$$y = Ax$$
$$A^{-1}y = A^{-1}Ax$$
$$A^{-1}y = x$$

A 矩阵是线性变换.

如果变换 A 不会将空间塌陷到更低的维度的话，也就是说 A 的行列式不为 0，那么 x 经过 A 线性变换后就会找到唯一的 Y 与之对应．同理．我们对 y 进行 A 逆变换，也会得到唯一与之对应的 x.

A matrix is linear transformation.

If transformation A doesn’t squish all space into a lower dimension, that mean determinant of A is non-zero, then ‘y’ and ‘x’ are one-to-one correspondents, that mean given a ‘x’ vector always have a unique ‘y’ vector for corresponding via A linear transformation. Similarly, given a ‘y’ vector we can also find a unique ‘x’ vector by inverse of A transformation, this is an inverse linear transformation.

$$A = U\Sigma V^T$$

奇异值分解：Singular Value Decomposition, SVD

奇异值分解：给定一个任意矩阵 A，我们都可以分解成为三个矩阵相乘的形式。其中 U，V 是两个方阵，∑是一个对角阵。对角上面的就是奇异值。

What is SVD? Given any Matrix A, we can decompose A to form of multiply with three Matrix. Where U, V is square Matrix, Σ is a diagonal Matrix, in which ones on diagonal are the singular value.

$A = U\Sigma V^T$：

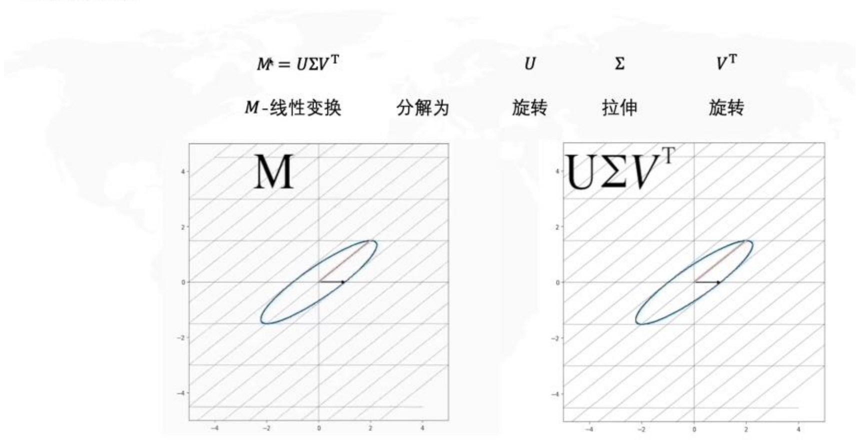
$$\begin{bmatrix} \quad & \quad \\ \quad & \quad \\ \quad & \quad \end{bmatrix}_{(m,n)} = \begin{bmatrix} \quad & \quad \\ \quad & \quad \end{bmatrix}_{(m,m)} \begin{bmatrix} \quad & \quad \\ \quad & \quad \\ \quad & \quad \end{bmatrix}_{(m,n)} \begin{bmatrix} \quad & \quad \\ \quad & \quad \end{bmatrix}_{(n,n)}$$

为什么任意一个矩阵可以被分解为上述这种形式。解释如下：矩阵就是一个变换，任何一个矩阵都可以被看成是一种线性变换。那么线性变换，在空间上就表示过原点和等距分布，那么满足这种变换有两种形式：拉伸和旋转。因此一个矩阵就可以看成是含有拉伸，旋转的变换的载体。所以一个矩阵可以通过 SVD 分解成为三个矩阵相乘，也就是旋转，拉伸，再旋转。

Why given any matrix can be decomposed to form of above. interpretation in this here: Matrix is a transformation, and Matrix can be view a linear transformation, which represent through the origin and equally spaced in space, then there are two forms that satisfy this transformation: *stretch* and *rotation*. so, a Matrix can be view a supporter include stretch and rotation. so a matrix can decomposition to three Matrix multiplication, that is rotation, stretch and rotation again.

$$stretch\text{: } s = \begin{bmatrix} i & 0 \\ 0 & j \end{bmatrix} \qquad rotation\text{: } r = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

