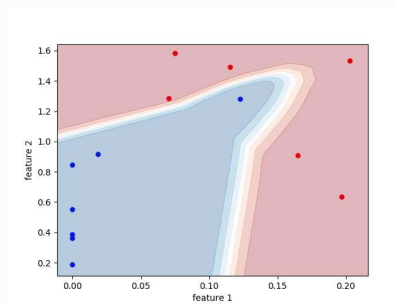


# 媒体与认知

## Media and Cognition



V 1.0 2022.3.17

清华大学电子工程系

章磊 助教

Email: [zhanglei21@mails.tsinghua.edu.cn](mailto:zhanglei21@mails.tsinghua.edu.cn)

# 第二次习题课 基于多层感知机的非线性分类

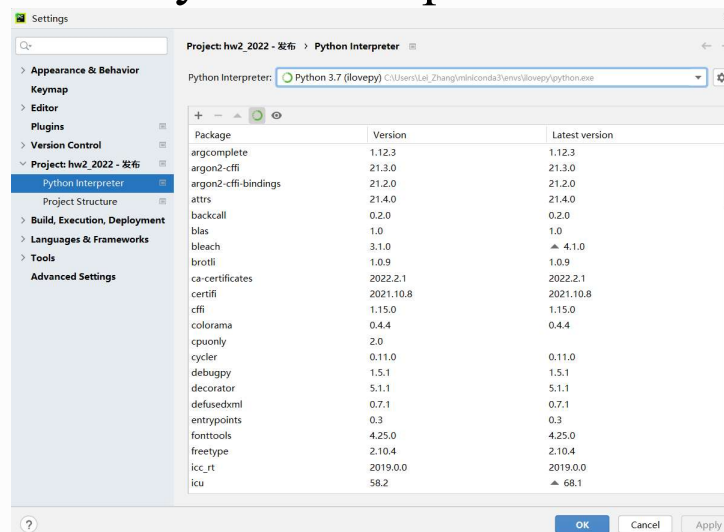
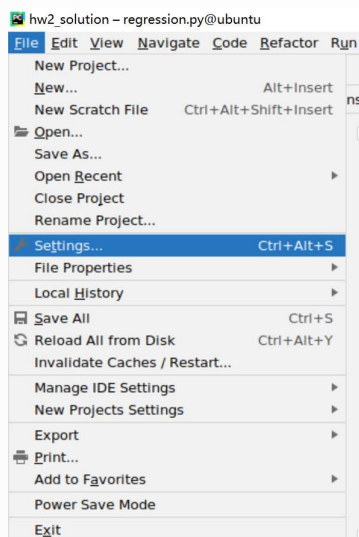
- 一、Python程序运行与调试
- 二、PyTorch中的张量Tensor
- 三、向量和矩阵求导方法
- 四、第二次作业说明

# 一、Python 程序运行与调试

◆ 建议采用PyCharm或VS Code 运行、调试Python程序

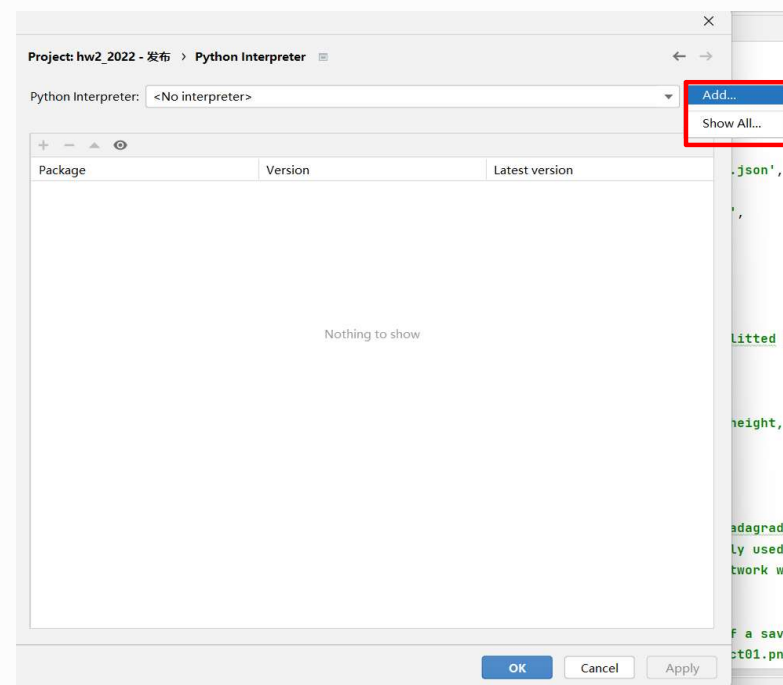
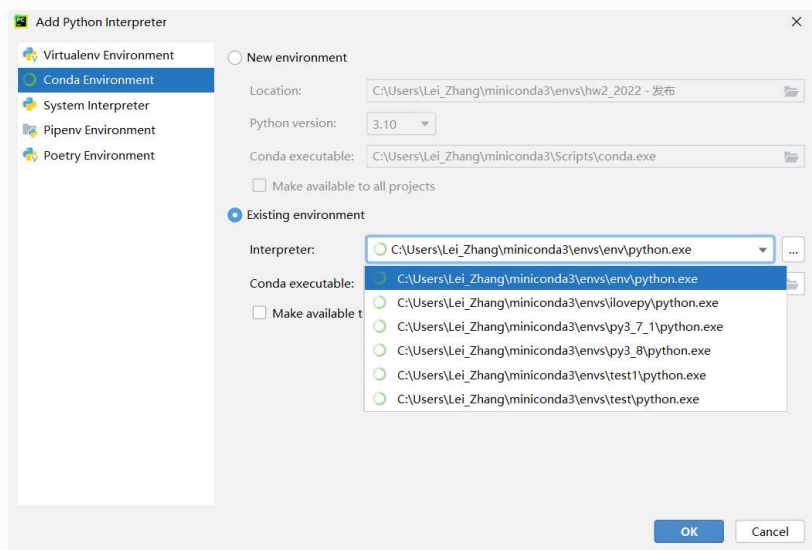
◆ **方案一：** 采用PyCharm运行、调试Python程序

- PyCharm是一款用于Python开发的专业化集成开发环境IDE
  - 可下载免费的Community 版本: <https://www.jetbrains.com/zh-cn/pycharm/>
- 安装后在File → Settings → Project → Python Interpreter中切换到对应的conda环境



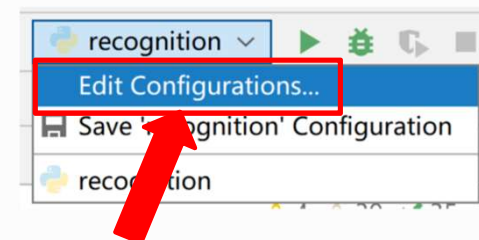
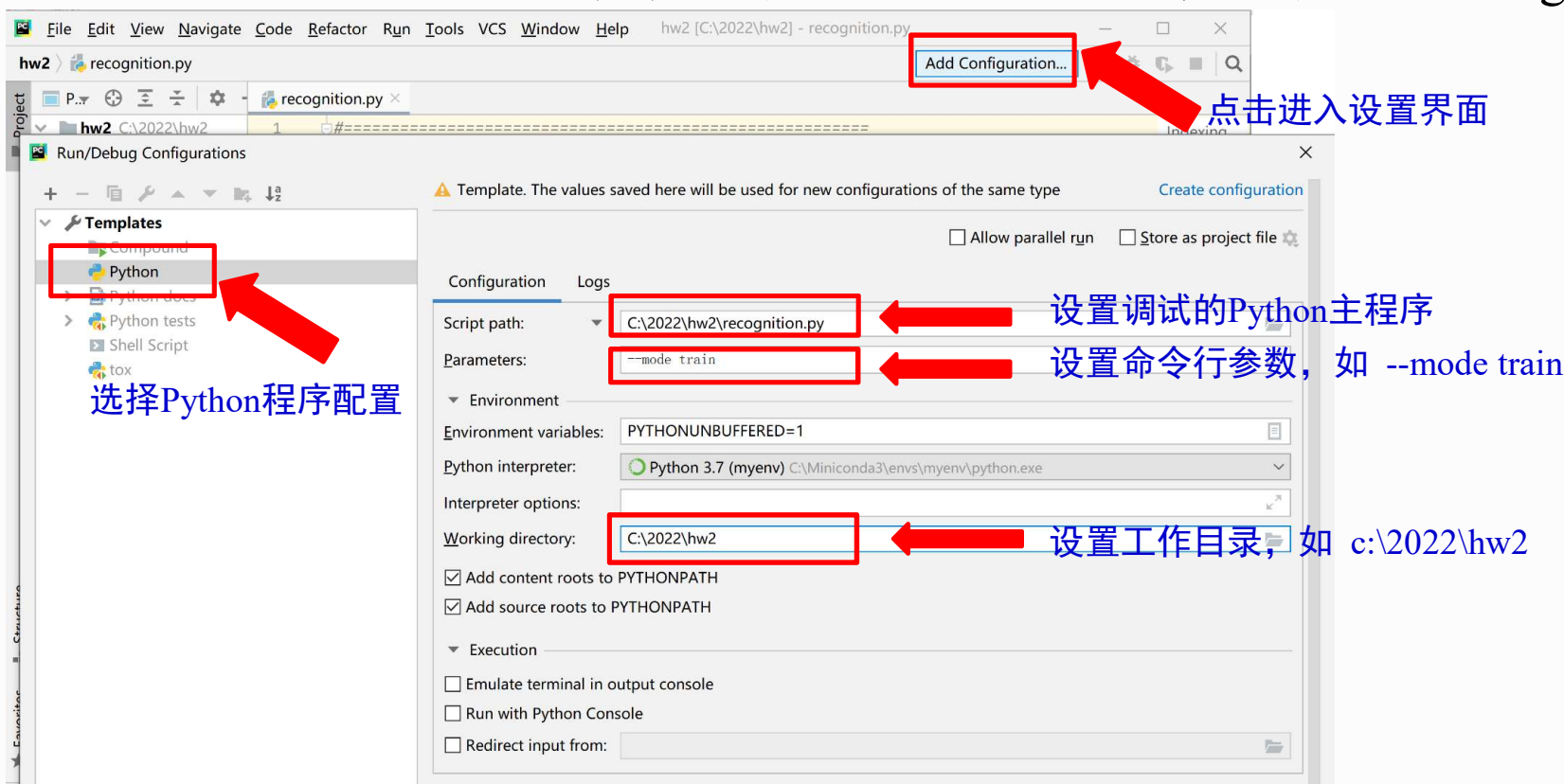
# 在PyCharm中设置Python解释器

