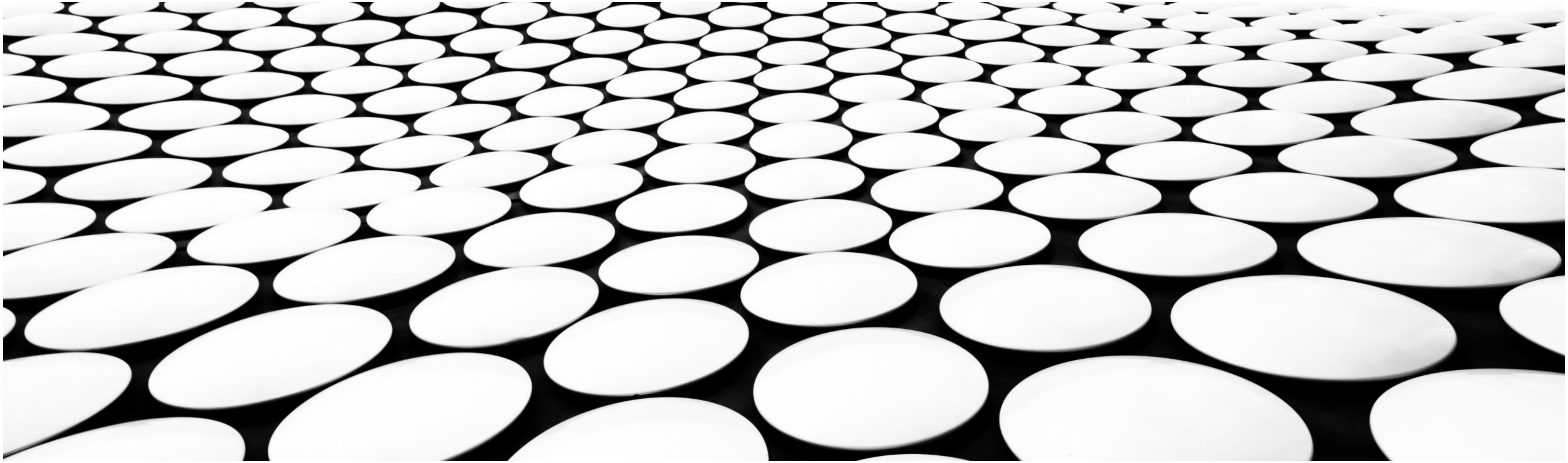
深度学习

邱怡轩



今天的主题

- 卷积神经网络实践
- 一般化建模方法
- 优化方法——SGD

实践

练习

- 参见 `lec7-convnet.ipynb`

一般化建模方法

神经网络建模

- 从建模的角度看，CNN 与前馈神经网络并没有本质上的不同
- 神经元 + 参数 => 新的神经元

预测 分类

模块化

- 可以将各种类型的层抽象成具有统一接口的模块
- 输入 => 输出
- 可学习的参数
- 导数计算机制

模块化

- 全连接层
  - 线性变换
  - 参数为权重矩阵和偏置向量
- 激活函数层
  - 固定的非线性变换
  - 无参数（某些会带少量参数）
- 卷积层
  - 卷积运算（也是一种线性变换）
  - 参数为卷积核和偏置向量

一般流程

- 搭建模型
  - 模型 = 结构 + 参数
- 计算损失函数（标量，通常按输入取平均）
- 对参数求导
- 最优化、参数更新
- 重复、迭代

问题

- 模型怎么选?
- 损失函数怎么选?
- 导数怎么求?
- 最优化方法怎么选?

模型

- 根据数据和问题特点选择层的类型
- 例如卷积层的引入正是为了处理图片数据
- 神经元的数目、层的深度等超参数通常需要微调和试验，也可借鉴已有文献成果
- 一些最新的研究在关注自动化的神经网络架构搜索（neural architecture search，NAS）

损失函数

- 本质上是一个统计问题
- 大部分情况下根据极大似然准则导出
- 也有许多其他的损失函数，如 SVM

自己的原则

导数

- 手动计算反向传播
- 现代软件框架的自动微分

最优化

- 基础的随机梯度下降
- 更多改良的优化算法

优化方法

传统模型

- 在传统统计模型中，经常使用牛顿法迭代
- 同时利用一阶导和二阶导的信息
- 如果参数数量是 p
- 一阶导是梯度 g，大小为 $p \times 1$
- 二阶导是 Hessian 矩阵 H，大小为 $p \times p$
- 牛顿法需计算 $H^{-1}g$，复杂度为 $O(p^3)$

