计算机视觉

张健

https://villa.jianzhang.tech/

信息工程学院
北京大学深圳研究生院

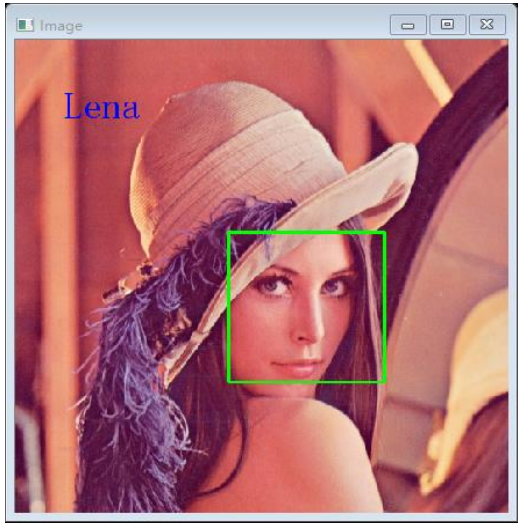
2022.10.12

课程内容

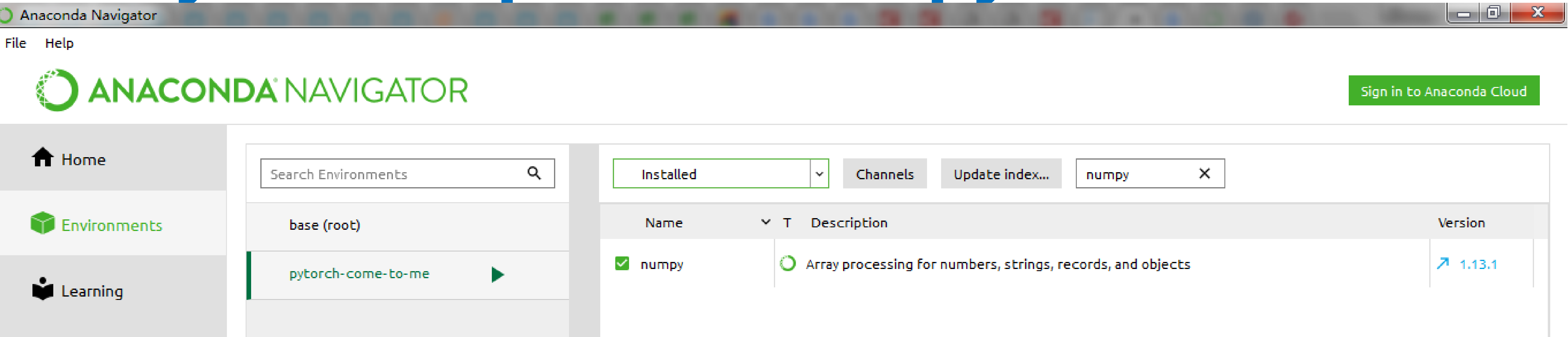
- OpenCV, Numpy
- 矩阵求导基础
- PyTorch基础

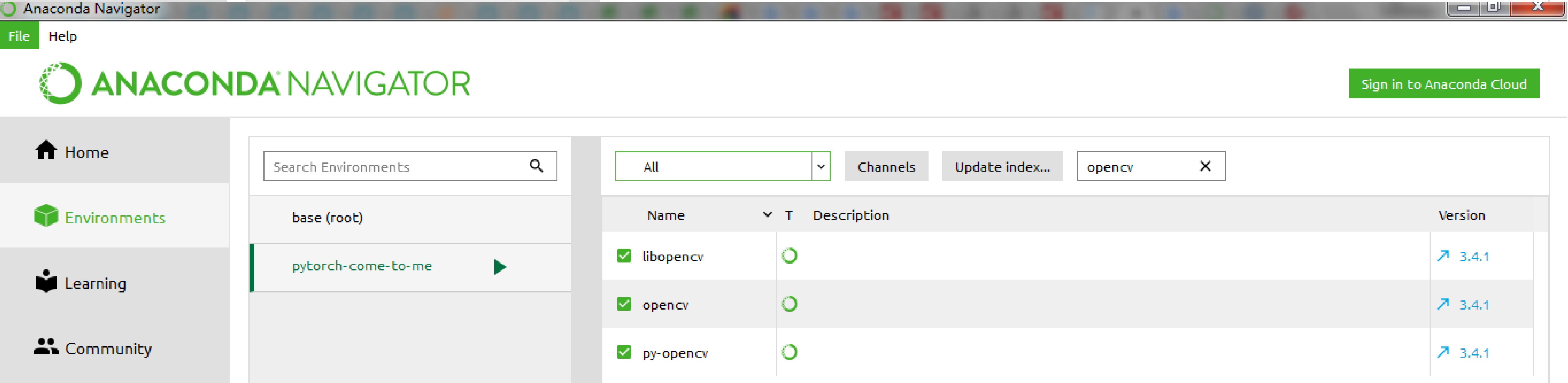
作业情况

- 安装Anaconda
- 安装虚拟环境，删除
- 安装OpenCV, Numpy, Matplotlib
- 学习Numpy和CV2工具包的使用
- 用OpenCV检测人脸
  - 读取图片
  - 调用opencv的cv2.CascadeClassifier检测出人脸位置
  - 画出方框，写上文本，显示图片



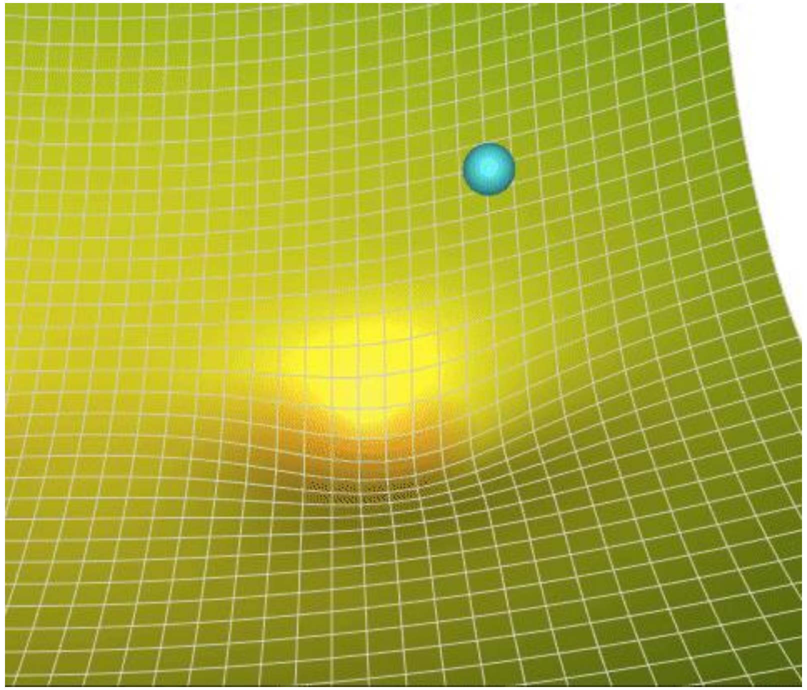
Python (Opencv, Numpy)

