动手学深度学习

[toc]

前言

基础了解

深度学习与一门学科结合：

- 以特定的方式提出问题的动机
- 给定建模方法的数学
- 将模型拟合数据的优化算法
- 有效训练模型、克服数值计算并最大限度地利用现有硬件的工程方法

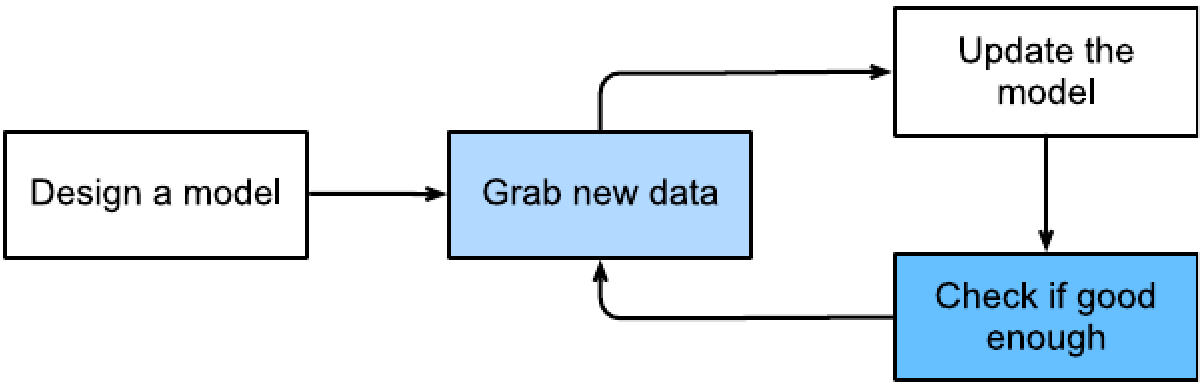
个人需要的能力：

- 批判性思维技能
- 解决问题所需要的数学知识
- 解决方案所需的软件工具

生活中的机器学习

基本词汇

- 参数：通过调整参数，来调节程序的行为。
- 模型族：通过调整参数而生成不同程序的集合(程序：输入输出的映射)。
- 学习算法：使用数据集选择参数的元程序。
- 训练流程：



> 在使用及其学习解决问题之前，必须精确定义问题，确定输入输出的性质，选择合适的模型族。
> 1．确定一个模型结构，并初始化参数
> 2．获取数据样本
> 3．调整参数
> 4．重复获取2、3两个步骤

关键组件

机器学习的关键组件包括四个部分：

- 我们可以学习的数据
- 如何转换数据的模型
- 一个目标函数，用来量化模型的有效性
- 调整模型参数以优化目标函数的算法

1. 数据：

```
每个数据集由样本构成，每个样本由特征构成。每个样本遵循iid的条件。

每个样本的特征数量都一致，有相同的长度，长度被成为数据的维度。

数据量越大，越有助于训练更强大的模型，从而减少对余弦设想的依赖。

Garbage in, garbage out。错误的数据会造成效果差。比如：数据分布不均匀（某种情况数据量少）；数据本身有偏见（数据本身就存在偏见，低质量）。
数据分为训练集和测试集。

数据集被分为三个部分：
    1. 训练集(用于拟合模型参数)
    2. 验证集(用于评估拟合效果)
    3. 测试集(通常测试集是不可以见的，验证集用于代替测试集)
```

2. 模型

```
机器学习从数据中学习，学习是指自主提高模型完成某些任务的效能。
深度学习注重强大的模型，将神经网络进行堆叠交织。
```

3. 目标函数

```
目标函数定义为模型的优劣程度，根据模型的参数（受决定于数据集）的具体情况，定义一个可以优化的目标函数，寻找到最佳的模型。


目标函数：
    预测数值： 平方误差。
    分类问题： 错误率
```

4. 优化算法

```
剃度下降方法：梯度的方向是函数在给定点上升最快的方向，那么梯度的反方向就是函数在给定点下降最快的方向
```

分类

监督学习

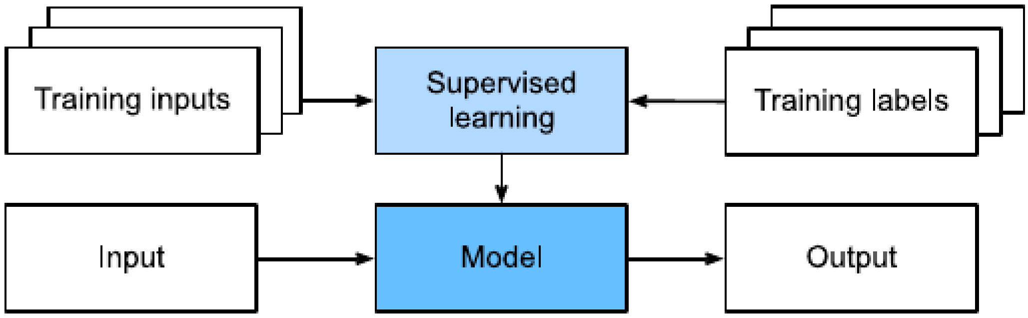


图1.3.1: 监督学习

- 回归
- 分类
- 标记
- 搜索
- 推荐系统
- 序列学习

无监督学习

- 聚类
- 主成分分析
- 因果关系和概率图模型问题
- 生成对抗性网络

与环境互动

```
主要应对以下几种问题：
    1. 环境对模型是否重要
    2. 环境是否有助于建模（语音识别）
    3. 环境是否想打败模型（垃圾邮件过滤）
    4. 环境是否变化（distribution shift）
```

