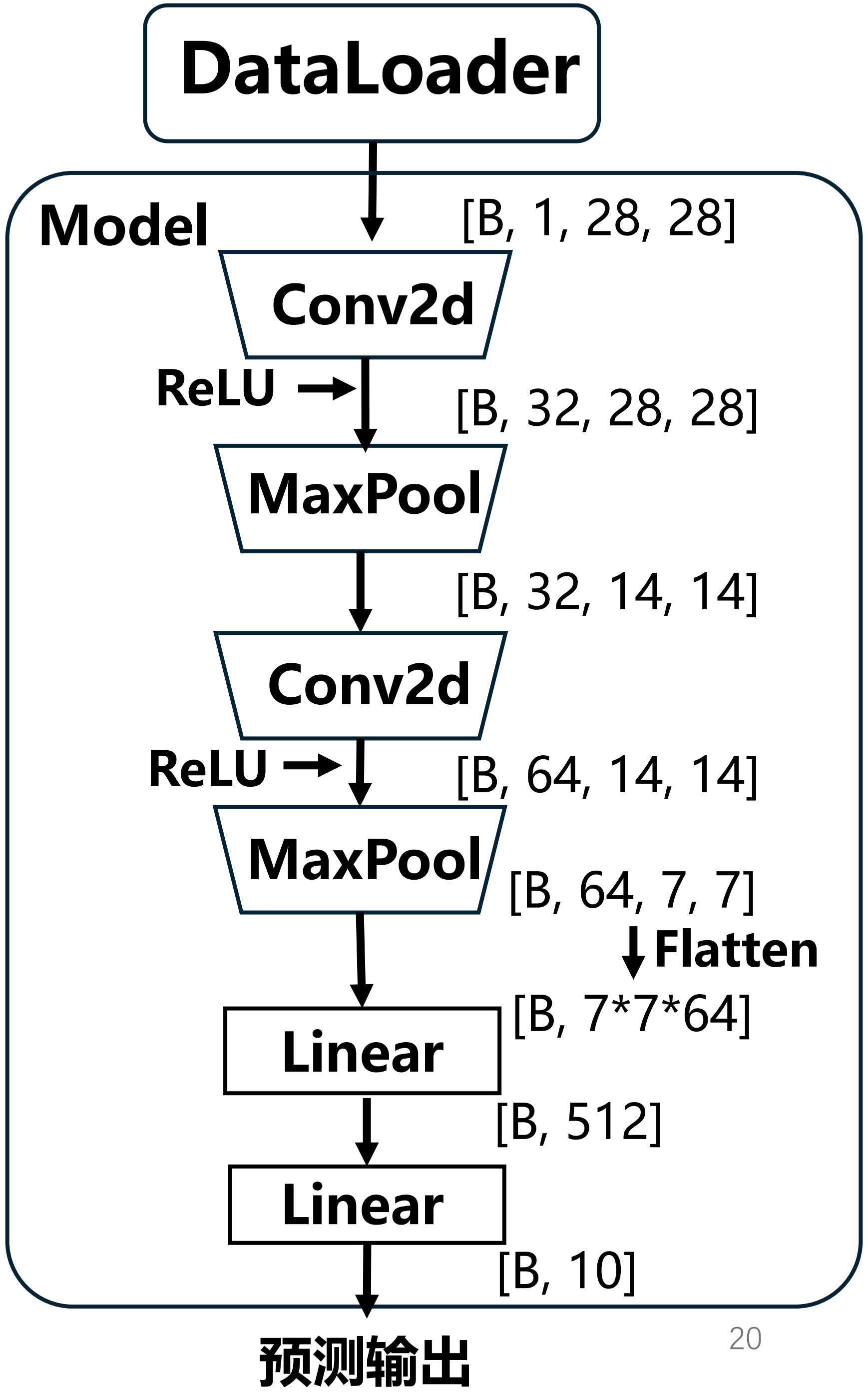
➢ **out_feature:** 输出的二维张量大小，最终输出的二维张量形状为[batch_size, out_feature]

➢ (out_feature同时也是线性层的神经元个数)

