

<< [2023-03-13](#) | [2023-03-15](#) >>

*Don't look back. Something might be gaining on you.*  
— Satchel Paige

# 1. Introduction

## 1.1 课程安排

[动手学深度学习课程安排](#)

### 1.1.1 课程目标

[00:11 课程目标](#)

- 介绍深度学习经典和最新模型
  - LeNet, ResNet, LSTM, BERT, ...
- 机器学习基础
  - 损失函数、目标函数、过拟合、优化
- 实践
  - 使用Pytorch实现介绍的知识点
  - 在真实数据上体验算法效果

### 1.1.2 内容

#### 内容



- 深度学习基础 — 线性神经网络, 多层感知机
- 卷积神经网络 — LeNet, AlexNet, VGG, Inception, ResNet
- 循环神经网络 — RNN, GRU, LSTM, seq2seq
- 注意力机制 — Attention, Transformer
- 优化算法 — SGD, Momentum, Adam
- 高性能计算 — 并行, 多GPU, 分布式
- 计算机视觉 — 目标检测, 语义分割
- 自然语言处理 — 词嵌入, BERT



[动手学深度学习 v2 • https://courses.d2l.ai/zh-v2](#)

**深度学习基础：线性神经网络, 多层感知机**

**卷积神经网络：LeNet, AlexNet, VGG, Inception, ResNet**

**循环神经网络：RNN, GRU, LSTM, seq2seq**

注意力机制: Attention, Transformer

优化算法: SGD, Momentum, Adam

高性能计算: 并行, 多GPU, 分布式

计算机视觉: 目标检测, 语义分割

自然语言处理: 词嵌入, BERT

### 1.1.3 学到什么

04:48 可以学到什么

- What: 深度学习有哪些技术, 以及哪些技术可以帮你解决问题
- How: 如何实现 (产品 or paper) 和调参 (精度or速度)
- Why: 背后的原因 (直觉、数学)

### 1.1.4 基本要求

- **AI相关从业人员** (产品经理等) : 掌握What, 知道名词, 能干什么
- **数据科学家、工程师**: 掌握What、How, 手要快, 能出活
- **研究员、学生**: 掌握What、How、Why, 除了知道有什么和怎么做, 还要知道为什么, 思考背后的原因, 做出新的突破

### 1.1.5 课程资源

07:33

- 课程主页: <https://courses.d2l.ai/zh-v2/>
- 教材: <https://zh-v2.d2l.ai/>
- 课程论坛讨论: <https://discuss.d2l.ai/c/chinese-version/16>
- Pytorch论坛: <https://discuss.pytorch.org/>
- b站视频合集: [<https://space.bilibili.com/1567748478/channel/detail?sid=358497>]

## 1.2 深度学习介绍

Introduction to DL Link

### 1.2.1 概述

0:1 什么是深度学习



动手学深度学习 v2

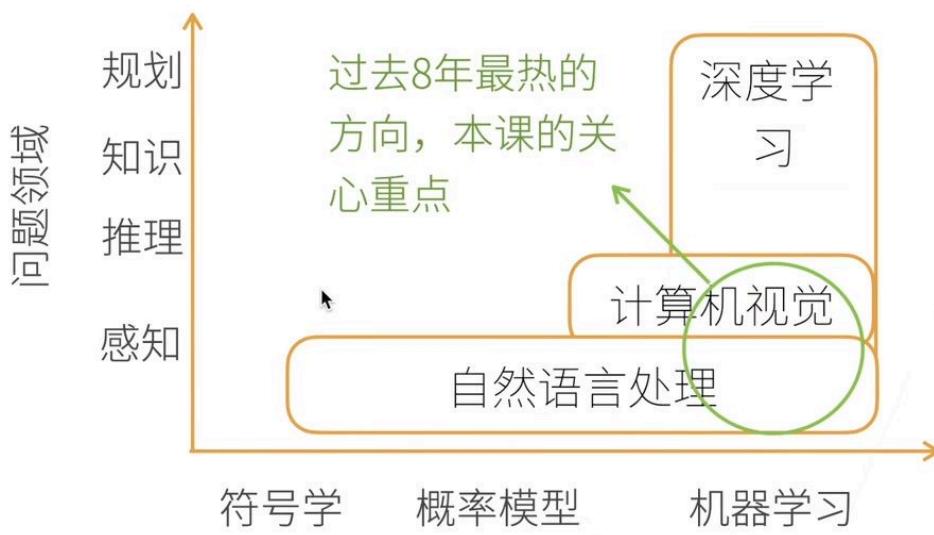
李沐 · AWS



[0:13 AI 地图](#)

首先画一个简单的人工智能地图：

## AI 地图



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

- x轴表示不同的模式or方法：最早的是符号学，接下来是概率模型，之后是机器学习
- y轴表示可以达到的层次：由底部向上依次是

感知：了解是什么，比如能够可以看到物体，如面前的一块屏幕

**推理**: 基于感知到的现象，想象或推测未来会发生什么

**知识**: 根据看到的数据或者现象，形成自己的知识

**规划**: 根据学习到的知识，做出长远的规划

## AI地图解读

- 问题领域的一个简单分类

- 自然语言处理**:

- 停留在比较简单的**感知**层面，比如自然语言处理用的比较多的机器翻译，给一句中文翻译成英文，很多时候是人的潜意识里面大脑感知的一个问题。一般来说，人可以几秒钟内反应过来的东西，属于感知范围。
- 自然语言处理最早使用的方法是**符号学**，由于语言具有符号性；之后一段时间比较流行的有**概率模型**，以及现在也用的比较多的**机器学习**。

- 计算机视觉**:

- 在简单的感知层次之上，可以对图片做一些**推理**。
- 图片里都是一些像素，很难用符号学解释，所以一般采用**概率模型**和**机器学习**。

- 深度学习**

- 机器学习的一种，更深层的神经网络。
- 可以做计算机视觉，自然语言处理，强化学习等。

- 过去八年最热的方向，也是本课程关心的重点：

- 深度学习+计算机视觉 / 自然语言处理**

---

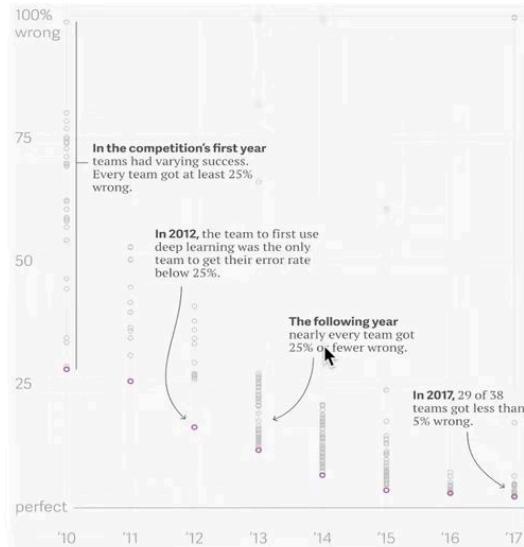
### 1.2.2 深度学习的应用

3:59 图片分类

深度学习最早是在图片分类上有比较大的突破，[ImageNet](#) 是一个比较大的图片分类数据集，



# 图片分类



<https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world/>

动手学深度学习 v2 • <https://courses.d2l.ai/zh-v2>

x轴：年份 y轴：错误率 圆点：表示某年份某研究工作/paper的错误率 [IMAGENET 数据来源](#)

在2010年时，错误率比较高，最好的工作错误率也在26%、27%左右；

在2012年，有团队首次使用深度学习将错误率降到25%以下；

在接下来几年中，使用深度学习可以将误差降到很低。

2017年基本所有的团队可以将错误率降到5%以下，基本可以达到人类识别图片的精度。

4:43 物体检测和分割

## 物体检测和分割



[https://github.com/matterport/Mask\\_RCNN](https://github.com/matterport/Mask_RCNN)  
动手学深度学习 v2 • <https://courses.d2l.ai/zh-v2>

当你不仅仅想知道图片里有什么内容，还想知道物体是什么，在什么位置，这就是**物体检测**。**物体分割**是指每一个像素属于什么，属于飞机还是属于人(如下图)，这是图像领域更深层次的一个应用。

### 5:15 样式迁移

## 样式迁移



<https://github.com/zhanghang1989/MXNet-Gluon-Style-Transfer>  
动手学深度学习 v2 • <https://courses.d2l.ai/zh-v2>

原图片+想要迁移的风格=风格迁移后的图片，加了一个可以根据输入改变图片风格的滤镜。

### 6:1 人脸合成

下图中所有的人脸都是假的，由机器合成的图片：

## 人脸合成



动手学深度学习 v2 • <https://courses.d2l.ai/zh-v2>

Karras et al, ICLR 2018

## 文字生成图片

TEXT PROMPT  
an illustration of a baby daikon radish in a tutu walking a dog

AI-GENERATED IMAGES

TEXT PROMPT  
an armchair in the shape of an avocado [...]

AI-GENERATED IMAGES

<https://openai.com/blog/dall-e/>

动手学深度学习 v2 • <https://courses.d2l.ai/zh-v2>

1. 描述：一个胡萝卜宝宝遛狗的图片。
2. 描述：一个牛油果形状的靠背椅。

## 7:2 文字生成

## 文字生成

Q: How to run an Effective Board Meeting

Step 1: Spend time building an effective board

Startup boards typically include one or more founders representing the common shareholders of the financing round), and one or more independent board directors. Importantly, these independent directors have more reputational skin in the game, actual legal liability, critical feedback to the CEO.

Most founders I've met don't appreciate the significant impact that high quality board members can have on high quality, however that's not always entirely in your control, since it's dependent on how hot you are picking you. The quality of your independent board members is far more in your control. It can take a board member but the effort is absolutely worth it.

When you start recruiting independent board members can vary from startup to startup, I typically can delay it if your current board is decent enough and covers the areas of knowledge critical to you.

A while back, one of the startups we'd invested in had seen some early success with large enterprise clients but encountered some really tough challenges. Mostly because the DNA of the founders was primarily spinning their wheels and rotating through heads of sales, the founders decided to try and accelerate an expert in enterprise sales. The effects were immediate. Within weeks they were making headway on deals and team delivering results. Over the following few years, this independent board member was critical to the company's success.

Founders also tend to not realize they can stretch even higher in recruiting board members than in-house talent. It's an incredibly flattering, a relatively low time commitment, and the topic can be broached with very little risk. You can shoot for the stars. Typically compensation for an external board member like this should be the current stage, with immediate monthly vesting; no cliff.

Here's #gpt3 writing some SQL for me.

```
Text: Select the "Students" from the "School" table joined with the "Class" table.
Code: SELECT * FROM Students
INNER JOIN Class
ON Students.ID = Class.StudentID
```

示例1：

问题输入：如何举行一个有效的董事会议

机器输出：生成篇章回答

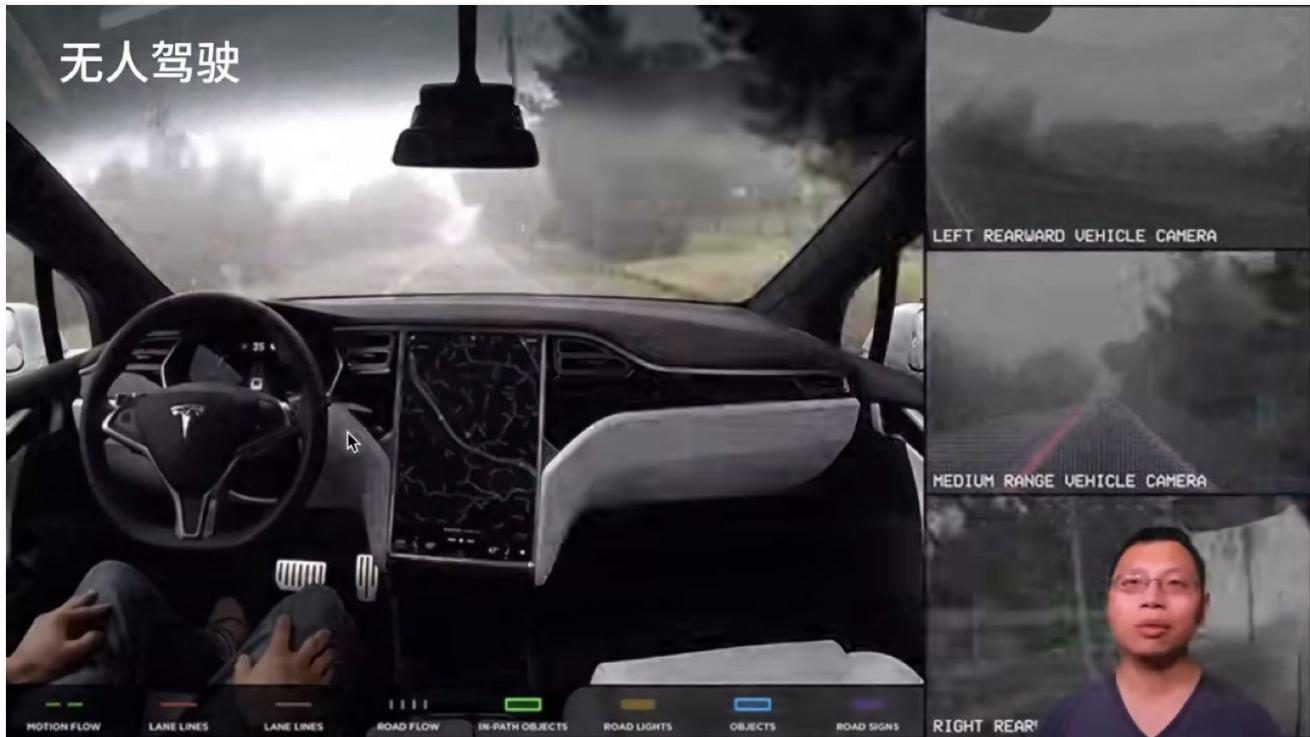
示例2：

输入：将Students从School这个table中选出来

输出：用于查询的SQL语言

## 7:56 无人驾驶

识别车、道路以及各种障碍物等，并规划路线。



## 8:29 案例研究-广告点击

用户输入想要搜索的广告内容，如：baby toy

网站呈现最具有效益的广告(用户更可能点击，且给网站带来更高经济效益)



# 案例研究 – 广告点击

Amazon Prime  
Deliver to Mu  
Palo Alto 94303

All - baby toy

Hello, Mu Account & Lists Orders Prime Cart

Fresh Whole Foods Today's Deals Help Best Sellers

Shop deals for the New Year

Sort by: Featured

1-48 of over 30,000 results for "baby toy"

**Amazon Prime**  
 prime  
 prime | FREE Same-Day Delivery on qualifying orders over \$35  
**Delivery Day**  
 Get It Today  
 Get It by Tomorrow  
**Pantry**  
 prime pantry  
**Prime Wardrobe**  
 prime wardrobe  
**Local Stores**  
 Amazon Fresh  
 Whole Foods Market  
**Department**  
Toys & Games  
Baby & Toddler Toys  
Baby Musical Toys  
Stuffed Animals & Plush Toys  
Baby Rings & Plush Rings  
Baby Activity Play Centers  
See more  
Baby Products  
Baby Teether Toys  
Baby Bibs  
Baby Toothbrushes  
Baby Health Care Products  
Education  
Children's Vocabulary & Spelling Books  
Children's Basic Concepts Books

**SPONSORED BY SIMILAC**  
#1 brand chosen by parents  
Shop now!

Similac Pro-Advance Non-GMO Infant Formula with Iron, with 2'-FL HMO, Baby Formula, 35 oz Box, 1 Count  
4.5 stars, 428 reviews  
\$17.99 prime

Similac Pro-Total Comfort Infant Formula OPTI-GRO, Non-GMO, Baby Formula, 35 oz Box, 1 Count  
4.5 stars, 188 reviews  
\$16.99 prime

Price and other details may vary based on size and color

**Sponsored** Baby Einstein Magic Touch Mini Piano Wooden Musical Toy, 5 Months+  
★★★★★ ~ 658  
\$19.99

**Sponsored** Splashin' Kids Infant Toys Beginner Crawler Game Ball Drop Maze Tummy Time Activity Center Early Development Jum...  
★★★★★ ~ 81

**Sponsored** iPlay, Learn 5 Dinosaur Baby Rattles, Teether, Shaker, Grab and Spin Rattle, Musical Toy Set, Early Educational Toys, Unique Gifts...  
★★★★★ ~ 84

**4 stars and above: Sponsored** Page 1 of 5

TOMY Bright 'n' Bold Toddler Toys For 1-2 Year Old Boy And Girl Gift Wooden Race Track Toy  
★★★★★ ~ 815  
\$21.99 prime

*(The rest of the page continues with more sponsored products and a sidebar for related items.)*


## 步骤：

- 触发：用户输入关键词，机器先找到一些相关的广告
- 点击率预估：利用机器学习的模型预测用户对广告的点击率
- 排序：利用点击率x竞价的结果进行排序呈现广告，排名高的在前面呈现

