***python:***

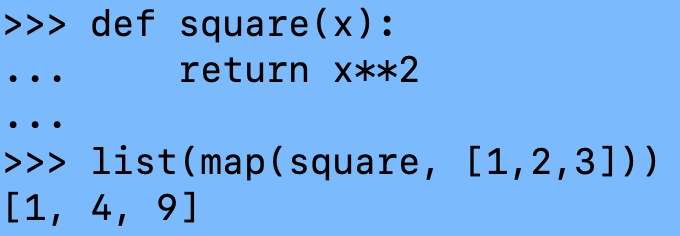
1. **map()**

:根据提供的函数, 为指定序列做映射.

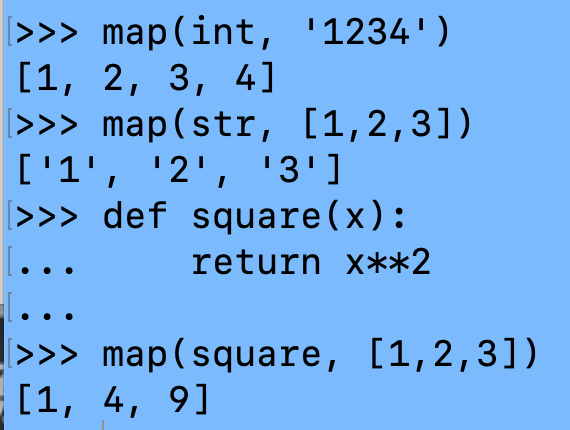
:接收: **一个函数**(特定的数据类型, 或一个function)**, 可迭代对象**(list, string之类的..,**一个或多个**)

:map(function, iterable, ...)

*python3:*



*python2:*

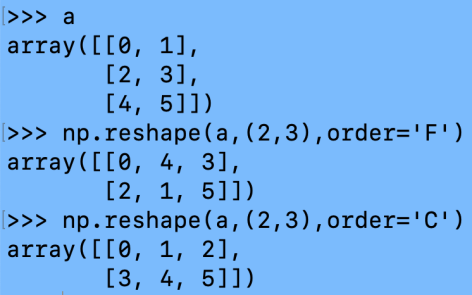


1. **transpose(), reshape()**

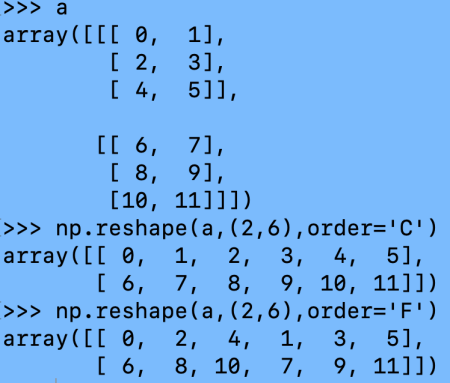
transpose: *矩阵的转置*

reshape: *先打散, 再按设置的顺序插入*

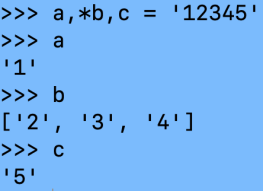
: np.reshape(a, (2, 3), order='F') [*order是打散+后续插入的顺序*, C 是按行顺序, F 是按列顺序, A 是按数据在内存中存储的顺序]

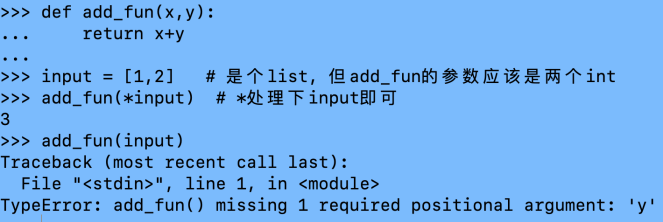


***更一般的理解：***C 顺序是从最里面的轴开始读+写, F顺序是从最外面的轴开始读+写.



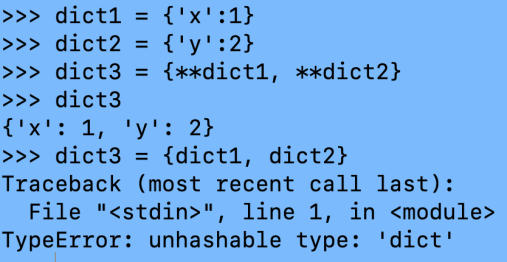
1. ***\*星号***表达式: 任意长度的可迭代数据.