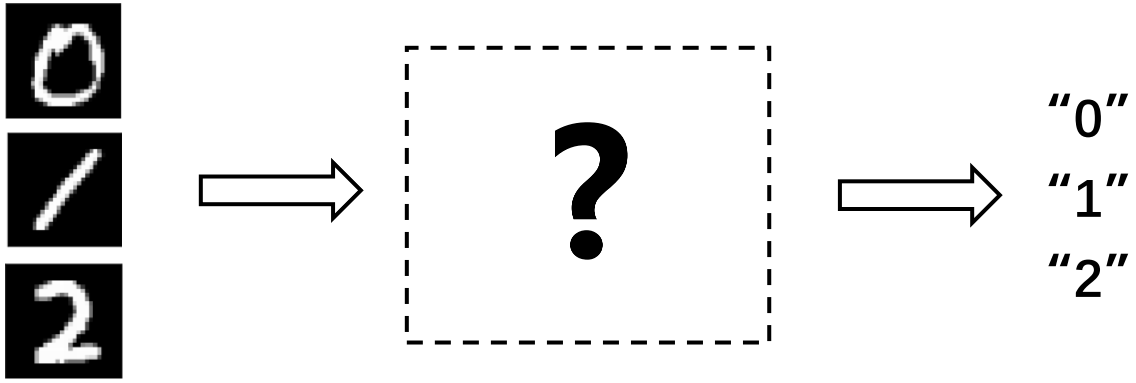


2. 模型

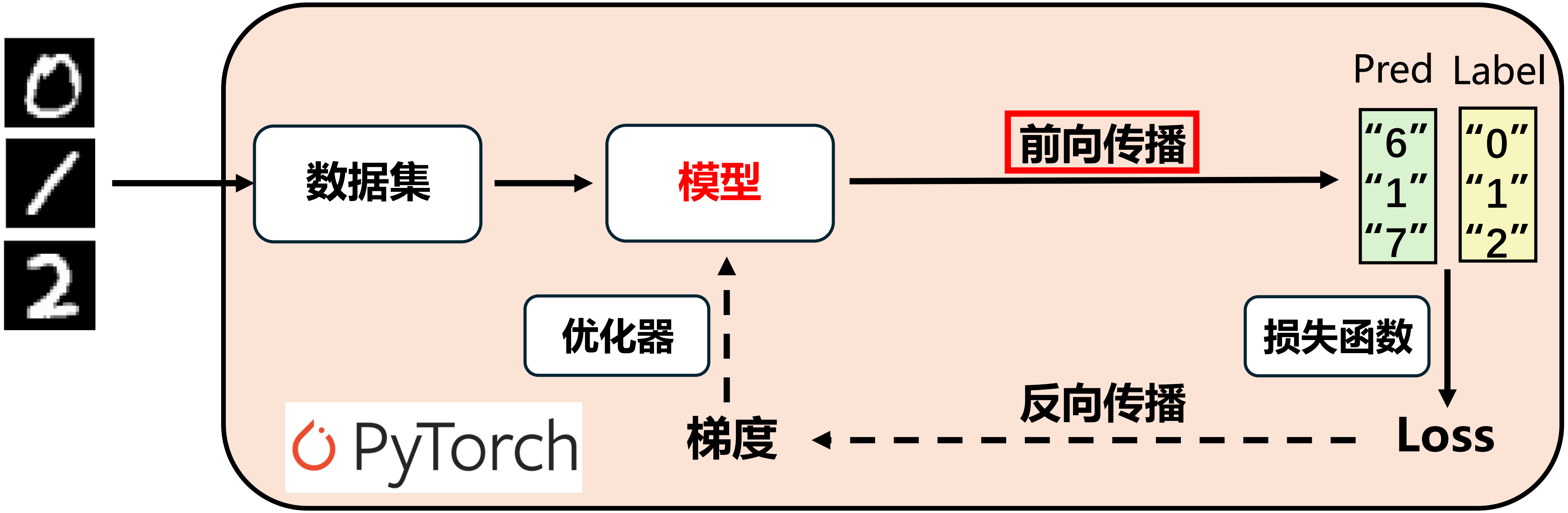
```python
class Mnist_CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=32,
        kernel_size=5, stride=1, padding=2)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2,
        padding=0)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64,
        kernel_size=5, stride=1, padding=2)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2,
        padding=0)
        self.fc1 = nn.Linear(7*7*64, 512)
        self.fc2 = nn.Linear(512, 10)

    def forward(self, inputs):
        tensor = F.relu(self.conv1(inputs))
        tensor = self.pool1(tensor)
        tensor = F.relu(self.conv2(tensor))
        tensor = self.pool2(tensor)
        tensor = tensor.view(-1, 7*7*64)
        tensor = F.relu(self.fc1(tensor))
        tensor = self.fc2(tensor)
        return tensor
```



逐步构建训练流程



- 针对手写数字识别这一任务，如何建立一个分类模型，并通过模型的训练过程不断进行优化，让计算机能较为准确地识别手写数字图片呢?