## 模型的训练与预测：

上述步骤的第二步中涉及到模型预测用户的点击率，具体过程如下：



# 预测与训练



动手学深度学习 v2 • <https://courses.d2l.ai/zh-v2>



- **模型预测**

数据 (待预测广告)  $\rightarrow$  特征提取  $\rightarrow$  模型  $\rightarrow$  点击率预测

- **模型训练**

训练数据 (过去广告展现和用户点击)  $\rightarrow$  特征(X)和用户点击(Y)  $\rightarrow$  喂给模型训练



# 完整的故事

领域专家



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>



- **领域专家**: 对特定的应用有比较深的了解, 根据展现情况以及用户点击分析用户的行为, 期望模型对应用做一些拟合, 符合真实数据和分析情况。
- **数据科学家**: 利用数据训练模型, 训练后模型投入使用, 进行预测呈现。
- **AI专家**: 应用规模扩大, 用户数量增多, 模型更加复杂, 需要进一步提升精度和性能。

## 1.2.3 总结

- 通过AI地图, 课程从纵向和横向两个维度解读了深度学习在重要问题领域的概况。
- 介绍了深度学习在CV和NLP方面的一些应用
- 简单分析并研究了深度学习实例——广告点击。

## 1.3 安装

[课程链接](#)

### 1.3.1 安装python

[02:04 安装python](#)

首先前提是安装python, 这里推荐安装python3.8 输入命令 `sudo apt install python3.8` 即可

### 1.3.2 安装Miniconda/Anaconda

- 然后第二步, 安装 Miniconda (如果已经安装conda或者Miniconda, 则可以跳过该步骤)。

#### 2.1 安装Miniconda

- 安装Miniconda的好处是可以创建很多虚拟环境，并且不同环境之间互相不会有依赖关系，对日后的项目有帮助，如果只想在本地安装的话，不装Miniconda只使用pip即可，第二步可以跳过。
- 如果是Windows系统，输入命令 [wget https://repo.anaconda.com/miniconda/Miniconda3-py38\\_4.10.3-Windows-x86\\_64.exe](https://repo.anaconda.com/miniconda/Miniconda3-py38_4.10.3-Windows-x86_64.exe)
- 如果是macOS，输入命令 [wget https://repo.anaconda.com/miniconda/Miniconda3-py38\\_4.10.3-MacOSX-x86\\_64.sh](https://repo.anaconda.com/miniconda/Miniconda3-py38_4.10.3-MacOSX-x86_64.sh) 之后要输入命令 `sh Miniconda3-py38_4.10.3-MacOSX-x86_64.sh -b`
- 如果是Linux系统，输入命令 [wget https://repo.anaconda.com/miniconda/Miniconda3-py38\\_4.10.3-Linux-x86\\_64.sh](https://repo.anaconda.com/miniconda/Miniconda3-py38_4.10.3-Linux-x86_64.sh) 之后输入命令 `sh Miniconda3-py38_4.10.3-Linux-x86_64.sh -b`
- 以上都是基于python3.8版本，对于其他版本，可以访问 <https://docs.conda.io/en/latest/miniconda.html>，下载对应版本即可。

## 2.2 Miniconda环境操作

- 对于第一次安装Miniconda的，要初始化终端shell，输入命令 `~/miniconda3/bin/conda init`
- 这样我们就可以使用 `conda create --name d2l python=3.8 -y` 来创建一个名为xxx的环境，这里命名为d2l
- 打开xxx环境命令：`conda activate xxx`；关闭命令：`conda deactivate xxx`。对于基础conda环境不用添加名

### 1.3.3 安装Pytorch, d2l, jupyter包

- 第三步，安装深度学习框架和d2l软件包

在安装深度学习框架之前，请先检查你的计算机上是否有可用的GPU（为笔记本电脑上显示器提供输出的GPU不算）。例如，你可以查看计算机是否装有NVIDIA GPU并已安装[CUDA](#)。如果你的机器没有任何GPU，没有必要担心，因为你的CPU在前几章完全够用。但是，如果你想流畅地学习全部章节，请提早获取GPU并且安装深度学习框架的GPU版本。

- 你可以按如下方式安装PyTorch的CPU或GPU版本：

```
pip install torch==1.8.1
pip install torchvision==0.9.1
```

- 也可以访问官网 <https://pytorch.org/get-started/locally/> 选择适合自己电脑pytorch版本下载！
- 本课程的jupyter notebook代码详见 <https://zh-v2.d2l.ai/d2l-zh.zip>
- 下载jupyter notebook：输入命令 `pip install jupyter notebook`（若pip失灵可以尝试pip3），输入密令 `jupyter notebook` 即可打开。

### 1.3.4 总结

- 本节主要介绍[安装Miniconda](#)、[CPU环境下的Pytorch](#)和其它课程所需[软件包](#)(d2l, jupyter)。对于前面几节来说，CPU已经够用了。
  - 如果您[已经安装](#)了Miniconda/Anaconda, Pytorch框架和jupyter记事本，您只需再安装[d2l包](#)，就可以跳过本节视频了[开启深度学习之旅](#)了；如果希望后续章节在[GPU下跑深度学习](#)，可以[新建环境安装CUDA版本的Pytorch](#)。
  - 如果需要在Windows下[安装CUDA和Pytorch](#)(cuda版本)，用[本地GPU跑深度学习](#)，可以参考李沐老师[Windows下安装CUDA和Pytorch跑深度学习](#)，如果网慢总失败的同学可以参考[cuda11.0如何安装pytorch？ - Glenn1Q84的回答 - 知乎](#)。当然，如果不方便在本地进行配置(如无GPU, GPU显存过低等)，也可以选择[Colab](#)(需要科学上网)，或其它[云服务器](#)GPU跑深度学习。
- 如果pip安装比较慢，可以用镜像源安装：

```
pip install torch torchvision -i http://mirrors.aliyun.com/pypi/simple/ --trusted-host language=bash
mirrors.aliyun.com
```

- 如果安装时经常报错，可以参考课程评论区部分。

## 1.4 数据操作与数据预处理

### 1.4.1 数据处理

数据处理

00:19 N维数组样例

为了能够完成各种数据操作，我们需要某种方法来存储和操作数据。通常，我们需要做两件重要的事：

1. 获取数据；
2. 将数据读入计算机后对其进行处理。

如果没有某种方法来存储数据，那么获取数据是没有意义的。

首先，我们介绍 n 维数组，也称为张量 (tensor)。PyTorch 的张量类与 Numpy 的 ndarray 类似。但在深度学习框架中应用 PyTorch 的张量类，又比 Numpy 的 ndarray 多一些重要功能：

1. tensor 可以在很好地支持 GPU 加速计算，而 NumPy 仅支持 CPU 计算；
  2. tensor 支持自动微分。
- 低维数组

### N维数组样例

The diagram illustrates N-dimensional arrays (tensors) from scalar to matrix. It shows three categories: 0-d (标量), 1-d (向量), and 2-d (矩阵).

- 0-d (标量):** Represented by a single blue square. Below it is the value **1.0**. Below that is the text **一个类别**.
- 1-d (向量):** Represented by a horizontal blue bar. Below it is the list **[1.0, 2.7, 3.4]**. Below that is the text **一个特征向量**.
- 2-d (矩阵):** Represented by a 2x3 grid of blue squares. Below it is the list **[[1.0, 2.7, 3.4], [5.0, 0.2, 4.6], [4.3, 8.5, 0.2]]**. Below that is the text **一个样本—特征矩阵**.

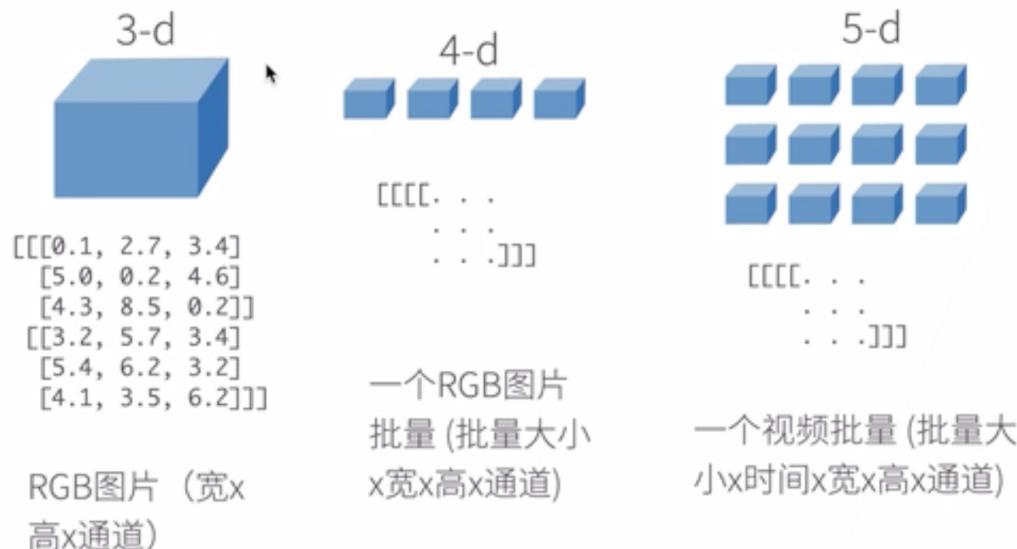


动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

- 高维数组



## N维数组样例 (续)



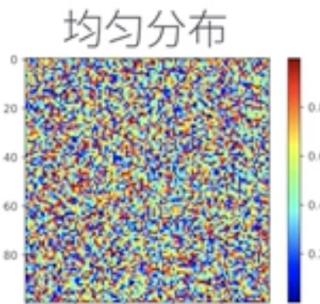
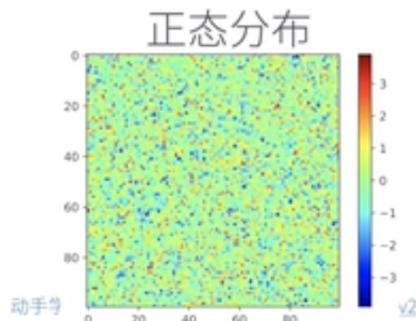
动手学深度学习 v2 • <https://courses.d2l.ai/zh-v2>



[02:36 创建数组](#)

## 创建数组

- 创建数组需要
  - 形状：例如  $3 \times 4$  矩阵
  - 每个元素的数据类型：例如32位浮点数
  - 每个元素的值，例如全是0，或者随机数



[03:06 访问元素](#)



# 访问元素

一个元素: [1, 2]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

一行: [1, :]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

一列: [1, :]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

子区域: [1:3, 1:]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

子区域: [::3, ::2]

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
3	13	14	15	16

动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>



## 基本操作

### 数据操作

张量表示由一些数值组成的数组，这个数组可能有多个维度。

- 一个轴：对应数学上的向量（vector）；
- 两个轴：对应数学上的矩阵（matrix）；
- 两个轴以上：没有特殊的数学名称。

1. 可使用 `arange` 创建行向量 `x`，默认创建为浮点数，张量中的每个值都称为张量的元素（element）。

**Note**: 除非额外指定，新的张量默认将存储在内存中，并采用基于CPU的计算。

```
>>> x = torch.arange(12)
      tensor([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
```

language=python

2. 访问张量的形状

```
>>> x.shape # 访问张量的形状
      torch.Size([12])
```

language=python

3. 张量的大小

```
>>> x.size() # 元素总数
      torch.Size([12])
```

4. 元素个数。在处理更高维度的张量时，可以用 `numel` 获取张量中元素的个数。

```
>>> x.numel() # number of elements
      12
```

language=python

5. 改变张量形状。

```
>>> X = x.reshape(3, 4)
      tensor([[ 0,  1,  2,  3],
              [ 4,  5,  6,  7],
              [ 8,  9, 10, 11]])
```

language=python

```
>>> x.reshape(-1, 4)
      tensor([[ 0,  1,  2,  3],
              [ 4,  5,  6,  7],
              [ 8,  9, 10, 11]])
```

## 6. 常量矩阵

```
>>> torch.zeros((2, 3, 4))
      tensor([[[0.,  0.,  0.,  0.],
                [0.,  0.,  0.,  0.],
                [0.,  0.,  0.,  0.]],
                [[0.,  0.,  0.,  0.],
                [0.,  0.,  0.,  0.],
                [0.,  0.,  0.,  0.]]])
```

```
>>> torch.ones((2, 3, 4))
      tensor([[[1.,  1.,  1.,  1.],
                [1.,  1.,  1.,  1.],
                [1.,  1.,  1.,  1.]],
                [[1.,  1.,  1.,  1.],
                [1.,  1.,  1.,  1.],
                [1.,  1.,  1.,  1.]]])
```

language=python

## 7. 符合某种分布的矩阵。通过从某个特定的概率分布中随机采样来得到张量中每个元素的值。例如，当构造数组来作为神经网络中的参数时，通常会随机初始化参数的值，使得其服从均值为0、标准差为1的标准正态分布。

```
>>> torch.randn(3, 4)
      tensor([[ 0.1364,  0.3546, -0.9091, -1.8926],
              [ 0.5786, -0.9019, -0.1305, -0.1899],
              [ 0.5696,  1.1626, -0.5987,  0.4085]])
```

language=python

## 8. 基于列表。

```
>>> torch.tensor([[2, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])
      tensor([[2, 1, 4, 3],
              [1, 2, 3, 4],
              [4, 3, 2, 1]])
```

language=python

## 1.4.2 简单运算

我们想在这些数据上执行数学运算，其中最简单且最有用的操作是按元素（elementwise）运算。我们通过将标量函数升级为按元素向量运算来生成向量值 $F: \mathbb{R}^d, \mathbb{R}^d \rightarrow \mathbb{R}^d$ 。

- 基本运算

```
>>> x = torch.tensor([1.0, 2, 4, 8])
>>> y = torch.tensor([2, 2, 2, 2])
>>> x + y, x - y, x * y, x / y, x ** y
      (tensor([ 3.,  4.,  6., 10.]),
       tensor([-1.,  0.,  2.,  6.]),
       tensor([ 2.,  4.,  8., 16.]),
```

language=python

```
tensor([0.5000, 1.0000, 2.0000, 4.0000],  
      tensor([ 1., 4., 16., 64.]))
```

- 指数

```
>>> torch.exp(x)  
tensor([2.7183e+00, 7.3891e+00, 5.4598e+01, 2.9810e+03])
```

language=python

- 连接。

```
>>> X = torch.arange(12, dtype=torch.float32).reshape(3,4)  
>>> Y = torch.tensor([[2.0, 1, 4, 3], [1, 2, 3, 4], [4, 3, 2, 1]])  
>>> torch.cat((X, Y), dim=0), torch.cat((X, Y), dim=1)  
(tensor([[ 0., 1., 2., 3.],  
        [ 4., 5., 6., 7.],  
        [ 8., 9., 10., 11.],  
        [ 2., 1., 4., 3.],  
        [ 1., 2., 3., 4.],  
        [ 4., 3., 2., 1.]]),  
 tensor([[ 0., 1., 2., 3., 2., 1., 4., 3.],  
        [ 4., 5., 6., 7., 1., 2., 3., 4.],  
        [ 8., 9., 10., 11., 4., 3., 2., 1.]]))
```

language=python

- 三维张量连接。由上述例子可见，当需要按轴-x连结两个张量时，我们就在第x+1层括号内将两张量中的元素相组合。类似地，我们将两个三维张量相连结。

```
>>> X = torch.arange(12, dtype=torch.float32).reshape(3, 2, 2)  
>>> Y = torch.tensor([[[2.0, 1], [4, 3]], [[1, 2], [3, 4]], [[4, 3], [2, 1]]])  
>>> torch.cat((X, Y), dim=0), torch.cat((X, Y), dim=1), torch.cat((X, Y), dim=2)  
(tensor([[[ 0., 1.],  
        [ 2., 3.]],  
  
        [[[ 4., 5.],  
          [ 6., 7.]],  
  
        [[[ 8., 9.],  
          [10., 11.]],  
  
        [[[ 2., 1.],  
          [ 4., 3.]],  
  
        [[[ 1., 2.],  
          [ 3., 4.]],  
  
        [[[ 4., 3.],  
          [ 2., 1.]]]]),  
 tensor([[[ 0., 1.],  
        [ 2., 3.],  
        [ 2., 1.],  
        [ 4., 3.]],  
  
        [[[ 4., 5.],  
          [ 6., 7.],  
          [ 1., 2.],  
          [ 3., 4.]],  
  
        [[[ 8., 9.],  
          [10., 11.],  
          [ 4., 3.],  
          [ 2., 1.]]]]),  
 tensor([[[[ 0., 1., 2., 1.],  
          [ 4., 3., 2., 1.],  
          [ 8., 9., 10., 11.],  
          [ 4., 3., 2., 1.]]]]))
```

language=python

```
[ 2.,  3.,  4.,  3.],
 [[ 4.,  5.,  1.,  2.],
 [ 6.,  7.,  3.,  4.]],
 [[ 8.,  9.,  4.,  3.],
 [10., 11.,  2.,  1.]])
```

- 逻辑运算。

```
>>> X == Y
tensor([[False,  True, False,  True],
 [False, False, False, False],
 [False, False, False, False]])
```

language=python

- 求和。

```
>>> X.sum()
tensor(66.)
```

language=python

## 广播机制

### 07:31 广播机制

在上面的部分中，我们看到了如何在相同形状的两个张量上执行按元素操作。在某些情况下，即使形状不同，我们仍然可以通过调用广播机制（broadcasting mechanism）来执行按元素操作。