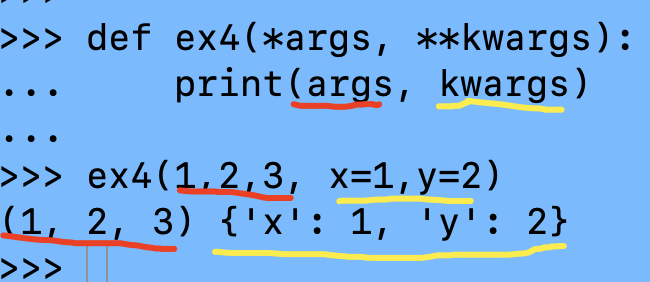


1. ***双\*星号***

合并两个字典:



在函数参数中的使用, 也是辅助完成dict的一些传递, 打包:



1. with: 处理*上下文环境,* 和一些*产生的异常* (with open(‘./red.txt’)) 可实现读入然后自行退出[enter()+exit()], and, 是否正常读入了, 无需手动加try检查.
2. 函数参数中的, 冒号: def Sum(num1: int, num2: int=100) -> int:

[传参类型的建议, 而已..]

1. lambda
2. 下划线

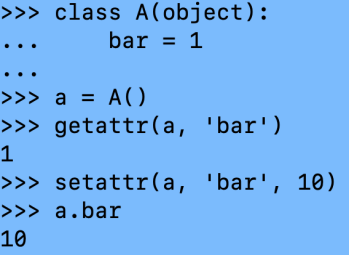
\_example: 表示该内容受保护, 如果是变量或函数, 在from some\_module import \* 事后则不会被导入. 如果是成员或方法, 仅允许类内部使用, 类的子类继承.

\_\_example: 表示该变量、函数、成员or方法私有, 无法以任何方式被外部使用, 也无法被子类继承. 本类可用a\_instance.\_ClassName\_\_MethodName()来调用.

\_\_example\_\_: 表示python里的特殊方法

example\_: 避免和Python关键词冲突, 无特殊含义

1. 类的内置函数: getattr() setattr()

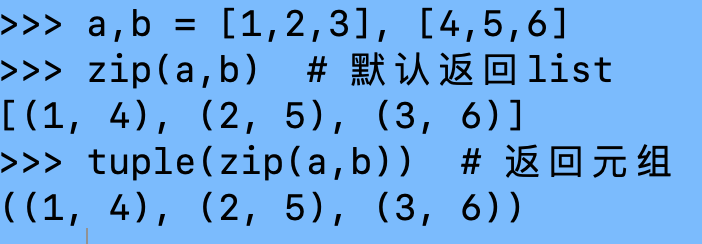


1. 数据类型:

不可变: string(ta可以做为dict的key啊), 元组,

可变: list, dict, set

1. argparse: 参数设置模块
2. zip: python2示例



1. 装饰器

<https://zhuanlan.zhihu.com/p/269012332>

/Users/chenjia/Downloads/Smartmore/2022/daydayup/python\_jicheng\_shujuleixing/decorator.py

1. 多线程和多进程: io密集多线程, cpu密集多进程 [python的GIL等待机制导致的]
2. 闭包
3. 装饰器
4. 内存管理

C++:

1. const define的区分

2. malloc和new的区别

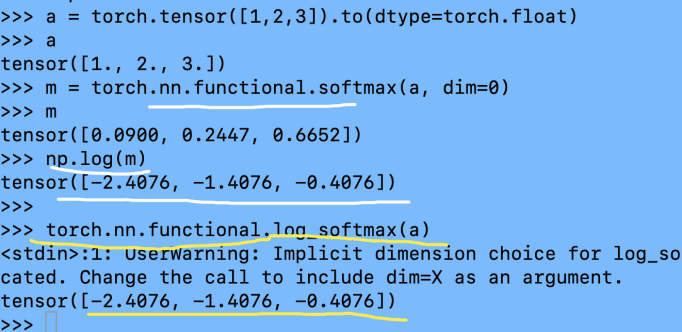
3. 虚函数

4. 结构体与类的区别

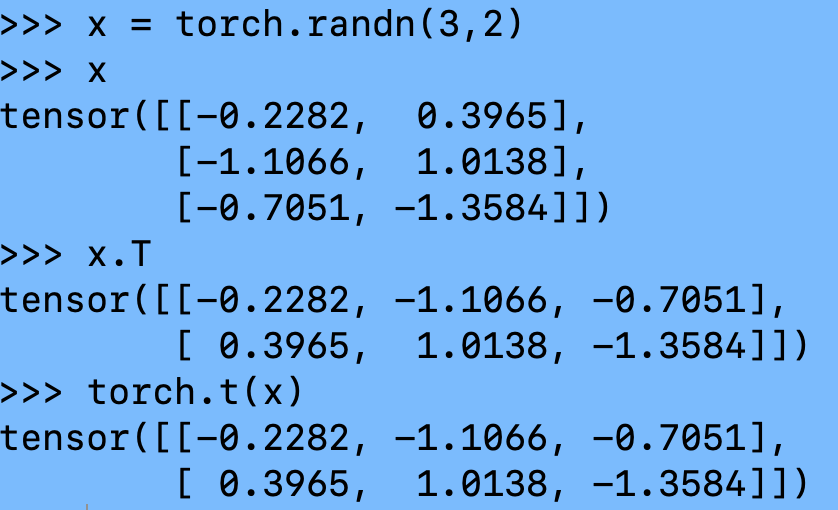
5. 死锁

pytorch:

1. torch.zeros() or torch.Tensor() 默认数值类型: float32
2. torch.nn.functional.softmax + np.log() 等于 torch.nn.functional.log\_softmax(a)



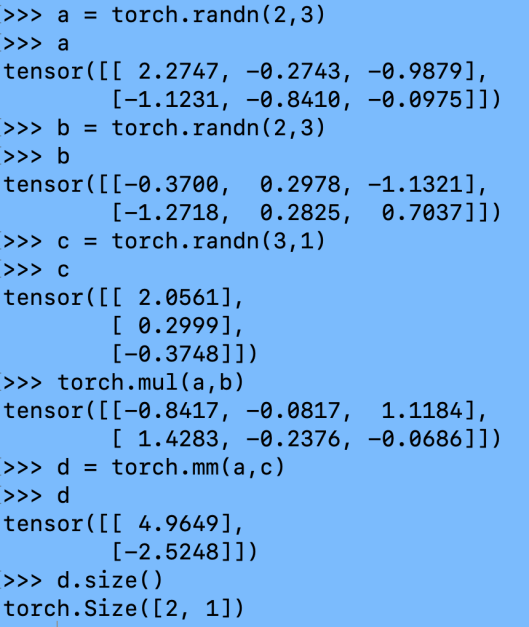
1. torch.t(x) 等于 x.T 均为转置 处理



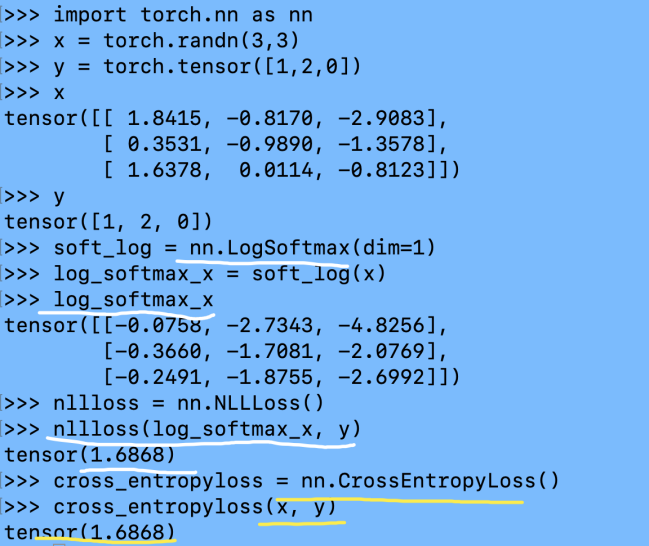
1. 矩阵 乘

torch.mul(a, b) a, b对应位置点对点乘, a, b 维度必须一致

torch.mm(a, b) 就是矩阵乘法了, a, b维度: [x, y]和[y, z] -> [x, z]