基于PyTorch实现手写数字识别

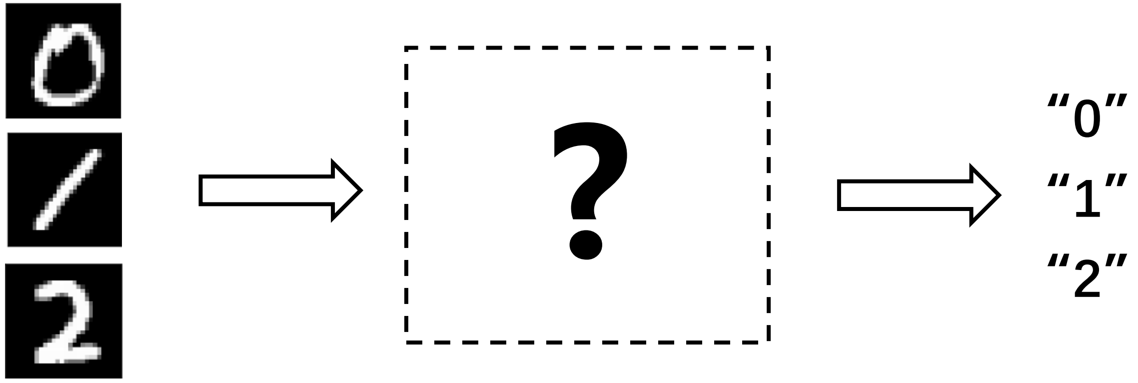
生成数据集、模型对象&定义损失函数、优化器

```
## 生成训练集
train_set = Mnist(
    root='data/MNIST/raw',
    train=True
)
train_loader = DataLoader(
    dataset=train_set, #输出的数据
    batch_size=32,
    shuffle=True #将元素随机排序
)
## 生成模型对象
net = Mnist_CNN()

## 选择数据和模型放置在 CPU/哪个GPU 上
device = torch.device("cuda", 0)#选择将程序放置到哪个GPU上
#device = torch.device('cpu')
net.to(device)

## 定义损失函数
loss_function = torch.nn.CrossEntropyLoss()
## 定义优化器
optimizer = optim.SGD(
    net.parameters(),#网络参数
    lr=0.001, #学习率
    momentum=0.9 #Momentum 用于加速 SGD（随机梯度下降）在某一方向上的
搜索以及抑制震荡的发生。
)
```

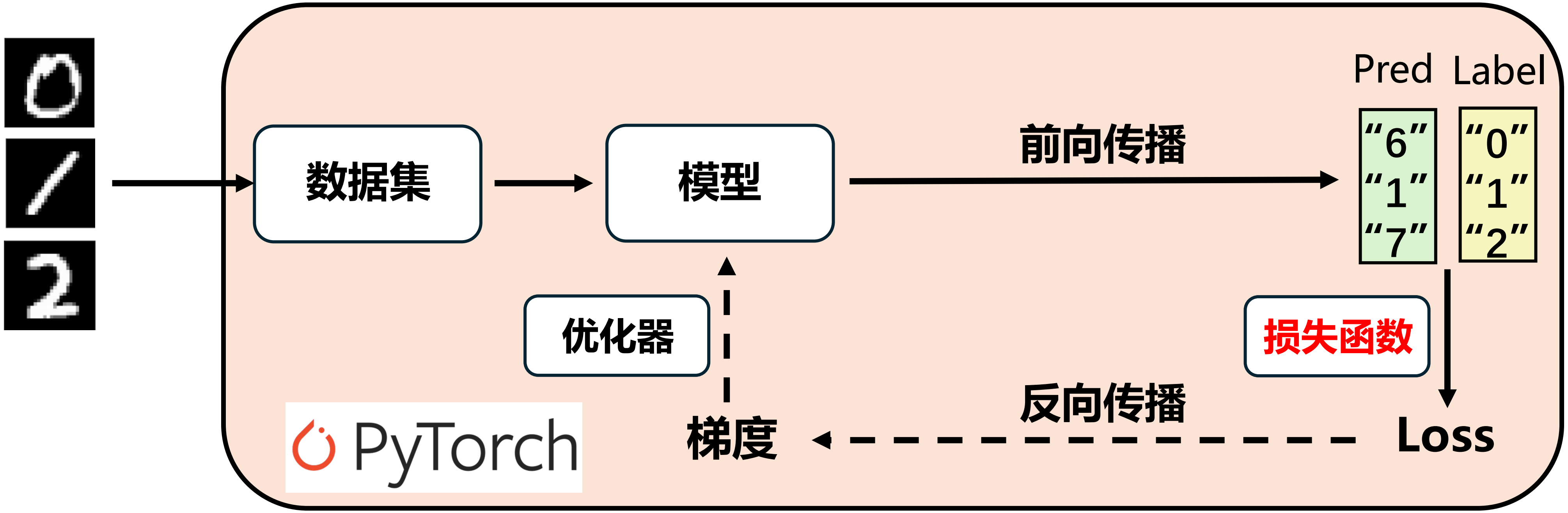
迭代训练（反向传播与模型优化）

```
loss_list,acc_list = [],[]
for epoch in range(10):#训练10次
    running_loss = 0.0
    total, correct=0,0
    for images, labels in tqdm(train_loader):
            # enumerate索引函数, start下标开始位置
        images=images.to(device)  #将images放进GPU
        labels=labels.to(device)  #将labels放进GPU

        optimizer.zero_grad()
            # 梯度清零, 初始化,如果不初始化, 则梯度会叠加
        outputs = net(images)                           # 前向传播
        loss = loss_function(outputs, labels)           # 计算误差
        loss.backward()                                 # 反向传播
        optimizer.step()                                # 权重更新

        running_loss += loss.item()                     # 误差累计
        _, predict = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predict == labels).sum()


    print('epoch:{:d} loss:{:.3f} acc:{:.3f}'
          .format(epoch+1, running_loss/len(train_loader),
            correct/total), flush=True)
            loss_list.append(running_loss/len(train_loader))
    acc_list.append(correct/total)

print('Finished Training!')

torch.save(net.state_dict(), "Linear.pth")      # 保存训练模型
```

逐步构建训练流程



- 针对手写数字识别这一任务，如何建立一个分类模型，并通过模型的训练过程不断进行优化，让计算机能较为准确地识别手写数字图片呢?