这种机制的工作方式如下：首先，通过适当复制元素来扩展一个或两个数组，以便在转换之后，两个张量具有相同的形状。其次，对生成的数组执行按元素操作。在大多数情况下，我们将沿着数组中长度为1的轴进行广播，如下例子：

```
>>> a = torch.arange(3).reshape(3, 1)
>>> b = torch.arange(2).reshape(1, 2)
>>> a, b, a + b
(tensor([[0],
 [1],
 [2]]),
 tensor([[0, 1]]))
(tensor([[0, 1],
 [1, 2],
 [2, 3]]))
```

language=python

**Note**: 广播机制只能扩展维度，而不能凭空增加张量的维度，例如在计算沿某个轴的均值时，若张量维度不同，则会报错：

```
>>> C = torch.arange(24, dtype=torch.float32).reshape(2, 3, 4)
>>> C / C.sum(axis=1)
RuntimeError: The size of tensor a (3) must match the size of tensor b (2) at non-singleton dimension 1
```

language=python

此时我们需要将`keepdim`设为True，才能正确利用广播机制扩展`C.sum(axis=1)`的维度：

```
>>> C.sum(axis=1).shape, C.sum(axis=1, keepdim=True).shape
torch.Size([2, 4]), torch.Size([2, 1, 4])

>>> C / C.sum(axis=1, keepdim=True)
tensor([[[0.0000, 0.0667, 0.1111, 0.1429],
 [0.3333, 0.3333, 0.3333, 0.3333],
 [0.6667, 0.6000, 0.5556, 0.5238]],

 [[0.2500, 0.2549, 0.2593, 0.2632],
```

language=python

```
[0.3333, 0.3333, 0.3333, 0.3333],  
[0.4167, 0.4118, 0.4074, 0.4035]]])
```

## 索引和切片

### 09:34 元素访问

- 通过索引访问。

```
>>> X[-1], X[1:3]  
(tensor([ 8.,  9., 10., 11.]),  
 tensor([[ 4.,  5.,  6.,  7.],  
        [ 8.,  9., 10., 11.])))
```

language=python

- 间隔访问。可以用`[:2]`每间隔一个元素选择一个元素，可以用`[:,::3]`每间隔两个元素选择一个元素：

```
>>> X[:,::2, ::3]  
tensor([[ 0.,  3.],  
       [ 8., 11.]])
```

language=python

- 写入数据。除读取外，我们还可以通过指定索引来将元素写入矩阵。

```
>>> X[1, 2] = 9  
tensor([[ 0.,  1.,  2.,  3.],  
       [ 4.,  5.,  9.,  7.],  
       [ 8.,  9., 10., 11.]])
```

language=python

- 多元素赋值。

```
>>> X[0:2, :] = 12  
tensor([[12., 12., 12., 12.],  
       [12., 12., 12., 12.],  
       [ 8.,  9., 10., 11.]])
```

language=python

## 节约内存

### 10:50 内存管理

如果在后续计算中没有重复使用`X`，我们也可以使用`X[:] = X + Y`或`X += Y`来减少操作的内存开销。

```
>>> before = id(X) # 内存地址  
>>> X += Y  
>>> id(X) == before  
True
```

language=python

## 转换为 NumPy 对象

将深度学习框架定义的张量转换为NumPy张量（`ndarray`）。torch张量和numpy数组将共享它们的底层内存，就地操作更改一个张量也会同时更改另一个张量。

```
>>> A = X.numpy()  
>>> B = torch.tensor(A)  
>>> type(A), type(B)  
(numpy.ndarray, torch.Tensor)
```

language=python

要(将大小为1的张量转换为Python标量)，我们可以调用`item`函数或Python的内置函数。

```
>>> a = torch.tensor([3.5])  
>>> a, a.item(), float(a), int(a)
```

language=python

```
tensor([3.5000], 3.5, 3.5, 3)
```

## 1.4.2 数据预处理

### 00:08 数据预处理

为了能用深度学习来解决现实世界的问题，我们经常从预处理原始数据开始，而不是从那些准备好的张量格式数据开始。在Python中常用的数据分析工具中，我们通常使用pandas软件包。像庞大的Python生态系统中的许多其他扩展包一样，pandas可以与张量兼容。本节我们将简要介绍使用pandas预处理原始数据，并将原始数据转换为张量格式的步骤。

### 读取数据集

举一个例子，我们首先(创建一个人工数据集，并存储在CSV（逗号分隔值）文件)`../data/house_tiny.csv`中。以其他格式存储的数据也可以通过类似的方式进行处理。下面我们将数据集按行写入CSV文件中。

```
>>> import os  
>>> os.makedirs(os.path.join('..', 'data'), exist_ok=True)  
>>> data_file = os.path.join('..', 'data', 'house_tiny.csv')  
>>> with open(data_file, 'w') as f:  
>>>     f.write('NumRooms,Alley,Price\n') # 列名  
>>>     f.write('NA,Pave,127500\n') # 每行表示一个数据样本  
>>>     f.write('2,NA,106000\n')  
>>>     f.write('4,NA,178100\n')  
>>>     f.write('NA,NA,140000\n')
```

language=python

要从创建的CSV文件中加载原始数据集，我们导入pandas包并调用`read_csv`函数。该数据集有四行三列。其中每行描述了房间数量 (“NumRooms”)、巷子类型 (“Alley”) 和房屋价格 (“Price”)。

```
>>> import pandas as pd  
>>> data = pd.read_csv(data_file)  
>>> print(data)
```

language=python

	NumRooms	Alley	Price
0	NaN	Pave	127500
1	2.0	NaN	106000
2	4.0	NaN	178100
3	NaN	NaN	140000

### 处理缺失值

“NaN”项代表缺失值。<sup>\*</sup>为了处理缺失的数据，典型的方法包括插值法和删除法\*.

- 插值法：用一个替代值弥补缺失值；
- 删除法：直接忽略缺失值。

```
>>> inputs, outputs = data.iloc[:, 0:2], data.iloc[:, 2]  
>>> inputs = inputs.fillna(inputs.mean())  
>>> print(inputs)
```

language=python

	NumRooms	Alley
0	3.0	Pave
1	2.0	NaN
2	4.0	NaN

	NumRooms	Alley
3	3.0	NaN
	NumRooms	Alley

利用删除法，我们删除缺失元素最多的一个样本。首先，`data.isnull()`矩阵统计每个元素是否缺失，之后在轴 1 的方向上（对列进行求和）将`data.isnull()`元素求和，得到每个样本缺失元素个数，取得缺失元素个数最大的样本的序号，并将其删除。

```
>>> nan_numer = data.isnull().sum(axis=1)                                     language=python
>>> nan_numer
    0      1
    1      1
    2      1
    3      2
   dtype: int64

>>> nan_max_id = nan_numer.idxmax()
>>> data_delete = data.drop([nan_max_id], axis=0) # 删除第 nan_max_id 行
```

	NumRooms	Alley	Price
0	NaN	Pave	127500
1	2.0	NaN	106000
2	4.0	NaN	178100

一般情况下，可以利用`dropna`删除数据：

```
dropna( axis=0, how='any', thresh=None, subset=None, inplace=False)
```

language=python

#### Note:

- `Axis`: 哪个维度
- `How`: 如何删除。`any` 表示有 nan 即删除，`all` 表示全为nan删除，`Thresh` 有多少个nan 删除，`Subset` 在哪些列中查找 nan
- `Inplace`: 是否原地修改。

## 离散值处理

对于类别值或离散值，可以将“NaN”视为一个类别。

```
>>> inputs = pd.get_dummies(inputs, dummy_na=True)
```

language=python

	NumRooms	Alley_Pave	Alley_nan
0	3.0	1	0
1	2.0	0	1
2	4.0	0	1
3	3.0	0	1

## 转换为张量格式

现在所有条目都是数值类型，可以转换为张量格式。

```
>>> import torch
>>> X, y = torch.tensor(inputs values), torch.tensor(outputs values)
     tensor([[3.,  1.,  0.],
            [2.,  0.,  1.],
            [4.,  0.,  1.],
            [3.,  0.,  1.]], dtype=torch float64),
     tensor([127500, 106000, 178100, 140000]))
```

language=python

### 1.4.3 Q&A

07:26 Q&A

**Q1: reshape和view的区别?**

*View为浅拷贝，只能作用于连续型张量；Contiguous函数将张量做深拷贝并转为连续型；Reshape在张量连续时和view相同，不连续时等价于先contiguous再view。*

**Q2: 数组计算吃力怎么办？**

*学习numpy的知识。*

**Q3: 如何快速区分维度？**

*利用a.shape或a.dim()。*

**Q4: Tensor和Array有什么区别？**

*Tensor是数学上定义的张量，Array是计算机概念数组，但在深度学习中有时将Tensor视为多维数组。*

**Q5: 新分配了y的内存，那么之前y对应的内存会自动释放吗？**

*Python会在不需要时自动释放内存。*

## 1.5 线性代数

线性代数

### 1.5.1 线性代数基础知识

这部分主要是由标量过渡到向量，再从向量拓展到矩阵操作，重点在于理解矩阵层面上的操作（都是大学线代课的内容，熟悉的可以自动忽略）

#### 1. 标量

- 简单操作

$$c = a + b$$

$$c = a \cdot b$$

$$c = \sin a$$

- 长度

$$|a| = \begin{cases} a & \text{if } a > 0 \\ -a & \text{otherwise} \end{cases}$$

$$|a + b| \leq |a| + |b|$$

$$|a \cdot b| = |a| \cdot |b|$$

## 2. 向量

## 向量

- 简单操作

$$c = a + b \quad \text{where } c_i = a_i + b_i$$

$$c = \alpha \cdot b \quad \text{where } c_i = \alpha b_i$$

$$c = \sin a \quad \text{where } c_i = \sin a_i$$

- 长度

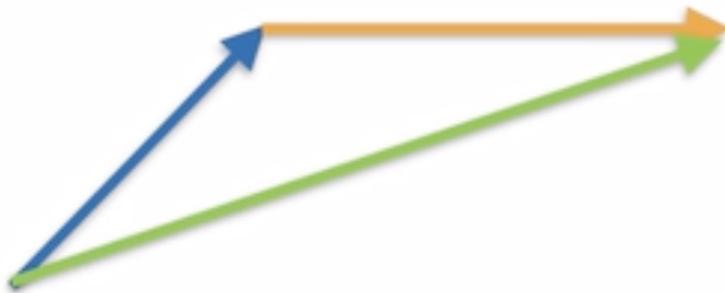
$$\|a\|_2 = \left[ \sum_{i=1}^m a_i^2 \right]^{\frac{1}{2}}$$

$$\|a\| \geq 0 \text{ for all } a$$

$$\|a + b\| \leq \|a\| + \|b\|$$

$$\|a \cdot b\| = |a| \cdot \|b\|$$

## 向量



$$c = a + b$$



$$c = \alpha \cdot b$$

## 矩阵的操作

# 矩阵

- 乘法 (矩阵乘以向量)

$$c = Ab \text{ where } c_i = \sum_j A_{ij} b_j$$



# 矩阵

- 范数

$$c = A \cdot b \text{ hence } \|c\| \leq \|A\| \cdot \|b\|$$

- 取决于如何衡量  $b$  和  $c$  的长度
- 常见范数
  - 矩阵范数：最小的满足的上面公式的值
  - Frobenius 范数

$$\|A\|_{\text{Frob}} = \left[ \sum_{ij} A_{ij}^2 \right]^{\frac{1}{2}}$$

动手学深度学习 v2 • <https://courses.d2l.ai/zh-v2>

(矩阵范数麻烦且不常用，一般用F范数)

特殊矩阵

# 特殊矩阵



- 对称和反对称



$$A_{ij} = A_{ji} \text{ and } A_{ij} = -A_{ji}$$



- 正定

$$\|x\|^2 = x^\top x \geq 0 \text{ generalizes to } x^\top Ax \geq 0$$



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

(深度学习里基本不会涉及到正定、置换矩阵，这里明确个概念就行)

## 特殊矩阵

- 正交矩阵

- 所以行都相互正交

- 所有行都有单位长度  $U$  with  $\sum_j U_{ij} U_{kj} = \delta_{ik}$

- 可以写成  $UU^\top = \mathbf{1}$

- 置换矩阵

$P$  where  $P_{ij} = 1$  if and only if  $j = \pi(i)$

- 置换矩阵是正交矩阵

## 特征向量和特征值

[08:30 特征向量和特征值](#)

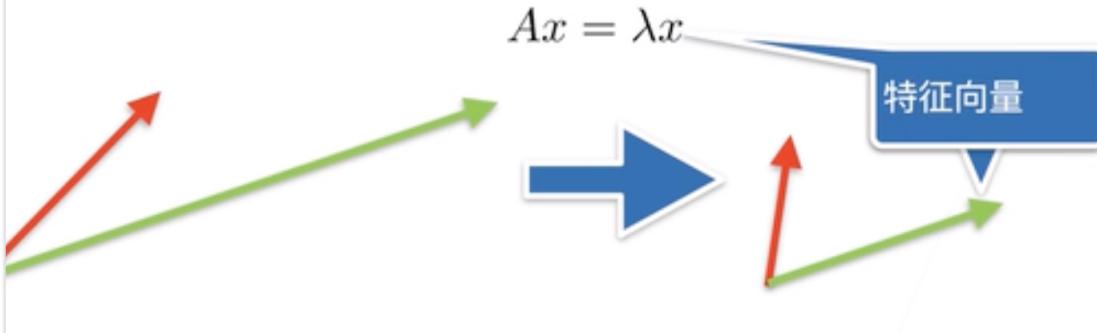
- 数学定义：设  $A$  是  $n$  阶方阵，如果存在常数  $\lambda$  及非零  $n$  向量  $x$ ，使得  $Ax = \lambda x$ ，则称  $\lambda$  是矩阵  $A$  的特征值， $x$  是  $A$  属于特征值  $\lambda$  的特征向量。

- 直观理解：不被矩阵  $A$  改变方向  
(大小改变没关系) 的向量  $x$  就是  $A$  的一个特征向量

# 矩阵



- 特征向量和特征值
  - 不被矩阵改变方向的向量



- 对称矩阵总是可以找到特征向量



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

- 矩阵不一定有特征向量，但是对称矩阵总是可以找到特征向量

## 1.5.2 线性代数实现

[00:09 线性代数实现](#)

这部分主要是应用pytorch实现基本矩阵操作，同样由标量过渡到向量最后拓展到矩阵

### 1. 标量

```
import torch      # 应用pytorch框架

# 标量由只有一个元素的张量表示
x = torch.tensor([3.0])      # 单独一个数字表示标量也可以
y = torch.tensor([2.0])      # 单独一个数字表示标量也可以
print(x + y)    # tensor([5.])
print(x * y)    # tensor([6.])
print(x / y)    # tensor([1.5000])
print(x ** y)   # tensor([9.]) 指数运算
```

language=python

### 2. 向量

```
# 向量可以看作是若干标量值组成的列表
x = torch.arange(4)      # tensor([0, 1, 2, 3])
                        # 生成[0, 4)范围内所有整数构成的张量tensor
print(x[3])            # tensor(3)
                        # 和列表相似，通过张量的索引访问元素
print(len(x))          # 4
                        # 获取张量x的长度
print(x.shape)          # torch.Size([4])
```

language=python

```
# 获取张量形状，这里x是只有一个轴的张量因此形状只有一个元素
```

### 3. 矩阵

#### 1. 创建

```
A = torch.arange(6)      # tensor([0, 1, 2, 3, 4, 5])
B = torch.tensor([[1, 2, 3], [2, 0, 4], [3, 4, 5]])
C = torch.tensor([[[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]],
                 [[0, 0, 0],
                  [1, 1, 1],
                  [2, 2, 2]]])
D = torch.arange(20, dtype=torch.float32)
```

language=python

#### 2. 转置

```
A = torch.arange(6)      # tensor([0, 1, 2, 3, 4, 5])
A = A.reshape(3, 2)      # tensor([[0, 1],
                                #                 [2, 3],
                                #                 [4, 5]]))

A = A.T                  # 转置 A.T
# tensor([[0, 2, 4],
#         [1, 3, 5]])
```

language=python

#### 3. reshape

```
# 使用reshape方法创建一个形状为3 x 2的矩阵A
A = torch.arange(6)      # tensor([0, 1, 2, 3, 4, 5])
A = A.reshape(3, 2)      # tensor([[0, 1],
                                #                 [2, 3],
                                #                 [4, 5]])
```

language=python

#### 4. clone

```
A = torch.arange(20, dtype=torch.float32)
A = A.reshape(5, 4)
B = A.clone()      # 通过分配新内存，将A的一个副本分给B，该边B并不影响A的值
print(B)

...
tensor([[ 0.,  1.,  2.,  3.],
        [ 4.,  5.,  6.,  7.],
        [ 8.,  9., 10., 11.],
        [12., 13., 14., 15.],
        [16., 17., 18., 19.]])
```

language=python

#### 5. 按元素乘

两个矩阵的按元素乘法称为 **哈达玛积 (Hadamard product)**，数学符号为  $\odot$ 。

```
>>> A*B
tensor([[ 0.,  1.,  2.,  3.],
        [ 4.,  5.,  6.,  7.],
        [ 8.,  9., 10., 11.],
```

language=python

```
[12., 13., 14., 15.],
[16., 17., 18., 19.]])
```

## 6. sum/mean

```
A = torch.tensor([[[1, 2, 3],
                  [4, 5, 6],
                  [7, 8, 9]],
                 [[0, 0, 0],
                  [1, 1, 1],
                  [2, 2, 2]]])

print(A.shape)
# torch.Size([2, 3, 3])

print(A.sum())
# tensor(54)

print(A.sum(axis=0))
#####
tensor([[ 1,  2,  3],
        [ 5,  6,  7],
        [ 9, 10, 11]])
#####

print(A.sum(axis=0, keepdims=True))
#####
tensor([[ 1,  2,  3],
        [ 5,  6,  7],
        [ 9, 10, 11]])
#####

print(A.sum(axis=1))
#####
tensor([[12, 15, 18],
        [ 3,  3,  3]])
#####

print(A.sum(axis=1, keepdims=True))
#####
tensor([[12, 15, 18],
        [[ 3,  3,  3]]])
#####

print(A.sum(axis=2))
#####
tensor([[ 6, 15, 24],
        [ 0,  3,  6]])
#####

print(A.sum(axis=2, keepdims=True)) # 保留维度信息
#####
tensor([[[ 6],
          [15],
          [24]],
         [[ 0],
          [ 3],
          [ 6]]])
#####
```

language=python

```
print(A.sum(axis=[0,1]))  
# tensor([15, 18, 21])  
  
print(A.sum(axis=[0,1], keepdim=True))  
# tensor([[15, 18, 21]])
```