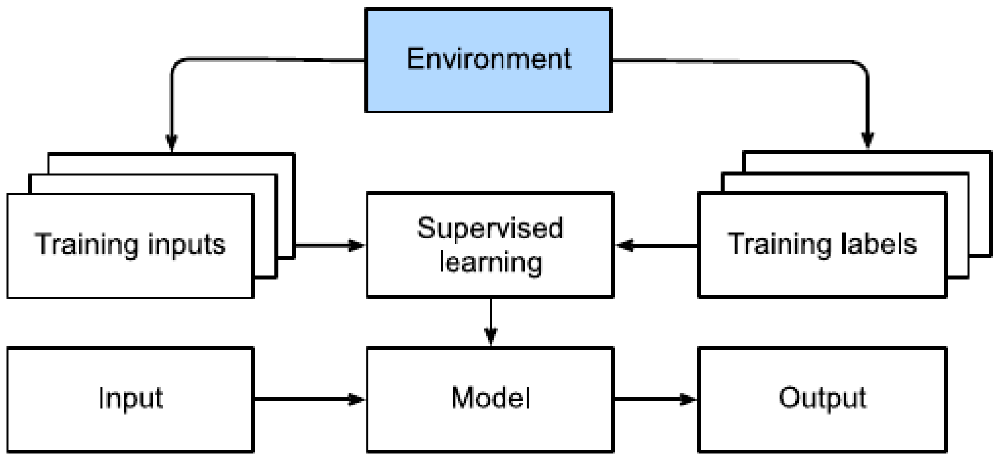


图1.3.6: 从环境中为监督学习收集数据。

强化学习

```
强化学习的过程是：agent持续接受environment的一些observation。

    agent接收到environment的observation
    agent接收，并选择一个action
    action通过某种机制传输到environment
    agent从environment获得reward
    循环第一步

目标是产生一个好的策略
```

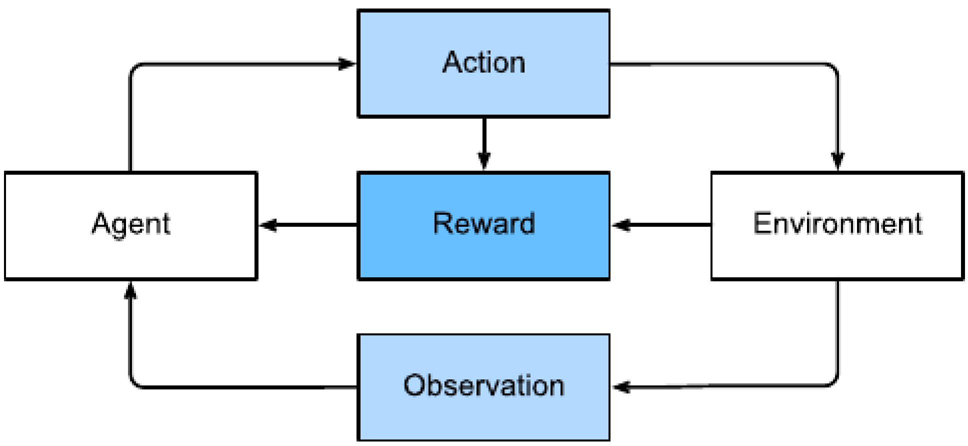


图1.3.7: 强化学习和环境之间的相互作用

近年的进展

dropout attention机制 多阶段设计 生成对抗网络 并行计算能力提升 可伸缩算法 深度学习框架

基础操作

基础运算符

按照元素进行计算(+ - * ** / ==)

基本运算

```
torch.cat
torch.dot
torch.mv
torch.mm
torch.exp
torch.cumsum
X.sum(dim) //sum默认会降维，可以通过keepdims属性设置
X.mean()
X.numel()


广播机制：维度不同的张量会扩展成相同类型，再进行基本操作。

切片：
    X[1:4:2, 1:-1:4]

节省内存:

    切片法，避免重新开辟空间(Memo[:] = <expression>)
    避免轻易对变量转换类型，可以通过调用变量本身的方法
```

数据预处理

```
1. 读取数据集 //pd.read_csv
2. NAN值处理 //data.fillna()
3. onehot编码 //data.getdummies()
4. 转换成张量 //torch.tensor(data.values)
```