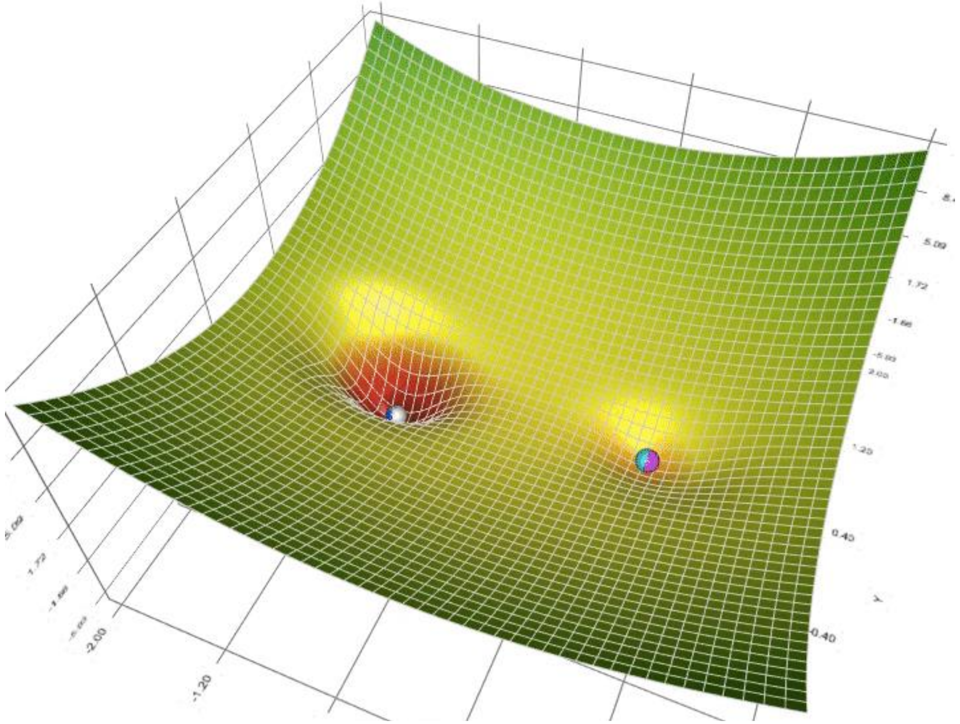




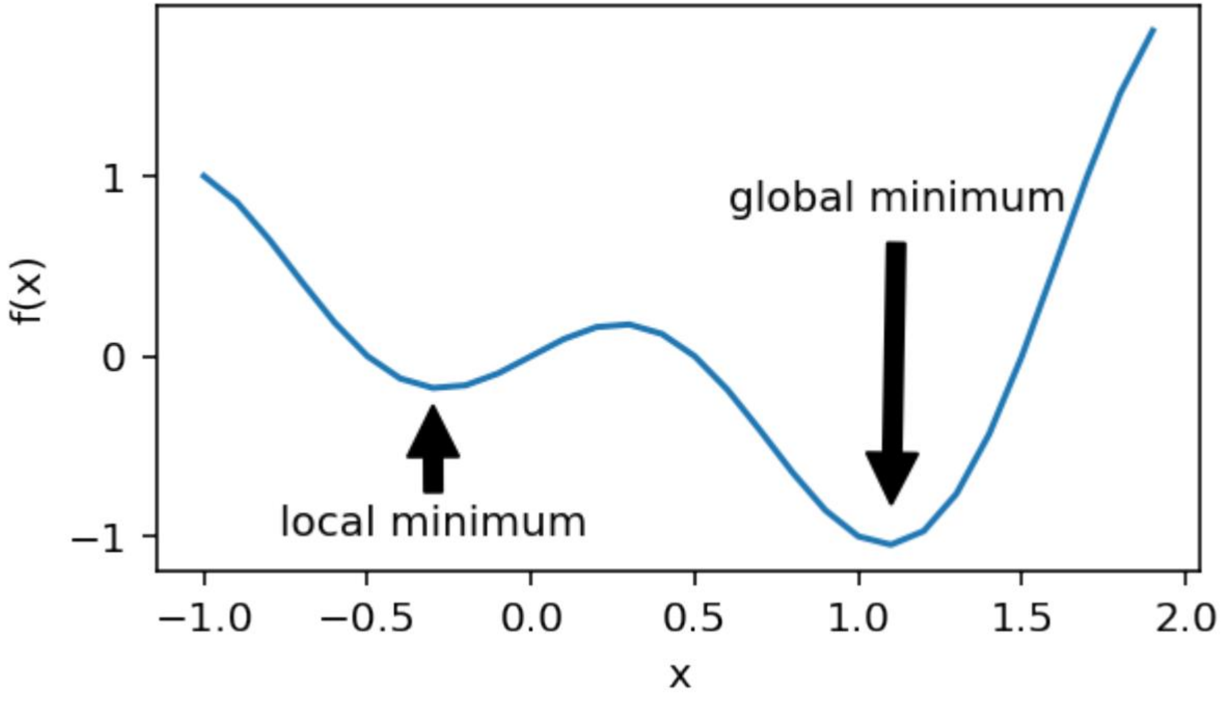
矩阵求导基础

梯度下降





1维举例



$$f(x+\epsilon) \approx f(x) + f'(x)\epsilon.$$

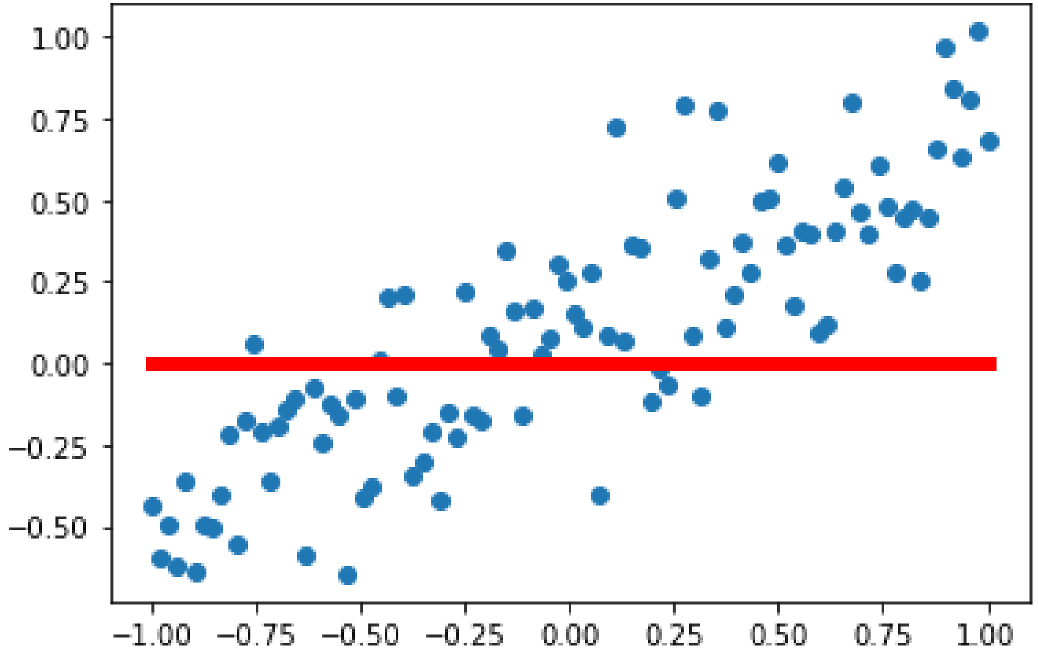
$$f(x-\eta f'(x)) \approx f(x) - \eta f'(x)^2.$$