## 7. cumsum 累加

```
>>> A, A.cumsum(axis = 1)  
  
tensor([[ 0.,  1.,  2.,  3.],  
       [ 4.,  5.,  6.,  7.],  
       [ 8.,  9., 10., 11.],  
       [12., 13., 14., 15.],  
       [16., 17., 18., 19.]]),  
tensor([[ 0.,  1.,  3.,  6.],  
       [ 4.,  9., 15., 22.],  
       [ 8., 17., 27., 38.],  
       [12., 25., 39., 54.],  
       [16., 33., 51., 70.]])
```

language=python

## 8. numel

```
A = torch.tensor([[0., 0., 0.], [1., 1., 1.]])  
print(A.numel())      # 6 元素个数
```

language=python

## 9. 2.3.7 mean

```
A = torch.tensor([[0., 0., 0.], [1., 1., 1.]])  
print(A.numel())      # 6 元素个数  
print(A.sum())        # tensor(3.)  
print(A.mean())       # tensor(0.5000)  
  
# 特定轴  
A = torch.tensor([[0., 0., 0.], [1., 1., 1.]])  
print(A.shape[0])      # 2  
print(A.sum(axis=0))   # tensor([1., 1., 1.])  
print(A.mean(axis=0))  # tensor([0.5000, 0.5000, 0.5000])      平均值
```

language=python

## 10. dot

```
>>> x = torch.tensor([0., 1., 2., 3.])  
>>> y = torch.tensor([1., 1., 1., 1.])  
>>> print(torch.dot(x, y))  
  
tensor(6.)
```

language=python

## 11. mm、mv

```
A = torch.tensor([[0, 1, 2],  
                 [3, 4, 5]])  
B = torch.tensor([[2, 2],  
                 [1, 1],  
                 [0, 0]])  
x = torch.tensor([3, 3, 3])  
  
print(torch.mm(A, x)) # matrix-vector product 向量积  
=====  
tensor([ 9, 36])  
=====
```

language=python

```
print(torch.mm(A, B)) # 矩阵积
.....
tensor([[ 1.,  1.],
       [10., 10.]])
....
```

## 12. L1、L2、F范数

```
x = torch.tensor([3.0, -4.0])
print(torch.abs(x).sum())    # 向量的L1范数: tensor(7.) x中的每个元素绝对值的和
print(torch.norm(x))         # 向量的L2范数: tensor(5.) x中的每个元素平方的和开根号

A = torch.ones((4, 9))
print(torch.norm(A))         # 矩阵的F范数: tensor(6.) A中的每个元素平方的和开根号
```

language=python

## 13. 运算

```
A = torch.arange(20, dtype=torch.float32)
A = A.reshape(5,4)
B = A.clone()

print(B)      # tensor([[ 0.,  1.,  2.,  3.],
#                  [ 4.,  5.,  6.,  7.],
#                  [ 8.,  9., 10., 11.],
#                  [12., 13., 14., 15.],
#                  [16., 17., 18., 19.]])  
  
print(A == B)
.....
tensor([[True, True, True, True],
       [True, True, True, True],
       [True, True, True, True],
       [True, True, True, True],
       [True, True, True, True]])
.....
  
  
print(A + B)
.....
tensor([[ 0.,  2.,  4.,  6.],
       [ 8., 10., 12., 14.],
       [16., 18., 20., 22.],
       [24., 26., 28., 30.],
       [32., 34., 36., 38.]])
.....
  
  
print(A * B)
.....
tensor([[ 0.,  1.,  4.,  9.],
       [16., 25., 36., 49.],
       [64., 81., 100., 121.],
       [144., 169., 196., 225.],
       [256., 289., 324., 361.]])
....
```

language=python

## 14. 广播

```
A = torch.tensor([[1., 2., 3.],
                  [4., 5., 6.]]))
B = A.sum(axis=1, keepdims=True)
```

language=python

```
print(B)
#####
tensor([[ 6.],
       [15.]])
#####

print(A / B)
#####
tensor([[0.1667, 0.3333, 0.5000],
       [0.2667, 0.3333, 0.4000]])
#####

print(A + B)
#####
tensor([[ 7.,  8.,  9.],
       [19., 20., 21.]])
#####

print(A * B)
#####
tensor([[ 6., 12., 18.],
       [60., 75., 90.]])
#####
```

### 1.5.3 按特定轴求和

[00:01 按特定轴求和](#)

[00:01 Q&A](#)

## 1.6 矩阵计算

[00:00 矩阵计算](#)

### 1.6.1 导数的概念及几何意义

#### 标量导数

- 导数是切线的斜率

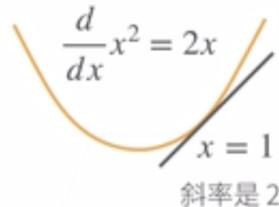


## 标量导数

$y$	$a$	$x^n$	$\exp(x)$	$\log(x)$	$\sin(x)$
$\frac{dy}{dx}$	0	$nx^{n-1}$	$\exp(x)$	$\frac{1}{x}$	$\cos(x)$

$a$  不是  $x$  的函数

导数是切线的斜率



$y$	$u + v$	$uv$	$y = f(u), u = g(x)$
$\frac{dy}{dx}$	$\frac{du}{dx} + \frac{dv}{dx}$	$\frac{du}{dx}v + \frac{dv}{dx}u$	$\frac{dy}{du}\frac{du}{dx}$



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

- 指向值变化最大的方向

## 亚导数（偏导数）

02:16 亚导数

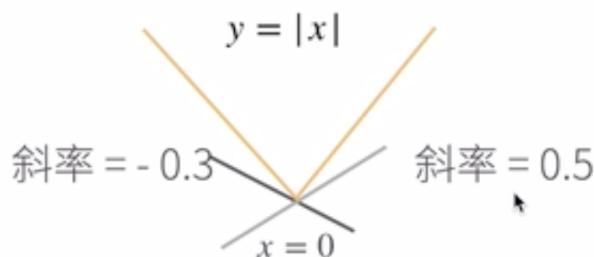
- 将导数拓展到不可微的函数，在不可导的点的导数可以用一个范围内的数表示

## 亚导数



- 将导数拓展到不可微的函数

另一个例子



$$\frac{\partial}{\partial x} \max(x, 0) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \\ a & \text{if } x = 0, \quad a \in [0, 1] \end{cases}$$

$$\frac{\partial |x|}{\partial x} = \begin{cases} 1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \\ a & \text{if } x = 0, \quad a \in [-1, 1] \end{cases}$$



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

## 梯度

# 梯度

- 将导数拓展到向量

			向量
标量	$x$	$\frac{\partial y}{\partial x}$	$\frac{\partial y}{\partial \mathbf{x}}$
标量	$y$		
向量	$\mathbf{y}$	$\frac{\partial \mathbf{y}}{\partial x}$	$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$

## 1.6.2 函数与标量，向量，矩阵

该部分结合课程视频和参考文章进行总结（参考了知乎文章：[矩阵求导的本质与分子布局、分母布局的本质（矩阵求导——本质篇） - 知乎 \(zhihu.com\)](#)）

- 考虑一个函数  $\text{function}(\text{input})$ ，针对  $\text{function}$  的类型、输入  $\text{input}$  的类型，可以将函数分为不同的种类：

### 一、 $\text{function}$ 为是一个标量

称函数  $\text{function}$  是一个**实值标量函数**。用细体小写字母  $f$  表示。

1. input 是一个标量。称函数的变元是标量。用细体小写字母  $x$  表示。

$$f(x) = x + 2$$

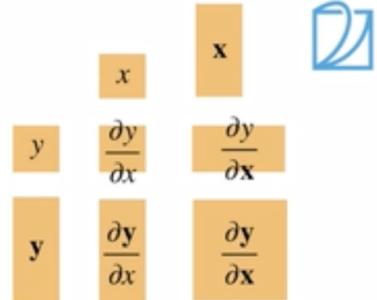
2. input 是一个向量。称 function 的变元是向量。用粗体小写字母  $\mathbf{x}$  表示。

设  $\mathbf{x} = [x_1, x_2, x_3]^T$   
 $f(\mathbf{x}) = a_1x_1^2 + a_2x_2^2 + a_3x_3^2 + a_4x_1x_2$

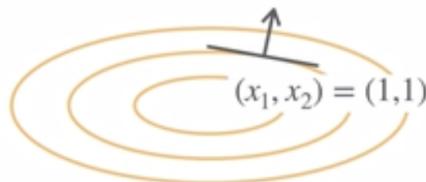
求导：

$$\partial \mathbf{y} / \partial \mathbf{x}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \left[ \frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \dots, \frac{\partial y}{\partial x_n} \right]$$



$$\frac{\partial}{\partial \mathbf{x}} x_1^2 + 2x_2^2 = [2x_1, 4x_2] \quad \text{方向 } (2, 4) \text{ 跟等高线正交}$$



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

样例：

## 样例

$$\begin{array}{c|ccccc} y & a & au & \text{sum}(\mathbf{x}) & \|\mathbf{x}\|^2 \\ \hline \frac{\partial y}{\partial \mathbf{x}} & \mathbf{0}^T & a \frac{\partial u}{\partial \mathbf{x}} & \mathbf{1}^T & 2\mathbf{x}^T \end{array}$$

$a$  is not a function of  $\mathbf{x}$

$\mathbf{0}$  and  $\mathbf{1}$  are vectors

$$\begin{array}{c|ccccc} y & u + v & uv & \langle \mathbf{u}, \mathbf{v} \rangle \\ \hline \frac{\partial y}{\partial \mathbf{x}} & \frac{\partial u}{\partial \mathbf{x}} + \frac{\partial v}{\partial \mathbf{x}} & \frac{\partial u}{\partial \mathbf{x}}v + \frac{\partial v}{\partial \mathbf{x}}u & \mathbf{u}^T \frac{\partial \mathbf{v}}{\partial \mathbf{x}} + \mathbf{v}^T \frac{\partial \mathbf{u}}{\partial \mathbf{x}} \end{array}$$

动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>



3. input 是一个矩阵。称 function 的变元是矩阵。用粗体大写字母  $\mathbf{X}$  表示。

设  $\mathbf{X}_{3 \times 2} = (x_{ij})_{i=1,j=1}^{3,2}$   
 $f(\mathbf{X}) = a_1x_{11}^2 + a_2x_{12}^2 + a_3x_{21}^2 + a_4x_{22}^2 + a_5x_{31}^2 + a_6x_{32}^2$

## 二、function 为是一个向量

称函数 function 是一个实向量函数。用粗体小写字母  $\mathbf{f}$  表示。

含义： $\mathbf{f}$  是由若干个  $f$  组成的一个向量。

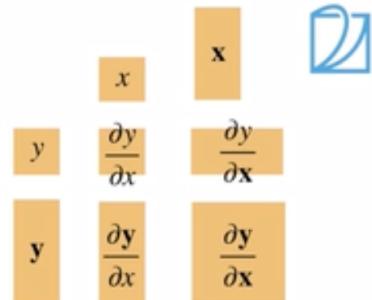
1. input 是标量。[08:39](#)

$$f_{3 \times 1}(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ f_3(x) \end{bmatrix} = \begin{bmatrix} x+1 \\ 2x+1 \\ 3x^2+1 \end{bmatrix}$$

求导：

$\partial \mathbf{y} / \partial x$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} \quad \frac{\partial \mathbf{y}}{\partial x} = \begin{bmatrix} \frac{\partial y_1}{\partial x} \\ \frac{\partial y_2}{\partial x} \\ \vdots \\ \frac{\partial y_m}{\partial x} \end{bmatrix}$$



$\partial \mathbf{y} / \partial \mathbf{x}$  是行向量,  $\partial \mathbf{y} / \partial x$  是列向量

这个被称之为分子布局符号, 反过来的  
版本叫分母布局符号



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

称之为分子布局符号, 反过来称为坟墓布局符号。

2. input 是向量。[09:15](#)

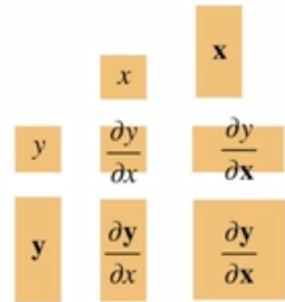
称 function 的变元是向量。用粗体小写字母  $\mathbf{x}$  表示。

$$\mathbf{f}_{3 \times 1}(\mathbf{x}) = \begin{bmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ f_3(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} x_1 + x_2 + x_3 \\ x_1^2 + 2x_2 + 2x_3 \\ x_1x_2 + x_2 + x_3 \end{bmatrix}$$

求导：

$\partial \mathbf{y} / \partial \mathbf{x}$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}$$



$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \frac{\partial y_1}{\partial \mathbf{x}} \\ \frac{\partial y_2}{\partial \mathbf{x}} \\ \vdots \\ \frac{\partial y_m}{\partial \mathbf{x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial y_1}{\partial x_1}, \frac{\partial y_1}{\partial x_2}, \dots, \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1}, \frac{\partial y_2}{\partial x_2}, \dots, \frac{\partial y_2}{\partial x_n} \\ \vdots \\ \frac{\partial y_m}{\partial x_1}, \frac{\partial y_m}{\partial x_2}, \dots, \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

样例：

## 样例

$\mathbf{y}$	$\mathbf{a}$	$\mathbf{x}$	$\mathbf{Ax}$	$\mathbf{x}^T \mathbf{A}$	$\mathbf{x} \in \mathbb{R}^n, \quad \mathbf{y} \in \mathbb{R}^m, \quad \frac{\partial \mathbf{y}}{\partial \mathbf{x}} \in \mathbb{R}^{m \times n}$
$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	$\mathbf{0}$	$\mathbf{I}$	$\mathbf{A}$	$\mathbf{A}^T$	$a, \mathbf{a}$ and $\mathbf{A}$ are not functions of $\mathbf{x}$ $\mathbf{0}$ and $\mathbf{I}$ are matrices

$\mathbf{y}$	$a\mathbf{u}$	$\mathbf{Au}$	$\mathbf{u} + \mathbf{v}$
$\frac{\partial \mathbf{y}}{\partial \mathbf{x}}$	$a \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	$\mathbf{A} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$	$\frac{\partial \mathbf{u}}{\partial \mathbf{x}} + \frac{\partial \mathbf{v}}{\partial \mathbf{x}}$



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

3. input 是矩阵。

$$\mathbf{f}_{3 \times 1}(\mathbf{X}) = \begin{bmatrix} f_1(\mathbf{X}) \\ f_2(\mathbf{X}) \\ f_3(\mathbf{X}) \end{bmatrix} = \begin{bmatrix} x_{11} + x_{12} + x_{21} + x_{22} + x_{31} + x_{32} \\ x_{11} + x_{12} + x_{21} + x_{22} + x_{31} + x_{32} + x_{11}x_{12} \\ 2x_{11} + x_{12} + x_{21} + x_{22} + x_{31} + x_{32} + x_{11}x_{12} \end{bmatrix}$$

三、function 为是一个矩阵

称函数 function 是一个**实矩阵函数**。用粗体大写字母  $\mathbf{F}$  表示。

含义:  $\mathbf{F}$  是由若干个  $f$  组成的一个**矩阵**。

1. input 是标量。

$$\mathbf{F}_{3 \times 2}(x) = \begin{bmatrix} f_{11}(x) & f_{12}(x) \\ f_{21}(x) & f_{22}(x) \\ f_{31}(x) & f_{32}(x) \end{bmatrix} = \begin{bmatrix} x+1 & 2x+2 \\ x^2+1 & 2x^2+1 \\ x^3+1 & 2x^3+1 \end{bmatrix}$$