- 若在File-Settings → Project → Python Interpreter中找不到对应conda环境名称，需要手动将其添加到PyCharm中
- 在右下图所示的红框处点击“Add Interpreter”，在弹出的窗口中选择Conda Environment → Existing environment
- 选择Interpreter为对应conda环境中的Python



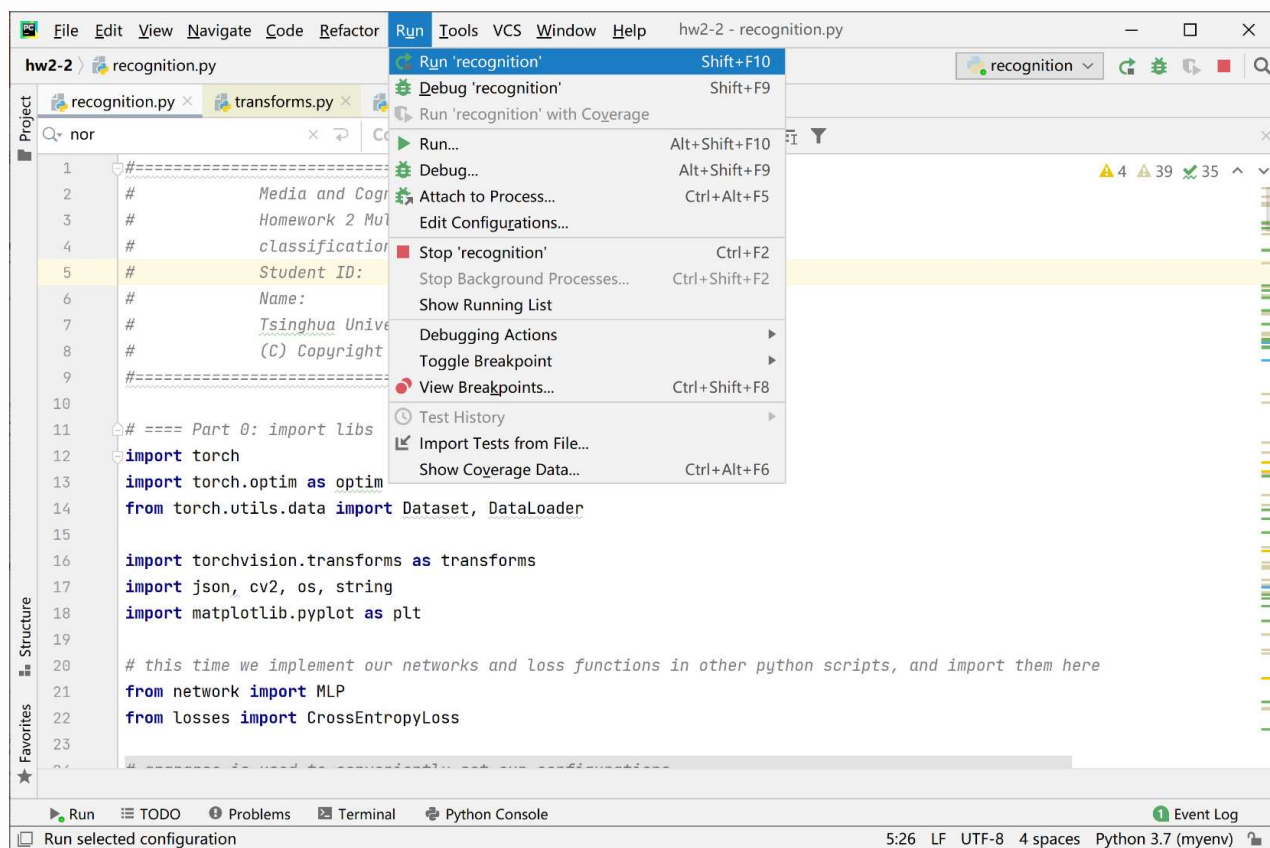
# 在PyCharm中设置Python程序工作目录和命令行参数等

- ◆ 通过File → Open打开Python文件或多个Python文件所在的文件夹
  - 点击窗口上方右侧的“Add Configuration”，选择左侧“Templates”中的Python
  - 或在当前配置下拉菜单（有个向下的箭头）中选择Edit Configurations



# 在PyCharm中运行或调试程序

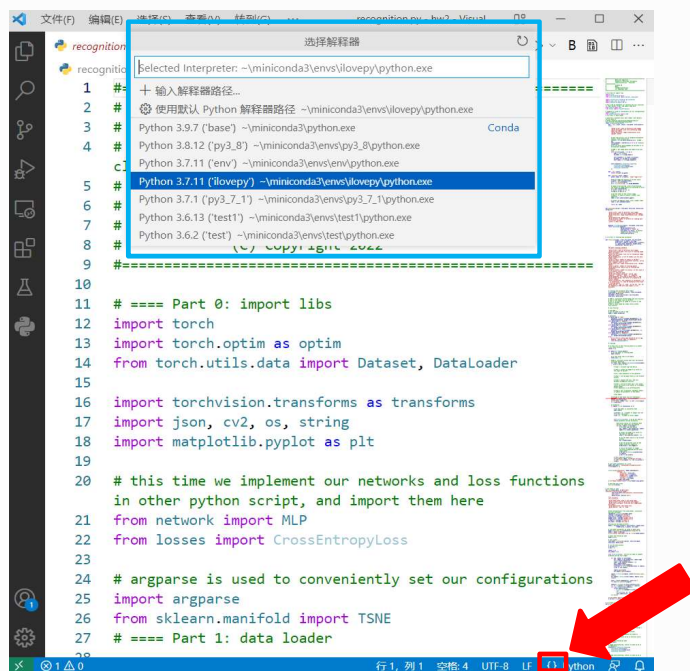
- ◆ 通过File → Run 或 File → Debug运行或调试程序



## 方案二：采用VS Code运行、调试Python程序

### ► 在VS Code中 运行、调试Python程序

- ◆ 打开需要运行的Python文件，若程序包含多个文件，则打开主文件
- ◆ 在VS Code中为Python文件选择解释器
  - 方法1：VS Code右下角点击python旁边的{}按钮可以选择解释器

