



智能系统安全实践：手写数字识别

复旦白泽智能
系统软件与安全实验室



大纲



- 理解手写数字识别任务
 - 如何通过有监督学习来解决此任务
- 理解神经网络模型
 - 理解神经网络的基本结构
- 手写数字识别数据集上的实验
 - 基于自动微分实现手写数字识别
 - 基于PyTorch框架实现手写数字识别

有监督学习：回顾

■ 有监督学习的过程

■ 准备工作

收集训练集 $(x^{(i)}, y^{(i)})$

建立模型 $f(x)$

■ 训练模型

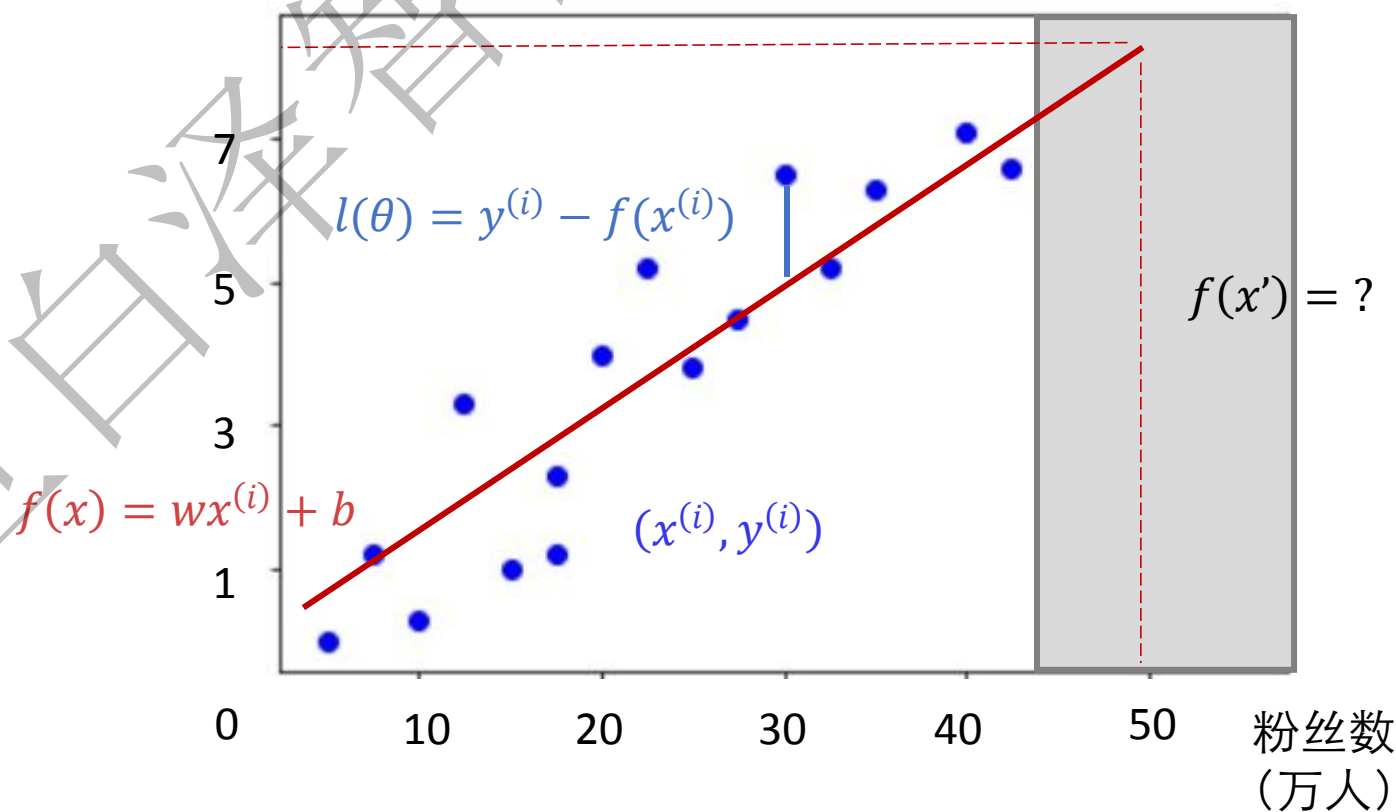
设定损失函数 $l(\theta)$

通过梯度下降最小化 $l(\theta)$

■ 模型预测

在新数据点上计算 $f(x)$

直播间销量
(万元)



有监督学习：回顾

■ 实例: 红酒分类

■ 准备工作

收集训练集 $(x^{(i)}, y^{(i)}) \rightarrow$ 收集每款红酒的属性值和等级

建立模型 $f(x) \rightarrow f(x) = Wx + b$ (多维线性模型)

$\underbrace{\quad}_{p(\text{等级1}), p(\text{等级2}), \dots}$
 \uparrow
 $\underbrace{\quad}_{\text{属性1, 属性2, } \dots}$

■ 训练模型

设定损失函数 $l(\theta) \rightarrow$ 交叉熵损失函数

通过梯度下降最小化 $l(\theta)$

■ 模型预测

在新数据点上计算 $f(x) \rightarrow f_j(x)$ 最大的 j 即为预测的等级

属性	值
固定酸度	8.319637
挥发物	0.527821
柠檬酸	0.270976
糖分	2.538806
氯化物	0.087467
游离SO ₂	15.874922
总SO ₂	46.467792
密度	0.996747
PH值	3.311113
硫酸盐	0.658149
酒精度	10.422983

红酒等级: {0, 1, 2}

有监督学习：手写数字识别

■ 应用: 手写数字识别

■ 准备工作

收集训练集 $(x^{(i)}, y^{(i)}) \rightarrow$ 手写数字图像数据集的形式？

建立模型 $f(x) \rightarrow f(x) = ?$

■ 训练模型

设定损失函数 $l(\theta) \rightarrow$ 交叉熵损失函数

通过梯度下降最小化 $l(\theta) \rightarrow$ 如何高效更新？

■ 模型预测

在新数据点上计算 $f(x)$



数字识别: 0-9

手写数字识别：数据集形式

■ 手写数字图像数据集

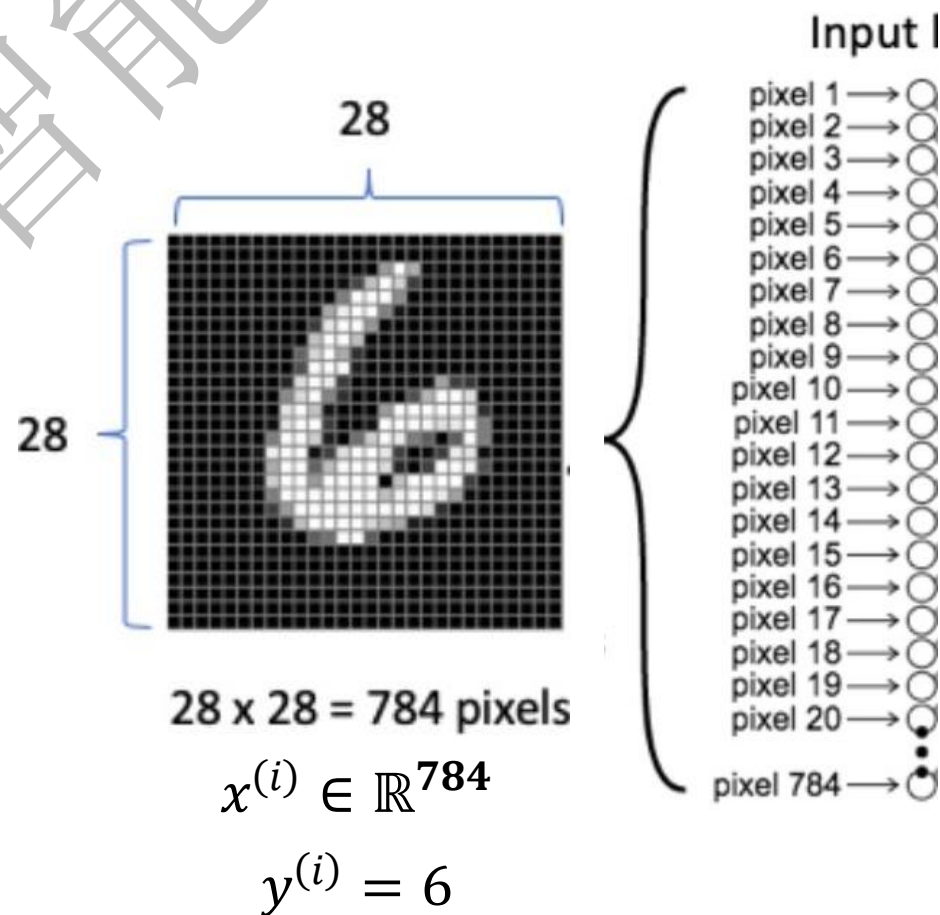
- 图像存储为28x28的灰度图
- 每个像素值0-1，越接近1越白
- 每个图像 → 784个像素值