SVD在2×2矩阵



这种变换的具体推导：

在变换中，我们一般以一组基，来追踪线性变换，在 2D 中，我们以 V 作为一组基向量：那么这组基经过 A 矩阵变换后：可以分解为是基向量经过拉伸和旋转变换后得到的向量。
The specific derivation of this transformation:
In transformation, we generally use a group of bases to track linear transformation. In 2D, we use V as a group of base vectors, via A transformation then it can be decompose into vectors obtained by stretching and rotaing the base vector.

$$V=[\hat{v}_1\ \hat{v}_2]$$
$$MV=U\Sigma$$
$$\because V\, is\ othogonal\ Matrix,\ so\ right\ multiply\ V^T$$
$$MVV^T=U\Sigma V^T$$
$$M=U\Sigma V^T$$

同理 SVD 可以推广到任意矩阵的大小。
Σ对角的值就是奇异值。并且是从大到小排列的。奇异值就代表轴。
那么降维，就是保留重要的轴，然后将不那么重要的轴去除。这就是数据的压缩。
In the same way, SVD can be generalized to the any matrix,
The value of $\sum$ diagonally is the singular value, And arranged from large to small. The singular value represents the axis
so dimensionality reduction is keeping the import axis and then removing the less important axis. This is the compression of data.
那么如何求解 SVD：
How to calculate SVD:

$$A=U\Sigma V^T$$
$$A^TA=(U\Sigma V^T)^TU\Sigma V^T$$
$$A^TA=V\Sigma U^TU\Sigma V^T$$
$$A^TA=V\Sigma\Sigma V^T$$
$$order:\ L=\Sigma\Sigma=\begin{bmatrix}\lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n\end{bmatrix}\begin{bmatrix}\lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_n\end{bmatrix}=\begin{bmatrix}\lambda^2{}_1 & & & \\ & \lambda^2{}_2 & & \\ & & \ddots & \\ & & & \lambda^2{}_n\end{bmatrix}$$
$$A^TA=VLV^T$$
$$A^TAV=VL,\ similarly:\ AA^TU=UL$$
$$A^TA=VLV^T,\ AA^T=ULU^T$$
$$\because V,\ U\ is\ othogoal\ matrix,\ so\ their\ inverse\ is\ equal\ to\ the\ transpose$$
$$then\ form\ can\ be\ wrote\ is\ that:$$
$$A^TA=VLV^{-1},\ AA^T=ULU^{-1}$$
$$above\ formula\ is\ that\ form\ of\ eigenvector\ and\ eigenvalue$$
$$so,\ A^TA\text{‘}s\ eigenvector\ is\ V.\ AA^T\text{’}s\ eigenvector\ is\ U$$
$$A^TA\ or\ AA^T\text{’}s\ eigenvalue\ is\ L$$
$$then\ sqrt(L),\ get\ signual\ value:\ \lambda_1\ \cdots\ \lambda_n$$

基于 SVD 的图片压缩:

Image Compression Based on Singular Value Decomposition

```
Parameter:
   k: number of singular value retained

1. img = read image
2. for i in range(channel):
3.        U, Singular_value, V_T = svd(img)
4.        Sigma_Matrix = get_Sigma_Matrix(Singular_value, k)
5.        Compose_img = U[:,: k] × Sigma_Matrix × V_T[: k,]
6. Zip_img = normalization(Compose_img)
7. Show_finally_result()
```