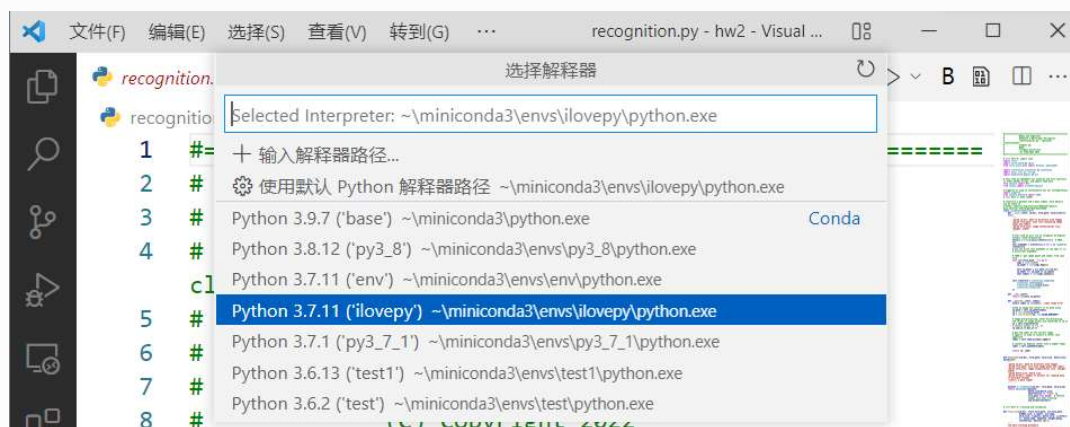
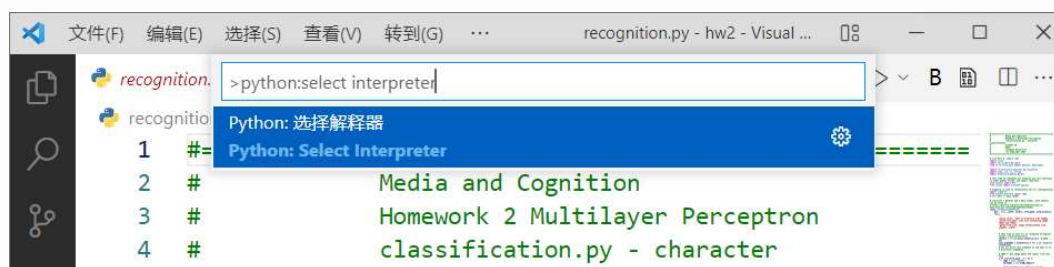




# VsCode

## ◆ 在VS Code中为Python文件选择解释器

- 方法2：在VS Code左上方打开菜单中的“查看”选项，选择命令面板，或者使用Ctrl+Shift+P打开命令面板，输入python:select interpreter，点击选择解释器





# VsCode

## ◆ 调试Python文件

- 对于已打开的Python文件，点击右侧图1所示的侧边栏“运行与调试”按钮（图中用红色框标出），然后点击运行与调试

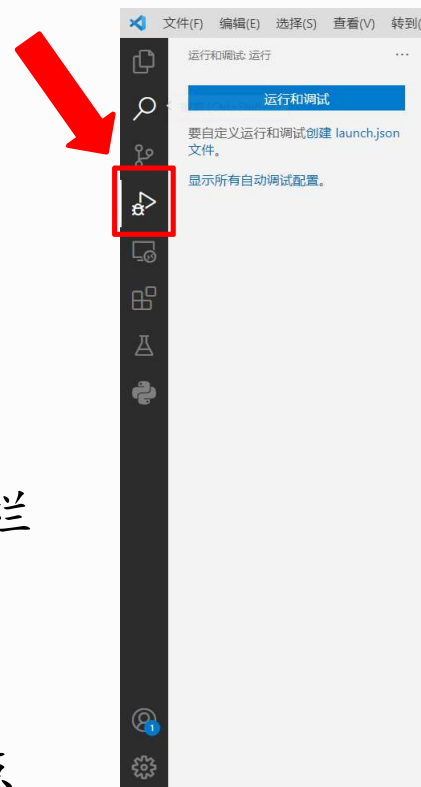


图1

- 或者打开python文件所在文件夹，点击侧边栏“运行与调试”按钮，如右侧图2所示选择“python: 当前文件”
  - 这时可以看到窗口左侧会显示变量和断点等调试工具

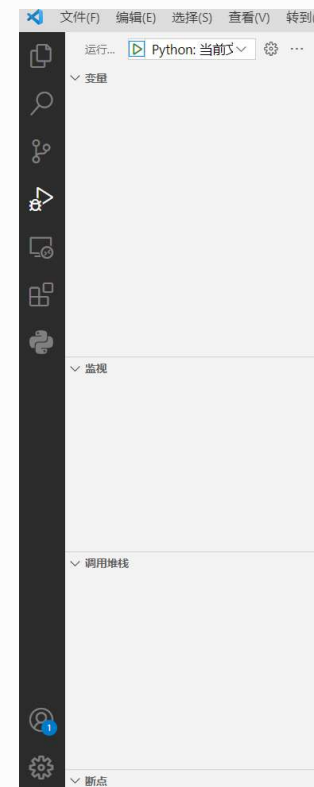


图2

# VS Code命令行终端中无法启动 conda 环境的解决方案

➤ 以Windows平台为例，MAC平台上操作类似

- ◆ 将所安装的Miniconda3或Anaconda3的工作目录（例如 C:\Miniconda3\Scripts）添加到系统环境变量“Path”中

- ◆ 在命令行终端中运行

**conda init**

➤ 在windows 搜索框中搜索 Power Shell，在其右侧菜单中选择“以管理员身份运行”

- ◆ 在Power Shell中运行如下命令

**set-ExecutionPolicy RemoteSigned**

根据提示信息选择[Y]或 [A]，确认更改执行策略

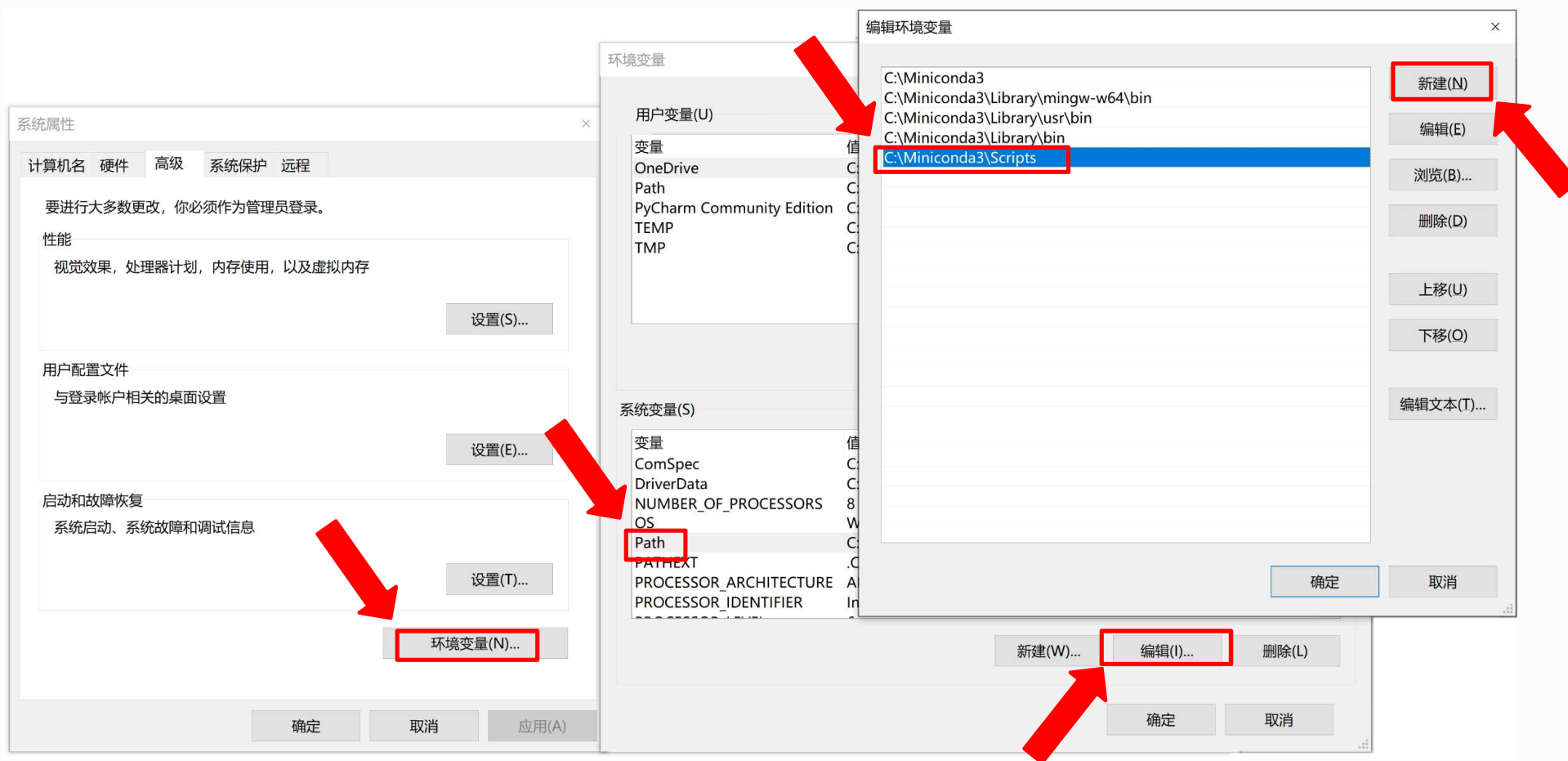
➤ 重启 VS Code，在右下方命令行终端中启动conda 环境，成功后提示符前会出现conda环境名