基于PyTorch实现手写数字识别

生成数据集、模型对象&定义损失函数、优化器

```
## 生成训练集
train_set = Mnist(
    root='data/MNIST/raw',
    train=True
)
train_loader = DataLoader(
    dataset=train_set, #输出的数据
    batch_size=32,
    shuffle=True #将元素随机排序
)
## 生成模型对象
net = Mnist_CNN()

## 选择数据和模型放置在 CPU/哪个GPU 上
device = torch.device("cuda", 0)#选择将程序放置到哪个GPU上
#device = torch.device('cpu')
net.to(device)

## 定义损失函数
loss_function = torch.nn.CrossEntropyLoss()
## 定义优化器
optimizer = optim.SGD(
    net.parameters(),#网络参数
    lr=0.001, #学习率
    momentum=0.9 #Momentum 用于加速 SGD（随机梯度下降）在某一方向上的
搜索以及抑制震荡的发生。
)
```

迭代训练（反向传播与模型优化）

```
loss_list,acc_list = [],[]
for epoch in range(10):#训练10次
    running_loss = 0.0
    total, correct=0,0
    for images, labels in tqdm(train_loader):
            # enumerate索引函数, start下标开始位置
        images=images.to(device)  #将images放进GPU
        labels=labels.to(device)  #将labels放进GPU

        optimizer.zero_grad()
            # 梯度清零, 初始化,如果不初始化, 则梯度会叠加
        outputs = net(images)                       # 前向传播
        loss = loss_function(outputs, labels)       # 计算误差
        loss.backward()                             # 反向传播
        optimizer.step()                            # 权重更新

        running_loss += loss.item()                 # 误差累计
        _, predict = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predict == labels).sum()


    print('epoch:{:d} loss:{:.3f} acc:{:.3f}'
          .format(epoch+1, running_loss/len(train_loader),
            correct/total), flush=True)
            loss_list.append(running_loss/len(train_loader))
    acc_list.append(correct/total)

print('Finished Training!')

torch.save(net.state_dict(), "Linear.pth")      # 保存训练模型
```

3. 损失函数

- **损失函数(Loss Function)**是模型在训练过程中需要优化的目标函数。Pytorch同样预设了非常简单易用的**类**来实现各种损失函数。例如，交叉熵损失函数：

