# python知识

# 一、pandas

## 1.pandas读取数据

python下pandas库中读取指定行或列数据（excel）_pandas 读取指定列-CSDN博客

Pandas读取csv数据-CSDN博客

史上最全！用Pandas读取CSV，看这篇就够了-CSDN博客史上最全！用Pandas读取CSV，看这篇就够了-CSDN博客

### 1.1 读取数据集

```
import pandas as pd
data=pd.read_csv('city.csv')#city.csv文件地址
#可以指定name参数，来给我们的列表的不同列命名
#data=pd.read_csv('city.csv',names=['A','B','C'])
print(data)
```

这是我们的data指定names之后的内容，可以发现我们的2018年到2016年全部为A，2015年为B，2014年为C

| | | | A | B | C |
|---|---|---|---|---|---|
| 城市 | 2018年 | 2017年 | 2016年 | 2015年 | 2014 |
| 北京 | 30319.98 | 28014.94 | 25669.13 | 23014.59 | 21330.83 |
| 天津 | 18809.64 | 18549.19 | 17885.39 | 16538.19 | 15726.93 |
| 石家庄 | NaN | 6460.88 | 5927.73 | 5440.6 | 5170.27 |
| 太原 | NaN | 3382.18 | 2955.6 | 2735.34 | 2531.09 |
| 呼和浩特 | NaN | 2743.72 | 3173.59 | 3090.52 | 2894.05 |
| 沈阳 | NaN | 5864.97 | 5460.01 | 7272.31 | 7098.71 |
| 大连 | NaN | 7363.92 | 6730.33 | 7731.64 | 7655.58 |
| 长春 | NaN | 6530.03 | 5917.94 | 5530.03 | 5342.43 |
| 哈尔滨 | NaN | 6355.05 | 6101.61 | 5751.21 | 5340.07 |
| 上海 | 32679.87 | 30632.99 | 28178.65 | 25123.45 | 23567.7 |
| 南京 | NaN | 11715.1 | 10503.02 | 9720.77 | 8820.75 |
| 杭州 | NaN | 12603.36 | 11313.72 | 10050.21 | 9206.16 |
| 宁波 | NaN | 9842.1 | 8686.49 | 8003.61 | 7610.28 |
| 合肥 | NaN | 7213.45 | 6274.38 | 5660.27 | 5157.97 |
| 福州 | NaN | 7103.4 | 6197.64 | 5618.08 | 5169.16 |

**index_col用来指定索引列**，

```
import pandas as pd
#指定行索引
data='city.csv'
data=pd.read_csv(data,index_col='2018年')
pd.read_csv(data, index_col=0) # 第几列是索引
pd.read_csv(data, index_col=['a','b']) # 多个索引
pd.read_csv(data, index_col=[0, 3]) # 按列索引指定多个索引
print(data)
```

```
              城市     2017年     2016年     2015年     2014
2018年
30319.98     北京   28014.94   25669.13   23014.59   21330.83
18809.64     天津   18549.19   17885.39   16538.19   15726.93
NaN         石家庄    6460.88    5927.73    5440.60    5170.27
NaN          太原    3382.18    2955.60    2735.34    2531.09
NaN        呼和浩特    2743.72    3173.59    3090.52    2894.05
NaN          沈阳    5864.97    5460.01    7272.31    7098.71
NaN          大连    7363.92    6730.33    7731.64    7655.58
NaN          长春    6530.03    5917.94    5530.03    5342.43
NaN         哈尔滨    6355.05    6101.61    5751.21    5340.07
32679.87     上海   30632.99   28178.65   25123.45   23567.70
NaN          南京   11715.10   10503.02    9720.77    8820.75
NaN          杭州   12603.36   11313.72   10050.21    9206.16
```

## 只读取部分列

```
#如果只使用数据的部分列，可以用usecols来指定
# 支持类似列表的序列和可调用对象
# 读取部分列
pd.read_csv(data, usecols=[0,4,3]) # 按索引只读取指定列，与顺序无关
pd.read_csv(data, usecols=['列1', '列5']) # 按列名，列名必须存在
# 以下用callable方式可以巧妙指定顺序，in后面的是我们要的顺序
pd.read_csv(data, usecols=lambda x: x.upper() in ['COL3', 'COL1'])
```