**conda activate myenv**



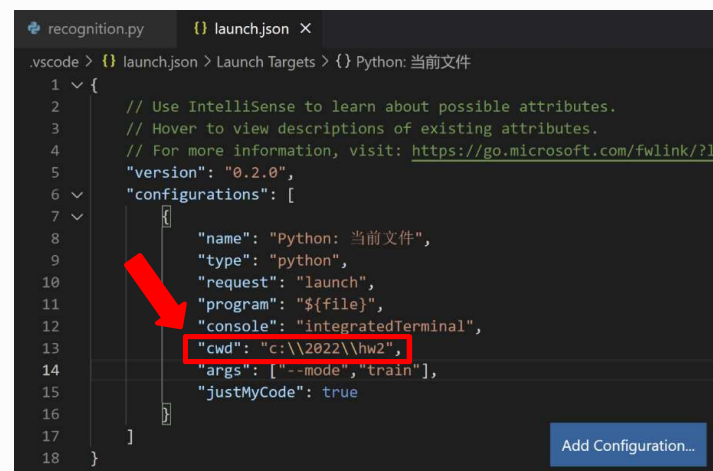
## 附：添加环境变量的方法

- 在windows 搜索框中搜索 “编辑系统环境变量”，编辑“Path”系统变量或用户变量



# 在VS Code中设置Python程序工作目录

- 运行/调试Python程序时，如果程序中使用相对路径访问数据文件等，需要设置当前工作目录（current working directory, cwd），相关设置在launch.json 配置文件中定义
- 在VS Code调试界面，可在“Python: 当前文件”处下拉菜单（有个向下的箭头）中选择Add configuration，则自动打开launch.json 配置文件
- 在launch.json文件Configurations中添加程序所在目录
  - ◆ 注意目录中的“\”前面需要多加一个转义符号控制“\”，即用双斜杆“\\”表示
  - ◆ 例如程序所在目录为“c:\2022\hw2”，添加  
**"cwd": "c:\\2022\\hw2"**
  - ◆ 各行在行尾用逗号分隔

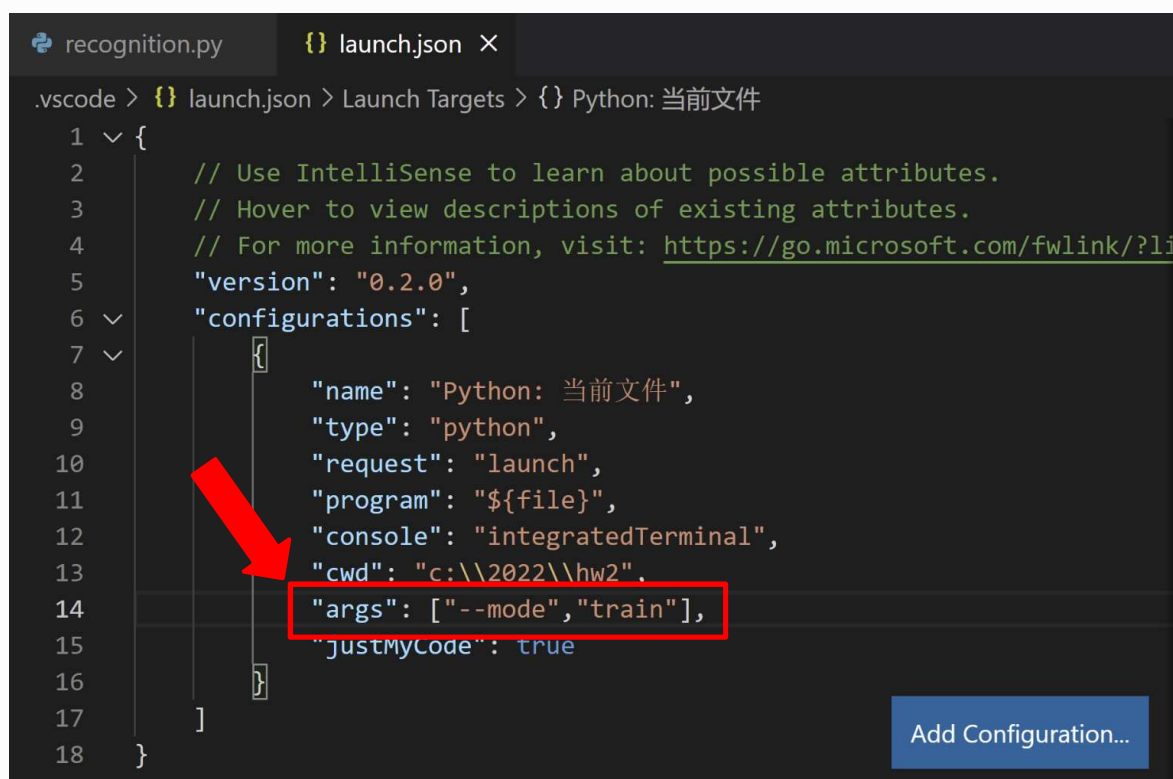


```
1 {  
2     // Use IntelliSense to learn about possible attributes.  
3     // Hover to view descriptions of existing attributes.  
4     // For more information, visit: https://go.microsoft.com/fwlink/?li  
5     "version": "0.2.0",  
6     "configurations": [  
7         {  
8             "name": "Python: 当前文件",  
9             "type": "python",  
10            "request": "launch",  
11            "program": "${file}",  
12            "console": "integratedTerminal",  
13            "cwd": "c:\\2022\\hw2",  
14            "args": ["--mode", "train"],  
15            "justMyCode": true  
16        }  
17    ]  
18 }
```

# 在VS Code中设置Python程序所需的命令行参数

- 在launch.json文件 configurations 中添加 “args” 命令行参数，多个参数以Python 列表 (list) 形式输入字符串形式的命令行参数，例如：

`"args": ["--mode", "train"]`



```
.vscode > {} launch.json > Launch Targets > {} Python: 当前文件
1  {
2      // Use IntelliSense to learn about possible attributes.
3      // Hover to view descriptions of existing attributes.
4      // For more information, visit: https://go.microsoft.com/fwlink/?li
5      "version": "0.2.0",
6      "configurations": [
7          {
8              "name": "Python: 当前文件",
9              "type": "python",
10             "request": "launch",
11             "program": "${file}",
12             "console": "integratedTerminal",
13             "cwd": "c:\\2022\\hw2",
14             "args": ["--mode", "train"],
15             "justMyCode": true
16         }
17     ]
18 }
```

## 二、PyTorch中的张量Tensor

### ◆ 什么是Tensor

- Tensor可以理解为任意维度的数组
- 例如，`[1,2]`是2维的Tensor，`[[1,2]]`是 $1 \times 2$ 维度的Tensor，`[[1],[2]]`是 $2 \times 1$ 维度Tensor

### ◆ 如何创建PyTorch Tensor

- 直接初始化一个Tensor，例如：`torch.tensor([1,2])`

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

```
>>> torch.tensor([1,2]).dtype  
torch.int64
```

- 由numpy数组转化

```
>>> torch.from_numpy(np.array([1,2]))  
tensor([1, 2], dtype=torch.int32)
```

- 随机初始化、创建全0/全1 Tensor

```
>>> torch.rand(2)  
tensor([0.1883, 0.8740])  
>>> torch.randn(2)  
tensor([-1.0343, 0.5256])  
>>> torch.zeros(2)  
tensor([0., 0.])  
>>> torch.ones(2)  
tensor([1., 1.])
```

- 更多方式可以参考

<https://pytorch.org/docs/stable/tensors.html?highlight=tensor#torch.Tensor>

# PyTorch中的张量Tensor

## ◆ 第二次作业相关的Tensor操作

- 对Tensor求和/取平均

```
>>> x = torch.ones(2, 3)
>>> x
tensor([[1., 1., 1.],
        [1., 1., 1.]])
>>> x.sum()
tensor(6.)
>>> x.sum(dim=0)
tensor([2., 2., 2.])
>>> x.sum(dim=1)
tensor([3., 3.])
```

```
>>> torch.sum(x)
tensor(6.)
>>> torch.sum(x, dim=0)
tensor([2., 2., 2.])
>>> torch.sum(x, dim=1)
tensor([3., 3.])
```

```
>>> x.mean()
tensor(1.)
>>> x.mean(0)
tensor([1., 1., 1.])
>>> x.mean(1)
tensor([1., 1.])
```