微分

目标函数反映模型的效果。拟合的主要体现在两个方面：优化和泛化。都需要通过微分来寻找

- 自动求导(反向传播、Y.sum().backward()、Y.detach())
- 反向传播

```
神经网络沿着输出层到输入层的顺序计算梯度的方法，依据的是链式法则
```

- 正向传播

> 神经网络沿着输入层到输出层的顺序，依次进行计算并存储模型的中间变量。比如：输入层、隐藏层、输出层、`loss`。

经验分布

经验分布：依据抽样数据的分布概率Fn(X)，对原数据分布F(X)进行估计。

将抽样的数据按照升序进行排列，然后估计 xi < X 的概率，

Fn(X) = 0 (X < X1) = k/n (Xk <= X <= Xk+1) = 1 (X > Xk)

Fn(X)的示性函数：I(X) = k/n (Xk<= X <= Xk+1) = 0 (其他情况)

E(Fn(X)) = F(X) Var(Fn(X)) = (1/n)(1-F(X))(F(X))

熵

- 熵：H(X)
- 交叉熵：H(P,Q) = P(X)logQ(X)
- 条件熵：H(Y|X) = H(X,Y) - H(X)
- 相对熵: DKL = P(X)log(P(X)/Q(X)) (P相对Q的相对熵)

> 训练集的数据分布`Ptrain(X)`，学习后的模型分布概率为`Pmodel(X)`，衡量模型的有效性可以通过相对来实现。
> `Ptrain(X)`相对与`Pmodel(X)`的相对熵，即：KL散度
> `Dkl(Ptrain, Pmodel) = Ptrain(X)log(Ptrain(X)/Pmodel(X)) = Ptrain(X)logPtrain(X) - Ptrain(X)log(Pmodel(X)) = H(Xtrain) - H(Xtrain, Xmodel)`
> 显而易见 训练集的熵是一个常数，直接通过交叉熵评价模型的优劣。

- 信息增益：G = H(X) - H(X|A)

MLE

MLE的核心思想：找到一组参数θ，是的现有的Y以最大的概率出现。 Y = P(X|θ)，在已知一部分X、Y数据的基础上，求取。

$$L(\theta) = L(\theta; x_1, x_2, \cdots, x_n) = \prod_{k=1}^{n} p(x_k, \theta)$$

采用的MLE，需要Y|X关于θ的函数达到最大值，此刻θ即所求。

- MLE: argmin[-sigma(logp(Xi))]
- 基于经验分布的MLE: argmin[-sigma(p(Xi)logp(Xi))]

线性回归

训练线性回归

循环在训练集训练epoch次。 每个epoch的训练如下：

```
每次迭代过程中，通过迭代器读取一个batch的数据，输入模型，通过正向传播计算每个batch上的loss，然后通过反向传播计算参数的梯度，通过剃度下降算法更新模型的参数。

其中超参数包括：epoch、batch、learning rate、decay(衰减系数)。

在读取数据集的过程中，通过迭代器进行读取，可以降低内存的压力，需要训练的时候才读取到内存进行训练。


tips

with no_grad():
    在代码块中，不追踪梯度
torch.tensor(vector, require_grad = True)
    参数增加梯度。

param.grad.zero_()
    更新梯度后，要将梯度重置
nn.MSELoss()
    nn.MSELoss(reduction='none') //返回向量
    nn.MSELoss(reduction='mean') //返回标量，MSELoss的默认选择
    nn.MSELoss(reduction='sum') //返回标量
```

分类问题

onehot编码

softmax运算

- softmax的定义

```
Yhat = softmax(o) yj = exp(yj) / sigma(yi) //在计算loss时候使用
Y = argmax(yj) = argmax(o)
softmax 虽然是一个非线性函数，但是它仍然是由一个仿射变换，因此仍然是一个线性模型。
```

- softmax的导数

```
loss(y, yhat) = yj / (exp(oj)/sigma(exp(o)))  //只有yj是1，其他概率为0
loss关于o的导数(即：输出层的导数)为：softmax(o)j - yj //即真实值与预测值之间的差值
```

- 对数似然

$$P(\mathbf{Y} \mid \mathbf{X}) = \prod_{i=1}^{n} P(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}). \tag{3.4.6}$$

根据最大似然估计，我们最大化 $P(\mathbf{Y} \mid \mathbf{X})$，相当于最小化负对数似然：

$$-\log P(\mathbf{Y} \mid \mathbf{X}) = \sum_{i=1}^{n} -\log P(\mathbf{y}^{(i)} \mid \mathbf{x}^{(i)}) = \sum_{i=1}^{n} l(\mathbf{y}^{(i)}, \hat{\mathbf{y}}^{(i)}), \tag{3.4.7}$$

其中，对于任何标签 $\mathbf{y}$ 和模型预测 $\hat{\mathbf{y}}$，损失函数为：

$$l(\mathbf{y}, \hat{\mathbf{y}}) = -\sum_{j=1}^{q} y_j \log \hat{y}_j. \tag{3.4.8}$$

根据上式子可以理解交叉熵：

```
1. 求似然函数的最大值
2. KL散度最小
```