■ 输入图像形式

- 把28x28的像素值按行拼接成一个长向量
- 每个图像 $x^{(i)} \in \mathbb{R}^{784}$ ，第 k 个维度 $x_k^{(i)} \in [0,1]$

■ 标签形式

- $y^{(i)} \in \{0,1,2,3,4,5,6,7,8,9\}$



手写数字识别：多层神经网络 (MLP)

■ 模型定义 (以3层为例)

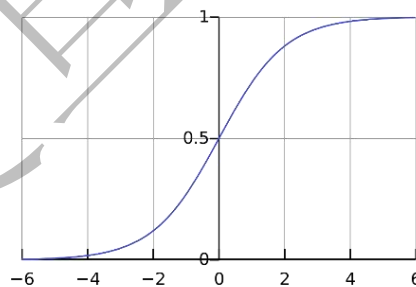
■ 输入层 - $x \in \mathbb{R}^{784}$

■ 隐藏层 - $h \in \mathbb{R}^{100}$ (包含100个神经元)

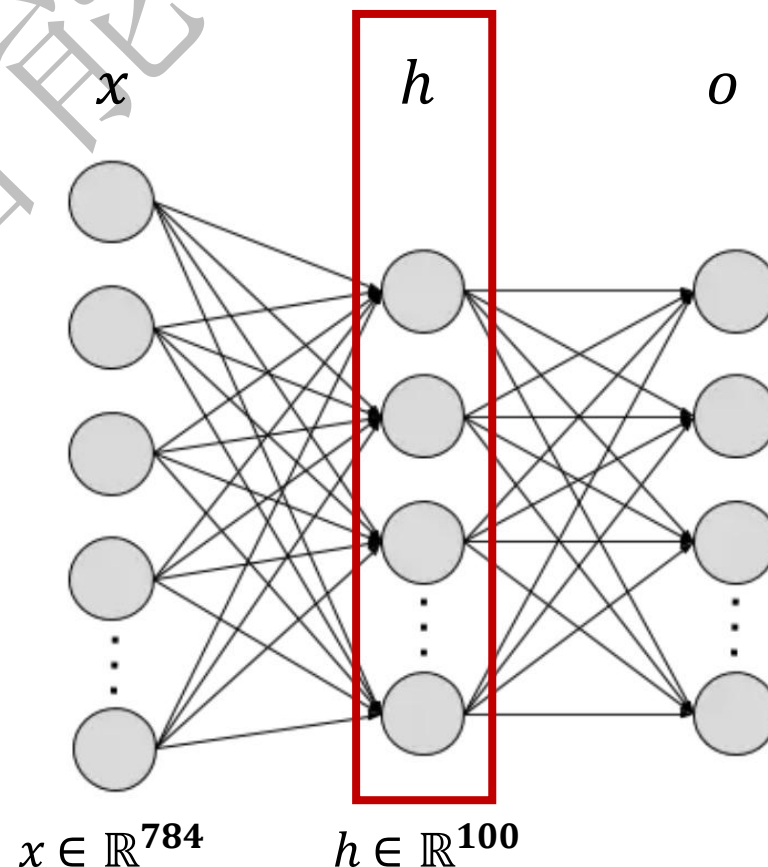
$$W_h \in \mathbb{R}^{100 \times 784}$$

$$h = \sigma(W_h x + b_h), \text{ 其中 } \sigma(h) = \frac{1}{1+e^{-h}}$$

$$b_h \in \mathbb{R}^{100}$$



激活函数不改变维度



手写数字识别：多层神经网络 (MLP)

■ 模型定义 (以3层为例)

■ 输入层 - $x \in \mathbb{R}^{784}$

■ 隐藏层 - $h \in \mathbb{R}^{100}$ (包含100个神经元)