- 逐元素计算指数/对数

```
>>> x = torch.tensor([1, 2.])
>>> x
tensor([1., 2.])
>>> torch.exp(x)
tensor([2.7183, 7.3891])
```

```
>>> torch.log(x)
tensor([0.0000, 0.6931])
```

- 逐元素乘积和矩阵乘法

```
>>> x = torch.eye(2)
>>> y = torch.ones(2, 2)
>>> x
tensor([[1., 0.],
        [0., 1.]])
>>> y
tensor([[1., 1.],
        [1., 1.]])
```

```
>>> x * y
tensor([[1., 0.],
        [0., 1.]])
>>> torch.matmul(x, y)
tensor([[1., 1.],
        [1., 1.]])
```

- 更多操作可以参考

<https://pytorch.org/docs/stable/tensors.html?highlight=tensor#torch.Tensor>



### 三、向量和矩阵求导方法

#### ➤ 标量对向量求导

◆ 思路：尽量转化为标量对标量的求导

• 若  $x = (x_1 \quad \dots \quad x_n)^T$ ,  $y$  为标量, 则

$$\frac{\partial y}{\partial x} = \left( \frac{\partial y}{\partial x_1} \quad \dots \quad \frac{\partial y}{\partial x_n} \right)^T$$

• 例1:  $y = a^T x$ ,  $a$  和  $x$  为列向量, 求  $\frac{\partial y}{\partial x}$

思路：把  $a$  和  $x$  展开, 转化为 **标量对标量求导**

$$y = a^T x \Rightarrow y = (a_1 \quad \dots \quad a_n) \begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix} = a_1 x_1 + \dots + a_n x_n \Rightarrow \frac{\partial y}{\partial x} = \begin{pmatrix} a_1 \\ \dots \\ a_n \end{pmatrix} = a$$

• 例2:  $y = ax^T$ ,  $a$  和  $x$  为行向量, 求  $\frac{\partial y}{\partial x}$

$$y = (a_1 \quad \dots \quad a_n) \begin{pmatrix} x_1 \\ \dots \\ x_n \end{pmatrix} = a_1 x_1 + \dots + a_n x_n \Rightarrow \frac{\partial y}{\partial x} = (a_1 \quad \dots \quad a_n) = a$$

# 向量和矩阵求导方法

## ► 标量对矩阵求导

◆ 思路：尽量转化为标量对标量的求导

• 若  $X = \begin{pmatrix} x_{11} & \cdots & x_{1n} \\ \cdots & \ddots & \cdots \\ x_{m1} & \cdots & x_{mn} \end{pmatrix}$ ,  $y$  为标量, 则

$$\frac{\partial y}{\partial X} = \begin{pmatrix} \frac{\partial y}{\partial x_{11}} & \cdots & \frac{\partial y}{\partial x_{1n}} \\ \cdots & \ddots & \cdots \\ \frac{\partial y}{\partial x_{m1}} & \cdots & \frac{\partial y}{\partial x_{mn}} \end{pmatrix}$$

• 例3:  $y = a^T X b$ , 求  $\frac{\partial y}{\partial X}$

不妨假设  $a = (a_1 \ a_2)^T$ ,  $b = (b_1 \ b_2 \ b_3)^T$ ,  $X = \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{pmatrix}$

$$y = (a_1 \ a_2) \begin{pmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

$$= a_1 x_{11} b_1 + a_1 x_{12} b_2 + a_1 x_{13} b_3 + a_2 x_{21} b_1 + a_2 x_{22} b_2 + a_2 x_{23} b_3$$

$$\frac{\partial y}{\partial X} = \begin{pmatrix} a_1 b_1 & a_1 b_2 & a_1 b_3 \\ a_2 b_1 & a_2 b_2 & a_2 b_3 \end{pmatrix} = a b^T$$

# 向量和矩阵求导方法

## ► 神经网络中的求导问题

### ◆ 为什么要求导

- 回忆梯度下降法，求导是为了更新模型参数

$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$

### ◆ 如何求导：链式法则

- 假设一个线性层的参数为 $W$ 和 $b$ ， $\sigma(\cdot)$ 为sigmoid函数， $y^*$ 为真值

$$z = W^T x + b$$

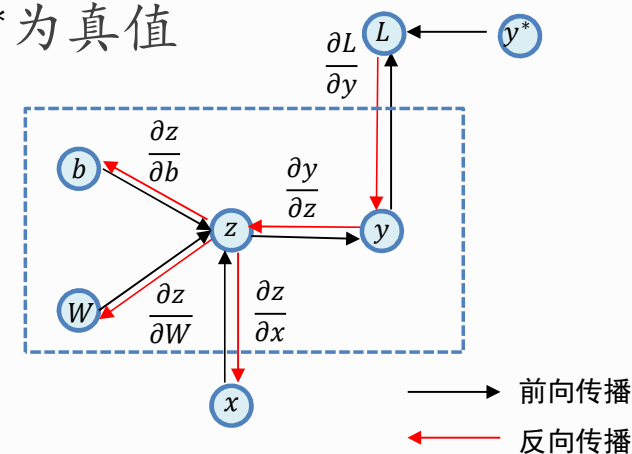
$$y = \sigma(z)$$

$$L = (y - y^*)^2$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial W}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial b}$$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial x}$$



- ◆ 问题：上述求导过程中存在向量对向量和向量对矩阵的求导： $\frac{\partial z}{\partial x}$  和  $\frac{\partial z}{\partial W}$

- 思考：该部分怎么操作？得到的结果的形状如何？

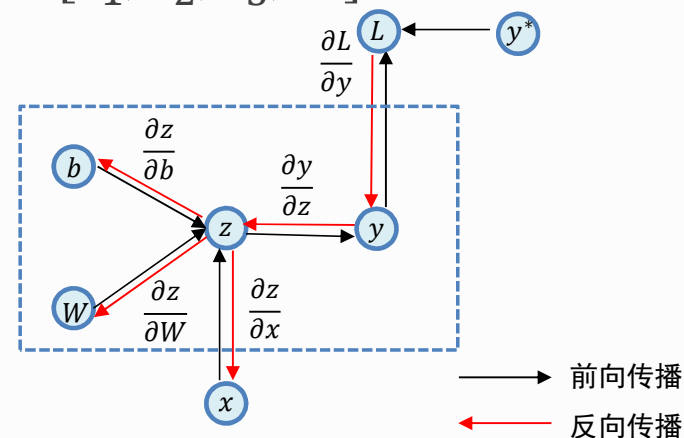
# 向量和矩阵求导方法

## ► 神经网络中的求导问题

- ◆ 如何求导：转化为标量对向量和矩阵的求导，利用  $z = [z_1, z_2, z_3, \dots]$ 。

$$z = W^T x + b \quad y = \sigma(z) \quad L = (y - y^*)^2$$

$$\begin{aligned} \frac{\partial L}{\partial W} &= \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial W} + \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial W} + \dots \\ \frac{\partial L}{\partial b} &= \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial b} + \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial b} + \dots \\ \frac{\partial L}{\partial x} &= \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial x} + \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial x} + \dots \end{aligned}$$



## ◆ 要求哪些导

- 终极目标：loss（标量）对模型各层网络参数（向量或矩阵）的导数： $\frac{\partial L}{\partial W}$  和  $\frac{\partial L}{\partial b}$
- 中间过程：loss对模型输出的导数 $\frac{\partial L}{\partial y}$ 、激活函数输出对激活函数输入的导数 $\frac{\partial y_i}{\partial z_i}$ 、中间层输出对中间层输入的导数 $\frac{\partial z_i}{\partial x}$ 、中间层输出对模型参数的导数 $\frac{\partial z_i}{\partial W}$ 和 $\frac{\partial z_i}{\partial b}$

# 向量和矩阵求导方法

## ➤ Loss对网络输出的求导

◆ MSE Loss  $L = (y - y^*)^2$  求  $\frac{\partial L}{\partial y}$

- MSE Loss也是按逐元素进行操作，最后把各个结果进行求和或平均
- MSE Loss的求导也可以简单地转化为标量求导

◆ Cross Entropy Loss

- 假设类别总数为C，某样本的真实类别为第i类，

$$y^* = (y_1, y_2, \dots, y_C) \quad \text{其中 } y_j = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases} \quad j = 1, 2, \dots, C$$