- loss_function = torch.nn.CrossEntropyLoss () # 定义损失函数

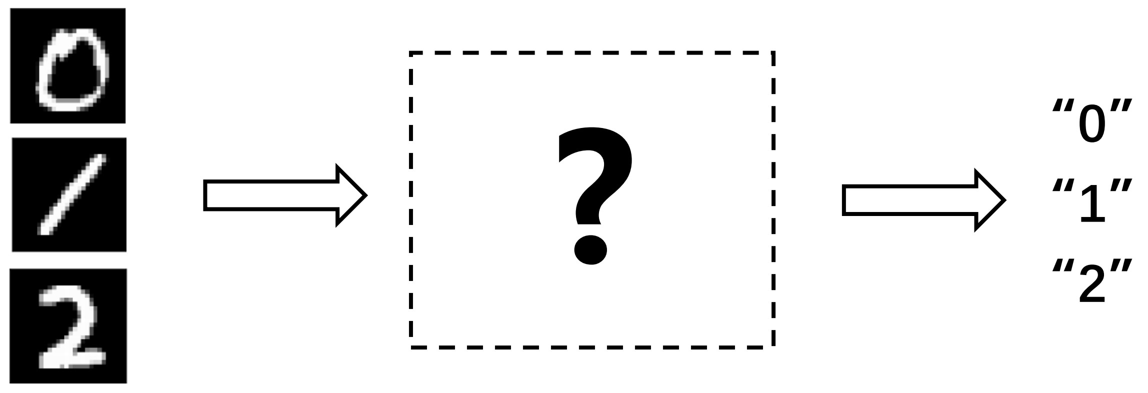
  ⋮

- loss = loss_function(outputs, labels) # 计算误差
- loss.backward() # 反向传播

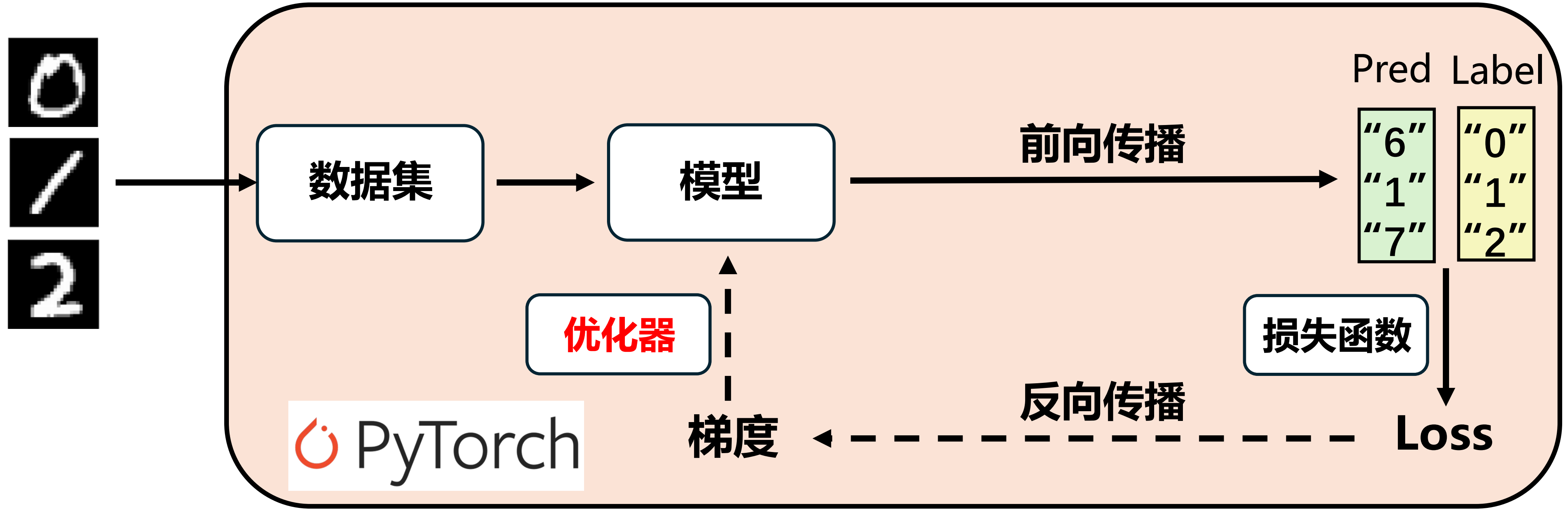
对所有需要进行梯度计算的变量 x (weight、bias) 进行梯度回传：

$$x.grad = x.grad + \frac{d}{dx}loss$$

逐步构建训练流程



- 针对手写数字识别这一任务，如何建立一个分类模型，并通过模型的训练过程不断进行优化，让计算机能较为准确地识别手写数字图片呢?



基于PyTorch实现手写数字识别

生成数据集、模型对象&定义损失函数、优化器

```
## 生成训练集
train_set = Mnist(
    root='data/MNIST/raw',
    train=True
)
train_loader = DataLoader(
    dataset=train_set, #输出的数据
    batch_size=32,
    shuffle=True #将元素随机排序
)
## 生成模型对象
net = Mnist_CNN()

## 选择数据和模型放置在 CPU/哪个GPU 上
device = torch.device("cuda", 0)#选择将程序放置到哪个GPU上
#device = torch.device('cpu')
net.to(device)

## 定义损失函数
loss_function = torch.nn.CrossEntropyLoss()
## 定义优化器
optimizer = optim.SGD(
    net.parameters(),#网络参数
    lr=0.001, #学习率
    momentum=0.9 #Momentum 用于加速 SGD（随机梯度下降）在某一方向上的搜索以及抑制震荡的发生。
)
```

迭代训练（反向传播与模型优化）

```
loss_list,acc_list = [],[]
for epoch in range(10):#训练10次
    running_loss = 0.0
    total, correct=0,0
    for images, labels in tqdm(train_loader):
            # enumerate索引函数，start下标开始位置
        images=images.to(device)  #将images放进GPU
        labels=labels.to(device)  #将labels放进GPU

        optimizer.zero_grad()
            # 梯度清零，初始化,如果不初始化，则梯度会叠加
        outputs = net(images)                         # 前向传播
        loss = loss_function(outputs, labels)         # 计算误差
        loss.backward()                               # 反向传播
        optimizer.step()                              # 权重更新

        running_loss += loss.item()                   # 误差累计
        _, predict = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predict == labels).sum()


    print('epoch:{:d} loss:{:.3f} acc:{:.3f}'
          .format(epoch+1, running_loss/len(train_loader),
            correct/total), flush=True)
    loss_list.append(running_loss/len(train_loader))
    acc_list.append(correct/total)

print('Finished Training!')

torch.save(net.state_dict(), "Linear.pth")      # 保存训练模型
```

4. 优化器

- **torch.optim**：由PyTorch提供的用于实现各种优化算法的包，其中以**对象**形式包含了各种不同的优化器，通过简单的定义即可在训练过程中使用不同的优化算法。例如SGD：

- optimizer = torch.optim.SGD(params, lr, momentum)

  - ➢ **params:** 需要优化的网络参数，如net.parameters，其中net就是用户定义的模型；
  - ➢ **lr:** 学习率（learning rate)，控制参数更新的步长；
  - ➢ **momentum**：即动量，用于加速 SGD（随机梯度下降）在某一方向上的搜索，并抑制震荡的发生。