$$f(x-\eta f'(x)) \leq f(x).$$

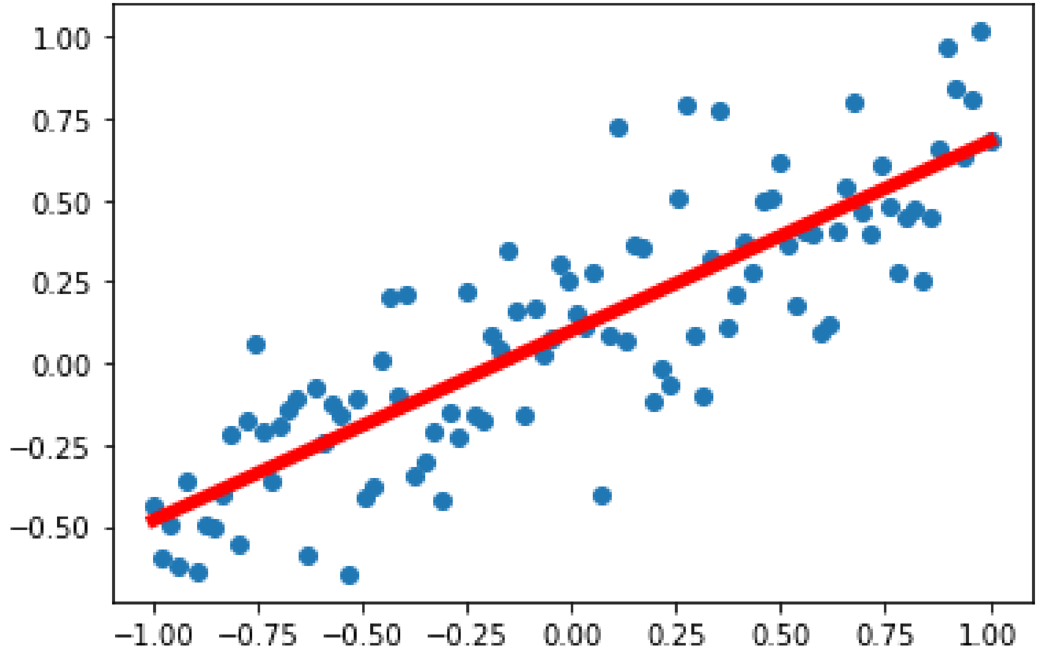
$$x \leftarrow x - \eta f'(x).$$

矩阵求导基础

$$(x_i, y_i)$$

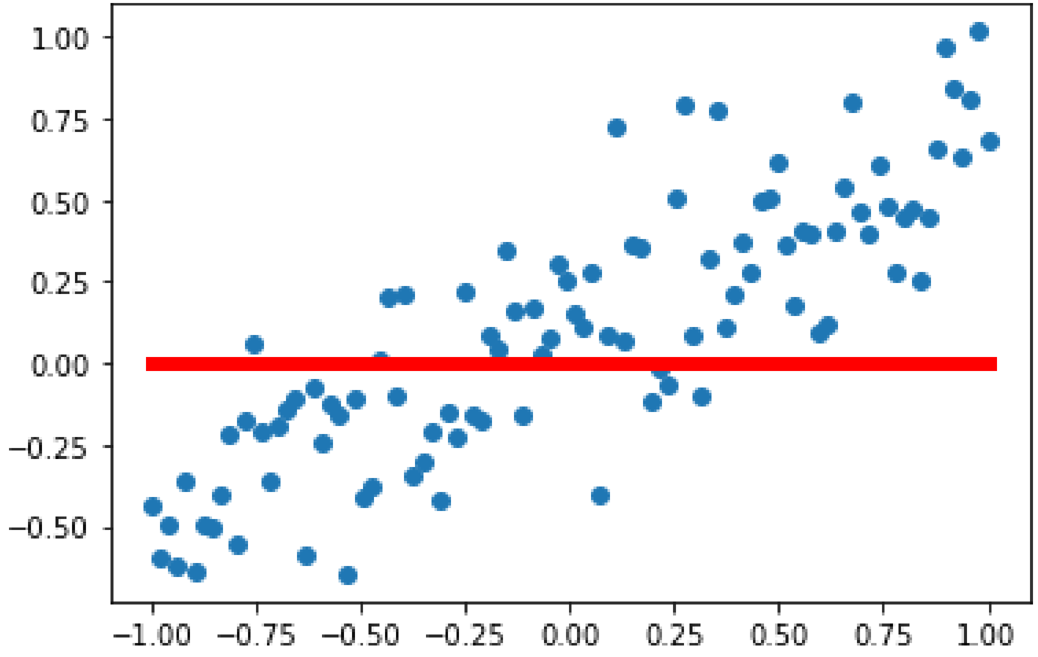


$$y = w^*x + b$$

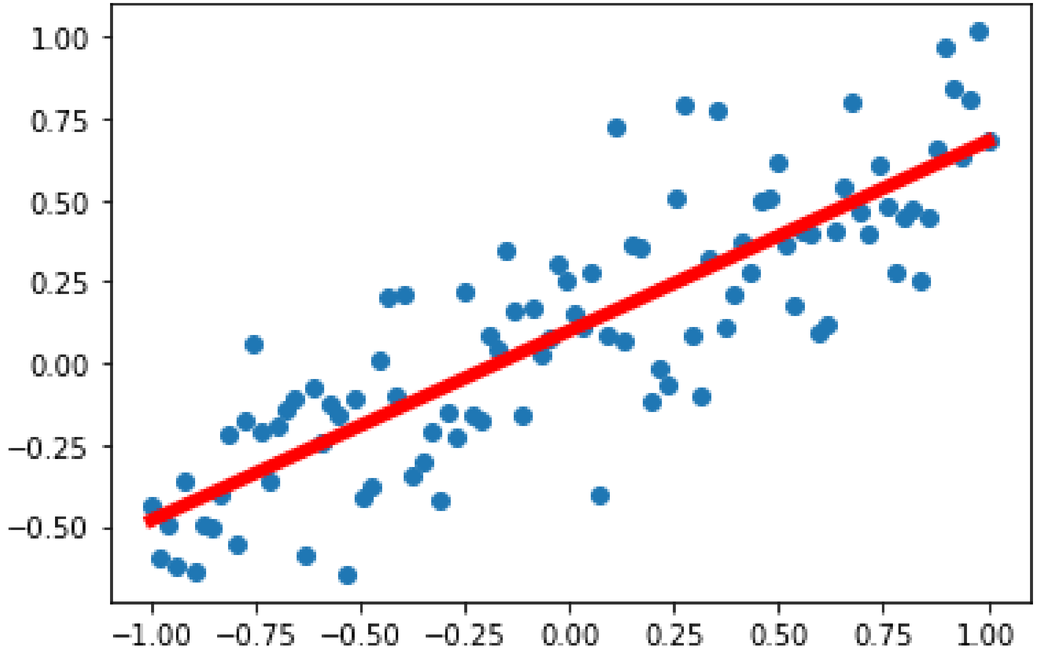


矩阵求导基础

$$(x_i, y_i)$$