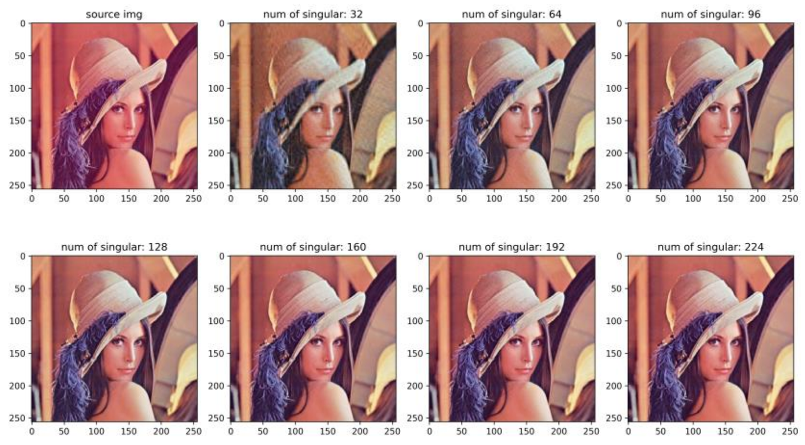
Function: get_Sigma_Matrix

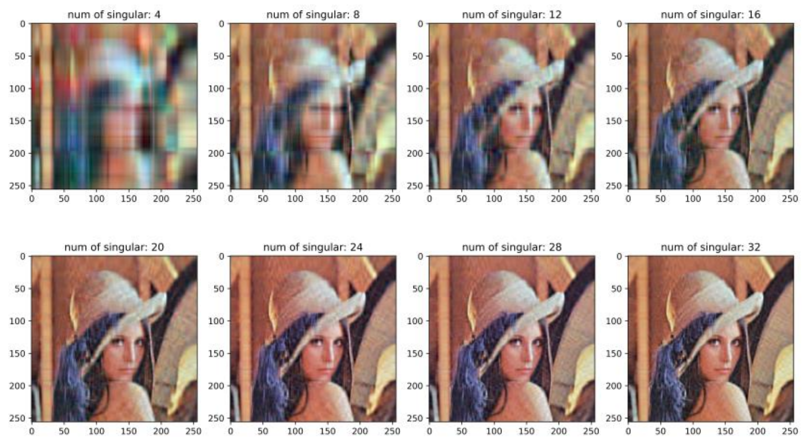
```
Input:
   Singular_value: np.ndarray, which dimension is one, and descending order
   k: int, number of singular value retained

Return:
   np.eye(k) * Singular_value[: k]
```

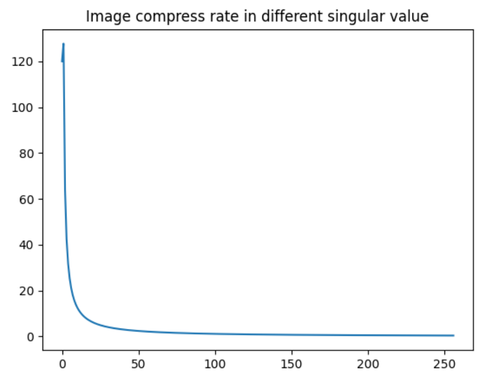
Function: svd

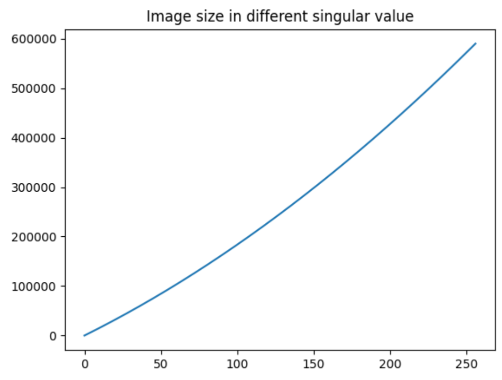
```
Input:
   Matrix: calculate svd of this matrix

Return:
   $U, singular\_value, V^T$

1. Singular, u = eig(Matrix)
2. Singular_matrix=get_Sigma_Matrix(singular, Singular.len)
3. s_idx=argsort(Singular)[:, ::-1]
4. u[:] = u[:, s_idx]
5. v = singular.inv × u.t × Matrix
Return:     $U, singular\_value, V^T$
```

result:





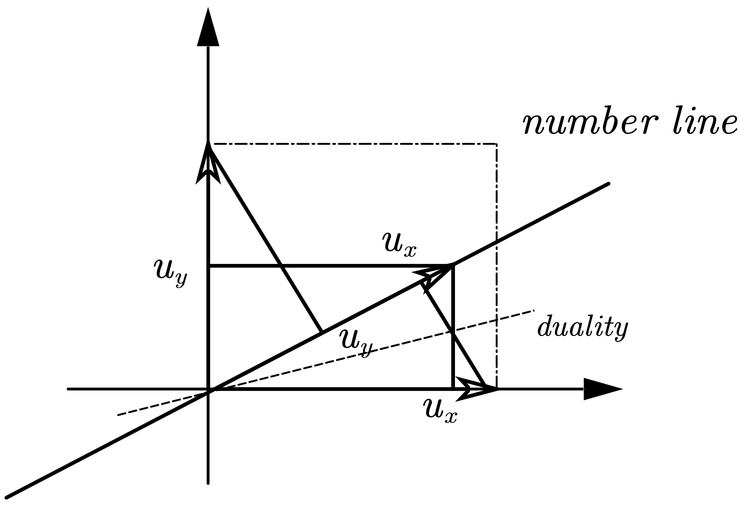




随着奇异值越来越多，在主要轴上保留的信息也越来越多，因此图片尺寸越来越大，压缩率越来越小。
With more and more singular values, more and more information is retained on the main axis, so the image size is larger and the compression ratio is smaller and smaller.

code link:
[postgraduate-studying/main.py at master · WakingHours-GitHub/postgraduate-studying]()

Understand *dot product* myself:



Why associate between dot product and projection? Actually, process in here. Form of dot product and linear transformation is same, and one of vectors by 2 dimension transformation is actually projecting onto the number line. So we can regard that there are have an associate between dot project and projection.为什么点积要和投影联系起来？说说我自己的理解。实际上过程是这样的。点积的形式，和向量做线性变化的形式一样，而向量做 2D 线性变换的实际上就是投影在数轴上。因此，我们可以说点积是可以和投影相联系起来的。

$$dot\ product\text{：}$$

$$\begin{bmatrix} a \\ b \end{bmatrix} \cdot \begin{bmatrix} c \\ d \end{bmatrix} = ac + bd$$

$$two\ dimension\ Vector\ linear\ transform\text{：}$$

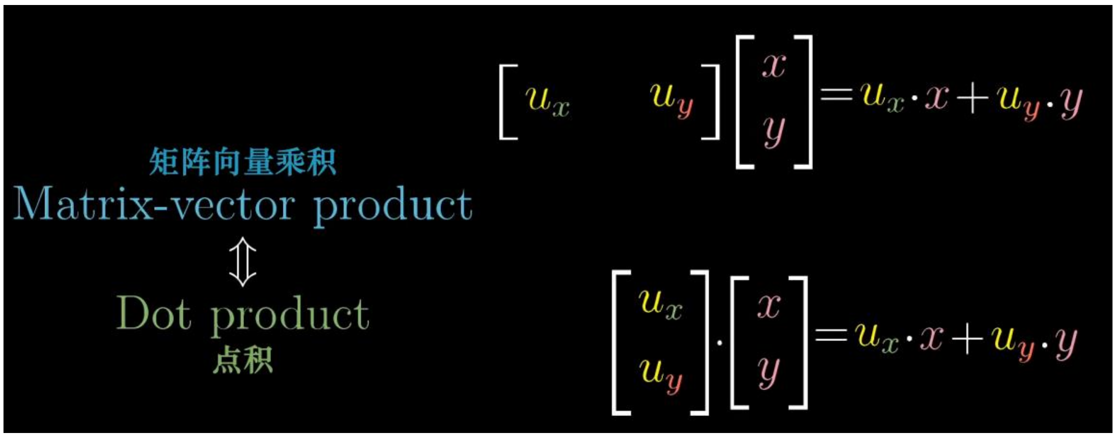
$$\begin{bmatrix} u_x \\ u_y \end{bmatrix} [a \quad b] = u_x a + u_y b$$

This linear transform, multiply with transform matrix, this form of transformation and dot product is same. This is why dot product can be interpreted projection a vector onto the span of that unit victor and taking the length. That's can associate matrix projection.

这样，这种线性变换，乘以待变换的矩阵，这种形式与 dot product 的形式完全相同。这就是为什么这种 dot product 可以解读为将向量投影到单位向量所在的直线上的所得到的投影长度（projection a vector onto the span of that unit victor and taking the length）。即将矩阵投影相联系起来。

这里，也有一种对偶性质：就是每当你看到一个向量从空间到数轴的线性变换. 那么总是能找到其对应的向量，满足这种点积的形式. 在叉积中，这种思想也会被体现到.