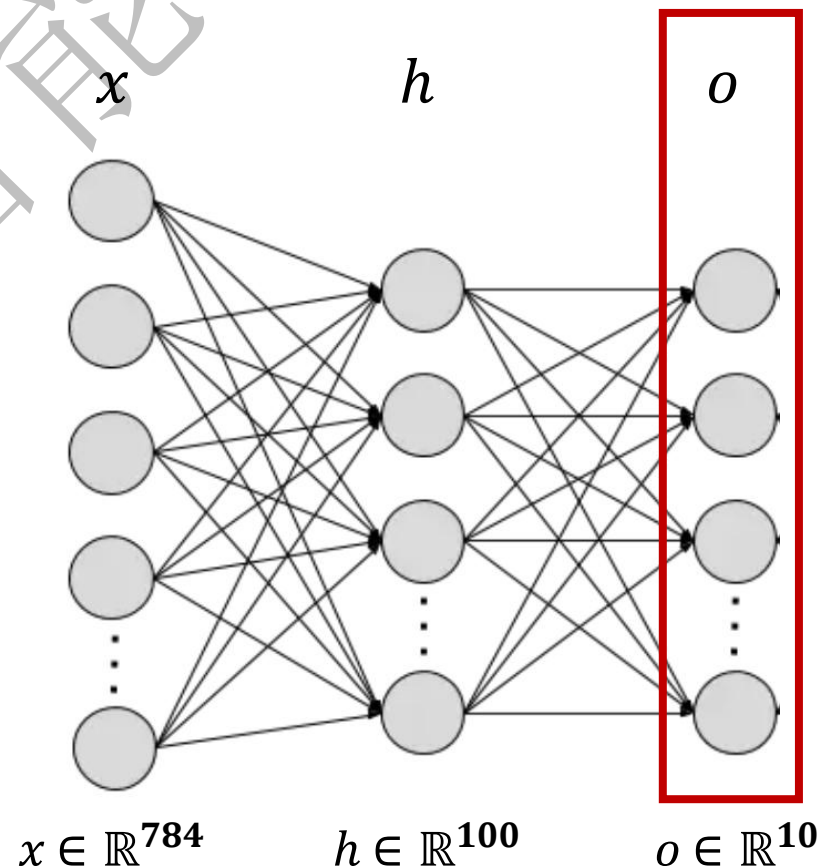
■ 输出层 - $o \in \mathbb{R}^{10}$

$$W_o \in \mathbb{R}^{10 \times 100}$$

$$o = W_o h + b_o$$

$$b_o \in \mathbb{R}^{10}$$

o_j 表示模型认为图像是数字 j 的置信度



手写数字识别：损失函数

■ 模型定义

- $h = \sigma(W_h x + b_h)$

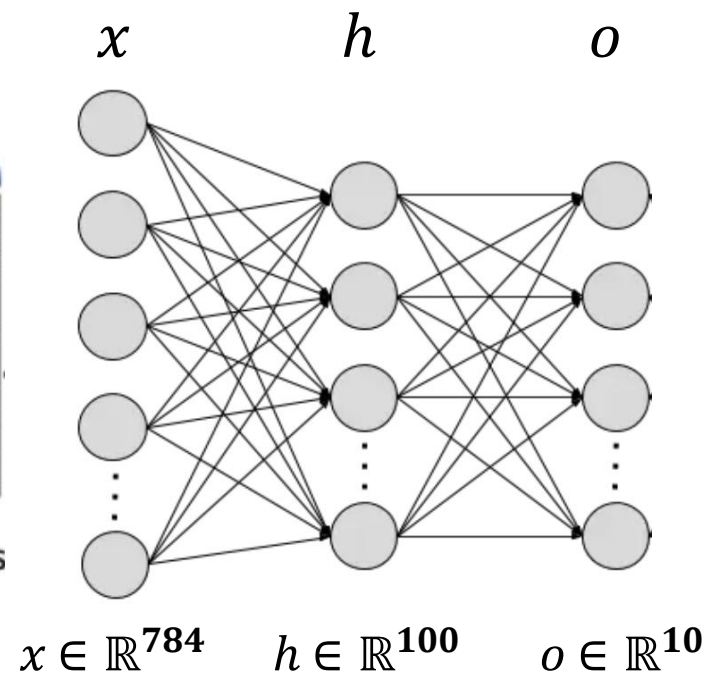
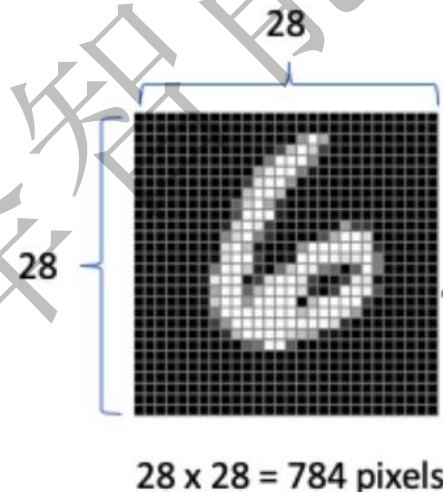
- $o = W_o h + b_o$

■ 损失函数

- 把标签 $y^{(i)}$ 转换为one-hot形式 $\tilde{y}^{(i)}$

- 把预测结果 $f(x^{(i)})$ 通过Softmax变成 $\tilde{f}(x^{(i)})$

- 计算 $\tilde{f}(x^{(i)})$ 与 $\tilde{y}^{(i)}$ 的交叉熵损失函数 $\ell(\theta) = \sum_{i=1}^N CE(f(x^{(i)}), y^{(i)})$



手写数字识别：模型训练

■ 模型定义

- $h = \sigma(W_h x + b_h)$

- $o = W_o h + b_o$

■ 模型训练

- 通过梯度下降训练参数 W_h, b_h, W_o, b_o

- 计算损失函数 → 计算参数的偏导数 → 更新参数 $\ell(\theta) = \sum_{i=1}^N CE(f(x^{(i)}), y^{(i)})$

- 存在问题: 手写数字识别的数据集中, $N = 60000$

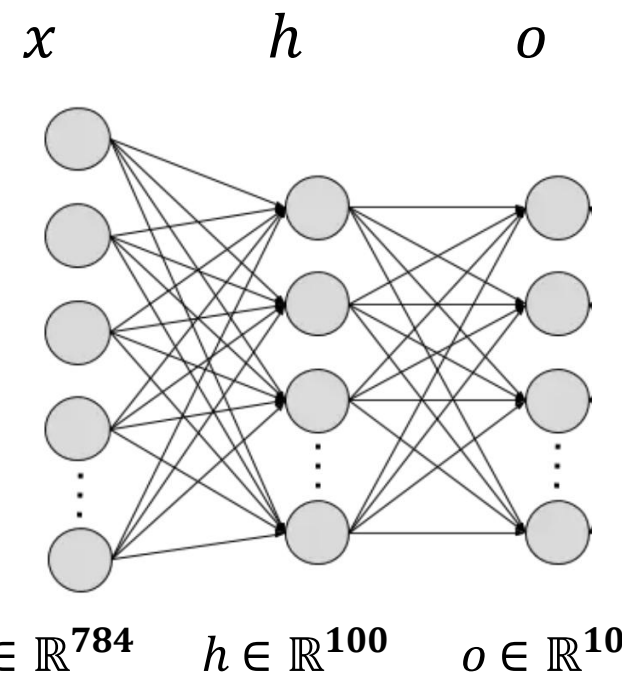
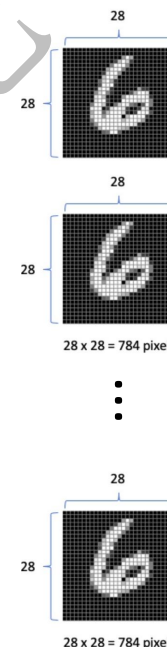
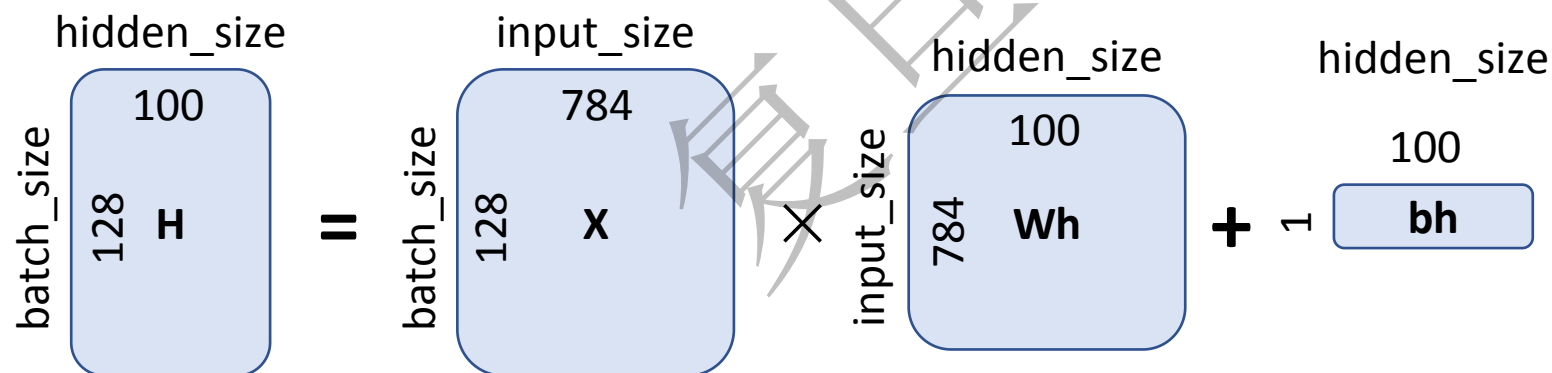
- 解决方案: 每次随机取一部分数据(batch)计算损失函数, 然后更新参数
(Batch Stochastic Gradient Descent, Batch SGD)