## 将指定列置空

```
import pandas as pd
data=pd.read_csv('city.csv',na_values=[28014.94,25669.13])
print(data)
```

| | 城市 | 2018年 | 2017年 | 2016年 | 2015年 | 2014年 |
|---|---|---|---|---|---|---|
| 0 | 北京 | 30319.98 | NaN | NaN | 23014.59 | 21330.83 |
| 1 | 天津 | 18809.64 | 18549.19 | 17885.39 | 16538.19 | 15726.93 |
| 2 | 石家庄 | NaN | 6460.88 | 5927.73 | 5440.60 | 5170.27 |
| 3 | 太原 | NaN | 3382.18 | 2955.60 | 2735.34 | 2531.09 |
| 4 | 呼和浩特 | NaN | 2743.72 | 3173.59 | 3090.52 | 2894.05 |
| 5 | 沈阳 | NaN | 5864.97 | 5460.01 | 7272.31 | 7098.71 |
| 6 | 大连 | NaN | 7363.92 | 6730.33 | 7731.64 | 7655.58 |

## 1.2 保存数据集

```
data.to_csv('out.csv')#保存数据集
```

## 1.3日期时间解析

### 1.3.1 创建范围日期

使用pandas的date_range方法创建指定范围的日期 指定开始日期start和结束日期end创建范围类的日期

```
pandas.date_range(start=None, end=None, periods=None, freq='D',
tz=None, normalize=False, name=None, closed=None, **kwargs)
```

- start：string 或 datetime-like，默认值是 None，表示日期的起点。
- end：string 或 datetime-like，默认值是 None，表示日期的终点。
- periods：integer 或 None，默认值是 None，表示你要从这个函数产生多少个日期索引值；如果是 None 的话，那么 start 和 end 必须不能为 None。
- freq：string 或 DateOffset，默认值是 'D'，表示以自然日为单位，这个参数用来指定计时单位，比如 '5H' 表示每隔 5 个小时计算一次。
- tz：string 或 None，表示时区，例如：'Asia/Hong_Kong'。
- normalize：bool，默认值为 False，如果为 True 的话，那么在产生时间索引值之前会先把 start 和 end 都转化为当日的午夜 0 点。
- name：str，默认值为 None，给返回的时间索引指定一个名字。

- closed：string 或者 None，默认值为 None，表示 start 和 end 这个区间端点是否包含在区间内，可以有三个值，'left' 表示左闭右开区间，'right' 表示左开右闭区间，None 表示两边都是闭区间。

```
date_list = ["2021-11-01", "2021-11-05"]
date_range = pd.date_range(date_list[0], date_list[1])
print(date_range)
```

一些串别名被赋予有用的常见时间序列频率。我们将把这些别名称为偏移别名。

```
B       工作日频率
D       日历日频率
W       每周频率
W-SUN   星期天为起始(Sundays). 等同 ‘W’
W-MON   星期一为起始(Mondays)
W-TUE   星期二为起始(Tuesdays)
W-WED   星期三为起始(Wednesdays)
W-THU   星期四为起始(Thursdays)
W-FRI   星期五为起始(Fridays)
W-SAT   星期六为起始(Saturdays)
WOM     每月的第几周的第几天

BH      工作小时级频率
H       小时级频率
T,min   分钟级频率
S       秒级频率
L,ms    毫秒
U,us    微秒
N       纳秒

M       月结束频率，如'2018-11-30', '2018-12-31'
SM      半月结束频率（15日和月末）
BM      工作月结束频率
MS      月起始频率，如'2018-12-01'
SMS     短信半月开始频率（第1和第15）
BMS     工作月份开始频率

Q       季末频率
BQ      工作季度结束频率
QS      季度开始频率
BQS     工作季度开始频率

A,Y     年结束频率，如'2000-12-31'
A-DEC   年级别的频率，以 十二月（December） 为年底，等同 'A'
A-JAN   年级别的频率，以 一月（January ）为年底。（其他二到十一月：FEB、MAR、APR、MAY、
JUN、JUL、AUG、SEP、OCT、NOV）
BA,BY   工作年度结束频率
AS,YS   年开始频率
BAS,BYS 工作年度开始频率
```