Here, also have a , you find linear transformation to the number line, will be able to match it to some vector, which is called the dual vector of that transformation, so that performing the linear transformation is the same as taking a dot product with that vector.



$$\begin{bmatrix} u_x & u_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = u_x \cdot x + u_y \cdot y$$

$$\begin{bmatrix} u_x \\ u_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} = u_x \cdot x + u_y \cdot y$$

Similarly, for any non-unit vector dot product, one of vectors always represent a compose of unit vector: $\hat{i}$, $\hat{j}$, and it is linear transformation matrix, then conduct linear transformation with another vector. That is why dot product between a vector and non-unit vector can interpret that first take projection onto a given vector, to get projection value, then multiply by vector, and then plus.

同理，对对于任意向量点积，其中一个向量总是可以被表示成单位向量的组合，那么此时他就是线性变换矩阵。然后再将另一个向量做线性变换。这就是为什么向量与给定非单位向量的点积 可以解读为：首先朝给定向量上做投影，然后将投影值，与给定向量长度相乘，然后相加。

In short: dot product is that change one of vectors to linear transformation. Then conduct linear transformation to another vector, because of it is one-by-two matrix (transformation), so, result of vector is mapped onto number line, and result is a number value.

总的来说：两个向量点乘就是将其中一个向量转化为线性变换。然后对另外一个向量做线性变换，因为是 1-by-2 transform，所以最终向量被映射到 number line，并且最终的结果是一个数。

10 月汇报

工作:

观看《线性代数的本质》系列视频，我的英语不是很好，一些视频经常需要多刷几遍，同时加深理解，边听边思考。目前已经看完，准备回顾。

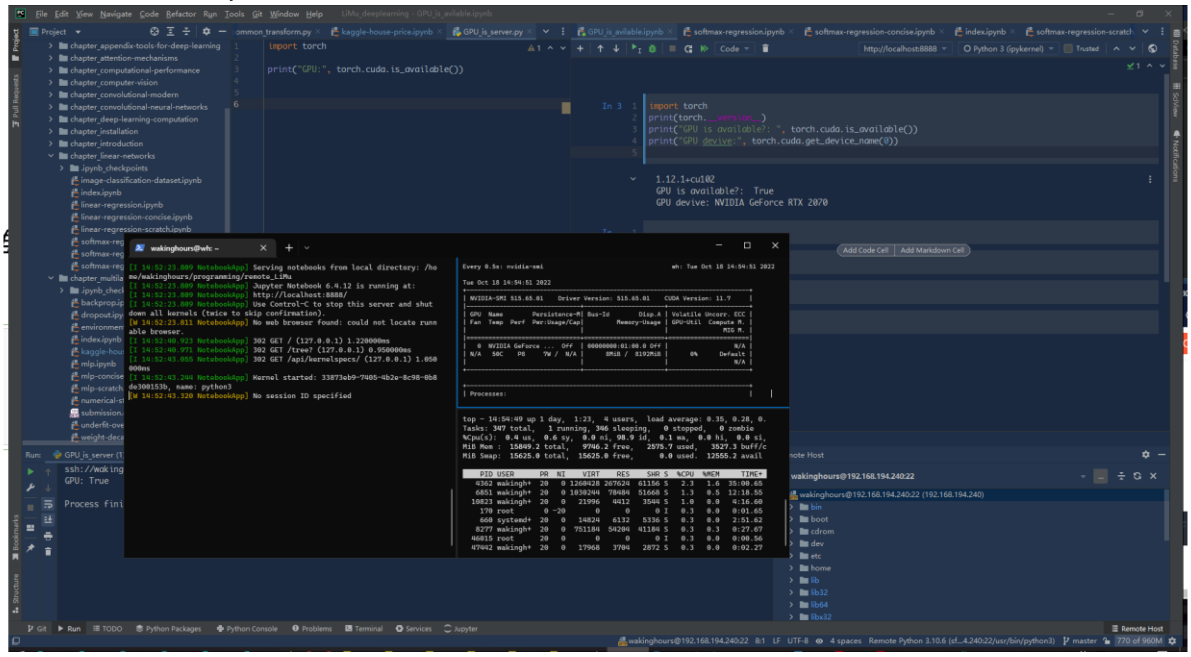
准备论文开题，搭建运行环境，配置服务器，在我另一台搭载 2070 的笔记本安装 ubuntu。并使用 ssh 和 ssh -L 进行远程连接和端口转发。配置 pycharm 以便可以远程服务器的训练和推理。