2. input 是一个向量。称 function 的**变元**是**向量**。用**粗体小写字母**  $\mathbf{x}$  表示。

$$\mathbf{F}_{3 \times 2}(\mathbf{x}) = \begin{bmatrix} f_{11}(\mathbf{x}) & f_{12}(\mathbf{x}) \\ f_{21}(\mathbf{x}) & f_{22}(\mathbf{x}) \\ f_{31}(\mathbf{x}) & f_{32}(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 + x_2 + x_3 & 2x_1 + 2x_2 + x_3 \\ 2x_1 + 2x_2 + x_3 & x_1 + 2x_2 + x_3 \\ 2x_1 + x_2 + 2x_3 & x_1 + 2x_2 + 2x_3 \end{bmatrix}$$

3. input 是矩阵。

$$\begin{aligned} \mathbf{F}_{3 \times 2}(\mathbf{X}) &= \begin{bmatrix} f_{11}(\mathbf{X}) & f_{12}(\mathbf{X}) \\ f_{21}(\mathbf{X}) & f_{22}(\mathbf{X}) \\ f_{31}(\mathbf{X}) & f_{32}(\mathbf{X}) \end{bmatrix} \\ &= \begin{bmatrix} x_{11} + x_{12} + x_{21} + x_{22} + x_{31} + x_{32} & 2x_{11} + x_{12} + x_{21} + x_{22} + x_{31} + x_{32} \\ 3x_{11} + x_{12} + x_{21} + x_{22} + x_{31} + x_{32} & 4x_{11} + x_{12} + x_{21} + x_{22} + x_{31} + x_{32} \\ 5x_{11} + x_{12} + x_{21} + x_{22} + x_{31} + x_{32} & 6x_{11} + x_{12} + x_{21} + x_{22} + x_{31} + x_{32} \end{bmatrix} \end{aligned}$$

求导:

## 拓展到矩阵

	标量	向量	矩阵
标量	$x$ (1,)	$\mathbf{x}$ ( $n, 1$ )	$\mathbf{X}$ ( $n, k$ )
向量	$y$ (1,)	$\frac{\partial y}{\partial \mathbf{x}}$ ( $1, n$ )	$\frac{\partial \mathbf{y}}{\partial \mathbf{X}}$ ( $k, n$ )
矩阵	$\mathbf{Y}$ ( $m, l$ )	$\frac{\partial \mathbf{Y}}{\partial \mathbf{x}}$ ( $m, l$ )	$\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ ( $m, l, n$ )

动手学深度学习 v2 · <https://courses.d2l.ai/zh/v2>

## 求导的本质

对于一个多元函数:

$$f(x_1, x_2, x_3) = x_1^2 + x_1 x_2 + x_2 x_3$$

可以将  $f$  对  $x_1, x_2, x_3$  的偏导分别求出来, 即

$$\begin{cases} \frac{\partial f}{\partial x_1} = 2x_1 + x_2 \\ \frac{\partial f}{\partial x_2} = x_1 + x_3 \\ \frac{\partial f}{\partial x_3} = x_2 \end{cases}$$

矩阵求导也是一样的，本质就是 function 中的每个  $f$  分别对变元中的每个元素逐个求偏导，只不过写成了向量、矩阵形式而已。

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}_{3 \times 1}} = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 2x_1 + x_2 \\ x_1 + x_3 \\ x_2 \end{bmatrix}$$

也可以按照 行向量形式展开：

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}_{3 \times 1}^T} = \left[ \frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3} \right] = [2x_1 + x_2, x_1 + x_3, x_2]$$

$\mathbf{X}$  为矩阵时，先把矩阵变元  $\mathbf{X}$  进行转置，再对转置后的每个位置的元素逐个求偏导，结果布局和转置布局一样。

$$\begin{aligned} D_{\mathbf{X}} f(\mathbf{X}) &= \frac{\partial f(\mathbf{X})}{\partial \mathbf{X}_{m \times n}^T} \\ &= \begin{bmatrix} \frac{\partial f}{\partial x_{11}} & \frac{\partial f}{\partial x_{21}} & \dots & \frac{\partial f}{\partial x_{11}} \\ \frac{\partial f}{\partial x_{12}} & \frac{\partial f}{\partial x_{22}} & \dots & \frac{\partial f}{\partial x_{m2}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f}{\partial x_{1n}} & \frac{\partial f}{\partial x_{2n}} & \dots & \frac{\partial f}{\partial x_{mn}} \end{bmatrix}_{n \times m} \end{aligned}$$

- 所以，如果 function 中有  $m$  个  $f$  (标量)，变元中有  $n$  个元素，那么，每个  $f$  对变元中的每个元素逐个求偏导后，我们就会产生  $m \times n$  个结果。

### 1.6.3 矩阵求导的布局

- 经过上述对求导本质的推导，关于矩阵求导的问题，实质上就是对求导结果的进一步排布问题

对于 2.2 ( $f$  为向量，input 也为向量) 中的情况，其求导结果有两种排布方式，一种是分子布局，一种是分母布局

1. 分子布局。就是分子是列向量形式，分母是行向量形式 (课上讲的)

$$\frac{\partial \mathbf{f}_{2 \times 1}(\mathbf{x})}{\partial \mathbf{x}_{3 \times 1}^T} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \frac{\partial f_1}{\partial x_3} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \frac{\partial f_2}{\partial x_3} \end{bmatrix}_{2 \times 3}$$

2. 分母布局，就是分母是列向量形式，分子是行向量形式

$$\frac{\partial \mathbf{f}_{2 \times 1}^T(\mathbf{x})}{\partial \mathbf{x}_{3 \times 1}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_2}{\partial x_1} \\ \frac{\partial f_1}{\partial x_2} & \frac{\partial f_2}{\partial x_2} \\ \frac{\partial f_1}{\partial x_3} & \frac{\partial f_2}{\partial x_3} \end{bmatrix}_{3 \times 2}$$

将求导推广到矩阵，由于矩阵可以看作由多个向量所组成，因此对矩阵的求导可以看作先对每个向量进行求导，然后再增加一个维度存放求导结果。

例如当  $F$  为矩阵，input 为矩阵时， $F$  中的每个元素  $f$  (标量) 求导后均为一个矩阵 (按照课上的展开方式)，因此每个  $f$  (包含多个  $f$  (标量)) 求导后为存放多个矩阵的三维形状，再由于矩阵  $F$  由多个  $f$  组成，因此  $F$  求导后为存放多个  $f$  求导结果的四维形状。

对于不同  $f$  和 input 求导后的维度情况总结如下图所示：



# 拓展到矩阵

	标量	向量	矩阵
标量	$x$ (1,)	$\mathbf{x}$ (n,1)	$\mathbf{X}$ (n, k)
向量	$y$ (1,)	$\frac{\partial y}{\partial x}$ (1,)	$\frac{\partial y}{\partial \mathbf{x}}$ (1,n)
矩阵	$\mathbf{Y}$ (m, l)	$\frac{\partial \mathbf{Y}}{\partial x}$ (m, l)	$\frac{\partial \mathbf{Y}}{\partial \mathbf{x}}$ (m, l, n)



动手学深度学习 v2 • <https://courses.d2l.ai/zh-v2>

## 1.6.4 Q&A

[00:00 Q&A](#)

## 1.7 自动求导

### 1.7.1 向量链式法则

[00:00 自动求导](#)

#### 1. 标量链式法则

$$y = f(u), u = g(x) \quad \frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \frac{\partial u}{\partial x}$$

#### 2. 拓展到向量

需要注意维数的变化

下图三种情况分别对应：

1.  $y$ 为标量,  $x$ 为向量
2.  $y$ 为标量,  $x$ 为矩阵
3.  $y$ 、 $x$ 为矩阵



# 向量链式法则

- 标量链式法则

$$y = f(u), u = g(x) \quad \frac{\partial y}{\partial x} = \frac{\partial y}{\partial u} \frac{\partial u}{\partial x}$$

- 拓展到向量

$$\frac{\partial y}{\partial \mathbf{x}} = \frac{\partial y}{\partial u} \frac{\partial u}{\partial \mathbf{x}}$$

(1,n) (1,) (1,n)

$$\frac{\partial y}{\partial \mathbf{x}} = \frac{\partial y}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

(1,n) (1,k) (k, n)

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

(m, n), (m, k) (k, n)



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

## 链式法则示例

[01:30 链式法则示例](#)

### 1. 标量对向量求导

这里应该是用分子布局，所以是X转置

#### 例子 1

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

假设

$$\mathbf{x}, \mathbf{w} \in \mathbb{R}^n, \quad y \in \mathbb{R}$$

$$z = (\langle \mathbf{x}, \mathbf{w} \rangle - y)^2$$

计算

$$\frac{\partial z}{\partial \mathbf{w}}$$

$$\begin{aligned} \frac{\partial z}{\partial \mathbf{w}} &= \frac{\partial z}{\partial b} \frac{\partial b}{\partial a} \frac{\partial a}{\partial \mathbf{w}} \\ &= \frac{\partial b^2}{\partial b} \frac{\partial a - y}{\partial a} \frac{\partial \langle \mathbf{x}, \mathbf{w} \rangle}{\partial \mathbf{w}} \\ &= 2b \cdot 1 \cdot \mathbf{x}^T \end{aligned}$$

分解

$$a = \langle \mathbf{x}, \mathbf{w} \rangle$$



$$b = a - y$$

$$z = b^2$$

$$= 2 (\langle \mathbf{x}, \mathbf{w} \rangle - y) \mathbf{x}^T$$

动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>



### 2. 涉及到矩阵的情况

$X$ 是 $m \times n$ 的矩阵,  $w$ 为 $n$ 维向量,  $y$ 为 $m$ 维向量;  $z$ 对 $Xw - y$ 做L2 norm, 为标量; 过程与例一大体一致;

## 例子 2

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \frac{\partial \mathbf{y}}{\partial \mathbf{u}} \frac{\partial \mathbf{u}}{\partial \mathbf{x}}$$

假设  $\mathbf{X} \in \mathbb{R}^{m \times n}$ ,  $\mathbf{w} \in \mathbb{R}^n$ ,  $\mathbf{y} \in \mathbb{R}^m$

$$z = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2$$

计算

$$\frac{\partial z}{\partial \mathbf{w}}$$

$$\begin{aligned}\frac{\partial z}{\partial \mathbf{w}} &= \frac{\partial z}{\partial \mathbf{b}} \frac{\partial \mathbf{b}}{\partial \mathbf{a}} \frac{\partial \mathbf{a}}{\partial \mathbf{w}} \\ &= \frac{\partial \|\mathbf{b}\|^2}{\partial \mathbf{b}} \frac{\partial \mathbf{a} - \mathbf{y}}{\partial \mathbf{a}} \frac{\partial \mathbf{X}\mathbf{w}}{\partial \mathbf{w}} \\ &= 2\mathbf{b}^T \times \mathbf{I} \times \mathbf{X} \\ &= 2(\mathbf{X}\mathbf{w} - \mathbf{y})^T \mathbf{X}\end{aligned}$$

分解

$$\mathbf{a} = \mathbf{X}\mathbf{w}$$

$$\mathbf{b} = \mathbf{a} - \mathbf{y}$$

$$z = \|\mathbf{b}\|^2$$



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

**Note:** 由于在神经网络动辄几百层, 手动进行链式求导是很困难的, 因此我们需要借助自动求导

## 1.7.2 自动求导

[04:20 自动求导](#)

### 自动求导



- 自动求导计算一个函数在指定值上的导数
- 它有别于
  - 符号求导

In[1]:= D[4 x^3 + x^2 + 3, x]

Out[1]= 2 x + 12 x<sup>2</sup>

- 数值求导

$$\frac{\partial f(x)}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h}$$



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

含义: 计算一个函数在指定值上的导数。它有别于:

- 符号求导

- 数值求导

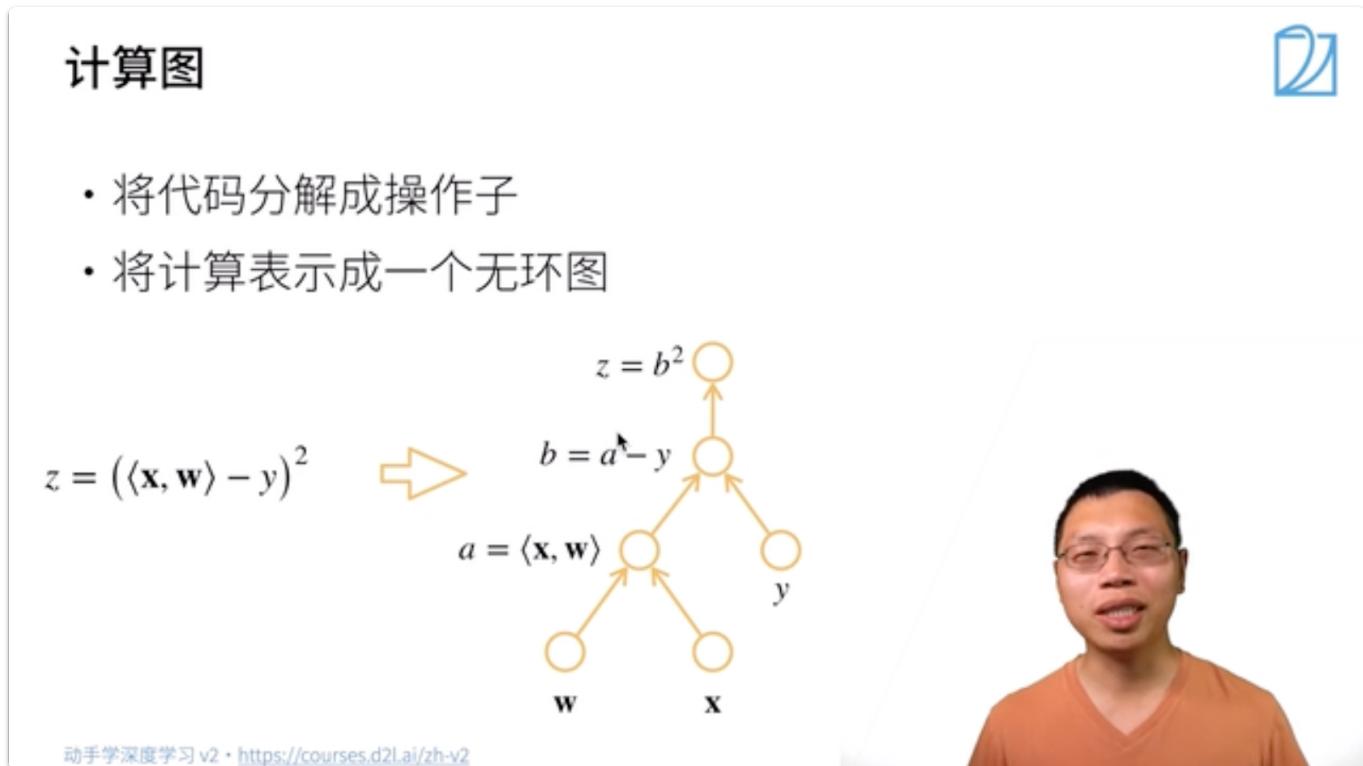
## 一、计算图

[05:11 自动求导](#)

一、步骤：

- 将代码分解成操作子
- 将计算表示成一个无环图

下图自底向上其实就类似于链式求导过程：



二、计算图有两种构造方式

- 显示构造

可以理解为先定义公式再代值

Tensorflow/Theano/MXNet

```
from mxnet import sym
a = sym.var()
b = sym.var()
c = 2 * a + b
# bind data into a and b later
```

language=python

- 隐式构造

系统将所有的计算记录下来

Pytorch/MXNet

```
from mxnet import autograd, nd

with autograd.record():
    a = nd.ones((2, 1))
    b = nd.ones((2, 1))
    c = 2 * a + b
```

language=python

## 二、自动求导的两种模式

[07:39 自动求导的两种模式](#)

### 自动求导的两种模式



- 链式法则:  $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u_n} \frac{\partial u_n}{\partial u_{n-1}} \cdots \frac{\partial u_2}{\partial u_1} \frac{\partial u_1}{\partial x}$
- 正向累积  $\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u_n} \left( \frac{\partial u_n}{\partial u_{n-1}} \left( \cdots \left( \frac{\partial u_2}{\partial u_1} \frac{\partial u_1}{\partial x} \right) \right) \right)$
- 反向累积、又称反向传递

$$\frac{\partial y}{\partial x} = \left( \left( \left( \frac{\partial y}{\partial u_n} \frac{\partial u_n}{\partial u_{n-1}} \right) \cdots \right) \frac{\partial u_2}{\partial u_1} \right) \frac{\partial u_1}{\partial x}$$



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

- 链式法则:

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u_n} \frac{\partial u_n}{\partial u_{n-1}} \cdots \frac{\partial u_2}{\partial u_1} \frac{\partial u_1}{\partial x}$$

#### 1. 正向累积

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u_n} \left( \frac{\partial u_n}{\partial u_{n-1}} \left( \cdots \left( \frac{\partial u_2}{\partial u_1} \frac{\partial u_1}{\partial x} \right) \right) \right)$$

#### 2. 反向累积 (反向传递back propagation)

$$\frac{\partial y}{\partial x} = \left( \left( \left( \frac{\partial y}{\partial u_n} \frac{\partial u_n}{\partial u_{n-1}} \right) \cdots \right) \frac{\partial u_2}{\partial u_1} \right) \frac{\partial u_1}{\partial x}$$