- 模型输出的logits为  $z = (z_1 \quad z_2 \quad \dots \quad z_C)$

$$q_i = \text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^C e^{z_j}} \quad L = - \sum_{j=1}^C y_j \log q_j$$

- Cross Entropy Loss的计算是逐元素操作的吗？

# 向量和矩阵求导方法

## ➤ Loss对网络输出的求导

### ◆ Cross Entropy Loss

- 假设类别总数为C，某样本的真实类别为第i类，即

$$y^* = (y_1, y_2, \dots, y_C) \quad \text{其中 } y_j = \begin{cases} 1 & j = i \\ 0 & j \neq i \end{cases} \quad j = 1, 2, \dots, C$$

- 模型输出的logits为  $z = (z_1 \quad z_2 \quad \dots \quad z_C)$

$$q_i = \text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^C e^{z_j}} \quad L = - \sum_{j=1}^C y_j \log q_j$$

- 由于只有  $j = i$  时， $y_j = 1$ ，因此  $L = -\log q_i$

- 因此  $\frac{\partial L}{\partial z} = \frac{\partial L}{\partial q_i} \cdot \frac{\partial q_i}{\partial z} = -\frac{1}{q_i} \frac{\partial q_i}{\partial z}$

- 把对向量的求导转化为对多个标量求导： $\frac{\partial q_i}{\partial z} = \left( \frac{\partial q_i}{\partial z_1} \quad \frac{\partial q_i}{\partial z_2} \quad \dots \quad \frac{\partial q_i}{\partial z_C} \right)$

- 因此只需计算  $\frac{\partial q_i}{\partial z_j}$ ，并讨论  $j = i$  和  $j \neq i$  的情况

- 请推导  $\frac{\partial L}{\partial z} = \mathbf{q} - \mathbf{y}^*$

# 向量和矩阵求导方法

## ► Loss对激活函数的求导

- ◆ 激活函数通常是逐元素操作
- ◆ 因此，激活函数的求导就是标量求导

$$y = \sigma(z) = (\sigma(z_1) \quad \sigma(z_2) \quad \dots \quad \sigma(z_n))^T$$

$$\frac{\partial L}{\partial z_j} = \frac{\partial L}{\partial y_1} \cdot \frac{\partial y_1}{\partial z_j} + \frac{\partial L}{\partial y_2} \cdot \frac{\partial y_2}{\partial z_j} + \dots + \frac{\partial L}{\partial y_n} \cdot \frac{\partial y_n}{\partial z_j} = \frac{\partial L}{\partial y_j} \cdot \frac{\partial y_j}{\partial z_j}$$

回忆sigmoid函数的导数为 $\sigma'(z) = \sigma(z)(1 - \sigma(z))$ ，因此  $\frac{\partial y_j}{\partial z_j} = y_j(1 - y_j)$

$$\frac{\partial L}{\partial z} = \left( \frac{\partial L}{\partial z_1} \quad \dots \quad \frac{\partial L}{\partial z_n} \right)^T = \left( \frac{\partial L}{\partial y_1} \cdot \frac{\partial y_1}{\partial z_1} \quad \dots \quad \frac{\partial L}{\partial y_n} \cdot \frac{\partial y_n}{\partial z_n} \right) = \frac{\partial L}{\partial y} * y * (1 - y)$$

其中\*为逐元素乘积。在PyTorch程序中，\*表示逐元素乘积，torch.matmul表示矩阵乘法



# 向量和矩阵求导方法

## ► Loss对线性层输入及参数的求导

$z = W^T x + b$ , 其中  $x \in R^{m \times 1}$ ,  $z \in R^{n \times 1}$ ,  $W \in R^{m \times n}$ ,  $b \in R^{n \times 1}$

已知  $\frac{\partial L}{\partial z} = (dz_1 \quad \dots \quad dz_n)^T = \left( \frac{\partial L}{\partial z_1} \quad \dots \quad \frac{\partial L}{\partial z_n} \right)^T$ , 求  $\frac{\partial L}{\partial x}, \frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}$

不妨设  $m = 3, n = 2$ , 则  $z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + b_1 \\ w_{12}x_1 + w_{22}x_2 + w_{32}x_3 + b_2 \end{pmatrix}$

把  $\frac{\partial L}{\partial x}$  转化为对多个标量的求导, 并使用链式法则:

$$\frac{\partial L}{\partial x_1} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial x_1} + \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial x_1} = dz_1 \cdot w_{11} + dz_2 \cdot w_{12}$$

$$\frac{\partial L}{\partial x_2} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial x_2} + \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial x_2} = dz_1 \cdot w_{21} + dz_2 \cdot w_{22}$$

$$\frac{\partial L}{\partial x_3} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial x_3} + \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial x_3} = dz_1 \cdot w_{31} + dz_2 \cdot w_{32}$$

$$\frac{\partial L}{\partial x} = \begin{pmatrix} dz_1 \cdot w_{11} + dz_2 \cdot w_{12} \\ dz_1 \cdot w_{21} + dz_2 \cdot w_{22} \\ dz_1 \cdot w_{31} + dz_2 \cdot w_{32} \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{pmatrix} \cdot \begin{pmatrix} dz_1 \\ dz_2 \end{pmatrix} = W \cdot \frac{\partial L}{\partial z}$$

# 向量和矩阵求导方法

## ► Loss对线性层输入及参数的求导

$z = W^T x + b$ , 其中  $x \in R^{m \times 1}$ ,  $z \in R^{n \times 1}$ ,  $W \in R^{m \times n}$ ,  $b \in R^{n \times 1}$

已知  $\frac{\partial L}{\partial z} = (dz_1 \quad \dots \quad dz_n)^T = \left( \frac{\partial L}{\partial z_1} \quad \dots \quad \frac{\partial L}{\partial z_n} \right)^T$ , 求  $\frac{\partial L}{\partial x}, \frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}$

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial x} + \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial x} + \dots \quad \text{该式中只有 } \frac{\partial z_i}{\partial x} \text{ 是未知的, 以下来计算该导数:}$$

由  $z = W^T x + b$ , 取其第  $i$  行, 我们可以得到公式:  $z_i = \mathbf{w}_i^T \mathbf{x} + b_i$

显然上述求导为标量对向量的求导, 我们根据之前的知识得到:  $\frac{\partial z_i}{\partial x} = \mathbf{w}_i$

代入  $\frac{\partial L}{\partial x}$  的公式中我们有: 
$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial z_1} \cdot \mathbf{w}_1 + \frac{\partial L}{\partial z_2} \cdot \mathbf{w}_2 + \dots + \frac{\partial L}{\partial z_n} \cdot \mathbf{w}_n = W \cdot \frac{\partial L}{\partial z}$$

和之前假设  $m = 3, n = 2$  的结果是相同的。

矩阵乘向量  $\Leftrightarrow$  矩阵列向量的线性组合

验证  $\frac{\partial L}{\partial x}$  与  $x$  具有相同维度:  $x \in R^{m \times 1} \quad W \in R^{m \times n} \quad \frac{\partial L}{\partial z} \in R^{n \times 1} \quad W \cdot \frac{\partial L}{\partial z} \in R^{m \times 1}$

# 向量和矩阵求导方法

## ► Loss对线性层输入及参数的求导

$$z = W^T x + b, \text{ 其中 } x \in R^{m \times 1}, z \in R^{n \times 1}, W \in R^{m \times n}, b \in R^{n \times 1}$$

$$\text{已知 } \frac{\partial L}{\partial z} = (dz_1 \quad \dots \quad dz_n)^T = \left( \frac{\partial L}{\partial z_1} \quad \dots \quad \frac{\partial L}{\partial z_n} \right)^T, \text{ 求 } \frac{\partial L}{\partial x}, \frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}$$

$$\text{不妨设 } m = 3, n = 2, \text{ 则 } z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + b_1 \\ w_{12}x_1 + w_{22}x_2 + w_{32}x_3 + b_2 \end{pmatrix}$$

把  $\frac{\partial L}{\partial W}$  转化为对多个标量的求导，并使用链式法则：

$$\frac{\partial L}{\partial w_{11}} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_{11}} + \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_{11}} = dz_1 \cdot x_1 \quad \frac{\partial L}{\partial w_{12}} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_{12}} + \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_{12}} = dz_2 \cdot x_1 \quad \dots$$

$$\frac{\partial L}{\partial W} = \begin{pmatrix} dz_1 \cdot x_1 & dz_2 \cdot x_1 \\ dz_1 \cdot x_2 & dz_2 \cdot x_2 \\ dz_1 \cdot x_3 & dz_2 \cdot x_3 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \cdot (dz_1 \quad dz_2) = x \cdot \left( \frac{\partial L}{\partial z} \right)^T$$