手写数字识别：模型训练

■ Batch SGD

- 每次随机取一部分数据(batch)输入模型，进行训练
- 输入从1x784变成128x784
- 对整个batch一起计算损失函数，反向更新参数

■ e.g., 隐藏层 $h = \sigma(W_h x + b_h)$



有监督学习：手写数字识别

■ 回顾整个框架

■ 准备工作

收集训练集 $(x^{(i)}, y^{(i)}) \rightarrow$ 图像表示为 $x^{(i)} \in \mathbb{R}^{784}$

建立模型 $f(x) \rightarrow$ MLP模型结构

■ 训练模型

设定损失函数 $l(\theta) \rightarrow$ 交叉熵损失函数

通过梯度下降最小化 $l(\theta) \rightarrow$ Batch SGD

■ 模型预测

在新数据点上计算 $f(x)$



数字识别: 0-9

Q&A



实验部分1

基于自动微分的手写数字识别

手写数字识别



■ 解决训练中的问题

■ 前向过程: 矩阵乘法

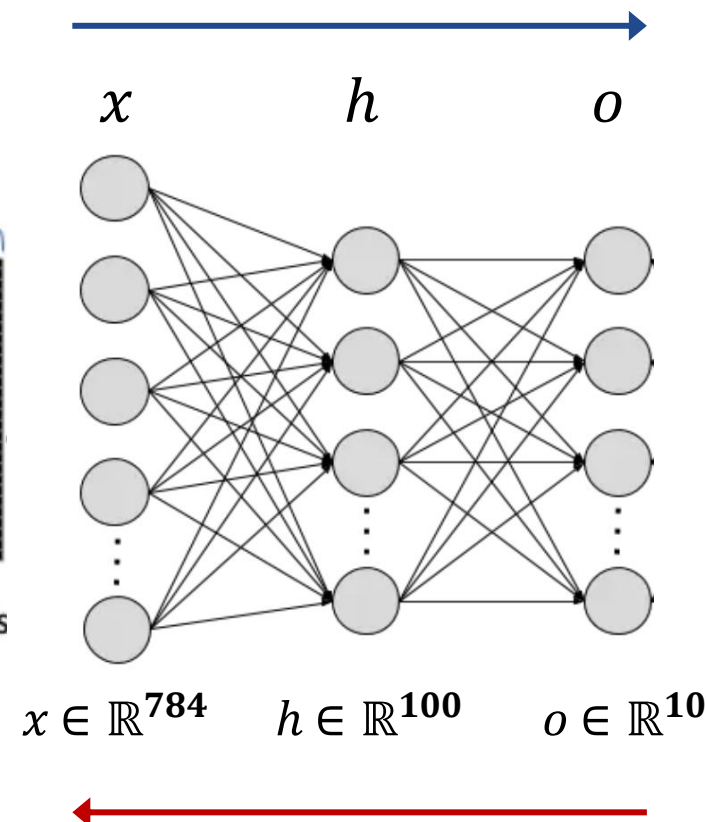
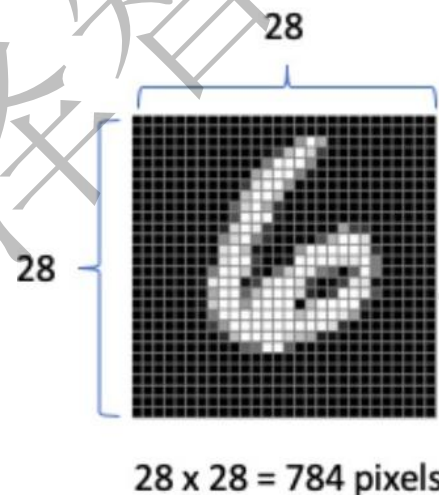
$$h = \sigma(W_h x + b_h)$$

$$o = W_o h + b_o$$

■ 反向传播: 如何对参数求偏导+更新?

$$l(\theta) = \sum_{i=1}^N CE(f(x^{(i)}), y^{(i)})$$

$$\frac{\partial l(\theta)}{\partial W_h}, \frac{\partial l(\theta)}{\partial b_h}, \frac{\partial l(\theta)}{\partial W_o}, \frac{\partial l(\theta)}{\partial b_o}?$$

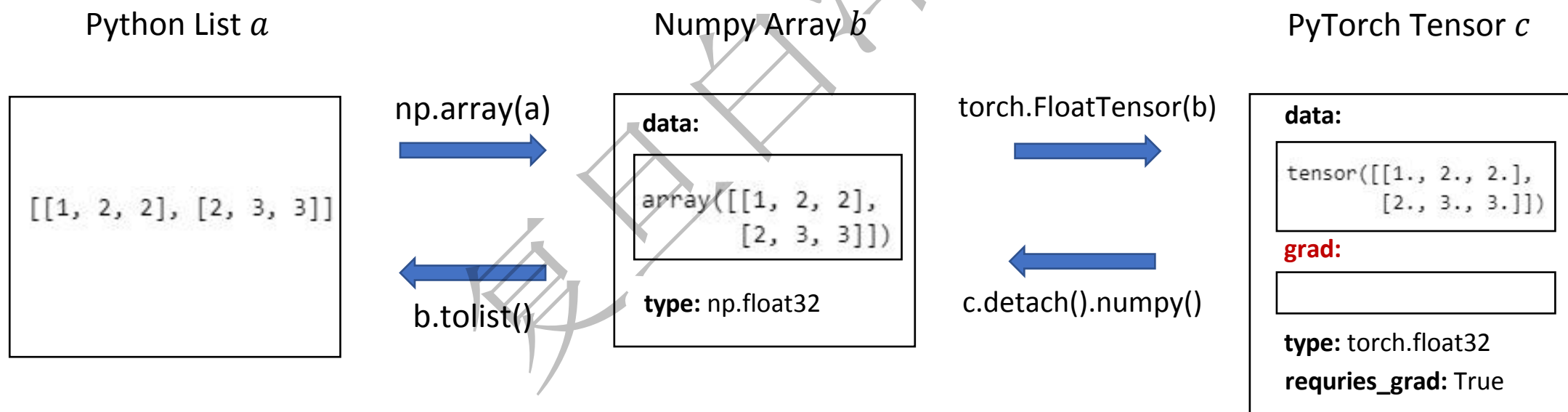


PyTorch: 科学计算库



■ Pytorch库

- 对NumPy做了进一步的封装，结合链式求导法则，能实现自动微分
- 参数向量以tensor形式存储，pytorch能自动求出梯度



PyTorch: 科学计算库



■ 自动微分

- 参数以tensor形式储存

- 训练时定义损失函数

- **loss.backward()** 计算每个参数的梯度

- **.grad** 命令获取某个变量的梯度值

```
import torch
```

```
a = torch.FloatTensor([3.0, 2.0, 3.0])  
a.requires_grad = True  
a
```

```
1]: tensor([3., 2., 3.], requires_grad=True)
```

```
loss = ((a - 1) ** 2).sum()
```

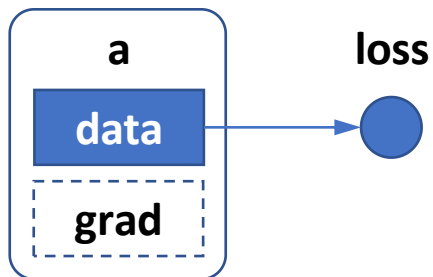
```
loss.backward()
```

```
a.grad
```

```
2]: tensor([4., 2., 4.]
```

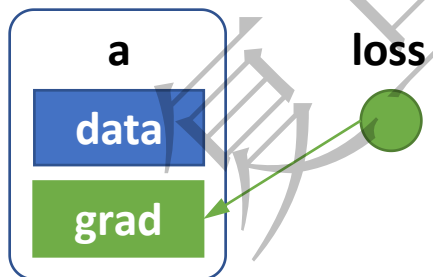
PyTorch: 自动微分

■ 正向过程



■ 反向传播: 自动微分

■ 调用loss.backward()后



```
import torch
```

```
a = torch.FloatTensor([3.0, 2.0, 3.0])
a.requires_grad = True
a
```

```
1]: tensor([3., 2., 3.], requires_grad=True)
```