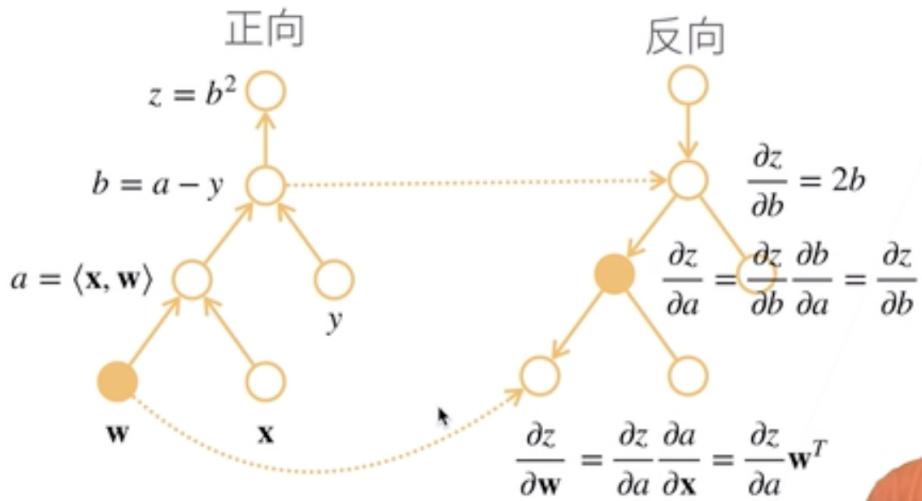
#### 反向累积计算过程

[09:01 反向传递](#)



## 反向累积

$$z = (\langle \mathbf{x}, \mathbf{w} \rangle - y)^2$$



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

注：

反向累积的正向过程：自底向上，需要存储中间结果

反向累积的反向过程：自顶向下，可以去除不需要的枝（图中的x应为w）

# 反向累积总结

- 构造计算图
- 前向：执行图，存储中间结果
- 反向：从相反方向执行图
  - 去除不需要的枝



## 三、复杂度比较

### 1. 反向累积

- 时间复杂度： $O(n)$ ,  $n$ 是操作子数
  - 通常正向和反向的代价类似
- 内存复杂度： $O(n)$ 
  - 存储正向过程所有的中间结果



# 复杂度

- 计算复杂度:  $O(n)$ ,  $n$  是操作子个数
  - 通常正向和反向的代价类似
- 内存复杂度:  $O(n)$ , 因为需要存储正向的所有中间结果
- 跟正向累积对比:
  - $O(n)$  计算复杂度用来计算一个变量的梯度
  - $O(1)$  内存复杂度



动手学深度学习 v2 • <https://courses.d2l.ai/zh-v2>

## 2. 正向累积

每次计算一个变量的梯度时都需要将所有节点扫一遍

- 时间复杂度:  $O(n)$
- 内存复杂度:  $O(1)$

### 1.7.3 代码部分

[00:03 自动求导代码实现](#)

#### 1. 对 $y = x.T \cdot x$ 关于列向量 $x$ 求导

```
# 对y = x.Tx关于列向量x求导
>>> import torch
>>> x = torch.arange(4.0)
>>> x

tensor([0., 1., 2., 3.])
```

language=python

#### 2. 存储梯度

```
#存储梯度
>>> x.requires_grad_(True) #等价于x = torch.arange(4.0, requires_grad=True)
>>> x.grad #默认值是None

>>> y = torch.dot(x, x)
>>> y
# PyTorch隐式地构造计算图, grad_fn用于记录梯度计算

tensor(14., grad_fn=<DotBackward0>)
```

language=python

#### 3. 通过调用反向传播函数来自动计算 $y$ 关于 $x$ 每个分量的梯度

```
>>> y.backward()  
>>> x.grad  
  
tensor([0., 2., 4., 6.])
```

language=python

验证：

```
>>> x.grad==2*x # 验证  
  
tensor([True, True, True, True])
```

language=python

在默认情况下，PyTorch会累积梯度，我们需要清除之前的值：

```
x.grad.zero_()  
# 如果没有上面这一步，结果就会加累上之前的梯度值，变为[1,3,5,7]  
y = x.sum()  
y.backward()  
x.grad
```

language=python

Result:

```
tensor([1., 1., 1., 1.])
```

#### 4. 哈达玛积 (Hadamard product)

```
>>> x.grad.zero_()  
>>> y=x*x # 哈达玛积，对应元素相乘
```

language=python

上述哈达玛积得到的  $y$  是一个向量，此时对  $x$  求导得到的是一个 **矩阵**。但是在深度学习中我们一般不计算微分矩阵，而是计算批量中每个样本单独计算的偏导数之和，如下：

```
>>> y.sum().backward() # 等价于y.backward(torch.ones(len(x)))  
>>> x.grad  
  
tensor([0., 2., 4., 6.])
```

language=python

#### 5. 将某些计算移动到记录的计算图之外。[03:53](#)。

后可用于用于将神经网络的一些参数固定住

```
# 后可用于用于将神经网络的一些参数固定住  
x.grad.zero_()  
y = x*x  
u = y.detach()#把y当作常数  
z = u*x  
  
z.sum().backward()  
x.grad == u
```

language=python

Results:

```
tensor([True, True, True, True])
```

language=python

#### 6. 控制流。[05:26](#)

即使构建函数的计算图需要用过Python控制流，仍然可以计算得到的变量的梯度。这也是隐式构造的优势，因为它会存储梯度计算的计算图，再次计算时执行反向过程就可以

```
def f(a):
    b = a * 2
    while b.norm()<1000:
        b = b * 2
    if b.sum() > 0:
        c = b
    else:
        c = 100 * b
    return c

a = torch.randn(size=(), requires_grad=True)
d = f(a)
d.backward()

a.grad == d / a
```

language=python

Results:

```
tensor(True)
```

language=python

### 1.7.3 Q&A

00:00 Q&A

**Q1: ppt上隐式构造和显式构造看起来为啥差不多？**

显式和隐式的差别其实就是数学上求梯度和python求梯度计算上的差别，不用深究

显式构造就是我们数学上正常求导数的求法，先把所有求导的表达式选出来再代值

**Q2:需要正向和反向都算一遍吗？**

需要正向先算一遍，自动求导时只进行反向就可以，因为正向的结果已经存储

**Q3:为什么PyTorch会默认累积梯度**

便于计算大批量；方便进一步设计

**Q4:为什么深度学习中一般对标量求导而不是对矩阵或向量求导**

loss一般都是标量

**Q5:为什么获取.grad前需要backward**

相当于告诉程序需要计算梯度，因为计算梯度的代价很大，默认不计算

**Q6:pytorch或mxnet框架设计上可以实现矢量的求导吗**

可以

## 2. Algorithm

### 2.1 线性回归

房价预测例子。00:02 线性回归



# 如何在美国买房

- 看中一个房，参观了解
- 估计一个价格，出价



**\$5,498,000** | 7 Beds | 5 Baths | 4,865 Sq. Ft. | \$1130 / Sq. Ft.

Redfin Estimate: \$5,390,037 On Redfin: 15 days

## Virtual Tour

- Branded Virtual Tour
- Virtual Tour (External Link)

## Parking Information

- Garage (Minimum): 2
- Garage (Maximum): 2
- Parking Description: Attached Garage, On Street
- Garage Spaces: 2

## Multi-Unit Information

- # of Stories: 2

## School Information

- Elementary School: El Carmelo
- Middle School: Palo Alto Middle School
- High School: Jane Lathrop High School
- College: Palo Alto High School District: Palo Alto Unified School District

## Interior Features

### Bedroom Information

- # of Bedrooms (Minimum): 7
- # of Bedrooms (Maximum): 7

动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

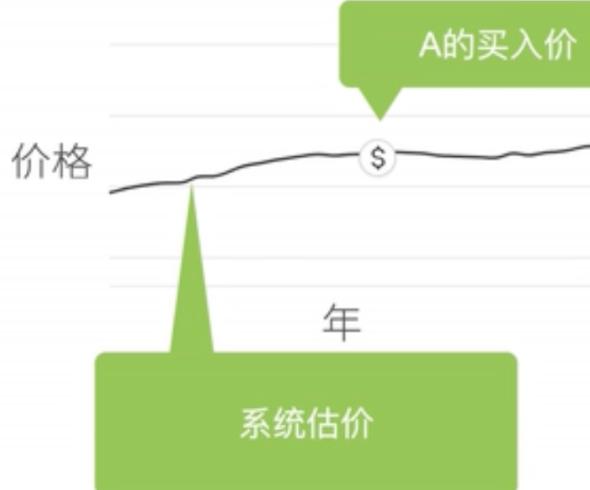


# 房价预测



很重要，这是真钱

\$100K+ 差价



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

## 模型简化。03:27 模型简化

- 假设一：影响房价的关键因素是卧室个数，卫生间个数和居住面积，记为  $x_1, x_2, x_3$ 。
- 假设二：成交价是关键因素的加权和

$$y = w_1x_1 + w_2x_2 + w_3x_3 + b$$

权重和偏差的实际值在后文给出。

## 2.1.1 线性模型

### 一、模型设定

#### 线性模型



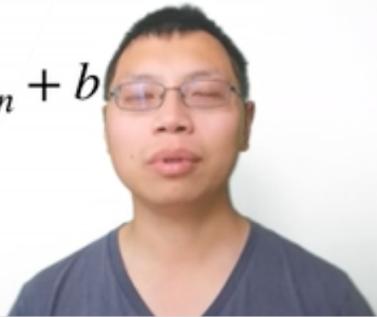
- 给定 n 维输入  $\mathbf{x} = [x_1, x_2, \dots, x_n]^T$
- 线性模型有一个 n 维权重和一个标量偏差

$$\mathbf{w} = [w_1, w_2, \dots, w_n]^T, b$$

- 输出是输入的加权和

$$y = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

向量版本：  $y = \langle \mathbf{w}, \mathbf{x} \rangle + b$



动手学深度学习 v2 · <https://courses.d2l.ai/zh-v2>

- 输入：  $x = [x_1, x_2, \dots, x_n]^T$
- 线性模型需要确定一个n维权重和一个标量偏差

$$w = [w_1, w_2, \dots, w_n]^T, b$$

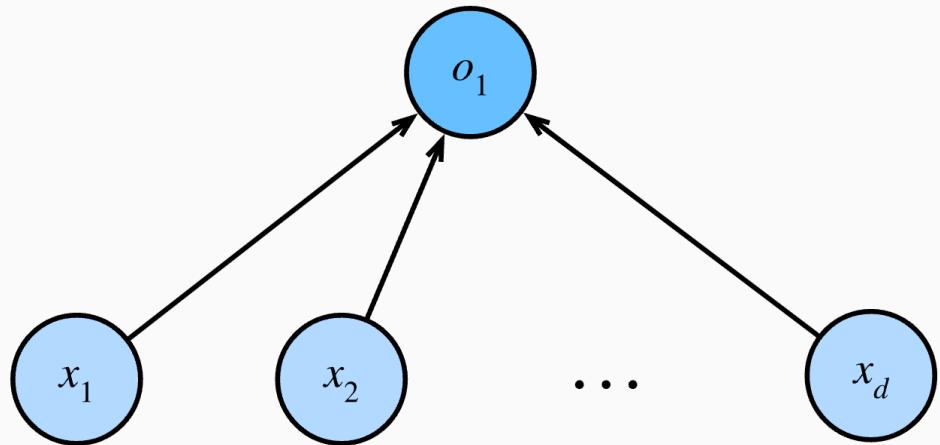
- 输出：

$$y = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$$

向量版本的是  $y = \langle w, x \rangle + b$

**Note**: 线性模型可以看作是单层神经网络 (图片)

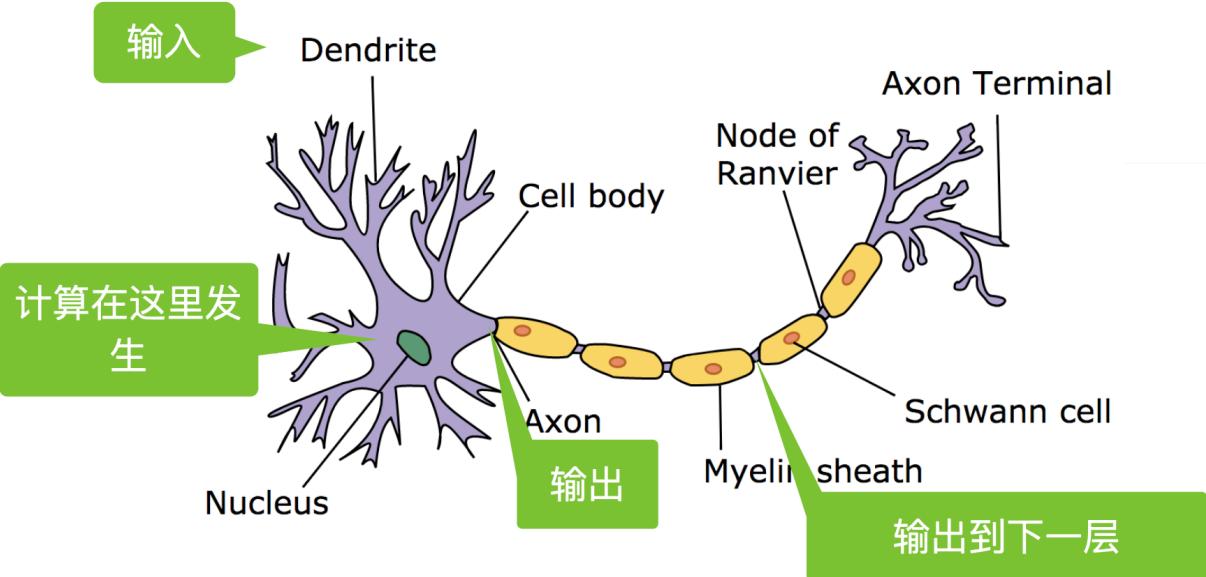
输出层



神经网络源于神经科学：

## 神经网络源于神经科学

真实的神经元



- 最早的神经网络是源自神经科学的，但是时至今日，很多神经网络已经远远高于神经科学，可解释性也不是很强，不必纠结

## 二、衡量估计质量

### 08:08 评价指标

- 需要估计模型的预估值和真实值之间的差距，例如房屋售价和股价
- 假设 $y$ 是真实值， $\tilde{y}$ 是估计值，我们可以比较

$$\ell(y, \tilde{y}) = \frac{1}{2}(y - \tilde{y})^2$$

这个叫做平方损失

### 三、训练数据

- 收集一些数据点来决定参数值（权重  $\omega$  和偏差  $b$ ），例如6个月内被卖掉的房子。这称之为训练数据
- 通常越多越好。

**Note**: 需要注意的是，现实世界的数据都是有限的，但是为了训练出精确的参数往往需要训练数据越多越好，当训练数据不足的时候，我们还需要进行额外处理。

- 假设我们有n个样本，记为

$$X = [x_1, x_2, \dots, x_n]^T, y = [y_1, y_2, \dots, y_n]^T$$

$X$ 的每一行是一个样本， $y$ 的每一行是一个输出的实数值。

### 四、参数学习

#### 10:41 参数学习

- 训练损失。

训练参数时，需要定义一个损失函数来衡量参数的好坏，应用前文提过的平方损失有公式：

$$\ell(\mathbf{X}, \mathbf{y}, \mathbf{w}, b) = \frac{1}{2n} \sum_{i=1}^n (y_i - \langle \mathbf{x}_i, \mathbf{w} \rangle - b)^2 = \frac{1}{2n} \|\mathbf{y} - \mathbf{X}\mathbf{w} - b\|^2$$

- 最小化损失来学习参数。

训练参数的目的就是使损失函数的值尽可能小（这意味着预估值和真实值更接近）。最后求得的参数值可表示为：

$$\mathbf{w}^*, \mathbf{b}^* = \arg \min_{\mathbf{w}, b} l(\mathbf{X}, \mathbf{y}, \mathbf{w}, b)$$

### 五、显示解

#### 12:58 显示解

线性回归有显示解，即可以矩阵数学运算，得到参数 $w$ 和 $b$ 的最优解，而不是用梯度下降，牛顿法等参数优化方式一点点逼近最优解。

推导过程：

- 为了方便矩阵表示和计算，将偏差加入权重， $X \leftarrow [X, 1]$ ,  $\mathbf{w} \leftarrow \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}$

$$\ell(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \frac{1}{2n} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 \quad \frac{\partial}{\partial \mathbf{w}} \ell(\mathbf{X}, \mathbf{y}, \mathbf{w}) = \frac{1}{n} (\mathbf{y} - \mathbf{X}\mathbf{w})^T \mathbf{X}$$

- 损失函数是凸函数，最优解满足导数为0，可解出显示解：

$$\frac{\partial}{\partial \omega} l(X, y, \omega) = 0$$

$$\text{有, } \frac{1}{n} (\mathbf{y} - \mathbf{X}\mathbf{w})^T \mathbf{X} = 0$$

$$\text{解得: } \omega^* = (X^T X)^{-1} X^T y$$

## 六、总结

- 线性回归是对  $n$  维输入的加权，外加偏差
- 使用平方损失来衡量预测值和真实值之间的误差
- 线性回归有显示解
- 线性回归可以看作单层神经网络

### 2.1.2 基础优化算法

[00:00 基础优化算法](#)