$$y = w^* x + b$$

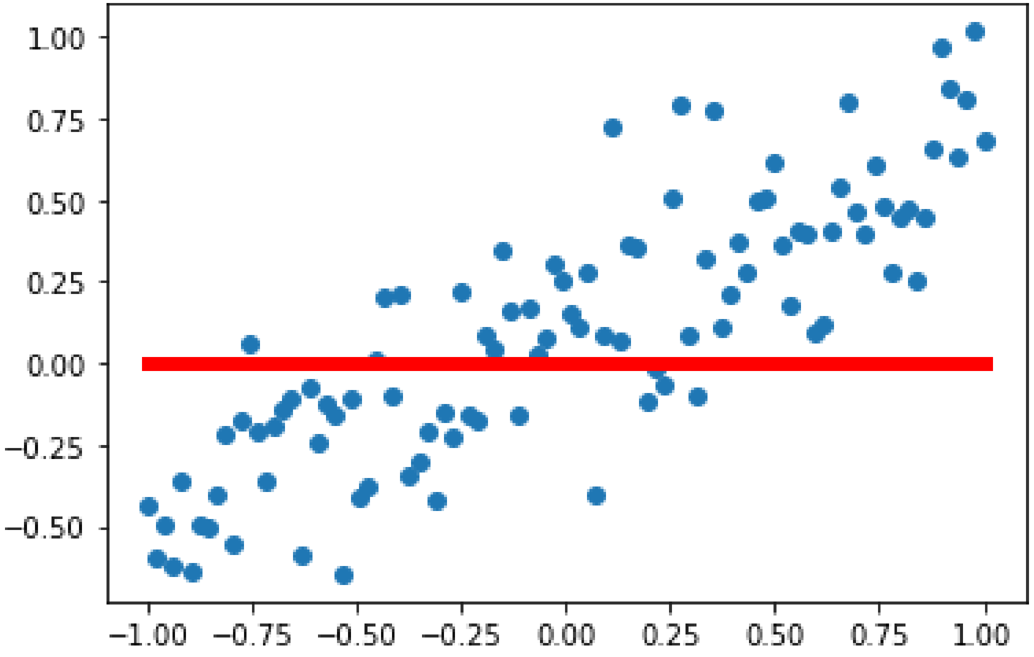


$$loss = \frac{\sum_{i=1}^{N}(w^* x_i + b - y_i)^2}{N}$$

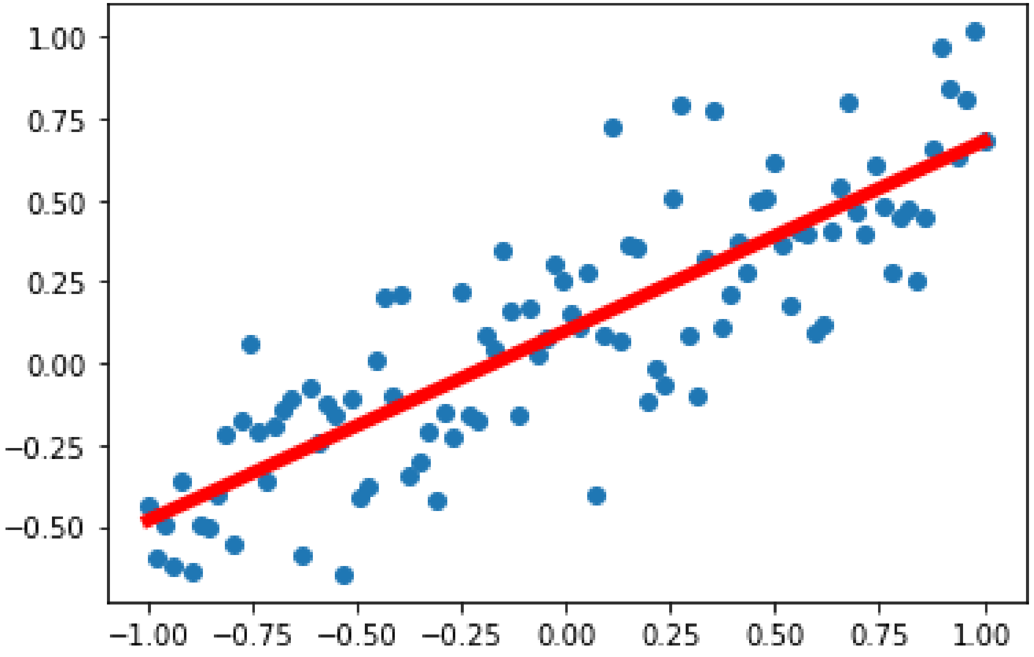
$$w^{(k+1)} = w^{(k)} - \eta^* \triangledown w^{(k)}, b^{(k+1)} = b^{(k)} - \eta^* \triangledown b^{(k)}$$