Pandas学习笔记（二）—— Pandas索引_pandas loc(lambda)-CSDN博客

# 1.4.索引

**一个完整的切片是包含三个参数和两个冒号" : " ,用于分隔三个参数(start_index、end_index、step)。当只有一个":"时，默认第三个参数step=1;**

## 1.4.1 loc方法

### Pandas中将loc设计为左右全闭

### 所有在loc中使用的切片全部包含右端点

```
data=pd.read_csv('data.csv')
##单行索引
data.loc[1103]
##多行索引
data.loc[[1102, 2304]]
##Pandas中将loc设计为左右全闭
data.loc[1304:2103]
data.loc[1304:2103].head
```

```
data=pd.read_csv('data.csv')
##单列索引
data.loc[:,'Height'].head()
#多列索引
data.loc[:,['Height', 'Math']].head()
#联合索引
data.loc[1102:2401:3,'Height':'Math'].head()
#函数式索引
data.loc[lambda x:x['Gender']=='M'].head()
#布尔
data.loc[df['Address'].isin(['street_7', 'street_4'])].head()
data.loc[[True if i[-1]=='4' or i[-1]=='7' else False
for i in data['Address'].values]].head()#i表示一个字符串时
```

## 1.4.2 iloc方法

### iloc方法（注意与loc不同，切片右端点不包含）

```
#单行索引
data.iloc[3]
#多行索引
data.iloc[3:5]
#多列索引
data.iloc[:,3].head()
#混合索引
data.iloc[3::4,7::-2].head()
#函数式索引
#布尔
data.iloc[(data['School']=='S_1').values].head()
```

### 1.4.3 []操作符

```
data[0:4]
```

### 1.4.4 布尔索引

**布尔符号：‘&’,‘|’,‘~’：分别代表和and，或or，取反not**

```
data[(data['Gender']=='F')&(data['Address']=='street_2')].head()
data[data['Math']>85 | (data['Address']=='street_2')].head()
data.loc[data['Math']>60, data.columns=='Physics'].head()
```

## 1.5.pandas常用函数

### 1.5.1 pd.concat()

```
#pd.concat()函数可以沿着指定的轴将多个dataframe或者series拼接到一起
pd.concat(
     objs,
     axis=0,
     join='outer',
     ignore_index=False,
     keys=None,
     levels=None,
     names=None,
     verify_integrity=False,
     copy=True)
```

objs：Series，DataFrame或Panel对象的序列或映射，如果传递了dict，则排序的键将用作键参数

axis：{0,1，...}，默认为0,也就是**纵向上进行合并**。沿着连接的轴。当axis = 1的时候，concat就是行对齐，然后将不同列名称的两张表合并

join：{'inner'，'outer'}，默认为"outer"。如何处理其他轴上的索引。outer为联合和inner为交集。加上join参数的属性，如果为'inner'得到的是两表的交集，如果是outer，得到的是两表的并集。

### 1.5.2 pd.get_dummies()

pd.get_dummies() 是 Pandas 库中用于独热编码（One-Hot Encoding）的函数。它的作用是将分类（离散）变量的每个不同取值都拓展为一个新的二进制特征（0或1），从而方便机器学习模型处理。

```
pd.get_dummies(data,
 prefix=None,
 prefix_sep='_',
dummy_na=False,
 columns=None,
sparse=False,
 drop_first=False,
 dtype=None)
```

其中一些常用参数包括：

data: 要进行独热编码的 DataFrame 或 Series。

prefix: 生成的独热编码列的前缀。

prefix_sep: 生成的独热编码列的前缀和原始列名之间的分隔符。

dummy_na: 是否为原始数据中的缺失值生成独热编码列。

columns: 要进行独热编码的列的名称，如果指定，则只对这些列进行操作。

drop_first: 是否删除第一个独热编码列，以避免共线性问题。

```
import pandas as pd
# 创建一个包含分类特征的DataFrame
data = {'color': ['red', 'green', 'blue', 'green', 'red']}
df = pd.DataFrame(data)

# 对分类特征进行独热编码
df_encoded = pd.get_dummies(df, prefix='color')
print(df_encoded)
```