#### 一、梯度下降

当模型没有显示解的时候，应用梯度下降法逼近最优解。梯度下降法的具体步骤：

- 挑选一个初始值  $w_0$ ,
- 重复迭代参数，迭代公式为：

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \frac{\partial \ell}{\partial \mathbf{w}_{t-1}}$$

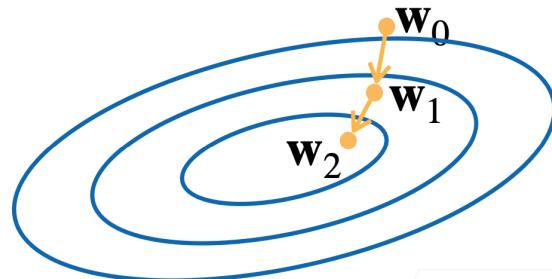
$-\frac{\partial l}{\partial w_{t-1}}$  为函数值下降最快的方向，学习率  $\eta$  为学习步长。

## 梯度下降

- 挑选一个初始值  $\mathbf{w}_0$
- 重复迭代参数  $t=1,2,3$

$$\mathbf{w}_t = \mathbf{w}_{t-1} - \eta \frac{\partial \ell}{\partial \mathbf{w}_{t-1}}$$

- 沿梯度方向将增加损失函数值
- 学习率：步长的超参数

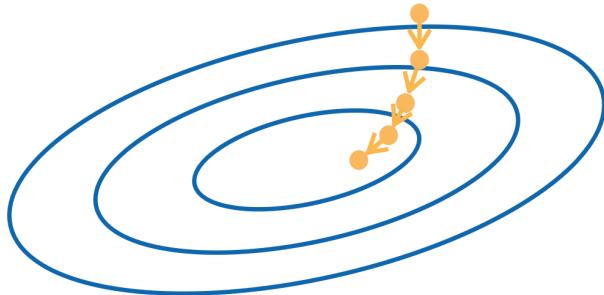


#### 二、选择学习率

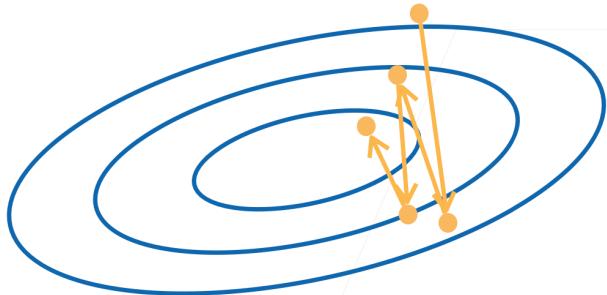
学习率  $\eta$  为学习步长，代表了沿负梯度方向走了多远，这是超参数（人为指定的值，不是训练得到的）

# 选择学习率

不能太小



也不能太大



- Note: 学习率不能太大，也不能太小，需要选取适当。

## 三、小批量随机梯度下降

### 04:15 小批量随机梯度下降

在实际应用中，很少直接应用梯度下降法，这是因为在整个训练集上计算梯度代价太大，一个深度神经网络模型可能需要数分钟至数小时。

为了减少运算代价，我们可以 随机采样  $b$  个样本  $i_1, i_2, \dots, i_b$  来近似损失，损失函数为：

$$\frac{1}{b} \sum_{i \in I_b} l(x_i, y_i, \omega)$$

其中  $b$  是批量大小 `batch size`，也是超参数。

## 四、选择批量大小

- $b$  也不能太大：内存消耗增加；浪费计算资源，一个极端的情况是可能会重复选取很多差不多的样本，浪费计算资源
- $b$  也不能太小：每次计算量太小，很难以并行，不能最大限度利用 GPU 资源

## 五、总结

- 梯度下降通过不断沿着负梯度方向更新参数求解
- 小批量随机梯度下降是深度学习默认的求解算法（简单，稳定）
- 两个重要的超参数：批量大小（`batch size`），学习率（lr）

### 2.1.3 线性回归的从零开始实现

#### 00:00 代码实现

在本节中，我们将介绍如何(通过使用深度学习框架来简洁地实现)  
`:numref:sec_linear_scratch`中的(线性回归模型)。

## 一、环境配置

```
!pip install d2l
import numpy as np
import torch
from torch.utils import data
from d2l import torch as d2l
```

language=python

**Note:** 笔者是 Mac-M1 系统，需要注意的是，此处推荐使用 Python 3.9 环境，如果使用 3.10 版本会报如下错误：

```
ValueError Traceback (most recent call last)
Input In [1], in <cell line: 4>()
1 #import torch
2 #print(torch.__version__)
----> 4 from d2l import torch as d2l

...
----> 13 from pandas._libs.interval import Interval
14 from pandas._libs.tslibs import (
15 NaT,
16 NaTType,
17 ...
21 iNaT,
22 )

File pandas/_libs/interval.pyx 1 in init pandas._libs.interval()

ValueError: numpy.ndarray size changed, may indicate binary incompatibility. Expected 96 from C
header, got 88 from PyObject
```

language=python

解决办法：重装pandas

```
pip install --force-reinstall pandas
```

language=shell

只需要重新安装 miniconda 和 python 版本就好，步骤：

- 从 [官网](#) 下载 3.9 版本的 miniconda，然后运行：

```
# execute the following at the download location:
sh Miniconda3-py39_23.1.0-1-MacOSX-arm64.sh -b
```

language=shell

- 初始化 miniconda：

```
# initiate the shell
conda activate ~/miniconda3
```

language=shell

或者（二选一）：

```
~/miniconda3/bin/conda init
```

language=shell

- 创建虚拟环境

```
conda create --name d2l python=3.9 -y
```

language=shell

- 激活虚拟环境

```
conda activate d2l
```

language-shell

- 下载 `torch-gpu` 版本（选）

考虑到后续可能需要 `gpu` 加速训练，因此此处直接下载 `torch-gpu` 版本。

```
pip install --pre torch torchvision torchaudio --extra-index-url  
https://download.pytorch.org/whl/nightly/cpu
```

language-shell

**Note:** 检查是否安装成功，首先在命令行输入 `python` 进入 `python` 编程环境，然后输入

```
>>> import torch  
>>> print(torch.backends.mps.is_available())  
  
True
```

language-python

若输出结果为 `True`，则表明安装成功。

- 下载 `ipykernel`

考虑到书中给的代码运行环境是 `jupyter`，应该此处安装 `ipykernel`，以确保 `jupyter-notebook` 中可以使用刚刚安装的 `d2l` 虚拟环境的内核。

```
pip install ipykernel  
  
python -m ipykernel install --user --name ENVNAME --display-name DISP_NAME
```

language-shell

**Note:** `ENVNAME` 和 `DISP_NAME` 分别为虚拟环境的名字

（此处为 `d2l`）和想要显示的名字。为了简便，此处将两个字段都设置为 `d2l`。

## 二、生成数据集

为了简单起见，我们将根据带有噪声的线性模型构造一个人造数据集。

任务：使用这个有限样本的数据集来恢复这个模型的参数。我们将使用低维数据，这样可以很容易地将其可视化。

在下面的代码中，我们生成一个包含 1000 个样本的数据集，每个样本包含从标准正态分布中采样的 2 个特征。我们的合成数据集是一个矩阵  $\mathbf{X} \in \mathbb{R}^{1000 \times 2}$ 。

使用线性模型参数  $\mathbf{w} = [2, -3.4]^\top$ 、 $b = 4.2$  和噪声项  $\epsilon$  生成数据集及其标签：

$$\mathbf{y} = \mathbf{X}\mathbf{w} + b + \epsilon$$

可以将  $\epsilon$  视为模型预测和标签时的潜在观测误差。在这里我们认为标准假设成立，即  $\epsilon$  服从均值为 0 的正态分布。为了简化问题，我们将标准差设为 0.01。下面的代码生成合成数据集。

```
def synthetic_data(w, b, num_examples):    #@save  
    """  
    生成y=Xw+b+噪声  
    @para w 权重  
    @para b 偏差  
    @para num_examples 样本数量  
    @return  
        X 随机生成的特征数据, (num_examples, len(w))  
        y X对应的标签 (num_examples, 1)  
    """
```

language-python

```
"""
X = torch.normal(0, 1, (num_examples, len(w)))#生成均值为0, 方差为1, 数据维度是 (num_examples,
len(w)) 的随机数据作为训练样本
y = torch.matmul(X, w) + b #生成X对应的预测值
y += torch.normal(0, 0.01, y.shape)# 加入噪音, 加入的是均值为0, 方差为0.01, 纬度和y.shape一致的噪音进行干
扰
return X, y.reshape(-1, 1)#返回X, y, y为列向量
```

```
true_w = torch.tensor([2, -3.4]) #真实权重
true_b = 4.2 #真实偏差
features, labels = synthetic_data(true_w, true_b, 1000) #随机生成1000组训练数据及标签
```

language=python

注意: `features` 中的每一行都包含一个二维数据样本, `labels` 中的每一行都包含一维标签值 (一个标量)。

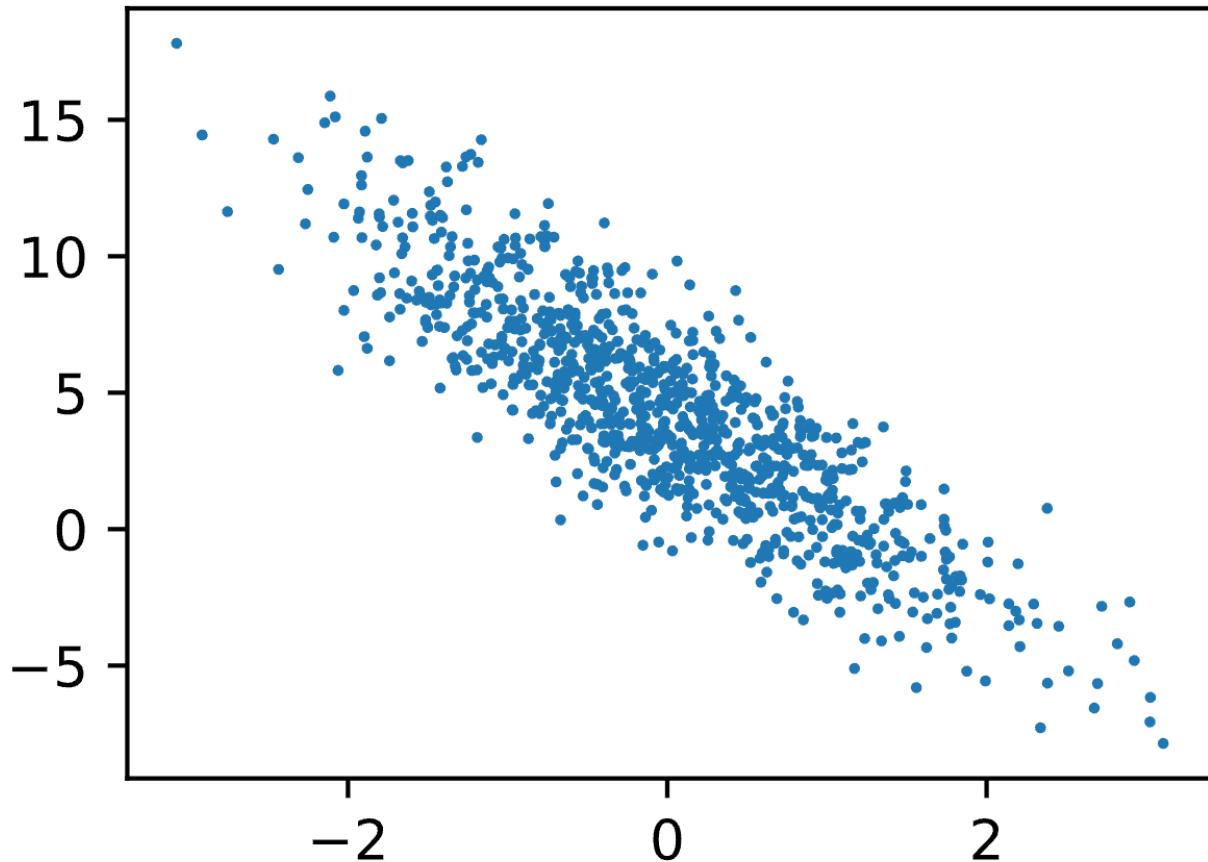
```
>>> print('features:', features[0], '\nlabel:', labels[0]) #打印一个样本数据和对应标签 language=python
features: tensor([-1.1678, -0.7740])
label: tensor([4.5072])
```

通过生成第二个特征 `features[:, 1]` 和 `labels` 的散点图, 可以直观观察到两者之间的线性关系。

```
#可视化数据
d2l.set_figsize()
d2l.plt.scatter(features[:, 1].detach().numpy(), labels.detach().numpy(), 1); #x轴为features的第一
列, y轴为标签值, 正相关
```

language=python

Results:



### 三、读取数据集

训练模型时要对数据集进行遍历，每次抽取一小批量样本，并使用它们来更新我们的模型。由于这个过程是训练机器学习算法的基础，所以有必要定义一个函数，该函数能打乱数据集中的样本并以小批量方式获取数据。

在下面的代码中，**定义了一个**data\_iter**函数**，  
**该函数接收批量大小、特征矩阵和标签向量作为输入，生成大小为**batch\_size**的小批量**。  
每个小批量包含一组特征和标签。

```
def data_iter batch_size  features, labels:
    """
    随机获取一小批样本的数据
    @para batch_size 批量的大小
    @para features 训练数据
    @para labels 训练数据对应的标签
    @return
        迭代器，每次返回batch_size大小的两组数据，一个是训练样本，一个是对应的标签
    """

    num_examples = len(features) #获取样本大小
    indices = list(range(num_examples)) #获取样本脚标的list
    # 这些样本是随机读取的，没有特定的顺序
    random.shuffle(indices) #随机变换indices
    for i in range(0, num_examples, batch_size): #开始循环
        batch_indices = torch.tensor(indices[i: min(i + batch_size, num_examples)]) #有可能不能整除，取i + batch_size和num_examples的较小值
        yield features[batch_indices], labels[batch_indices] #相当于是一个迭代器，每次返回batch_size个样本
```

通常，我们利用**GPU**并行运算的优势，处理合理大小的“小批量”。每个样本都可以并行地进行模型计算，且每个样本损失函数的梯度也可以被并行计算。**GPU**可以在处理几百个样本时，所花费的时间不比处理一个样本时多太多。

我们直观感受一下小批量运算：读取第一个小批量数据样本并打印。每个批量的特征维度显示批量大小和输入特征数。同样的，批量的标签形状与**batch\_size**相等。

```
batch_size = 10

for X, y in data_iter batch_size, features, labels):
    print(X, '\n', y) #X为10 x 2的tensor, y为10 x 1的tensor
    break

tensor([[ 0.6301, -0.3610],
       [-1.0436, -0.4737],
       [-0.2955,  0.2614],
       [-0.1482, -0.0150],
       [-0.7068,  1.1198],
       [ 1.3959,  0.2696],
       [ 0.9270,  0.1575],
       [-0.6020,  0.4045],
       [-1.8370,  0.6234],
       [ 0.1756,  0.0142]])
tensor([[ 6.6870],
       [ 3.7135],
       [ 2.7083],
       [ 3.9501],
       [-1.0375],
       [ 6.0819],
       [ 5.5011],
       [ 1.6290],
       [-1.5751],
       [ 4.5169]])
```

当我们运行迭代时，我们会连续地获得不同的小批量，直至遍历完整个数据集。上面实现的迭代对于教学来说很好，但它的执行效率很低，可能会在实际问题上陷入麻烦。

例如，它要求我们将所有数据加载到内存中，并执行大量的随机内存访问。在深度学习框架中实现的内置迭代器效率要高得多，它可以处理存储在文件中的数据和数据流提供的数据。

## 四、初始化模型参数

在我们开始用小批量随机梯度下降优化我们的模型参数之前，需要先有一些参数。

在下面的代码中，我们通过从均值为0、标准差为0.01的正态分布中采样随机数来初始化权重，并将偏置初始化为0。

```
# 由于训练的时候需要更细参数，计算梯度，所以requires_grad=True  
w = torch.normal(0, 0.01, size=(2, 1), requires_grad=True) #w初始化为均值为0，方差为0.001的符合正态分布的数  
组，纬度为2 x 1  
b = torch.zeros(1, requires_grad=True) #b初始化为0，纬度为1，就是一个实数
```

language=python

在初始化参数之后，需要更新这些参数，直到这些参数足够拟合我们的数据。

每次更新都需要计算损失函数关于模型参数的梯度。有了这个梯度，我们就可以向减小损失的方向更新每个参数。因为手动计算梯度很枯燥而且容易出错，所以没有人会手动计算梯度。我们引入自动微分来计算梯度。