学习李沐:《动手学深度学习》(torch)课程，以及土堆：动手学 PyTorch。目前正在学习神经网络基础部分。以及深度学习中的理论知识。进度一般，没有课的时间，每天都能抽出来时间进行学习。

论文：目前刚开始看，《SSD：single Shot MultiBox Detector》。一个硕士毕业论文：《基于机器视觉的河流漂浮异物检测方法研究_李善超》为毕业设计做准备。目前看，因为这几周实验课较多，并且下周还有一门考试，进度较慢。

结果:

在远程端运行本地的 Py 代码。



```python
import torch

print("GPU:", torch.cuda.is_available())
```

```python
import torch
print(torch.__version__)
print("GPU is available?: ", torch.cuda.is_available())
print("GPU devive:", torch.cuda.get_device_name(0))
```

```
1.12.1+cu102
GPU is available?:  True
GPU devive: NVIDIA GeForce RTX 2070
```

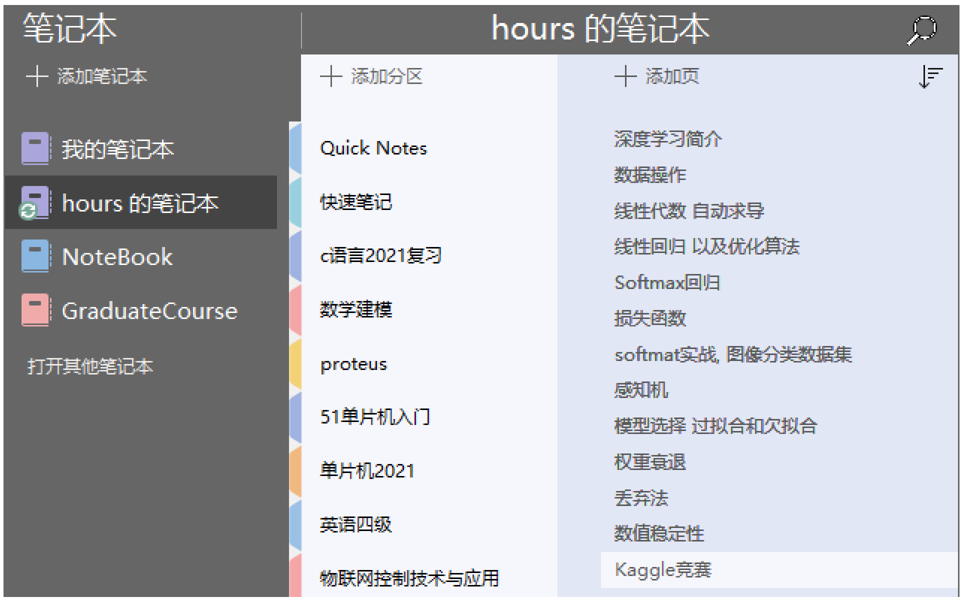
```
[I 14:52:23.809 NotebookApp] Serving notebooks from local directory: /ho
me/wakinghours/programming/remote_LiMu
[I 14:52:23.809 NotebookApp] Jupyter Notebook 6.4.12 is running at:
[I 14:52:23.809 NotebookApp] http://localhost:8888/
[I 14:52:23.809 NotebookApp] Use Control-C to stop this server and shut
down all kernels (twice to skip confirmation).
[W 14:52:23.811 NotebookApp] No web browser found: could not locate runn
able browser.
[I 14:52:40.923 NotebookApp] 302 GET / (127.0.0.1) 1.220000ms
[I 14:52:40.971 NotebookApp] 302 GET /tree? (127.0.0.1) 0.950000ms
[I 14:52:43.055 NotebookApp] 302 GET /api/kernelspecs/ (127.0.0.1) 1.050
000ms
[I 14:52:43.244 NotebookApp] Kernel started: 33873eb9-7405-4b2e-8c98-0b8
de300153b, name: python3
[W 14:52:43.320 NotebookApp] No session ID specified
```

```
Every 0.5s: nvidia-smi                               wh: Tue Oct 18 14:54:51 2022

Tue Oct 18 14:54:51 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 515.65.01    Driver Version: 515.65.01    CUDA Version: 11.7     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  Off  | 00000000:01:00.0 Off |                  N/A |
| N/A   50C    P8     7W /  N/A |      8MiB /  8192MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
```

```
top - 14:54:49 up 1 day,  1:23,  4 users,  load average: 0.35, 0.28, 0.
Tasks: 347 total,   1 running, 346 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.4 us,  0.6 sy,  0.0 ni, 98.9 id,  0.1 wa,  0.0 hi,  0.0 si,
MiB Mem :  15849.2 total,   9746.2 free,   2575.7 used,   3527.3 buff/c
MiB Swap:  15625.0 total,  15625.0 free,      0.0 used.  12555.2 avail

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+
   4362 wakingh+  20   0 1268428 267624  61156 S   2.3   1.6  35:00.65
   6851 wakingh+  20   0 1030244  78484  51668 S   1.3   0.5  12:18.55
  10823 wakingh+  20   0   21996   4412   3544 S   1.0   0.0   4:16.60
    170 root       0 -20       0      0      0 I   0.3   0.0   0:01.65
    660 systemd+  20   0   14824   6132   5336 S   0.3   0.0   2:51.62
   8277 wakingh+  20   0  751184  54204  41184 S   0.3   0.3   0:27.67
  46015 root      20   0       0      0      0 I   0.3   0.0   0:00.56
  47442 wakingh+  20   0   17968   3704   2872 S   0.3   0.0   0:02.27
```