输出

```
   color_blue  color_green  color_red
0           0            0          1
1           0            1          0
2           1            0          0
3           0            1          0
4           0            0          1
```

### 1.5.3 pd.DataFrame()函数

pd.DataFrame()函数-CSDN博客

通过传递一个numpy array，时间索引以及列标签来创建一个DataFrame1.DataFrame介绍 一个Datarame表示一个表格，类似电子表格的数据结构，包含一个经过排序的列表集，它的每一列都可以有不同的类型值（数字，字符串，布尔等等）。Datarame有行和列的索引；它可以被看作是一个Series的字典（Series们共享一个索引）。DataFrame里的面向行和面向列的操作大致是对称的。在底层，数据是作为一个或多个二维数组存储的，而不是列表，字典，或其它一维的数组集合。

通过传递一个numpy array，时间索引以及列标签来创建一个DataFrame

```
data = DataFrame(np.arange(10,26).reshape((4, 4)),
                 index=['Ohio', 'Colorado', 'Utah', 'New York'],
                 columns=['one', 'two', 'three', 'four'])
data
```

```
    one    two    three    four
Ohio    10    11    12    13
Colorado    14    15    16    17
Utah    18    19    20    21
New York    22    23    24    25
```

```
np.random.seed(10)
dates = pd.date_range('20190101', periods=6)
df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))
df
```

```
                A           B           C           D
2019-01-01  1.331587    0.715279   -1.545400   -0.008384
2019-01-02  0.621336   -0.720086    0.265512    0.108549
2019-01-03  0.004291   -0.174600    0.433026    1.203037
2019-01-04 -0.965066    1.028274    0.228630    0.445138
2019-01-05 -1.136602    0.135137    1.484537   -1.079805
2019-01-06 -1.977728   -1.743372    0.266070    2.384967
```

### 绘图

df.plot-CSDN博客

```
DataFrame.plot(x=None, y=None, kind='line', ax=None, subplots=False,
sharex=None, sharey=False, layout=None, figsize=None,
use_index=True, title=None, grid=None, legend=True,
style=None, logx=False, logy=False, loglog=False,
xticks=None, yticks=None, xlim=None, ylim=None, rot=None,
fontsize=None, colormap=None, position=0.5, table=False, yerr=None,
xerr=None, stacked=True/False, sort_columns=False,
secondary_y=False, mark_right=True, **kwds)
```

- subplots是否画子图 （1）将每一列作为一个子图 （2）默认False
- figsize图片大小 （1）元组类型，指定宽和高

[Pytorch系列-30]：神经网络基础 - torch.nn库五大基本功能：nn.Parameter、nn.Linear、nn.functioinal、nn.Module、nn.Sequentia-CSDN博客