## 五、定义模型

接下来，我们必须 定义模型，将模型的输入和参数同模型的输出关联起来。

回想一下，要计算线性模型的输出，

我们只需计算输入特征 $\mathbf{X}$ 和模型权重 $\mathbf{w}$ 的矩阵-向量乘法后加上偏置 $b$ 。

注意，上面的 $\mathbf{X}\mathbf{w}$ 是一个向量，而 $b$ 是一个标量。

回想一下 [数据操作与数据预处理](#) 中描述的广播机制：

当我们用一个向量加一个标量时，标量会被加到向量的每个分量上。

```
def linreg(X, w, b): #@save  
    """  
    线性回归模型  
    @param X 训练数据 (num_examples, len(w))  
    @param w 权重 (2, 1)  
    @param b 偏差 实数  
    @return  
        模型的预估值  
  
    """  
    return torch.matmul(X, w) + b
```

language=python

## 六、定义损失函数

因为需要计算损失函数的梯度，所以我们应该先定义损失函数。这里使用 [线性回归](#) 中描述的平方损失函数。

**Note:** 在实现中，我们需要将真实值 $y$ 的形状转换为和预测值 $y_{\text{hat}}$ 的形状相同。

```
def squared_loss(y_hat, y): #@save  
    """  
    均方损失  
    @param y_hat 训练数据的真实值 (num, 1)  
    @param y 训练数据的预测值  
    @return  
        均方误差，没有除以样本数目 (batch_size, 1)  
    """  
    return (y_hat - y.reshape(y_hat.shape)) ** 2 / 2
```

language=python

## 七、定义优化算法

正如我们在[线性回归](#)中讨论的，尽管线性回归有解析解，但本书中的其他模型却没有。这里我们介绍小批量随机梯度下降。

- 在每一步中，使用从数据集中随机抽取的一个小批量，然后根据参数计算损失的梯度。
- 接下来，朝着减少损失的方向更新我们的参数。

下面的函数实现小批量随机梯度下降更新。该函数接受模型参数集合、学习速率和批量大小作为输入。每一步更新的大小由学习速率`lr`决定。因为我们计算的损失是一个批量样本的总和，所以我们用批量大小（`batch_size`）来规范化步长，这样步长大小就不会取决于我们对批量大小的选择。

```
def sgd(params, lr, batch_size):    #@save
    """
    小批量随机梯度下降
    @param params 参数
    @param lr 学习率，人为指定
    @param batch_size 批量大小
    @return
    """
    with torch.no_grad(): #不需要计算梯度
        for param in params:
            param -= lr * param.grad / batch_size #梯度下降法更新参数
            param.grad.zero_() #手动梯度归零
```

language=python

## 八、训练

现在我们已经准备好了模型训练所有需要的要素，可以实现主要的[训练过程](#)部分了。

理解这段代码至关重要，因为从事深度学习后，

你会一遍又一遍地看到几乎相同的训练过程。

在每次迭代中，我们读取一小批量训练样本，并通过我们的模型来获得一组预测。

计算完损失后，我们开始反向传播，存储每个参数的梯度。

最后，我们调用优化算法`sgd`来更新模型参数。

概括一下，我们将执行以下循环：

- 初始化参数
- 重复以下训练，直到完成
  - 计算梯度  $\mathbf{g} \leftarrow \partial_{(\mathbf{w}, b)} \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} l(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, \mathbf{w}, b)$
  - 更新参数  $(\mathbf{w}, b) \leftarrow (\mathbf{w}, b) - \eta \mathbf{g}$

在每个[迭代周期](#)（epoch）中，我们使用`data_iter`函数遍历整个数据集，

并将训练数据集中所有样本都使用一次（假设样本数能够被批量大小整除）。

这里的迭代周期个数`num_epochs`和学习率`lr`都是超参数，分别设为3和0.03。

设置超参数很棘手，需要通过反复试验进行调整。

我们现在忽略这些细节，以后会在`:numref:chap_optimization`中详细介绍。

```
lr = 0.003 #学习率
num_epochs = 3 #训练次数
net = linreg #网络，之前定义的线性网络
loss = squared_loss #损失函数，之前定义的平方损失函数
```

language=python

```
#开始训练
for epoch in range(num_epochs):
    for X, y in data_iter(batch_size, features, labels):
        l = loss(net(X, w, b), y) # X和y的小批量损失
        # 因为l形状是(batch_size, 1)，而不是一个标量。l中的所有元素被加到一起，
        # 并以此计算关于[w, b]的梯度
        l.sum().backward()
```

language=python

```
sgd([w, b], lr, batch_size) # 使用参数的梯度更新参数
with torch.no_grad():
    train_l = loss(net(features, w, b), labels) #计算所有样本的损失函数
    print(f'epoch {epoch + 1}, loss {float(train_l.mean()):f}')
```

```
epoch 1, loss 9.117679
epoch 2, loss 4.993863
epoch 3, loss 2.736007
```

因为我们使用的是自己合成的数据集，所以我们知道真正的参数是什么。因此，可以通过 **比较真实参数和通过训练学到的参数来评估训练的成功程度**。

```
print(f'w的估计误差: {true_w - w.reshape(true_w.shape)}')
print(f'b的估计误差: {true_b - b}')
```

language=python

Results:

```
w的估计误差: tensor([-0.7622, -1.4047], grad_fn=<SubBackward0>)
b的估计误差: tensor([1.7077], grad_fn=<RsubBackward1>)
```

language=python

可以发现，真实参数和通过训练学到的参数确实非常接近。

**注：**我们不应该想当然地认为能够完美地求解参数。在机器学习中，我们通常不太关心恢复真正的参数，而更关心如何高度准确预测参数。幸运的是，即使是在复杂的优化问题上，随机梯度下降通常也能找到非常好的解。其中一个原因是，在深度网络中存在许多参数组合能够实现高度精确的预测。

## 九、小结

- 我们学习了深度网络是如何实现和优化的。在这一过程中只使用张量和自动微分，不需要定义层或复杂的优化器。
- 这一节只触及到了表面知识。在下面的部分中，我们将基于刚刚介绍的概念描述其他模型，并学习如何更简洁地实现其他模型。

### 2.1.4 调用 API 实现

```
true_w = torch.tensor([2, -3.4]) # 真实权重
true_b = 4.2 # 真实偏差
features, labels = d2l.synthetic_data(true_w, true_b, 1000)
# 应用linear-regression-concise里面的函数，生成训练数据

features
```

language=python

Results:

```
tensor([[[-1.4684,  0.0977],
        [ 1.7930,  0.9657],
        [ 1.1813,  2.1203],
        ...,
        [ 0.0698,  0.6992],
        [-1.6323,  2.0169],
        [-0.4198, -1.1236]])
```

language=python

## 二、读取数据集

可以 **调用框架中现有的API来读取数据**。

我们将 `features` 和 `labels` 作为API的参数传递，并通过数据迭代器指定 `batch_size`。  
此外，布尔值 `is_train` 表示是否希望数据迭代器对象在每个迭代周期内打乱数据。

```
def load_array data_arrays, batch_size, is_train=True): # @save
    """
    构造一个PyTorch数据迭代器
    @param data_arrays 训练数据
    @param batch_size 批量大小
    @param is_train=True 是否训练, 选择True会随机选择数据
    @return
        迭代器, 每次返回batch_size大小的两组数据, 一个是训练样本, 一个是对应的标签
    """
    dataset = data.TensorDataset(*data_arrays) # 数据集
    return data.DataLoader(dataset, batch_size, shuffle=is_train)
```

language=python

**Note:** dataloader可以理解为数据的一个接口, 说明见 [PyTorch中的Data.DataLoader](#),

- 其构造函数为:

```
class torch.utils.data.DataLoader dataset, batch_size=1, shuffle=False, sampler=None,
    batch_sampler=None, num_workers=0, collate_fn=<function default_collate>,
    pin_memory=False, drop_last=False, timeout=0, worker_init_fn=None)
```

- DataLoader** 的参数说明:

- `dataset (Dataset)`: 定义要加载的数据集;
- `batch_size (int, optional)`: 定义`batch_size` 大小, 也就是一次加载样本的数量, 默认是 1;
- `shuffle (bool, optional)`: 在每个epoch开始的时候, 是否进行数据重排序, 默认 False;
- `sampler (Sampler, optional)`: 定义从数据集中取样本的策略, 如果进行了指定, 那么 `shuffle` 必须是 False。
- `num_workers (int, optional)`: 定义加载数据使用的进程数, 0 代表所有的数据都被装载进主进程, 默认是 0。
- `drop_last (bool, optional)`: 这个是对最后的未完成的batch来说的, 比如你的 `batch_size` 设置为 64, 而一个 epoch 只有 100 个样本, (如果设置为True), 那么训练的时候后面的 36 个就被扔掉了。如果为 False (默认), 那么会继续正常执行, 只是最后的 `batch_size` 会小一点。

`Dataloader` 的处理逻辑是先通过 `Dataset` 类里面的 `__getitem__` 函数获取单个的数据, 然后组合成batch, 再使用 `collate_fn` 所指定的函数对这个batch做一些操作

```
batch_size = 10 # 批量大小
data_iter = load_array((features, labels), batch_size) # 随机返回的数据
```

language=python

使用`data_iter`的方式与我们在 :numref:`sec\_linear\_scratch`中使用`data_iter`函数的方式相同。为了验证是否正常工作, 让我们读取并打印第一个小批量样本。

与 :numref:`sec\_linear\_scratch`不同, 这里我们使用`iter`构造Python迭代器, 并使用`next`从迭代器中获取第一项。

```
next(iter(data_iter))# iter构造Python迭代器, 并使用next从迭代器中获取第一项。
```

language=python

```
tensor([[-0.4879,  0.5828],
       [-0.2912, -0.0160],
       [-2.0002,  2.0359],
       [-0.3812,  1.3547],
       [ 2.4167, -0.4777],
       [ 0.3231, -1.6874],
       [ 1.1544,  2.3415],
       [ 0.1994, -0.4488],
       [ 0.4064, -0.7453],
       [ 0.6125,  0.6447]]),
tensor([[ 1.2289,
         [ 3.6639],
         [-6.7296],
```

language=python

```
[ -1.1639 ,  
[ 10.6570 ,  
[ 10.5763 ,  
[ -1.4631 ,  
[ 6.1100 ,  
[ 7.5428 ,  
[ 3.2241 ] ) ] ]
```

### 三、定义模型

当我们在 :numref:`sec\_linear\_scratch` 中实现线性回归时，

我们明确定义了模型参数变量，并编写了计算的代码，这样通过基本的线性代数运算得到输出。

但是，如果模型变得更加复杂，且当你几乎每天都需要实现模型时，你会想简化这个过程。

这种情况类似于为自己的博客从零开始编写网页。

做一两次是有益的，但如果每个新博客你就花一个月的时间重新开始编写网页，那并不高效。

对于标准深度学习模型，我们可以[使用框架的预定义好的层]。这使我们只需关注使用哪些层来构造模型，而不必关注层的实现细节。

我们首先定义一个模型变量 `net`，它是一个 `Sequential` 类的实例。

`Sequential` 类将多个层串联在一起。

当给定输入数据时，`Sequential` 实例将数据传入到第一层，

然后将第一层的输出作为第二层的输入，以此类推。

在下面的例子中，我们的模型只包含一个层，因此实际上不需要 `Sequential`。

但是由于以后几乎所有的模型都是多层的，在这里使用 `Sequential` 会让你熟悉“标准的流水线”。

回顾 :numref:`fig\_single\_neuron` 中的单层网络架构，

这一单层被称为 **全连接层** (fully-connected layer) ，

因为它的每一个输入都通过矩阵-向量乘法得到它的每个输出。

在 PyTorch 中，全连接层在 `Linear` 类中定义。

值得注意的是，我们将两个参数传递到 `nn.Linear` 中。

第一个指定输入特征形状，即 2，第二个指定输出特征形状，输出特征形状为单个标量，因此为 1。

```
# nn 是神经网络的缩写  
from torch import nn  
  
# Sequential一个容器, list of layers  
net = nn.Sequential(nn.Linear(2, 1)) #nn.Linear为线性层函数, 输入的纬度2; 输出的纬度1,
```

language=python

## (初始化模型参数)

在使用 `net` 之前，我们需要初始化模型参数。

如在线性回归模型中的权重和偏置。

深度学习框架通常有预定义的方法来初始化参数。

在这里，我们指定每个权重参数应该从均值为 0、标准差为 0.01 的正态分布中随机采样，偏置参数将初始化为零。

正如我们在构造 `nn.Linear` 时指定输入和输出尺寸一样，

现在我们能直接访问参数以设定它们的初始值。

我们通过 `net[0]` 选择网络中的第一个图层，

然后使用 `weight.data` 和 `bias.data` 方法访问参数。

我们还可以使用替换方法 `normal_` 和 `fill_` 来重写参数值。

```
net[0].weight.data.normal_(0, 0.01) #权重初始化, 用normal方式  
net[0].bias.data.fill_(0) #初始化偏差为0
```

language=python

```
tensor([0.])
```

## 定义损失函数

[计算均方误差使用的是 `MSELoss` 类，也称为平方  $L_2$  范数]。

默认情况下，它返回所有样本损失的平均值。

```
loss = nn.MSELoss() #均方误差函数
```

language=python

## 定义优化算法

小批量随机梯度下降算法是一种优化神经网络的标准工具，

PyTorch在 `optim` 模块中实现了该算法的许多变种。

当我们(实例化一个 `SGD` 实例)时，我们要指定优化的参数

(可通过`net.parameters()`从我们的模型中获得) 以及优化算法所需的超参数字典。

小批量随机梯度下降只需要设置 `lr` 值，这里设置为 0.03。

```
trainer = torch.optim.SGD(net.parameters(), lr=0.03) #随机梯度下降优化算法SGD, 传入模型的
```

language=python

## 训练

通过深度学习框架的高级 API 来实现我们的模型只需要相对较少的代码。

我们不必单独分配参数、不必定义我们的损失函数，也不必手动实现小批量随机梯度下降。

当我们需要更复杂的模型时，高级 API 的优势将大大增加。

当我们有了所有的基本组件，[训练过程代码与我们从零开始实现时所做的非常相似]。

回顾一下：在每个迭代周期里，我们将完整遍历一次数据集 (`train_data`)，

不停地从中获取一个小批量的输入和相应的标签。

对于每一个小批量，我们会进行以下步骤：

- 通过调用 `net(X)` 生成预测并计算损失 `l` (前向传播)。
- 通过进行反向传播来计算梯度。
- 通过调用优化器来更新模型参数。

为了更好的衡量训练效果，我们计算每个迭代周期后的损失，并打印它来监控训练过程。

```
num_epochs = 3 #迭代周期为3
for epoch in range(num_epochs):
    for X, y in data_iter: #返回batch_size大小的数据
        l = loss(net(X), y) #计算损失函数
        trainer.zero_grad() #梯度清零
        l.backward() #计算梯度
        trainer.step() #更新参数
    l = loss(net(features), labels) #计算所有样本的loss
    print(f'epoch {epoch + 1}, loss {l:f}')
```

language=python

```
epoch 1, loss 0.000239
epoch 2, loss 0.000095
epoch 3, loss 0.000096
```

下面我们[比较生成数据集的真实参数和通过有限数据训练获得的模型参数]。

要访问参数，我们首先从 `net` 访问所需的层，然后读取该层的权重和偏置。

正如在从零开始实现中一样，我们估计得到的参数与生成数据的真实参数非常接近。

```
w = net[0].weight.data
print('w的估计误差: ', true_w - w.reshape(true_w.shape))
```

language=python

```
b = net[0].bias.data  
print('b的估计误差: ', true_b - b)
```

```
w的估计误差: tensor([0.0002, 0.0007])  
b的估计误差: tensor([-7.1526e-06])
```

## 小结

- 我们可以使用PyTorch的高级API更简洁地实现模型。
- 在PyTorch中，`data`模块提供了数据处理工具，`nn`模块定义了大量的神经网络层和常见损失函数。
- 我们可以通过`.zero_()`方法将参数替换，从而初始化参数。

## 练习

- 如果将小批量的总损失替换为小批量损失的平均值，你需要如何更改学习率？
- 查看深度学习框架文档，它们提供了哪些损失函数和初始化方法？用Huber损失代替原损失，即

$$l(y, y') = \begin{cases} |y - y'| - \frac{\sigma}{2} & \text{if } |y - y'| > \sigma \\ \frac{1}{2\sigma}(y - y')^2 & \text{其它情况} \end{cases}$$

- 你如何访问线性回归的梯度？

### Discussions

## 4.新型回归的简洁实现

- 代码

## 5.- 线性回归+基础优化算法

- [1.线性回归](#)
- [2.基础优化算法](#)
- [3.线性回归的从零开始实现](#)
- [4.新型回归的简洁实现](#)

- [5.QA](#)

- 1.为什么使用平方损失而不是绝对差值？**
  - 其实差别不大，最开始使用平方损失是因为它可导，现在其实都可以使用。
- 2.损失为什么要求平均？**
  - 本质上没有关系，但是如果不要求平均，梯度的数值会比较大，这时需要学习率除以n。如果不除以n，可能会随着样本数量的增大而让梯度变得很大。
- 3.不管是梯度下降还是随机梯度下降，怎么找到合适的学习率？**
  - 选择对学习率不敏感的优化方法，比如Adam
  - 合理参数初始化

- 4.训练过程中，过拟合和欠拟合情况下，学习率和batch\_size应该如何调整?
  - 理论上学习率和batch\_size对最后的拟合结果不会有影响
- 5.深度学习上，设置损失函数的时候，需要考虑正则吗?
  - 会考虑，但是和损失函数是分开的，深度学习中正则没有太大的用处，有很多其他的技术可以有正则的效果。
- 6.如果样本大小不是批量数的整数倍，需要随机剔除多余的样本吗?
  - 就取多余的样本作为一个批次
  - 直接丢弃
  - 从下一个epoch里面补少的样本