（这里使用的是内网穿透，虚拟化后的 IP）

进度:



笔记本

+ 添加笔记本

我的笔记本
hours 的笔记本
NoteBook
GraduateCourse
打开其他笔记本

hours 的笔记本

+ 添加分区

Quick Notes
快速笔记
c语言2021复习
数学建模
proteus
51单片机入门
单片机2021
英语四级
物联网控制技术与应用

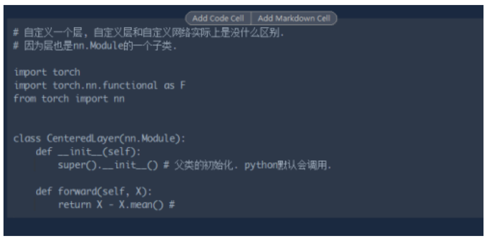
+ 添加页

深度学习简介
数据操作
线性代数 自动求导
线性回归 以及优化算法
Softmax回归
损失函数
softmat实战, 图像分类数据集
感知机
模型选择 过拟合和欠拟合
权重衰退
丢弃法
数值稳定性
Kaggle竞赛

一些问题:

来自 pycharm 连接远程服务器运行 jupyter notebook 时的一些细节问题，还在寻找解决答案中。

- 例如 PyCharm 在远程端无法直接运行 Jupyter Notebook 代码，只有在远程端开启 notebook 后，使用 SSH 将端口转发到本地后才能使用。
- 并且在远程端开启 Jupyter 后，映射到本地端 pycharm 居然不进行代码解析了!. 所有的代码不会高亮!所有的代码都是文本(显示白色)!这导致再 pycharm 中书写 jupyter notebook 的体验非常不好. 其代码补全逻辑和 JB 的软件相比是非常难用的. 目前还没有找到解决方案.



```
# 自定义一个层，自定义层和自定义网络实际上是没什么区别.
# 因为层也是nn.Module的一个子类.

import torch
import torch.nn.functional as F
from torch import nn


class CenteredLayer(nn.Module):
    def __init__(self):
        super().__init__() # 父类的初始化. python默认会调用.

    def forward(self, X):
        return X - X.mean() #
```

- PyCharm 中 SFTP 同步问题, 在运行 py 代码时, 我首先要使用 SFTP 进行本地端和远程端实现文件的同步, 并且有时候会断开，需要重新手动同步。

沐神的《动手学深度学习》中，一些代码没有注释，并且其中的一些逻辑晦涩难懂，仍然在啃，再加上这段时间比较忙，因此代码部分进度较慢。后期会对编写代码部分的时间增加。

目前还在加强矩阵的理解, 以及英语能力的学习.

将来

跟随导师，继续进行研究生基础前导课程的学习。
读论文，增加阅读量并提高阅读速度。(希望老师可以给些方向)
继续学习 PyTorch 框架。争取在下一月完成毕设内容的初步的实现。