矩阵求导基础

$$(x_i, y_i)$$



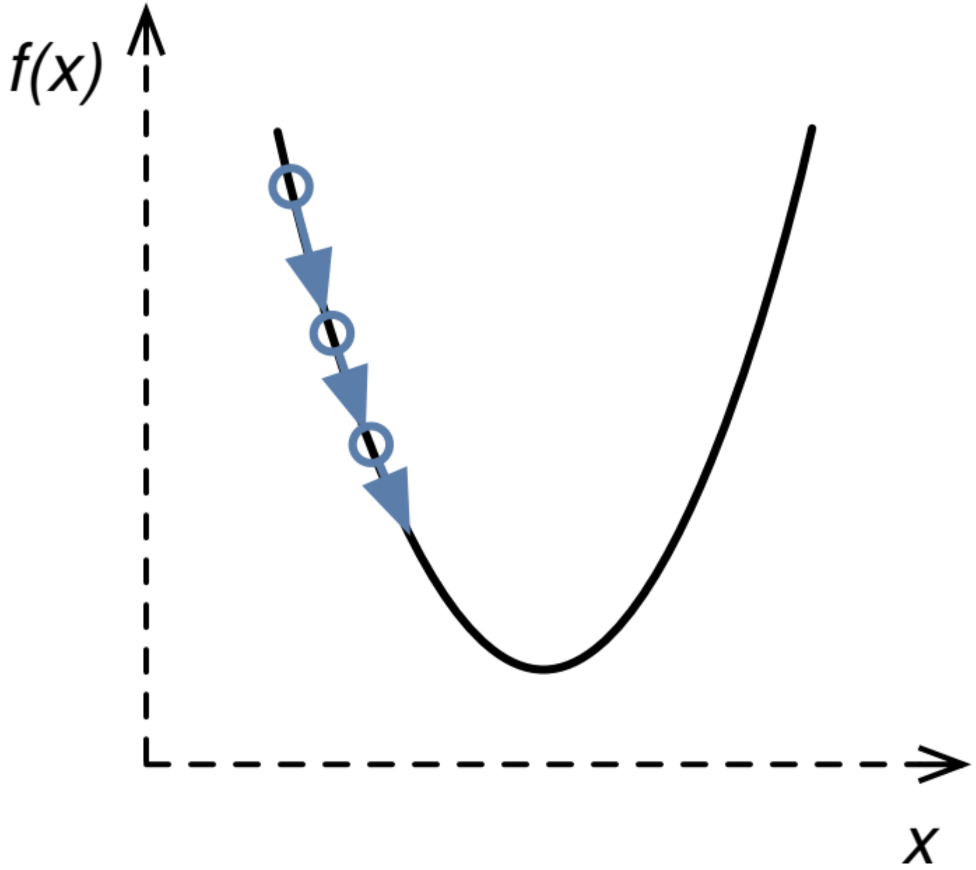
$$y = w^* x + b$$

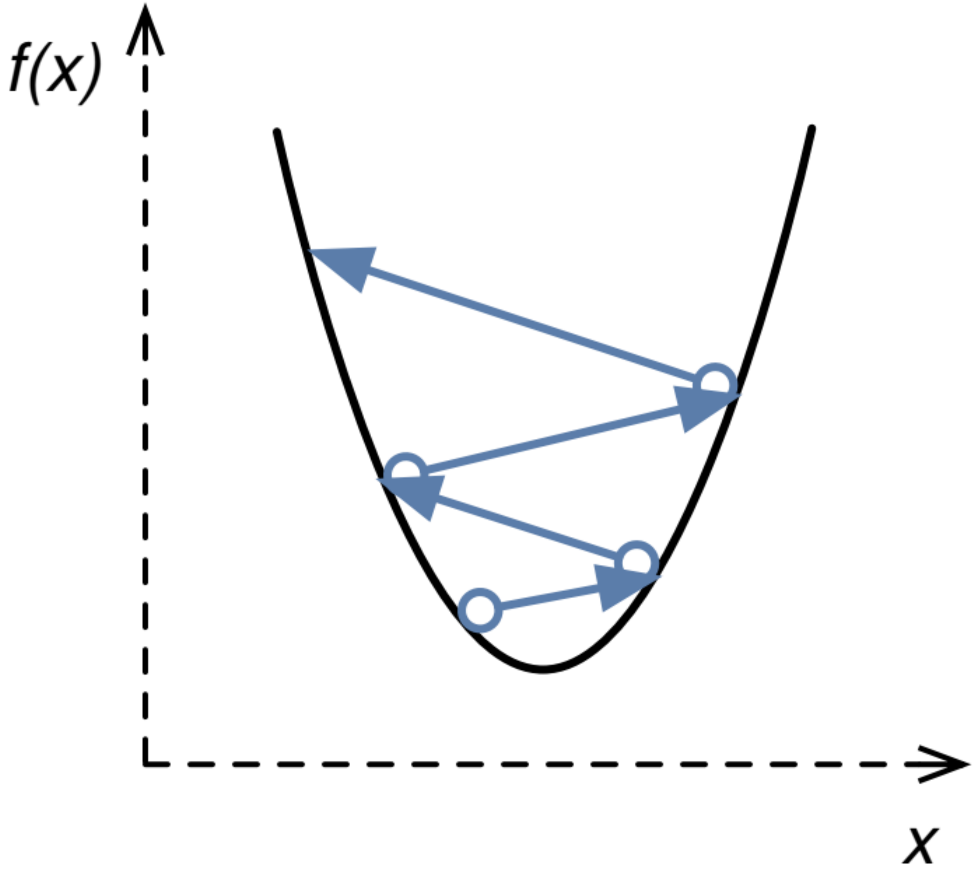


$$loss = \frac{\sum_{i=1}^{N} (w^* x_i + b - y_i)^2}{N} \qquad \nabla w^{(k)} = \frac{\sum 2^* x_i^* (w^{(k)*} x_i + b^{(k)} - y_i)}{N}, \quad \nabla b^{(k)} = \frac{\sum 2^* (w^{(k)*} x_i + b^{(k)} - y_i)}{N}$$

W4_Regression_Python.ipynb

矩阵求导基础





不同的学习率

多维

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \left[\frac{\partial f(\boldsymbol{x})}{\partial x_1}, \frac{\partial f(\boldsymbol{x})}{\partial x_2}, \ldots, \frac{\partial f(\boldsymbol{x})}{\partial x_d}\right]^\top$$

$$\boldsymbol{x} \leftarrow \boldsymbol{x} - \eta \nabla f(\boldsymbol{x})$$

标量f对矩阵X的导数

$$\frac{\partial f}{\partial X} = \left[\frac{\partial f}{\partial X_{ij}}\right]$$

- 定义在计算中并不好用
- 用矩阵运算更整洁
- 要找一个从整体出发的算法

矩阵求导基础

一元微积分中的导数（标量对标量的导数）与微分有联系：

$$df = f'(x)dx$$

多元微积分中的梯度（标量对向量的导数）也与微分有联系：

$$df = \sum_{i=1}^{n} \frac{\partial f}{\partial x_i} dx_i = \frac{\partial f}{\partial \boldsymbol{x}}^T d\boldsymbol{x}$$

第一个等号是全微分公式，第二个等号表达了梯度与微分的联系

全微分 df 是梯度向量 $\frac{\partial f}{\partial \boldsymbol{x}}$ (n×1)与微分向量 $d\boldsymbol{x}$ (n×1)的内积

矩阵求导基础

受前面一元和多元微积分启发，可以将矩阵导数与微分建立联系：

$$df = \sum_{i=1}^{m}\sum_{j=1}^{n}\frac{\partial f}{\partial X_{ij}}dX_{ij} = \operatorname{tr}\left(\frac{\partial f}{\partial X}^{T} dX\right)$$

其中tr代表迹(trace)是方阵对角线元素之和，满足性质：

对尺寸相同的矩阵A,B，$\operatorname{tr}(A^T B) = \sum_{i,j} A_{ij} B_{ij}$ 即 $\operatorname{tr}(A^T B)$ 是矩阵A,B的**内积**