# 二、torch

**nn是Neural Network的简称，帮助程序员方便执行如下的与神经网络相关的行为：**

（1）创建神经网络

（2）训练神经网络

（3）保存神经网络

（4）恢复神经网络

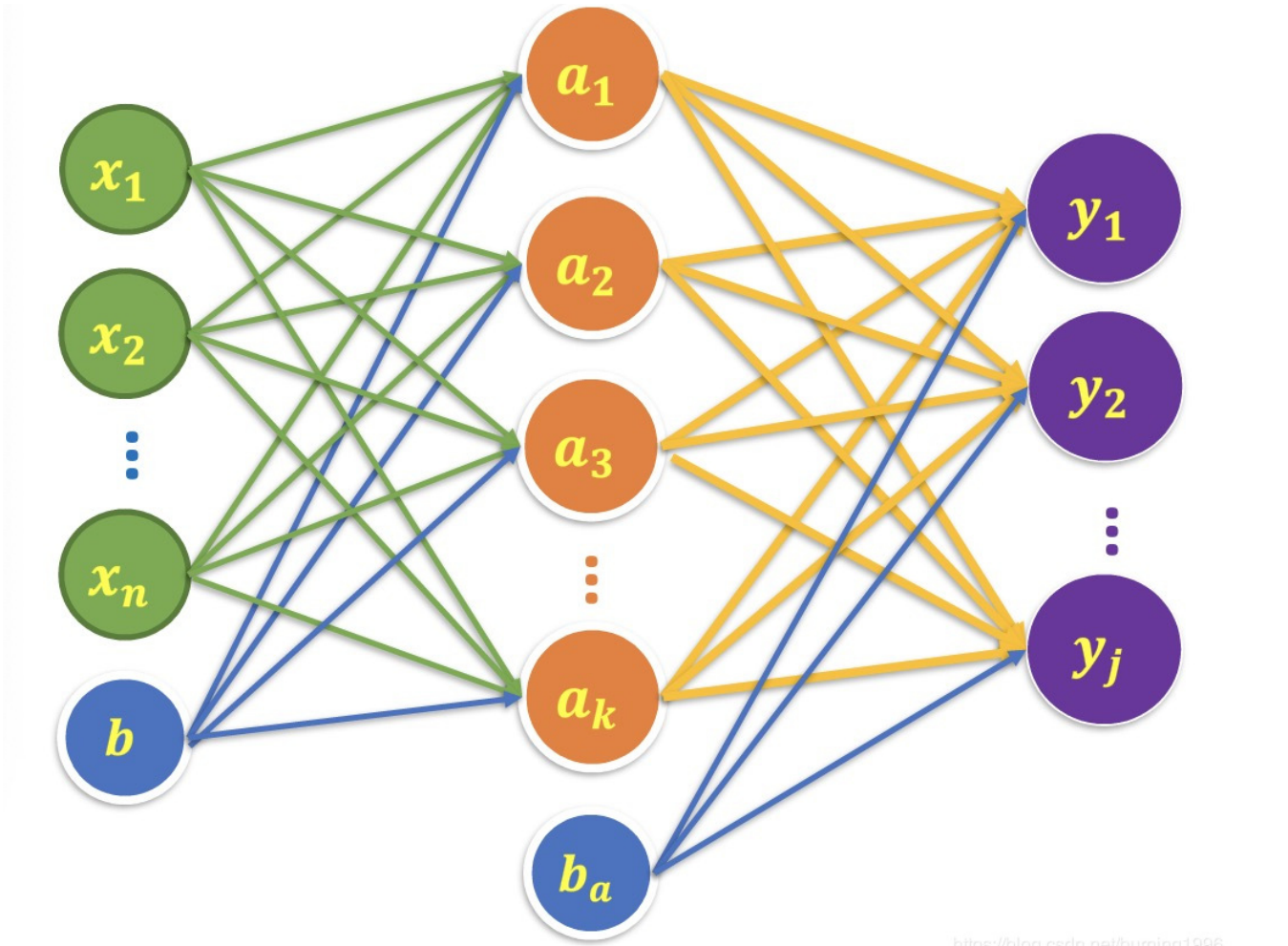
# torch.nn库

- **torch.nn是专门为神经网络设计的模块化接口。**
- **nn构建于autograd之上，可以用来定义和运行神经网络。**
  - **nn.Parameter**
  - **nn.Linear**
  - **nn.functional**
  - **nn.Module**
  - **nn.Sequential**



## 2.1 nn.Linear类（全连接层）

用于创建一个多输入、多输出的全连接层。








nn.Linear本身并不包含激活函数（Functional）

CLASS `torch.nn.Linear(in_features, out_features, bias=True)`

Applies a linear transformation to the incoming data: $y = xA^T + b$

Parameters

- **in_features** – size of each input sample
- **out_features** – size of each output sample
- **bias** – If set to `False`, the layer will not learn an additive bias. Default: `True`

- in_features：指的是输入的二维张量的大小，即输入的[batch_size, size]中的size。

in_features的数量，决定的参数的个数 Y = WX + b，X的维度就是in_features，X的维度决定的W的维度， 总的参数个数 = in_features + 1