4. 优化器

前向传播之前：

- optimizer.zero_grad()

使上一个batch计算的梯度清零，如果不清零，会叠加之前的梯度

反向传播之后：

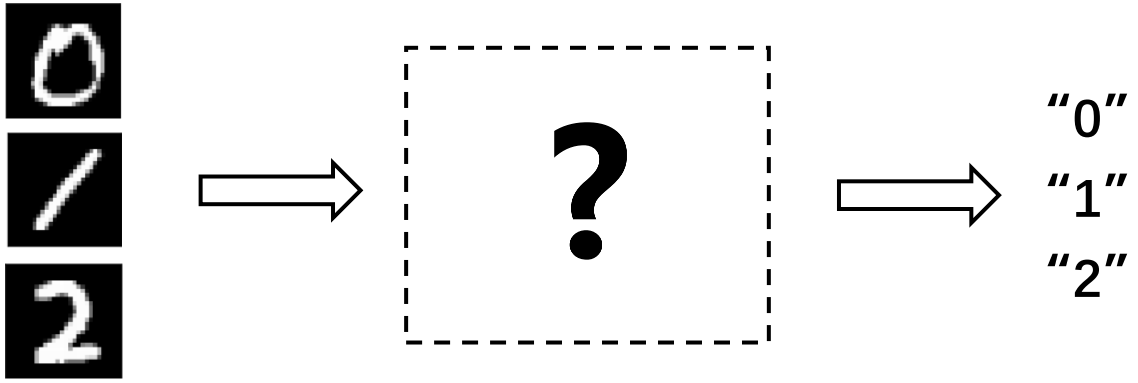
- optimizer.step()

调用后，模型各个参数将按照loss.backward() 时保存的梯度对模型参数进行更新

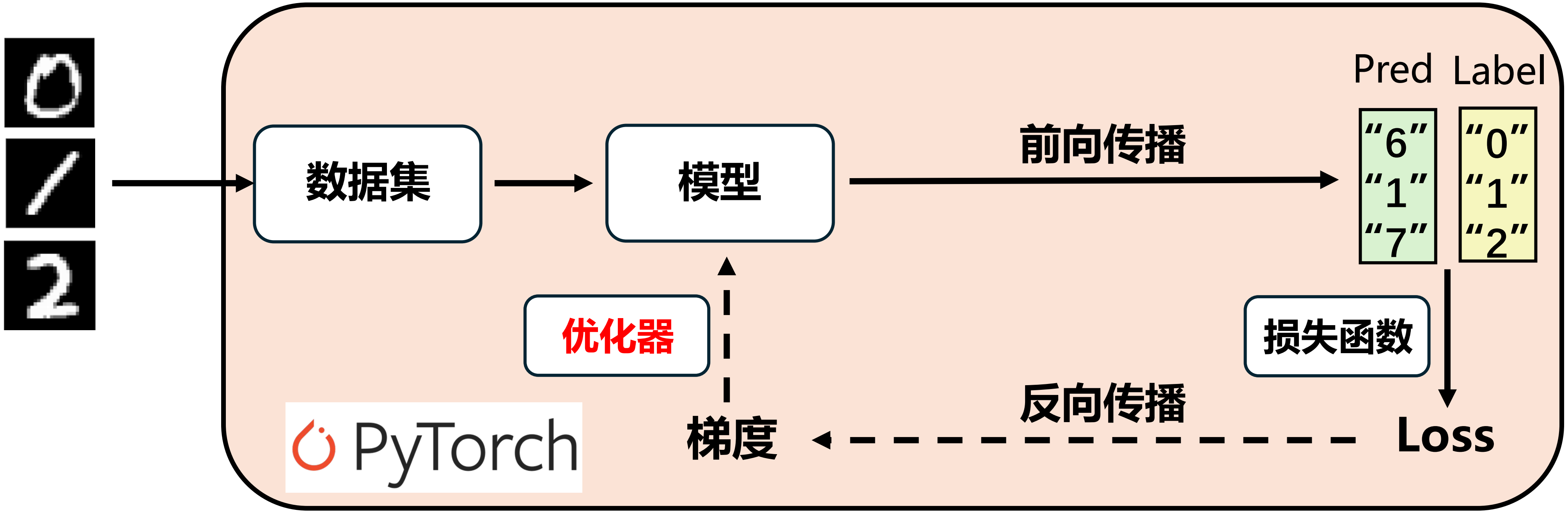
迭代训练（反向传播与模型优化）

```
loss_list,acc_list = [],[]
for epoch in range(10):#训练10次
    running_loss = 0.0
    total, correct=0,0
    for images, labels in tqdm(train_loader):
            # enumerate索引函数, start下标开始位置
        images=images.to(device)  #将images放进GPU
        labels=labels.to(device)  #将labels放进GPU

        optimizer.zero_grad()
            # 梯度清零, 初始化,如果不初始化, 则梯度会叠加
        outputs = net(images)                               # 前向传播
        loss = loss_function(outputs, labels)               # 计算误差
        loss.backward()                                     # 反向传播
        optimizer.step()                                    # 权重更新

        running_loss += loss.item()                         # 误差累计
        _, predict = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predict == labels).sum()


    print('epoch:{:d} loss:{:.3f} acc:{:.3f}'
          .format(epoch+1, running_loss/len(train_loader),
            correct/total), flush=True)
    loss_list.append(running_loss/len(train_loader))
    acc_list.append(correct/total)

print('Finished Training!')

torch.save(net.state_dict(), "Linear.pth")        # 保存训练模型
```