第一个等号是全微分公式，第二个等号表达了矩阵导数与微分的联系：

全微分 df 是导数 $\frac{\partial f}{\partial X}$ (m×n)与微分矩阵 dX (m×n)的内积。

然后通过矩阵微分运算法则可高效快速求解。

矩阵求导基础

常用的矩阵微分的运算法则：

加减法： $d(X \pm Y) = dX \pm dY$ ；矩阵乘法： $d(XY) = (dX)Y + XdY$ ；转置： $d(X^T) = (dX)^T$ ；迹： $d\mathrm{tr}(X) = \mathrm{tr}(dX)$ 。

逐元素乘法： $d(X \odot Y) = dX \odot Y + X \odot dY$ ， $\odot$ 表示尺寸相同的矩阵X,Y逐元素相乘。

逐元素函数： $d\sigma(X) = \sigma'(X) \odot dX$ ， $\sigma(X) = [\sigma(X_{ij})]$ 是逐元素标量函数运算， $\sigma'(X) = [\sigma'(X_{ij})]$ 是逐元素求导数。例如

$$X = \begin{bmatrix} X_{11} & X_{12} \\ X_{21} & X_{22} \end{bmatrix}, d\sin(X) = \begin{bmatrix} \cos X_{11} dX_{11} & \cos X_{12} dX_{12} \\ \cos X_{21} dX_{21} & \cos X_{22} dX_{22} \end{bmatrix} = \cos(X) \odot dX$$

矩阵迹的性质：

1. 标量套上迹： $a = \mathrm{tr}(a)$
2. 转置： $\mathrm{tr}(A^T) = \mathrm{tr}(A)$。
3. 线性： $\mathrm{tr}(A \pm B) = \mathrm{tr}(A) \pm \mathrm{tr}(B)$。
4. 矩阵乘法交换： $\mathrm{tr}(AB) = \mathrm{tr}(BA)$，其中 A 与 B^T 尺寸相同。两侧都等于 $\sum_{i,j} A_{ij} B_{ji}$。
5. 矩阵乘法/逐元素乘法交换： $\mathrm{tr}(A^T (B \odot C)) = \mathrm{tr}((A \odot B)^T C)$，其中 A, B, C 尺寸相同。两侧都等于 $\sum_{i,j} A_{ij} B_{ij} C_{ij}$。

在建立法则的最后，来谈一谈复合：假设已求得 $\frac{\partial f}{\partial Y}$ ，而Y是X的函数，如何求 $\frac{\partial f}{\partial X}$ 呢?

我们直接从微分入手建立复合法则：先写出 $df = \text{tr}\left(\frac{\partial f}{\partial Y}^T dY\right)$ ，再将dY用dX表示出来代入，并使用迹技巧将其他项交换至dX左侧，即可得到 $\frac{\partial f}{\partial X}$ 。

矩阵求导基础

最常见的情形是$Y=AXB$，此时 假设已求得$\dfrac{\partial f}{\partial Y}$，而Y是X的函数，如何求$\dfrac{\partial f}{\partial X}$呢?