# 向量和矩阵求导方法

## ► Loss对线性层输入及参数的求导

$z = W^T x + b$ , 其中  $x \in R^{m \times 1}$ ,  $z \in R^{n \times 1}$ ,  $W \in R^{m \times n}$ ,  $b \in R^{n \times 1}$

已知  $\frac{\partial L}{\partial z} = (dz_1 \ \dots \ dz_n)^T = \left( \frac{\partial L}{\partial z_1} \ \dots \ \frac{\partial L}{\partial z_n} \right)^T$ , 求  $\frac{\partial L}{\partial x}, \frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial W} + \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial W} + \dots \quad \text{该式中只有 } \frac{\partial z_i}{\partial W} \text{ 是未知的, 以下来计算该导数:}$$

由  $z = W^T x + b$ , 取其第  $i$  行, 我们可以得到公式:  $z_i = \mathbf{w}_i^T \mathbf{x} + b_i$

矩阵  $\Leftrightarrow$  列向量乘行向量

显然上述求导为标量对向量的求导, 我们根据之前的知识得到:

$$\frac{\partial z_i}{\partial \mathbf{w}_i} = \mathbf{x}, \quad \frac{\partial z_i}{\partial \mathbf{w}_j} = \mathbf{0}, \quad \forall j \neq i \quad \Longrightarrow \quad \frac{\partial z_i}{\partial W} = [\mathbf{0}, \mathbf{0}, \dots, \mathbf{x}, \dots \mathbf{0}] = \mathbf{x} * \mathbf{e}_n^T$$

$$\text{代入 } \frac{\partial L}{\partial W} \text{ 的公式中我们有: } \frac{\partial L}{\partial W} = \frac{\partial L}{\partial z_1} \cdot \mathbf{x} \cdot \mathbf{e}_1^T + \frac{\partial L}{\partial z_2} \cdot \mathbf{x} \cdot \mathbf{e}_2^T + \dots + \frac{\partial L}{\partial z_n} \cdot \mathbf{x} \cdot \mathbf{e}_n^T = \mathbf{x} \cdot \left( \frac{\partial L}{\partial z} \right)^T$$

$$\text{验证 } \frac{\partial L}{\partial W} \text{ 与 } W \text{ 具有相同维度: } W \in R^{m \times n} \quad x \in R^{m \times 1} \quad \frac{\partial L}{\partial z} \in R^{n \times 1} \quad x \cdot \left( \frac{\partial L}{\partial z} \right)^T \in R^{m \times n}$$

# 向量和矩阵求导方法

## ► Loss对线性层输入及参数的求导

$z = W^T x + b$ , 其中  $x \in R^{m \times 1}, z \in R^{n \times 1}, W \in R^{m \times n}, b \in R^{n \times 1}$

已知  $\frac{\partial L}{\partial z} = (dz_1 \quad \dots \quad dz_n)^T = \left( \frac{\partial L}{\partial z_1} \quad \dots \quad \frac{\partial L}{\partial z_n} \right)^T$ , 求  $\frac{\partial L}{\partial x}, \frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}$

不妨设  $m = 3, n = 2$ , 则  $z = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{21} & w_{31} \\ w_{12} & w_{22} & w_{32} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} w_{11}x_1 + w_{21}x_2 + w_{31}x_3 + b_1 \\ w_{12}x_1 + w_{22}x_2 + w_{32}x_3 + b_2 \end{pmatrix}$

把  $\frac{\partial L}{\partial b}$  转化为对多个标量的求导, 并使用链式法则:

$$\frac{\partial L}{\partial b_1} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_1} + \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial b_1} = dz_1 \quad \frac{\partial L}{\partial b_2} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial b_2} + \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial b_2} = dz_2$$

$$\frac{\partial L}{\partial b} = \begin{pmatrix} dz_1 \\ dz_2 \end{pmatrix} = \frac{\partial L}{\partial z}$$

- 本次作业中, 需要计算  $x$  和  $y$  为行向量情况下的导数计算
- 使用行向量的原因: 便于批量处理, 输入  $N$  个样本, 数据矩阵  $X \in R^{N \times m}, Y \in R^{N \times n}$
- 方法并无二致: 学会标量化, 行/列/矩阵都不怕

# 向量和矩阵求导方法

## ► Loss对线性层输入及参数的求导

$z = W^T x + b$ , 其中  $x \in R^{m \times 1}$ ,  $z \in R^{n \times 1}$ ,  $W \in R^{m \times n}$ ,  $b \in R^{n \times 1}$

已知  $\frac{\partial L}{\partial z} = (dz_1 \quad \dots \quad dz_n)^T = \left( \frac{\partial L}{\partial z_1} \quad \dots \quad \frac{\partial L}{\partial z_n} \right)^T$ , 求  $\frac{\partial L}{\partial x}, \frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}$

$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial z_1} \cdot \frac{\partial z_1}{\partial b} + \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial b} + \dots$  该式中只有  $\frac{\partial z_i}{\partial b}$  是未知的, 以下来计算该导数:

由  $z = W^T x + b$ , 取其第  $i$  行, 我们可以得到公式:  $z_i = \mathbf{w}_i^T \mathbf{x} + b_i$

显然上述求导为标量对标量的求导, 我们根据之前的知识得到:

$$\frac{\partial z_i}{\partial b_i} = 1, \quad \frac{\partial z_i}{\partial b_j} = 0, \quad \forall j \neq i \quad \Longrightarrow \quad \frac{\partial z_i}{\partial \mathbf{b}} = [0, 0, \dots, 1, \dots, 0]^T = \mathbf{e}_i$$

代入  $\frac{\partial L}{\partial b}$  的公式中我们有:

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial z_1} \cdot \mathbf{e}_1 + \frac{\partial L}{\partial z_2} \cdot \mathbf{e}_2 + \dots + \frac{\partial L}{\partial z_n} \cdot \mathbf{e}_n = \frac{\partial L}{\partial z}$$

- 可以看到使用标量对向量和矩阵的求导可以更直观和简洁的得到推导结果
- 不要求掌握只要求了解标量对向量和矩阵的简单求导。(关键:  $\frac{\partial L}{\partial b}$  与  $\mathbf{b}$  具有相同维度)

## 四、第二次作业说明

### ➤ 理论部分：

- ◆ 选择题：5道，涉及神经网络的优化及正则化
- ◆ 计算题：1道，涉及误差反向传播方法

### ➤ 编程部分：

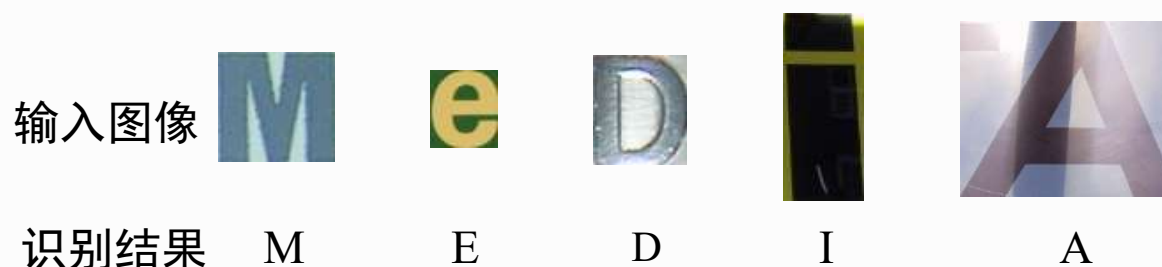
- ◆ 完成基于多层感知机的非线性分类器代码，并进行训练、测试和可视化
- ◆ 撰写作业报告：
  - 随作业发布的LaTeX模板打包文件为HW2-template.zip
  - 鼓励同学们使用作业文档的LaTeX模板完成报告
- ◆ 自选课题进展报告（选择自选课题替代编程作业的同学）



# 第二次作业的非线性分类任务

## ► 英文字符图像识别

- ◆ 输入英文字符图像，输出图像中包含的英文字符



- ◆ 多分类问题：可以采用多层感知机(Multi-Layer Perceptron, MLP)进行分类
  - MLP的输入应该为1维向量，而非二维图像
  - 解决方案：把图像拉直成向量作为MLP的输入
  - 例如，大小为 $32 \times 32$ 的图像，拉成 $1 \times 1024$ 的向量作为输入 $x$

# 第二次作业需要完成的代码

## ➤ 需要完成的代码

- ◆ 损失函数、线性层的前向传播和反向传播过程
- ◆ 使用多个线性层和激活函数层构建MLP
- ◆ 模型预测(predict)过程
- ◆ PyTorch中如何自定义反向传播过程?
  - 使用torch.autograd.Function方法
  - 详情可参考