逐步构建训练流程



- 针对手写数字识别这一任务，如何建立一个分类模型，并通过模型的训练过程不断进行优化，让计算机能较为准确地识别手写数字图片呢?



基于PyTorch实现手写数字识别

导入PyTorch相关库

```
import torch
from torch.utils.data import Dataset, DataLoader

import torch.nn as nn
import torch.nn.functional as F

import torch.optim as optim
```

模型

```
class Mnist_CNN(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(in_channels=1, out_channels=32,
             kernel_size=5, stride=1, padding=2)
        self.pool1 = nn.MaxPool2d(kernel_size=2, stride=2,
             padding=0)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64,
             kernel_size=5, stride=1, padding=2)
        self.pool2 = nn.MaxPool2d(kernel_size=2, stride=2,
             padding=0)
        self.fc1 = nn.Linear(7*7*64, 512)
        self.fc2 = nn.Linear(512, 10)

    def forward(self, inputs):
        tensor = F.relu(self.conv1(inputs))
        tensor = self.pool1(tensor)
        tensor = F.relu(self.conv2(tensor))
        tensor = self.pool2(tensor)
        tensor = tensor.view(-1, 7*7*64)
        tensor = F.relu(self.fc1(tensor))
        tensor = self.fc2(tensor)
        return tensor
```

数据集

```
class Mnist(Dataset):
    def __init__(self, root, train=True, transform=torch.tensor):
        self.file_pre = 'train' if train == True else 't10k'
        self.transform = transform #定义变换函数
        self.label_path = os.path.join(root,
                                    '%s-labels-idx1-ubyte.gz' % self.file_pre)
        self.image_path = os.path.join(root,
                                    '%s-images-idx3-ubyte.gz' % self.file_pre)
        # 读取文件数据，返回图片和标签
        self.images, self.labels = self.__read_data__(
            self.image_path,
            self.label_path)

    def __read_data__(self, image_path, label_path):
        # 数据集读取
        with gzip.open(label_path, 'rb') as lbpath:
            labels = np.frombuffer(lbpath.read(), np.uint8, offset=8) #将data以流的
             形式读入转化成ndarray对象，ndarray对象是用于存放同类型元素的多维数组
        with gzip.open(image_path, 'rb') as imgpath:
            images = np.frombuffer(imgpath.read(), np.uint8,
             offset=16).reshape(len(labels), 1, 28, 28) #将图片以标签文件的元素个数读
             取，设置大小为28*28
        return images, labels

    def __getitem__(self, index):
        image, label = np.array(self.images[index], dtype=np.float32)/255,
             int(self.labels[index])

        if self.transform is not None:
            image = self.transform(image)  # 此处需要用 np.array(image)，转化为数组
        return image, label

    def __len__(self):
        return len(self.labels)
```