最常见的情形是$Y = AXB$，此时 假设已求得$\frac{\partial f}{\partial Y}$，而Y是X的函数，如何求$\frac{\partial f}{\partial X}$呢?

$$df = \operatorname{tr}\left(\frac{\partial f}{\partial Y}^T dY\right) = \operatorname{tr}\left(\frac{\partial f}{\partial Y}^T AdXB\right) = \operatorname{tr}\left(B\frac{\partial f}{\partial Y}^T AdX\right) = \operatorname{tr}\left((A^T\frac{\partial f}{\partial Y}B^T)^T dX\right)$$

，可得到$\frac{\partial f}{\partial X} = A^T\frac{\partial f}{\partial Y}B^T$。注意这里

$dY = (dA)XB + AdXB + AXdB = AdXB$，由于$A, B$是常量，

$dA = 0, dB = 0$，以及我们使用矩阵乘法交换的迹技巧交换了$\frac{\partial f}{\partial Y}^T AdX$与$B$。

矩阵求导基础

例　$f = \boldsymbol{a}^T X \boldsymbol{b}$，求 $\dfrac{\partial f}{\partial X}$。其中 $\boldsymbol{a}$ 是 $m \times 1$ 列向量，X 是 $m \times n$ 矩阵，$\boldsymbol{b}$ 是 $n \times 1$ 列向量，f 是标量。

例 $f=\boldsymbol{a}^T X\boldsymbol{b}$，求 $\dfrac{\partial f}{\partial X}$。其中 $\boldsymbol{a}$ 是 $m\times 1$ 列向量，X 是 $m\times n$ 矩阵，$\boldsymbol{b}$ 是 $n\times 1$ 列向量，f 是标量。

解：先使用矩阵乘法法则求微分，$df=d\boldsymbol{a}^T X\boldsymbol{b}+\boldsymbol{a}^T dX\boldsymbol{b}+\boldsymbol{a}^T Xd\boldsymbol{b}=\boldsymbol{a}^T dX\boldsymbol{b}$，注意这里的 $\boldsymbol{a},\boldsymbol{b}$ 是常量，$d\boldsymbol{a}=\boldsymbol{0},d\boldsymbol{b}=\boldsymbol{0}$。由于df是标量，它的迹等于自身，$df=\operatorname{tr}(df)$，套上迹并做矩阵乘法交换：$df=\operatorname{tr}(\boldsymbol{a}^T dX\boldsymbol{b})=\operatorname{tr}(\boldsymbol{b}\boldsymbol{a}^T dX)=\operatorname{tr}((\boldsymbol{a}\boldsymbol{b}^T)^T dX)$，注意这里我们根据 $\operatorname{tr}(AB)=\operatorname{tr}(BA)$ 交换了 $\boldsymbol{a}^T dX$ 与 $\boldsymbol{b}$。对照导数与微分的联系 $df=\operatorname{tr}\left(\dfrac{\partial f}{\partial X}^T dX\right)$，得到 $\dfrac{\partial f}{\partial X}=\boldsymbol{a}\boldsymbol{b}^T$。

矩阵求导基础

例 【线性回归】：$l=\|X\boldsymbol{w}-\boldsymbol{y}\|^2$，求 $\boldsymbol{w}$ 的最小二乘估计，即求 $\dfrac{\partial l}{\partial \boldsymbol{w}}$ 的零点。其中 $\boldsymbol{y}$ 是 $m\times 1$ 列向量，X 是 $m\times n$ 矩阵，$\boldsymbol{w}$ 是 $n\times 1$ 列向量，l 是标量。

矩阵求导基础

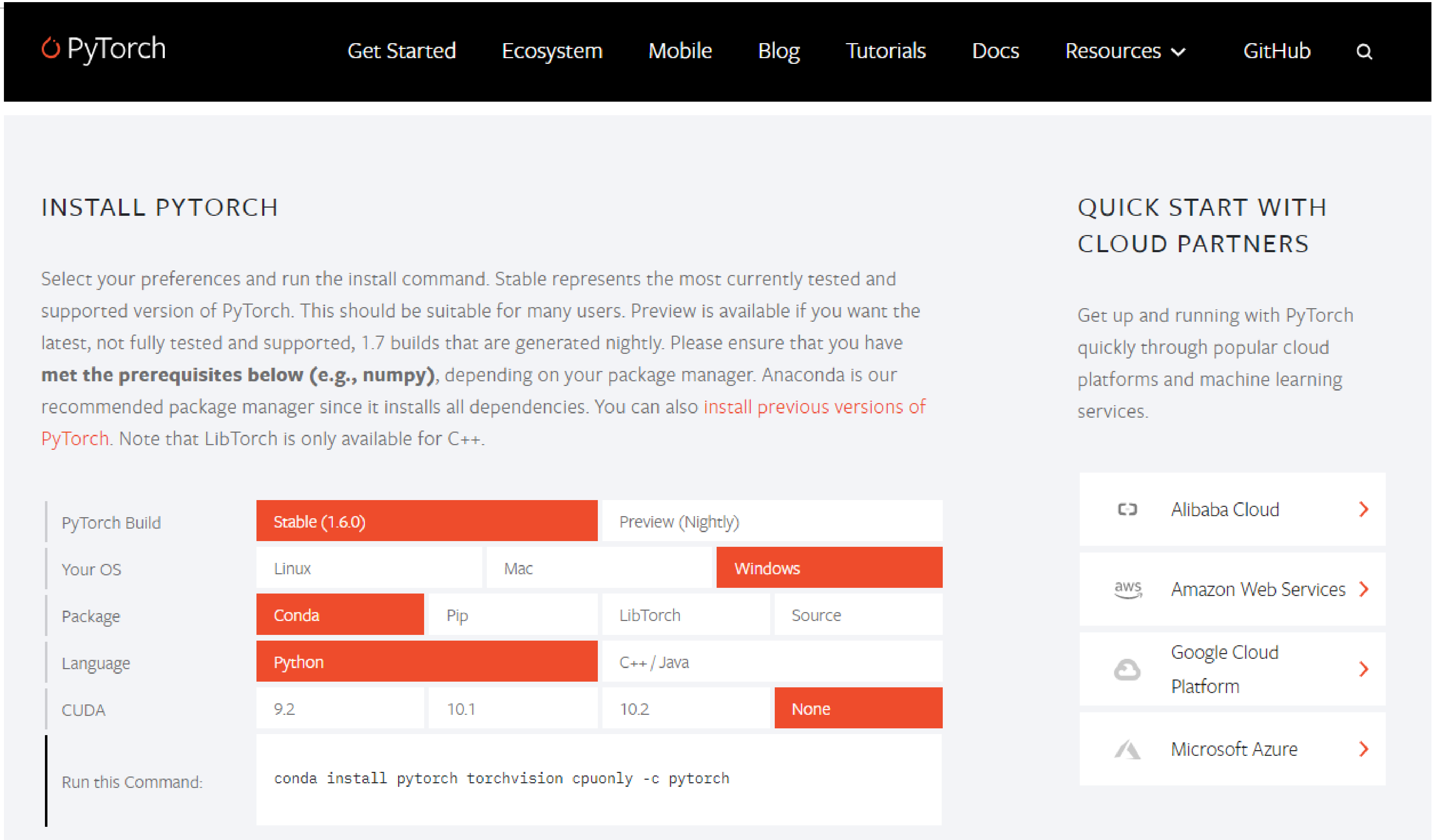
例 【线性回归】: $l=\|X\boldsymbol{w}-\boldsymbol{y}\|^2$，求 $\boldsymbol{w}$ 的最小二乘估计，即求 $\dfrac{\partial l}{\partial \boldsymbol{w}}$ 的零点。其中 $\boldsymbol{y}$ 是 $m\times 1$ 列向量，X 是 $m\times n$ 矩阵，$\boldsymbol{w}$ 是 $n\times 1$ 列向量，l 是标量。