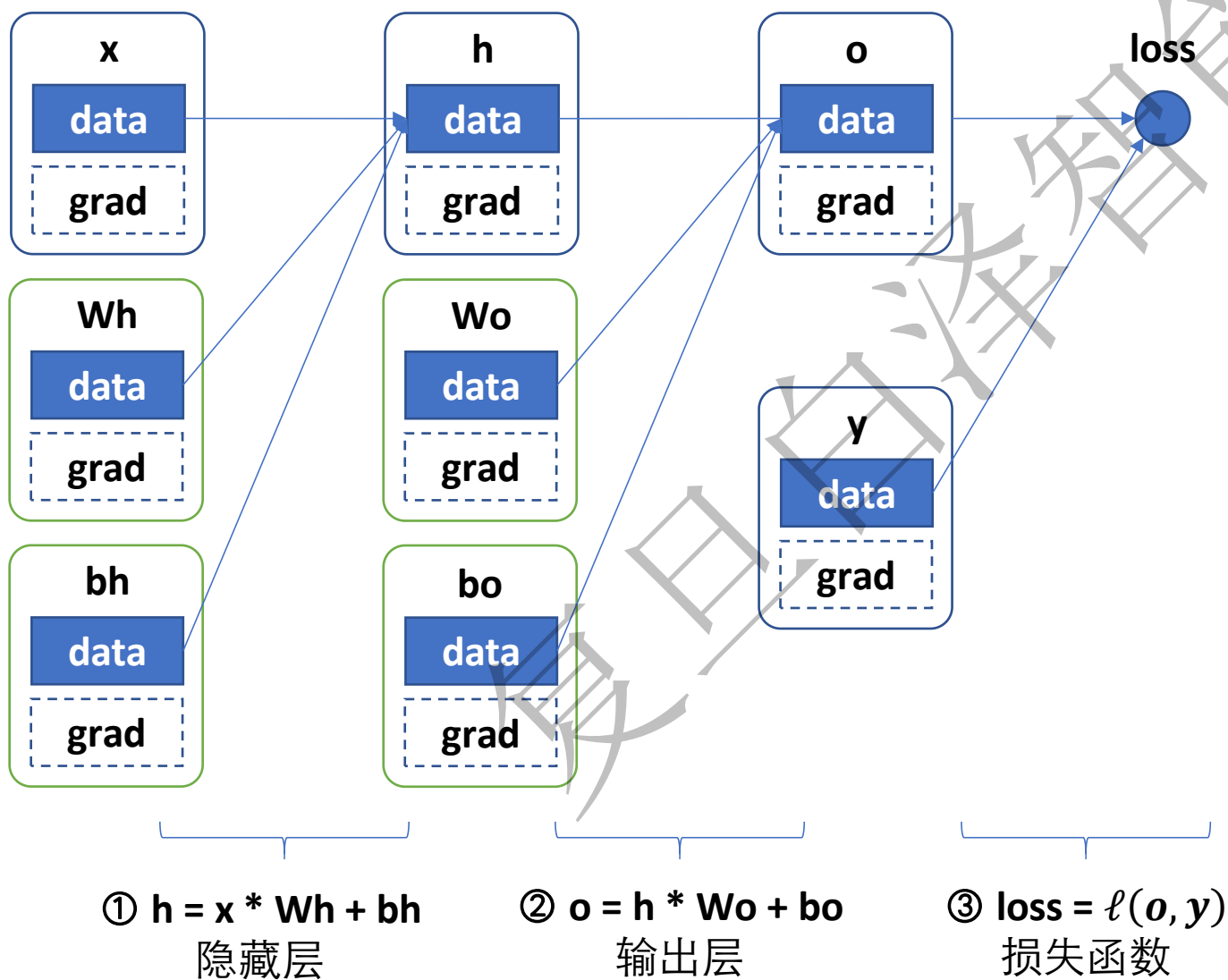
```
loss = ((a - 1) ** 2).sum()
```

```
loss.backward()
```

```
a.grad
```

```
2]: tensor([4., 2., 4.])
```

PyTorch: 自动微分

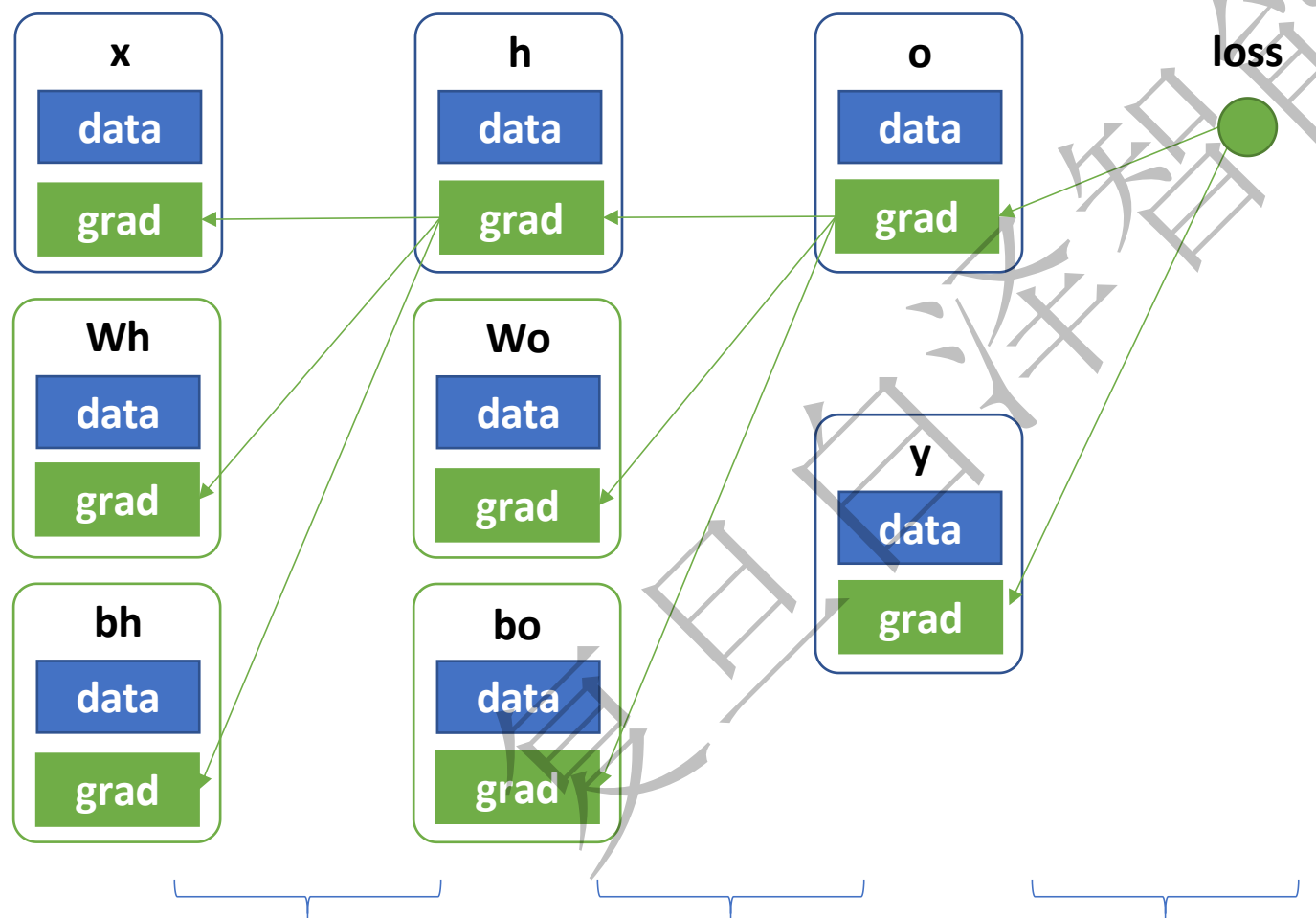


```
1 Wh = torch.FloatTensor(Wh)
2 bh = torch.FloatTensor(bh)
3 Wo = torch.FloatTensor(Wo)
4 bo = torch.FloatTensor(bo)
5
6 Wh.requires_grad = True
7 bh.requires_grad = True
8 Wo.requires_grad = True
9 bo.requires_grad = True
```

```
1 def forward(x, Wh, bh, Wo, bo):
2
3
4
5     return o
```

```
1 loss_func = torch.nn.CrossEntropyLoss()
2
3 loss = loss_func(o, y)
4
```