- out_features：指的是输出的二维张量的大小，即输出的二维张量的形状为[batch_size output_size]。

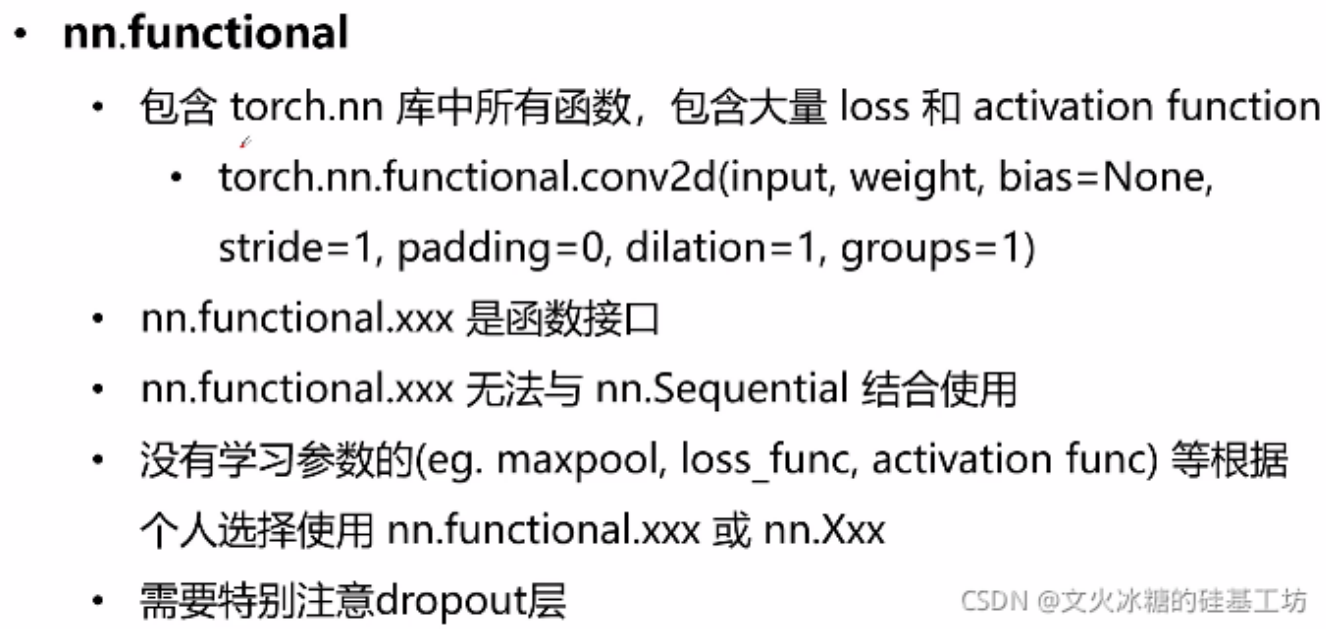
out_features的数量，决定了全连接层中神经元的个数，因为每个神经元只有一个输出。

多少个输出，就需要多个个神经元。

从输入输出的张量的shape角度来理解，相当于一个输入为[batch_size, in_features]的张量变换成了[batch_size, out_features]的输出张量。

## 2.2 nn.functional（常见函数）

nn.functional定义了创建神经网络所需要的一些常见的处理函数。如没有激活函数的神经元，各种激活函数等。




**nn.functional包括神经网络前向和后向处理所需要到的常见函数。**

### 2.2.1神经元处理函数

### 2.2.2激活函数

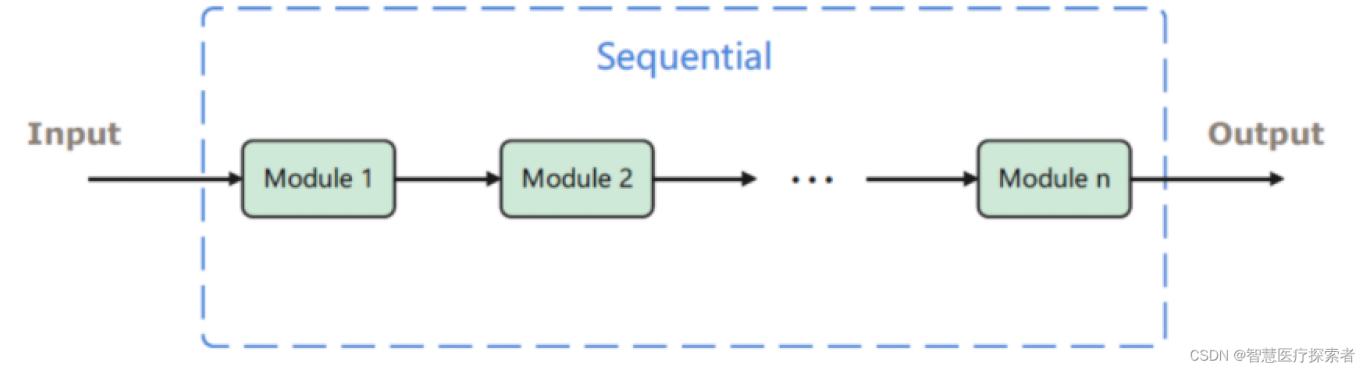
```
nn.functional.relu()
nn.functional.sigmoid()
```