1. 损失函数: nn.CrossEntropy() 等于 nn.LogSoftmax() + nn.NLLLoss()



**多gpu训练**

1. model = nn.DataParallel(model) model.to(device) [慢]

2. nn.DistributedDataParallel [推荐, 也是主流开源codebase的写法..~]

1. **多线程数据读入**

*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)

1. **DataLoader(可迭代数据装饰器) 和 DataSet(读数据的类)**

DataSet: 根据**数据索引**, 读取数据和标签

DataLoader: 根据定义好的DataSet, **构建可迭代的数据装载器**. 处理后数据即可train, eval了.

1. **detach(), detach\_(), .data:**

detach(): 截断梯度, **返回一个无梯变量/tensor** (*即使重新将它的requires\_grad=true, 它也不会再具有梯度*) and (x->y->z, y.detach()*只让y无梯, x的梯还是可以传递给z的.*.)

.data: 类似引用的概念 *可强行求导但可能是错误的导*..

detach\_(): 梯断裂... *y.detach\_()使得x向z的传播直接失败.*. [因为是inplace处理, 直接原地修改]

***detach\_()是对本身的修改in-place操作, detach()是生成了一个新tensor***

***[直接改变tensor的操作, 都包含以下下划线的. \_ ]***

1. **nn.functional和nn.models的区别**

两者都可定义layer层, 但:

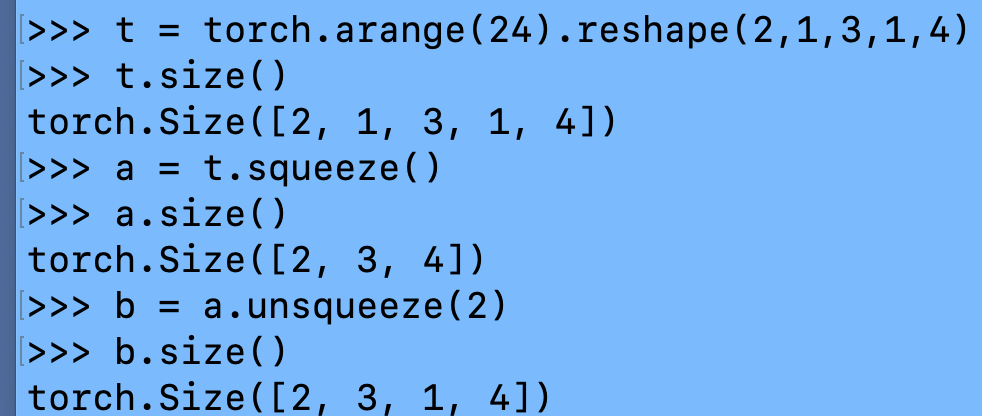
1. nn.functional实现的layer是一个函数, 可以直接调用不需要实例化, 无可学习参数.
2. nn.Module实现的是类, 继承父类nn.Module, 内部自动实现了forward功能, 可自动提取要学习的参数.

so: 有可学习参数的, 用nn.Module定义更方便, 无需自动传入params

无可学习参数的, 两者都可用.

**BN, dropout等这些train, eval不一致的层, 用nn.Module也更方便**(可根据nn.train(), nn.eval()识别出状态的不同.)

1. **squeeze()消除1维度, unsqueeze()添加1维度**

****

**DL base:**

1. leaky relu和relu:

relu: 小于0的值直接置0, 会导致部分神经元在训练中被"杀死", 但这个特点也达到: 特征稀疏的作用.

leaky relu: 负数值scale下: -10 变 -5. 没有绝对的好坏, 这俩激活函数, 根据场景而定.