PyTorch : 自动微分



① $h = x * Wh + bh$
隐藏层

② $o = h * Wo + bo$
输出层

③ $loss = \ell(o, y)$
损失函数

```
1 Wh = torch.FloatTensor(Wh)
2 bh = torch.FloatTensor(bh)
3 Wo = torch.FloatTensor(Wo)
4 bo = torch.FloatTensor(bo)
5
6 Wh.requires_grad = True
7 bh.requires_grad = True
8 Wo.requires_grad = True
9 bo.requires_grad = True
```

```
1 def forward(x, Wh, bh, Wo, bo):
2
3
4
5     return o
```

```
4
5 loss.backward()
6
```

实验任务1: 手写数字识别



■ 数据形式

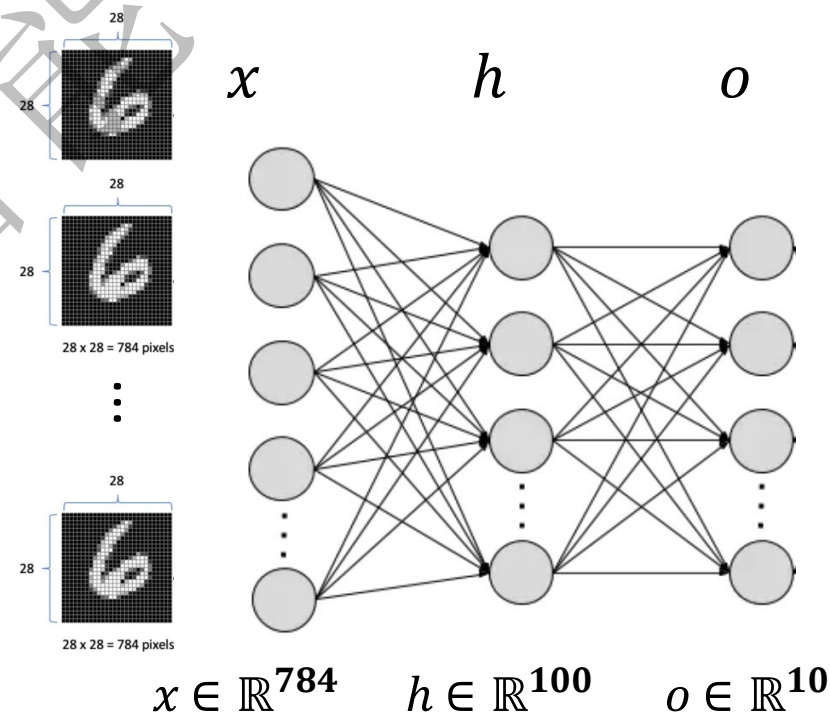
- 输入一个batch: $x \in \mathbb{R}^{128 \times 784}$
- batch对应的标签: $y = \{0 - 9\}^{128}$

■ 参数初始化

- 定义 W_h, b_h, W_o, b_o 并随机初始化

■ 前向/反向过程

- 根据 x, y 输出 o , 计算损失函数
- 调用`loss.backward()`计算梯度
- 利用梯度信息更新参数



实验任务1: 手写数字识别

■ Week3_Task1_Question.ipynb :

```
1 def forward(x, Wh, bh, Wo, bo):
2
3     # TODO: 根据上述函数描述, 利用给定的API实现模型前向预测计算
4
5     return o
```

■ Step1:

实现模型的正向过程(forward函数)

```
: 1 def train(dataloader, lr, Wh, bh, Wo, bo):
2     for x, y in dataloader:
3         # reshape x from [batch_size, 28, 28] to [batch_size, 784]
4         x = x.reshape(-1, 28 * 28)
5
6         # TODO: 根据上述函数描述, 利用已经实现好的forward, calc_grad函数, 获取参数梯度, 实现参数的SGD更新
7
8     return Wh, bh, Wo, bo
```

■ Step2:

调用反向传播, 获取参数梯度
更新模型参数

```
1 def predict(x, Wh, bh, Wo, bo):
2     # TODO: 根据上述函数描述, 利用已经实现好的forward函数和给定的参考API, 实现模型的预测
3
4     return prediction
5
6
7 def test(dataloader, Wh, bh, Wo, bo):
8     correct = 0
9     total = 0
10    for x, y in dataloader:
11        # reshape x from [batch_size, 28, 28] to [batch_size, 784]
12        x = x.reshape(-1, 28 * 28)
13
14        # forward & predict
15        prediction = predict(x, Wh, bh, Wo, bo)
16
17        # TODO: 根据上述函数描述, 实现模型的预测准确率统计
18
19    return correct / total
```

■ Step3:

在整个数据集上训练模型
观察准确率

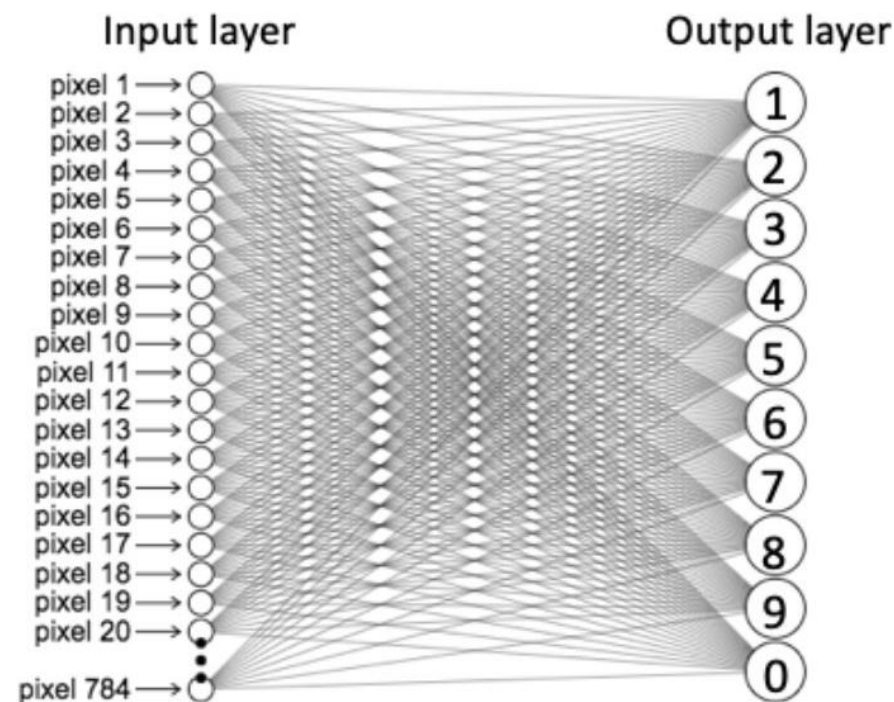
实验任务1: 检查内容

■ 默认参数设置

- 反向更新时的学习率 $lr=0.01$ ，迭代轮数 $epoches=10$
- batch大小=128，对模型参数进行随机初始化

■ 检查内容

- 默认参数下的测试集准确度
- 自行尝试不同的超参数，观察结果变化
- 记录下3组不同的(超参数-准确度)数据



实验任务1: Bonus



■ 梯度计算

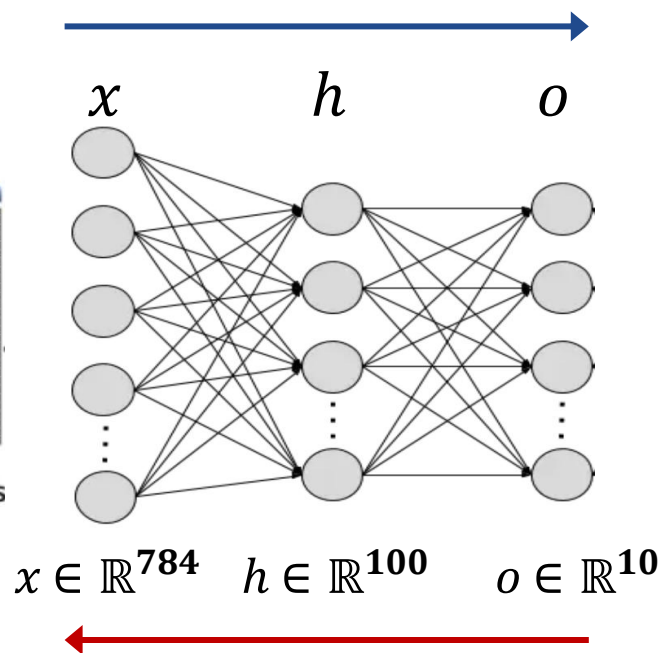
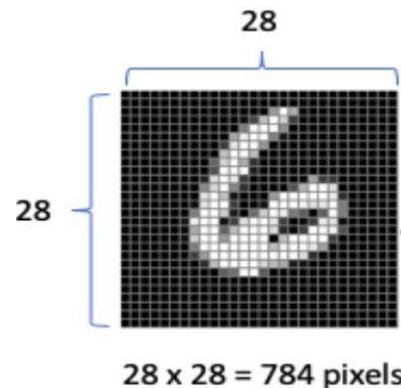
- 代码中使用了Pytorch框架进行自动微分
- 如何利用链式法则，计算参数 W_h, b_h, W_o, b_o 的偏导数？

$$h = \sigma(W_h x + b_h)$$

$$o = W_o h + b_o$$

$$l(\theta) = \sum_{i=1}^N CE(f(x^{(i)}), y^{(i)})$$

$$\frac{\partial l(\theta)}{\partial W_h}, \frac{\partial l(\theta)}{\partial b_h}, \frac{\partial l(\theta)}{\partial W_o}, \frac{\partial l(\theta)}{\partial b_o}?$$



实验任务1: Bonus

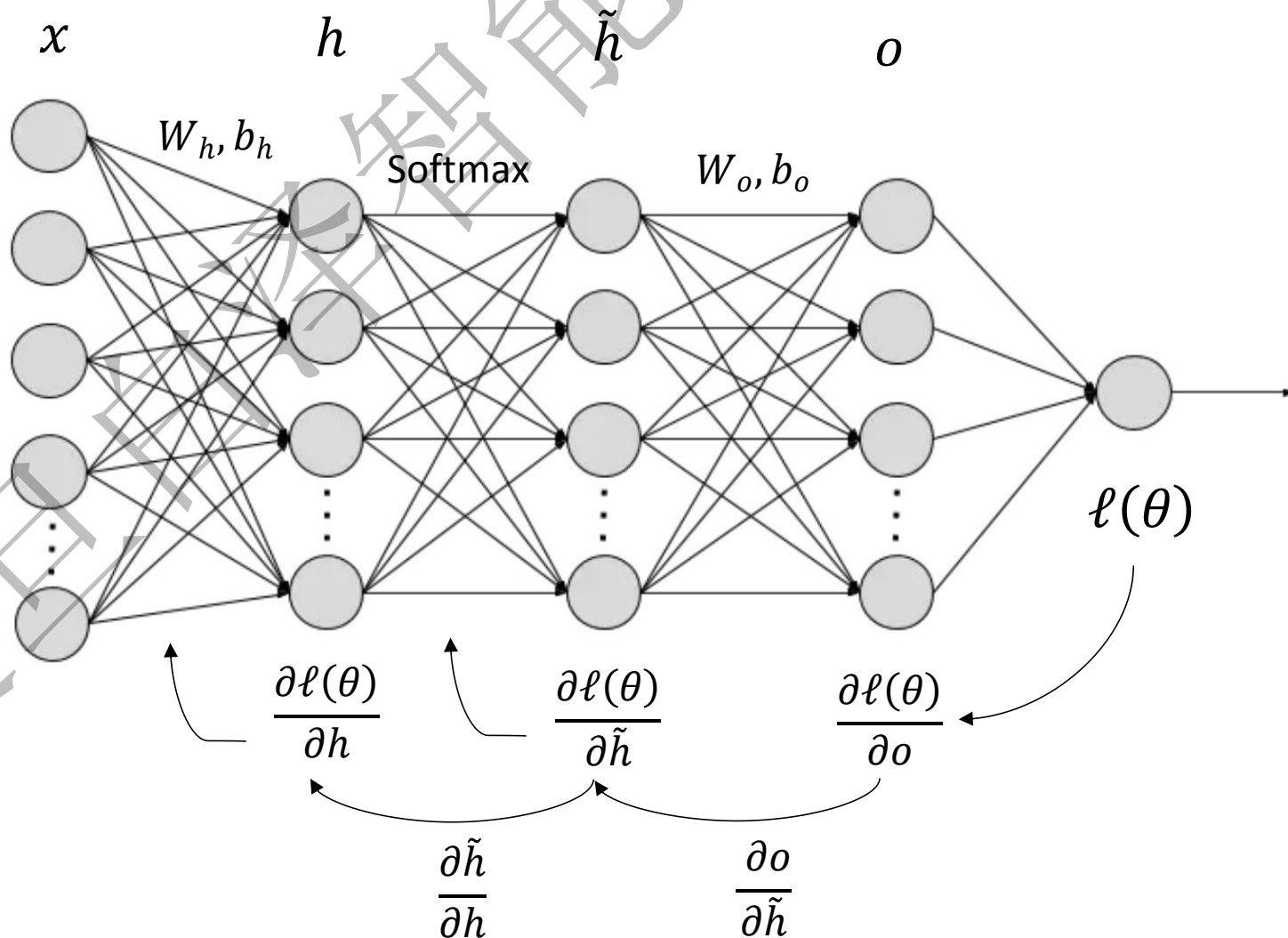


■ 链式法则

$$\frac{\partial \ell(\theta)}{\partial W_o} = \frac{\partial \ell(\theta)}{\partial o} \cdot \frac{\partial o}{\partial W_o}$$

$$\frac{\partial \ell(\theta)}{\partial W_h} = \frac{\partial \ell(\theta)}{\partial h} \cdot \frac{\partial h}{\partial W_h}$$

$$\begin{aligned} &= \frac{\partial \ell(\theta)}{\partial o} \cdot \frac{\partial o}{\partial h} \cdot \frac{\partial h}{\partial W_h} \\ &= \frac{\partial \ell(\theta)}{\partial o} \cdot \frac{\partial o}{\partial \tilde{h}} \cdot \frac{\partial \tilde{h}}{\partial h} \cdot \frac{\partial h}{\partial W_h} \end{aligned}$$



实验任务1: Bonus

■ 检查内容

- 在Week3_Task1_Question.ipynb中自行补充train_handcraft()函数
- 不使用loss.backward()函数
- 手动推导 W_h, b_h, W_o, b_o 的梯度
- 通过矩阵运算来实现梯度计算

```
9 def train_handcraft(x, y, lr):
10     global Wh, bh, Wo, bo
11     features, prediction = predict(x)
12     Wh_grad = torch.zeros(Wh.shape)
13     bh_grad = torch.zeros(bh.shape)
14     Wo_grad = torch.zeros(Wo.shape)
15     bo_grad = torch.zeros(bo.shape)
16
17     for batch in range(batch_size):
18         pred = prediction[batch]
19         y_vector = np.zeros(bo.shape)
20         y_vector[y[batch]] = 1.
21         y_vector = torch.Tensor(y_vector)
22
23         # TODO:手动计算当前batch的梯度, 并实现对模型参数的梯度更新过程
24
```