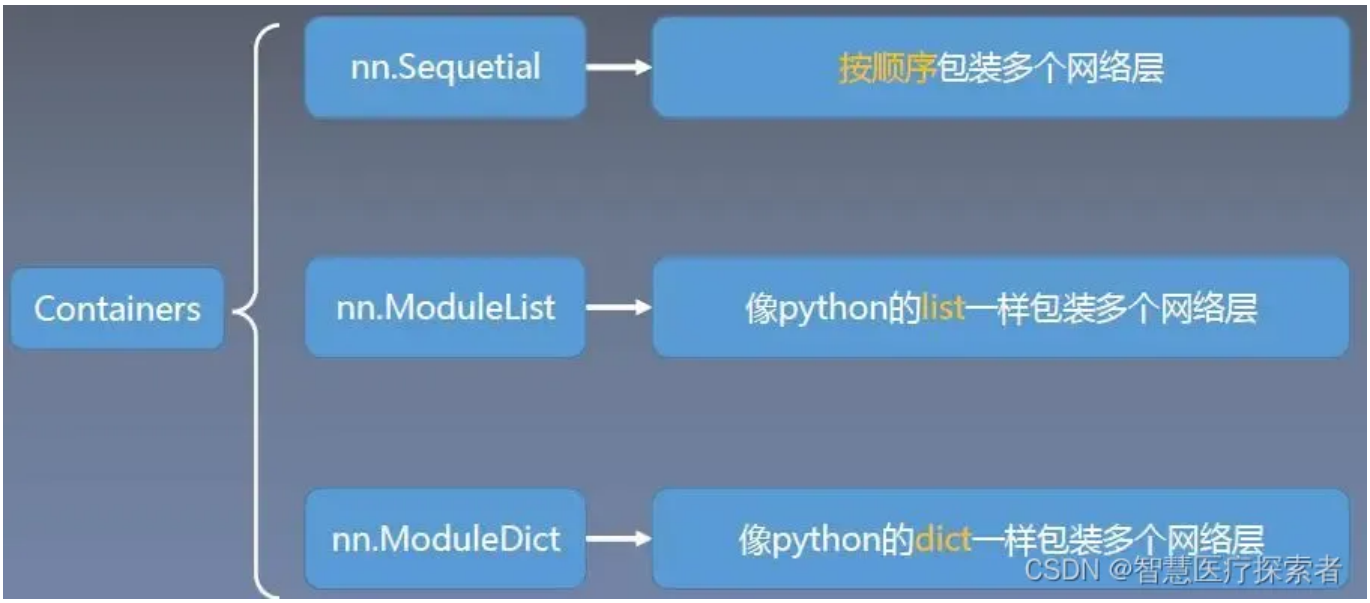
nn.functional.xxx是API函数接口，而nn.Xxx是对原始API函数nn.functional.xxx的类封装。

## 2.3 nn.Sequential类创建神经网络（继承与nn.Module类）

nn.Sequential是一个有序的容器，该类将按照传入构造器的顺序，依次创建相应的函数，并记录在Sequential类对象的数据结构中，同时以神经网络模块为元素的有序字典也可以作为传入参数。

因此，Sequential可以看成是有多个函数运算对象，串联成的神经网络，其返回的是Module类型的神经网络对象。




利用系统提供的神经网络模型类：Sequential,以参数列表的方式来实例化神经网络模型对象 Sequential( (0): Linear(in_features=784, out_features=32, bias=True) (1): ReLU() (2): Linear(in_features=32, out_features=10, bias=True) (3): Softmax(dim=1) )

## 2.4 torch.clamp()

```
torch.clamp(input, min, max, out=None)
```

将输入input张量每个元素的范围限制到区间 [min,max]，返回结果到一个新张量。

input (Tensor) – 输入张量 min (Number) – 限制范围下限 max (Number) – 限制范围上限 out (Tensor, optional) – 输出张量