基于PyTorch实现手写数字识别

生成数据集、模型对象&定义损失函数、优化器

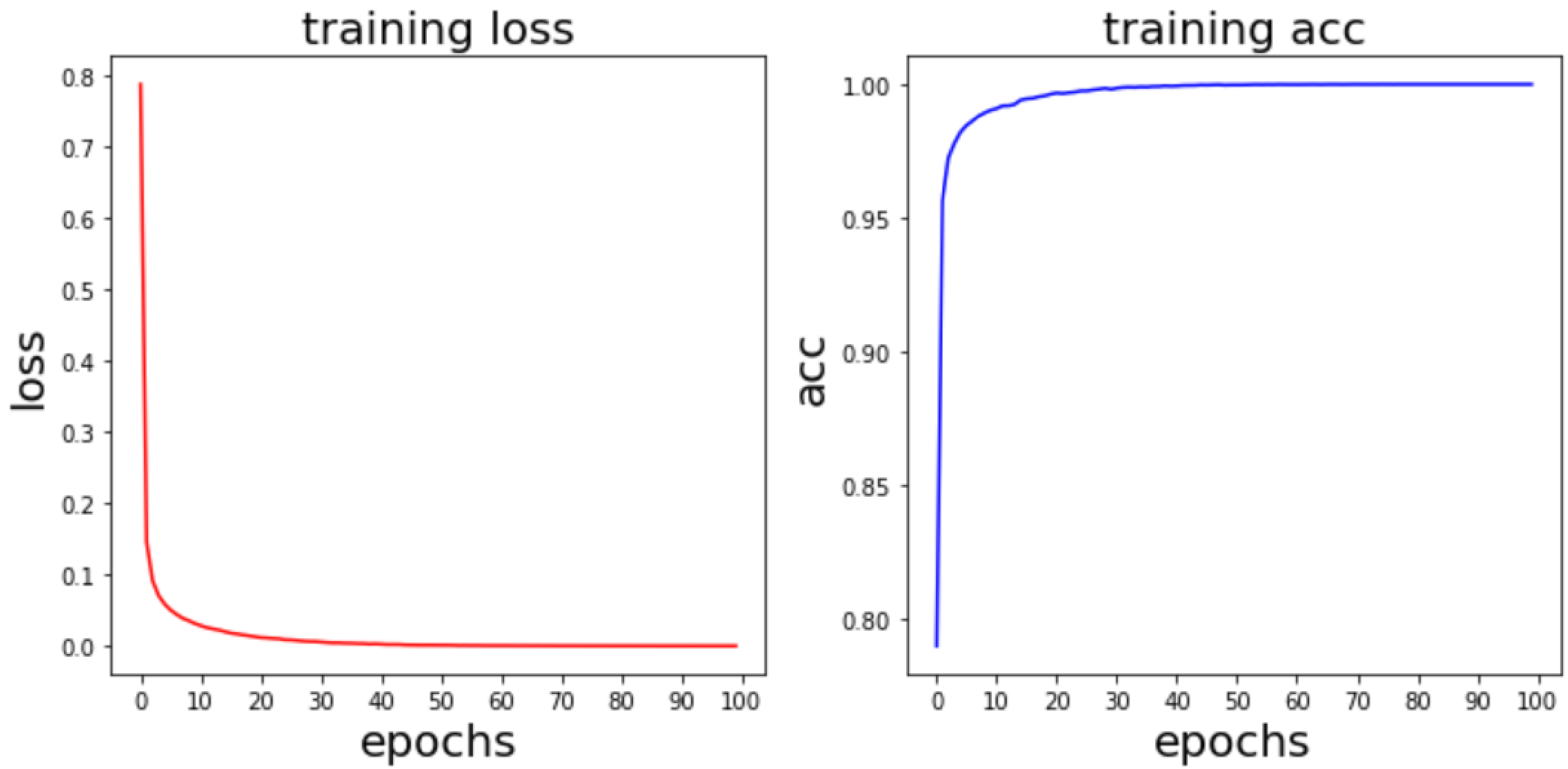
```
## 生成训练集
train_set = Mnist(
    root='data/MNIST/raw',
    train=True
)
train_loader = DataLoader(
    dataset=train_set, #输出的数据
    batch_size=32,
    shuffle=True #将元素随机排序
)
## 生成模型对象
net = Mnist_CNN()

## 选择数据和模型放置在 CPU/哪个GPU 上
device = torch.device("cuda", 0)#选择将程序放置到哪个GPU上
#device = torch.device('cpu')
net.to(device)

## 定义损失函数
loss_function = torch.nn.CrossEntropyLoss()
## 定义优化器
optimizer = optim.SGD(
    net.parameters(),#网络参数
    lr=0.001, #学习率
    momentum=0.9 #Momentum 用于加速 SGD（随机梯度下降）在某一方向上的
搜索以及抑制震荡的发生。
)
```

迭代训练（反向传播与模型优化）

```
loss_list,acc_list = [],[]
for epoch in range(5):#训练10次
    running_loss = 0.0
    total, correct=0,0
    for images, labels in tqdm(train_loader):
            # enumerate索引函数, start下标开始位置
        images=images.to(device)  #将images放进GPU
        labels=labels.to(device)  #将labels放进GPU

        optimizer.zero_grad()
            # 梯度清零, 初始化,如果不初始化, 则梯度会叠加
        outputs = net(images)                        # 前向传播
        loss = loss_function(outputs, labels)        # 计算误差
        loss.backward()                              # 反向传播
        optimizer.step()                             # 权重更新

        running_loss += loss.item()                  # 误差累计
        _, predict = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predict == labels).sum()


    print('epoch:{:d} loss:{:.3f} acc:{:.3f}'
          .format(epoch+1, running_loss/len(train_loader),
            correct/total), flush=True)
    loss_list.append(running_loss/len(train_loader))
    acc_list.append(correct/total)

print('Finished Training!')

torch.save(net.state_dict(), "Linear.pth")      # 保存训练模型
```

模型开始训练

——如果模型能够收敛，training loss将不断下降，training acc将不断上升



课后练习

1. nn.CrossEntropyLoss()可以直接用模型输出的预测概率进行计算而不需要转换成 one-hot形式，其他损失函数可以吗？请你调研一下PyTorch中其他常用损失函数的使用方法。
2. PyTorch对所有模型参数都以Tensor的数据类型保存，请你通过PyTorch官网等资源学习Tensor的性质与常见操作。

Reference

- https://pytorch.org/tutorials
- https://speech.ee.ntu.edu.tw/~hylee/ml/ml2023-course-data/Pytorch_Tutorial_1_rev_1.pdf