解：这是标量对向量的导数，不过可以把向量看做矩阵的特例。先将向量模平方改写成向量与自身的内积：$l=(X\boldsymbol{w}-\boldsymbol{y})^T(X\boldsymbol{w}-\boldsymbol{y})$，求微分，使用矩阵乘法、转置等法则：$dl=(Xd\boldsymbol{w})^T(X\boldsymbol{w}-\boldsymbol{y})+(X\boldsymbol{w}-\boldsymbol{y})^T(Xd\boldsymbol{w})=2(X\boldsymbol{w}-\boldsymbol{y})^TXd\boldsymbol{w}$。对照导数与微分的联系 $dl=\dfrac{\partial l}{\partial \boldsymbol{w}}^T d\boldsymbol{w}$，得到 $\dfrac{\partial l}{\partial \boldsymbol{w}}=2X^T(X\boldsymbol{w}-\boldsymbol{y})$。$\dfrac{\partial l}{\partial \boldsymbol{w}}=0$ 即 $X^TX\boldsymbol{w}=X^T\boldsymbol{y}$，得到 $\boldsymbol{w}$ 的最小二乘估计为 $\boldsymbol{w}=(X^TX)^{-1}X^T\boldsymbol{y}$。

PyTorch基础

安装

https://pytorch.org/



PyTorch Get Started Ecosystem Mobile Blog Tutorials Docs Resources GitHub

INSTALL PYTORCH

Select your preferences and run the install command. Stable represents the most currently tested and supported version of PyTorch. This should be suitable for many users. Preview is available if you want the latest, not fully tested and supported, 1.7 builds that are generated nightly. Please ensure that you have **met the prerequisites below (e.g., numpy)**, depending on your package manager. Anaconda is our recommended package manager since it installs all dependencies. You can also install previous versions of PyTorch. Note that LibTorch is only available for C++.

PyTorch Build	Stable (1.6.0)		Preview (Nightly)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
CUDA	9.2	10.1	10.2	None
Run this Command:	conda install pytorch torchvision cpuonly -c pytorch			

QUICK START WITH CLOUD PARTNERS

Get up and running with PyTorch quickly through popular cloud platforms and machine learning services.

- Alibaba Cloud
- Amazon Web Services
- Google Cloud Platform
- Microsoft Azure

安装

C:\Windows\system32\cmd.exe - conda install pytorch torchvision cpuonly -c pytorch

```
(virenv) C:\Users\Jian>conda install pytorch torchvision cpuonly -c pytorch
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: C:\Users\Jian\anaconda3\envs\virenv

  added / updated specs:
    - cpuonly
    - pytorch
    - torchvision


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    cpuonly-1.0                |                0           2 KB  pytorch
    ninja-1.10.1               |   py38h7ef1ec2_0         249 KB
    pytorch-1.6.0              |       py3.8_cpu_0       142.0 MB  pytorch
    torchvision-0.7.0          |         py38_cpu         5.8 MB  pytorch
    ------------------------------------------------------------
                                           Total:       148.0 MB

The following NEW packages will be INSTALLED:

  cpuonly            pytorch/noarch::cpuonly-1.0-0
  ninja              pkgs/main/win-64::ninja-1.10.1-py38h7ef1ec2_0
  pytorch            pytorch/win-64::pytorch-1.6.0-py3.8_cpu_0
  torchvision        pytorch/win-64::torchvision-0.7.0-py38_cpu


Proceed ([y]/n)?
```

安装

```
C:\Windows\system32\cmd.exe - conda  install pytorch torchvision cpuonly -c pytorch

  pytorch            pytorch/win-64::pytorch-1.6.0-py3.8_cpu_0
  torchvision        pytorch/win-64::torchvision-0.7.0-py38_cpu


Proceed ([y]/n)? y


Downloading and Extracting Packages
torchvision-0.7.0    | 5.8 MB    | ############################################################################ | 100%
ninja-1.10.1         | 249 KB    | ############################################################################ | 100%
pytorch-1.6.0        | 142.0 MB  | ############################################################################ | 100%
cpuonly-1.0          | 2 KB      | ############################################################################ | 100%
Preparing transaction: done
Verifying transaction: done
Executing transaction: done

(virenv) C:\Users\Jian>
```

```
(virenv) C:\Users\Jian>python
Python 3.8.5 | packaged by conda-forge | (default, Sep 24 2020, 16:20:24) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> print(torch.__version__)
1.6.0
>>>
```

- 自动求导机制
- 与Numpy互相转换注意

W4_Tensor_Tutorial.ipynb

W4_PyTorch_Basic.ipynb

本次作业

```
import torch
torch.manual_seed(0)

x = torch.randn(10,4, requires_grad=True)
W = torch.randn(4,4, requires_grad=True)
y = torch.randn(10,4, requires_grad=True)
```

目标函数： $f = ||\max(XW, 0) - Y||_F^2$

手动写出以下表达式，并用PyTorch进行验证：

$$\frac{\partial f}{\partial W} \qquad \frac{\partial f}{\partial X} \qquad \frac{\partial f}{\partial Y}$$

交流&问题？