Q&A



实验部分2

基于PyTorch框架的手写数字识别

PyTorch: 内置模块

- 除了提供矩阵计算和自动微分，PyTorch还提供神经网络模块接口
- 定义神经网络：继承`torch.nn.Module`类
 - 重写类函数，实现三个功能：定义参数 W, b 、初始化参数、forward计算
- 步骤：
 1. 重写init方法，定义参数和初始化
 2. 重写forward方法，定义前向计算过程
 3. 初始化该类，得到模型实例
 4. 通过`model(x)`调用forward函数，执行预测

```
class MyLinear(torch.nn.Module):  
    def __init__(self, in_dim, out_dim):  
        super(MyLinear, self).__init__()  
        self.W = ...  
        self.b = ...  
  
    def forward(self, x):  
        o = torch.mm(self.w, x) + self.b  
        return o  
  
linear = MyLinear(764, 100)  
o = linear(x)
```


PyTorch: 内置模块

- 定义模型时，PyTorch内置了大量计算模块

- e.g., torch.nn.Linear类 (线性模型)

- torch.nn.Module类还支持嵌套调用

- 在一个module内使用另一个module
- 利用上述两点，可以快速搭建模型

- 定义模型后，如何训练模型参数？

```
class MLP(torch.nn.Module):  
    def __init__(self):  
        super(MLP, self).__init__()  
        self.fc1 = torch.nn.Linear(764, 100)  
        self.fc2 = ...  
  
    def forward(self, x):  
        h = self.fc1(x)  
        ...  
        return o
```

PyTorch: 模型优化

■ 之前思路 (Task 1)

- 调用反向传播，获取参数梯度 \Rightarrow 更新模型参数
- 通过parameters()函数，可以获取一个Module内所有的模型参数

```
1 mlp = MLP()  
2 mlp.parameters()
```

```
<generator object Module.parameters at 0x7f8c678af820>
```

■ 利用Optimizer类 (Task 2)

- PyTorch提供了一种更简洁的方式
- 如torch.optim.SGD类 (SGD梯度下降策略)
 - PyTorch在类内帮我们实现了获取参数梯度，并做SGD更新的步骤

PyTorch: 模型优化



■ 定义优化器实现参数更新

■ Optimizer类用法

1. 初始化torch.optim.SGD实例
2. 输入模型参数、定义的学习速率
3. 在每一步更新中：

先用optimizer.zero_grad()清除上一步的梯度信息
对梯度进行反向传播，loss.backward()
调用optimizer.step()完成参数更新

```
optimizer = torch.optim.SGD(mlp.parameters(), lr=lr)

for x, y in dataloader:
    optimizer.zero_grad()

    o = model(x)
    loss = loss_func(o, y)
    loss.backward()

    optimizer.step()
```

小结：PyTorch内置模块

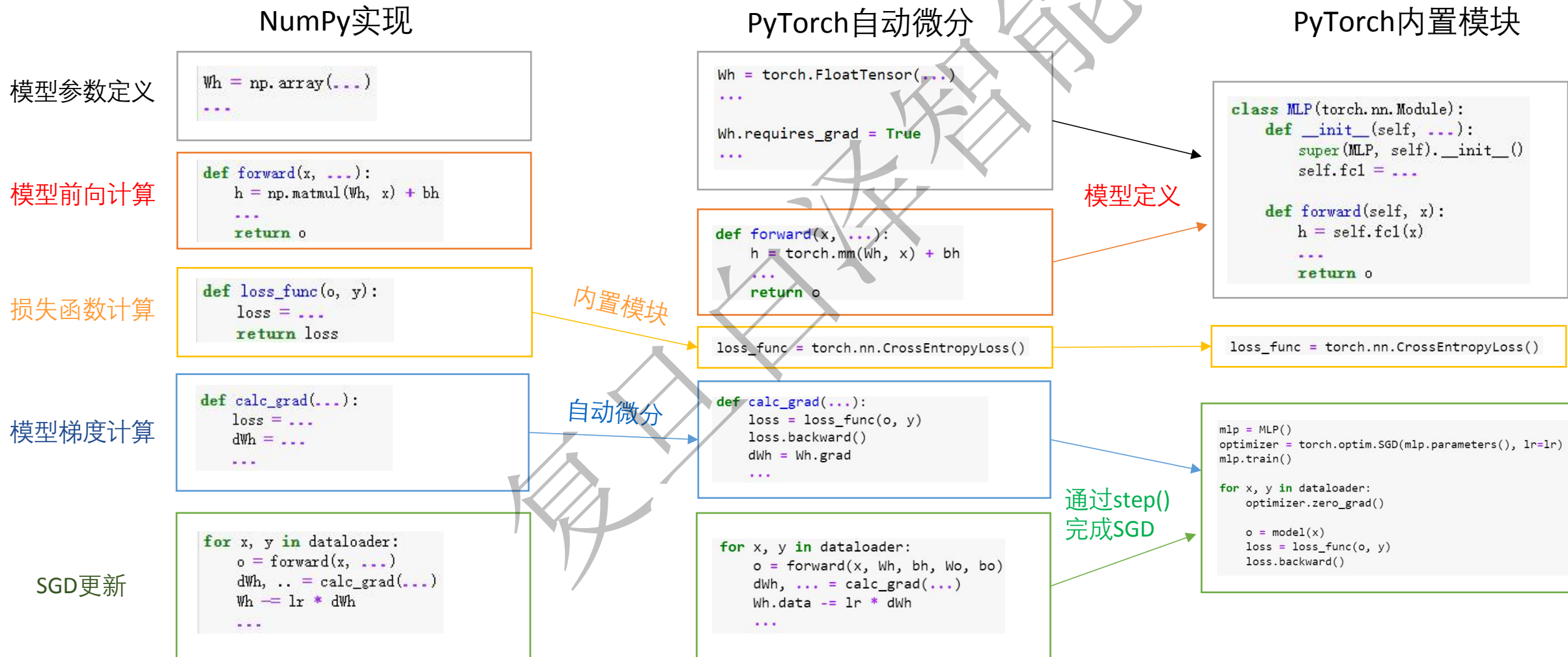
■ 三个核心类

- 继承`torch.nn.Module`，实现自定义的神经网络模块
- 初始化`torch.nn.loss`，用于计算损失函数
- 初始化`torch.optim.SGD`类，用于自动SGD更新

■ 训练步骤

- 通过`model(x)`获取模型输出
- 计算损失函数，并使用`backward()`计算梯度
- 利用`optimizer.step()`做更新

小结：MLP模型的训练



实验任务2: 基于Pytorch框架的手写数字识别



■ 实现一个4层神经网络

- Week3_Task2_Question.ipynb

- 完成init函数中的模型定义

- 完成forward函数中的正向过程

- 完成训练过程中的反向梯度传播、参数更新

- 训练模型，观察准确度，与Task1的结果比较

■ 额外尝试

- 查找PyTorch的API

- 用torch.optim中其他优化器完成训练

```
class MLP(nn.Module):  
    def __init__(self):  
        super(MLP, self).__init__()  
        # TODO: 定义上文要求的MLP模型结构  
  
    def forward(self, x):  
        # TODO: 定义MLP模型的前向过程  
  
        return 0
```

```
for e in range(epochs):  
    t = tqdm(train_loader)  
    for img, label in t:  
        # Forward img and compute loss  
        pred = mlp(img)  
        loss = criterion(pred, label)  
  
        # TODO: 基于优化器的使用方法，完成反向梯度传播、参数更新  
  
    t.set_postfix(epoch=e, train_loss=loss.item())
```

Q&A