[https://pytorch.org/tutorials/beginner/examples\\_autograd/two\\_layer\\_net\\_custom\\_function.html](https://pytorch.org/tutorials/beginner/examples_autograd/two_layer_net_custom_function.html)

# 第二次作业需要完成的代码

## ◆ PyTorch中如何自定义反向传播过程？

```
class MyReLU(torch.autograd.Function):
```

```
    """
```

```
    We can implement our own custom autograd Functions by subclassing
    torch.autograd.Function and implementing the forward and backward passes
    which operate on Tensors.
    """
```

```
    @staticmethod
```

```
    def forward(ctx, input):
```

```
        """
```

```
        In the forward pass we receive a Tensor containing the input and return
        a Tensor containing the output. ctx is a context object that can be used
        to stash information for backward computation. You can cache arbitrary
        objects for use in the backward pass using the ctx.save_for_backward method.
        """
```

```
        ctx.save_for_backward(input)
```

```
        return input.clamp(min=0)
```

```
    @staticmethod
```

```
    def backward(ctx, grad_output):
```

```
        """
```

```
        In the backward pass we receive a Tensor containing the gradient of the loss
        with respect to the output, and we need to compute the gradient of the loss
        with respect to the input.
        """
```

```
        input, = ctx.saved_tensors
```

```
        grad_input = grad_output.clone()
```

```
        grad_input[input < 0] = 0
```

```
        return grad_input
```

静态方法：允许程序在不初始化类对象的情况下直接调用该函数

例如：y = MyReLU.apply(x)

ctx保存forward()中的变量

ctx是context的缩写，翻译成“上下文；环境”；

python类函数都要带self参数，但是在静态方法中，使用ctx参数而不是self

ctx可以保存forward()中的变量，以便在backward()中继续使用。

ctx读取forward()中保存的变量

# 搭建网络的细节

- 除了使用`nn.CrossEntropyLoss`类以外，`CrossEntropyLoss` 还可以如何实现？
  - ◆ `CrossEntropyLoss`中包括了对输出层logits进行归一化的softmax激活函数
  - ◆ 利用预测的类别与类别真值计算损失

# 第二次作业需要完成的代码

## ➤ 需要完成的代码

- ◆ 损失函数、线性层的前向传播和反向传播过程
- ◆ 使用多个线性层和激活函数层构建MLP
- ◆ 模型预测(predict)过程
- ◆ PyTorch中如何完成多个层的组合？
  - 使用nn.Sequential()
  - 下列代码构建了一个两层MLP

网络的构建：

```
num_inputs, num_outputs, num_hiddens = 784, 10, 256
net = nn.Sequential(
    nn.Linear(num_inputs, num_hiddens),
    nn.ReLU(),
    nn.Linear(num_hiddens, num_outputs),
)
```

网络的调用：`y = net(x)`

## 第二次作业需要完成的代码

### ◆ PyTorch中如何完成多个层的组合？

- 联合使用nn.ModuleList()和nn.Sequential()能更灵活地构建网络

```
num_inputs, num_outputs, num_hiddens = 784, 10, 256
net = nn.Sequential(
    nn.Linear(num_inputs, num_hiddens),
    nn.ReLU(),
    nn.Linear(num_hiddens, num_outputs),
)
```

等价于：

```
num_inputs, num_outputs, num_hiddens = 784, 10, 256
layers = nn.ModuleList()
layers.append(nn.Linear(num_inputs, num_hiddens))
layers.append(nn.ReLU())
layers.append(nn.Linear(num_hiddens, num_outputs))

net = nn.Sequential(*layers)
```

```
>>> a = [1, 2, 3]
>>> print(a)
[1, 2, 3]
>>> print(*a)
1 2 3
```

Python列表前使用星号\*表示  
将列表解开成独立的元素作为形参

# 第二次作业需要完成的代码

- ◆ 本次作业中，为了便于自定义不同的层数、节点数、激活函数，我们使用nn.ModuleList()方法

```
# initialize a list to save layers
layers = nn.ModuleList()

if n_layers == 1:
    # if n_layers == 1, MLP degenerates to a Linear layer
    layer = Linear(input_size, output_size)
    # append the layer into layers
    layers.append(layer)
    layers.append(self.act)

# TODO 4: Finish MLP with at least 2 layers
else:
    # step 1: initialize the input layer
    # step 2: append the input layer and the activation layer into layers

    # step 3: construct the hidden layers and add it to layers
    for i in range(1, n_layers - 1):
        # initialize a hidden layer and activation layer
        # hint: Noting that the output size of a hidden layer is hidden_size[i], so what is its input size?

    # step 4: initialize the output layer and append the layer into layers
    # hint: what is the output size of the output layer?
    # hint: here we do not need activation layer

# End TODO 4

# Use nn.Sequential to get the neural network
self.net = nn.Sequential(*layers)
```

建立一个ModuleList对象

初始化输入层

使用循环灵活地为layers中添加隐含层

初始化各个隐含层

初始化输出层

将layers中的各层组合成整体网络



## 第二次作业相关文件

### ► 对于可以复用的模块，可以单独写成Python文件

- ◆ 在编写特定任务时，从这些文件中调用写好的模块
- ◆ 在工程较为复杂时，把不同模块分文件写，可以提高代码可读性，提高开发效率

本次作业文件结构：



调用losses.py和network.py中的类：  
`from network import MLP`  
`from losses import CrossEntropyLoss`

提示：模型测试阶段的可视化部分使用t-SNE (t-distributed stochastic neighbor embedding) 对特征降维，需要安装sklearn：

`conda install scikit-learn` 或者 `pip install scikit-learn`

## 第二次作业主程序的命令行参数

### ➤ 使用argparser程序库方便地管理模型参数、训练及测试参数

- ◆ 导入argparser库: `import argparse`

- ◆ 定义可调节参数

```
parser = argparse.ArgumentParser() → 初始化一个ArgumentParser类, 命名为parser  
parser.add_argument('--mode', type=str, default='train', help='train or predict')
```

为parser添加一个名为mode的参数, 类型为字符串, 缺省值为train

```
opt = parser.parse_args() → 将parser转化为可在程序内调用的对象, 命名为opt  
opt中包含一个名为mode的属性
```

```
if opt.mode == 'train':
```

```
    # run training process
```

```
elif opt.mode == 'predict': → 可以根据opt.mode方便地调整程序运行模式
```

```
    # run predicting process
```

- ◆ 在命令行 (Anaconda Prompt) 中运行:

`python filename.py --mode train` 或 `python filename.py` 运行训练程序

`python filename.py --mode predict` 运行预测程序

# 第二次作业主程序的命令行参数

## ◆ 本次作业程序的命令行参数说明：

```
parser = argparse.ArgumentParser()
parser.add_argument('--mode', type=str, default='train', help='train, test or predict')
parser.add_argument('--im_dir', type=str, default='data/character_classification/images',
                    help='path to directory with images')
parser.add_argument('--train_file_path', type=str, default='data/character_classification/train.json',
                    help='file list of training image paths and labels')
parser.add_argument('--val_file_path', type=str, default='data/character_classification/validation.json',
                    help='file list of validation image paths and labels')
parser.add_argument('--test_file_path', type=str, default='data/character_classification/test.json',
                    help='file list of test image paths and labels')
parser.add_argument('--batchsize', type=int, default=8, help='batch size')
parser.add_argument('--device', type=str, default='cpu', help='cpu or cuda')

# configurations for training
parser.add_argument('--hsize', type=str, default='32', help='hidden size for each hidden layer, splitted by comma')
parser.add_argument('--layer', type=int, default=2, help='number of layers in the MLP')
parser.add_argument('--act', type=str, default='relu',
                    help='type of activation function, can be none, sigmoid, tanh, or relu')
parser.add_argument('--norm_size', type=tuple, default=(32, 32), help='image normalization size, (height, width)')
parser.add_argument('--epoch', type=int, default=50, help='number of training epochs')
parser.add_argument('--n_classes', type=int, default=26, help='number of classes')
parser.add_argument('--valInterval', type=int, default=10, help='the frequency of validation')
parser.add_argument('--lr', type=float, default=5e-4, help='learning rate')
parser.add_argument('--optim_type', type=str, default='sgd', help='type of optimizer, can be sgd, adagrad, rmsprop, adam, or adadelat')
parser.add_argument('--momentum', type=float, default=0.9, help='momentum of the SGD optimizer, only used if optim_type is sgd')
parser.add_argument('--weight_decay', type=float, default=0., help='the factor of L2 penalty on network weights')

# configurations for test and prediction
parser.add_argument('--model_path', type=str, default='saved_models/recognition.pth', help='path of a saved model')
parser.add_argument('--im_path', type=str, default='data/character_classification/new_images/predict01.png',
```

谢谢大家！