一阶方法

- 对于深度学习模型，参数数量非常多
- 几乎只能依赖于梯度
- 通常称为一阶优化方法

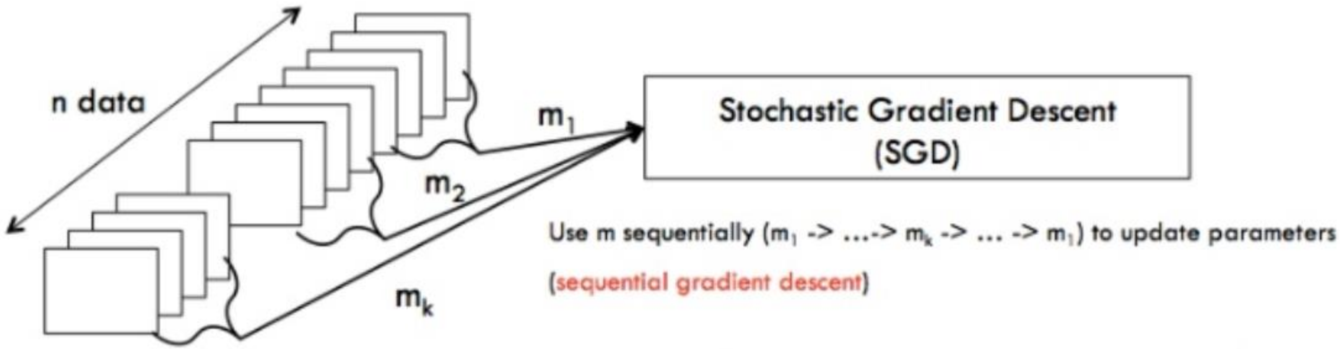
GD

- Gradient descent
- 梯度下降法，或最速下降法
- $\theta^{(k+1)} = \theta^{(k)} - \eta^{(k)} \cdot \nabla_\theta L(\theta^{(k)})$
- η 称为步长或学习率
- $\nabla_\theta L(\theta^{(k)})$ 是精确的梯度，由所有观测计算

SGD

- Stochastic gradient descent
- 随机梯度下降 每次使用单个样本
- $\theta^{(k+1)} = \theta^{(k)} - \eta^{(k)} \cdot \nabla_\theta l(\theta^{(k)})$
- $\nabla_\theta l(\theta^{(k)})$ 是 $\nabla_\theta L(\theta^{(k)})$ 的一个无偏估计
- 例如通过 mini-batch 计算而来

Mini-batch



n data
m_1
m_2
m_k
Stochastic Gradient Descent
(SGD)
Use m sequentially (m_1 -> ...-> m_k -> ... -> m_1) to update parameters
(sequential gradient descent)

思考题

- 数据共有 n 个观测，每个 mini-batch 大小为 m，如有剩余则单独形成一个 mini-batch

- 如何简单计算 mini-batch 的数量?
- 用 Python 实现

$(n-1)//m+1$

SGD

- SGD 的重要意义在于，即使利用随机的、不精确的梯度
- 在一定条件下也可以保证收敛到导数为0的点
- $\sum_{k=1}^{\infty} \eta^{(k)} = \infty$, $\sum_{k=1}^{\infty} \left(\eta^{(k)}\right)^2 < \infty$

  充分不必要条件
- 由统计学家 Herbert Robbins 和 Sutton Monro 在1951年提出
- A Stochastic Approximation Method

SGD

- $\eta^{(k)}$ 的选取

$\frac{1}{k}$

- 令 $S_n = \frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n}$， $T_n = \frac{1}{1^2} + \frac{1}{2^2} + \cdots + \frac{1}{n^2}$
- 则 $S_n - \log(n) \to \underline{0.577216}\ldots$，故 $S_n \to \infty$

画这两个图像 欧拉常数

- $T_n \to \frac{\pi^2}{6} < \infty$

常用模板

图片个数

```
obs_id = np.arange(n)  # [0, 1, ..., n-1]
# Run the whole data set `nepoch` times
for i in range(nepoch):    迭代次数
    # Shuffle observation IDs
    np.random.shuffle(obs_id)  打乱

    # Update on mini-batches
    for j in range(0, n, batch_size):  分成不同batch
        # Create mini-batch
        x_mini_batch = x[obs_id[j:(j + batch_size)]]
        # Compute loss
        loss = model(x_mini_batch)
        # Compute gradient and update parameters
        optimizer.zero_grad()
        loss.backward()            优化小部分
        optimizer.step()
```

改进 SGD

- SGD 虽然具有较好的理论性质
- 但在实际中会遇到各种挑战，如：
  - 确定合适的学习率 η，过大导致优化不收敛，过小耗费大量迭代次数
  - 对每个参数使用了相同的学习率 η

$n \times p + \not{R} \times p$

$n \times d \quad d \times p.$

一般化流程

1. 获取数据

导入，训练集，测试集

2. 封装

```
class MyModel(torch.nn.Module):
    def __init__(self)
        super __init__()
        self.conv1 = torch.nn.Conv2d(in_channel=1, out_channel=10,
                                     kernel_size=5, strides=2)
        self.pool1 = torch.nn.MaxPool2d(kernel_size=2)
        self.fc1 = torch.nn.Linear(in_features=720, out_features=10)

    def forward(self, x)
        x = self.conv1(x)
        x = torch.relu(x)
        x = self.pool1(x)
        x = torch.flatten(x, start_dim=1)
        x = self.fc1(x)
```

```
x = torch.nn.functional.softmax(x, dim=1)

nepoch = 30
batch_size = 200
lr = 0.001

np.random.seed(123)
torch.manual_seed(123)

model = MyModel
losses = []
opt = torch.optim.Adam(model.parameters(), lr=lr)
```

优化器

```
n = x.shape[0]
obs_id = np.arange(n)

for i in range(nepoch):
    np.random.shuffle(obs_id)
```

```
for j in range(0, n, batch_size):

    x_mini_batch = x[obs_id[j: (j+batch_size)]]
    y_mini_batch = y[obs_id[j: (j+batch_size)]]

    pred = model(x_mini_batch)
    lossfn = torch.nn.NLLoss()          损失函数
    loss = lossfn(torch.log(pred), y_mini_batch)

    opt.zero_grad()
    loss.backward()
    opt.step()
    losses.append(loss.item())

    if (j// batch_size) % 20 == 0:
        print
```