2. label smooth作用和原理:

分类来说, 一定程度上缓解过拟合(对于样本间严重不平衡的时候可轻微缓解.)

3. 移动端部署, 网络压缩和加速

4. loss nan 怎么定位和解决

先查数据是否有问题; 再code bug, 网络层写错啥之类的; 再是训练方式: 初始化方式不当, bn没加, 学习率过大, 损失函数用的不当等;

最后: 可能除以了0, 考虑是网络过程中出现了0值, 定位到对应去改;

5. 7\*7卷积和3个3\*3卷积的使用场景有什么不同

7x7大卷积, 可作为大图的特征提取conv, 常在网络的第一层使用. 感受野一下子就上来了. (ConvNeXt, resnet的第一层)

多几个3x3可近似7x7, 计算量更少(中间的激活次数还更多增强非线性)

**计算机视觉:**

1. yolo系的优势:

快, pipline轻, 实时性不错, 工业上部署也较成熟.

2. yolox:

1. 添加EMA权值更新, Cosine学习率机制等训练技巧

2. 使用IOU loss做reg优化, BCE loss训cls

3. 添加RandomHorizontalFlip, ColorJitter, 多尺度数据增广, 移除RandomResizedCrop

3. two-stage效果比one-stage好的原因:

1. 一定程度在RPN节点, 先解决了部分样本不平衡问题(正负的个数, 难度差异)

2. 有点cascade的意思, RPN筛选了一遍好的proposal, 然后还ROI pooling, 再做cls类别细分和reg.

(cls细分相对one-stage也会精度更好些的.)

3. two的小目标效果会更好(one的input size开太大容易OOM吃不住, 这直接就影响了小目标的精度. 把输入放大是有助于小目标检出的!)

4. 实现focal loss: -alpha\*(torch.pow((1-probs), self.gamma))\*log\_p

ce loss: loss = -log(pt)

focal loss: loss = -alpha\*(1-pt)^gamma \* log(pt)

5. 针对小目标检测的改进:

6. attention如何计算, self attention除以根号k的原因:

attention计算: kq三个向量做"相关性"计算,再加权到v上.

self attention除以根号k: 主要为了scale, 原因:

1. 内积值可能很大, 不normalize下的话计算量会较大; 2. 内积值太大送给softmax,可能导致梯度很小.

[softmax的梯度函数: (a: S(xi)\*(1-S(xi) or b: (-S(xi)\*S(xj))). 先看a, xi太大了, S(xi)趋近于1, 则a值趋近0; 再看b, 当各xi,j间方差很大, 则xi与xj要么一个0要么一个1, b值仍会趋于0.].

So除以sqrt(dk), 把内积方差值化为1,就可解决上面俩"隐患".

7. 用LN（layer norm）的原因:

因为样本长度不一致, 用BN需要把各个句子padding到size一致, 添加太多冗余信息.

另外, 一个句子在内部做统计量抓取就可了, 句子本身就有独立性和分布特性, 无需跨越不用样本(尤其样本间差异很大)

8. swin transformer:

1. 在局部小window内做attention, 计算量从图像分辨率的平方倍优化至线性倍; 2. 使用shifted层级结构+patch merging, 实现多层级多感受野目的(第二点也是为分割,检测等密集型预测任务做的必须行优化).

9. transformer编码顺序信息:

用sin, cosin实现每个句子中词index等价线性, 不受句子长度影响. 4个词的句子和10个词的句子, index=2处的编码力度一致.

10. transformer一般用什么优化器:

建议选择Adamw(搭配大点的learning rate decay), Adam, 毕竟数据需求大且显存要求高.

理论上: AdamW=Adam+Weight Decay

11. 双摄像头如何实现3d检测:

**ML：**

***PCA:*** 降维算法, 目的: 数据间**各个维度上方差够大**, **维度降低**了.

1. 去中心归一化
2. 求协方差矩阵
3. 求协方差矩阵的特征值, 特征向量
4. 按特征值大小, 排序特征向量, 取topk, 形成矩阵P
5. new\_X = P \* X (完成了数据X降维到k维..)

***图像下采样:*** *pooling和卷积, 均可.(设置好stride即可)*

***检测:***