## 2.5 torch.cat（）函数

函数将两个张量（tensor）按指定维度拼接在一起，注意：除拼接维数dim数值可不同外其余维数数值需相同，方能对齐

```
#列拼接
a = torch.randn(2,3)
b =  torch.randn(3,3)
c = torch.cat((a,b),dim=0)
a,b,c
```

```
#行拼接
a = torch.randn(2,3)
b =  torch.randn(2,4)
c = torch.cat((a,b),dim=1)
a,b,c
```

```
a = torch.randn(2,3,4)
b =  torch.randn(1,3,4)
c = torch.cat((a,b),dim=0)
a,b,c
```

## 2.6 torch.optim

### 2.6.1 结合PyTorch中的optimizer谈几种优化方法

一大类方法是SGD及其改进（加Momentum）；

另外一大类是Per-parameter adaptive learning rate methods（逐参数适应学习率方法），包括AdaGrad、RMSProp、Adam等。

**1.1 SGD(stochastic gradient descent)** 当训练数据N很大时，计算总的cost function来求梯度代价很大，所以一个常用的方法是计算训练集中的小批量（minibatches），这就是SGD。 minibatch的大小是一个超参数，通常使用2的指数，是因为在实际中许多向量化操作实现的时候，如果输入数据量是2的倍数，那么运算更快。

```
PyTorch中的SGD：
torch.optim.SGD(params, lr=<required parameter>
, momentum=0, dampening=0, weight_decay=0, nesterov=False)
```

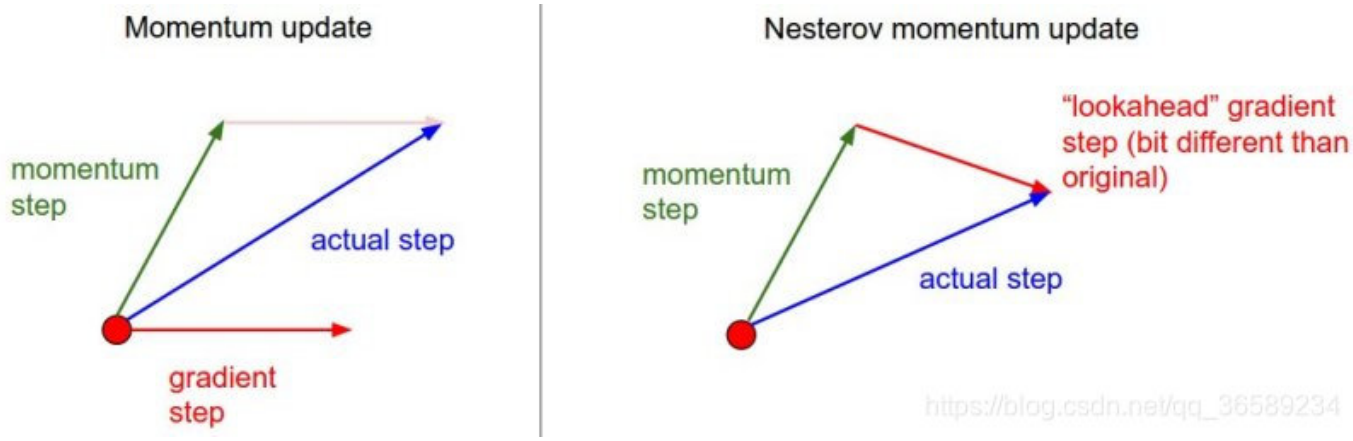
- params (iterable)：iterable of parameters to optimize or dicts defining parameter groups
- lr (float)：learning rate
- momentum (float, optional)：momentum factor (default: 0)
- weight_decay (float, optional)：weight decay (L2 penalty) (default: 0)即L2regularization，选择一个合适的权重衰减系数λ非常重要，这个需要根据具体的情况去尝试，初步尝试可以使用 1e-4 或者 1e-3
- dampening (float, optional)：dampening for momentum (default: 0)
- nesterov (bool, optional)：enables Nesterov momentum (default: False)

### 1.2 SGD+Momentum

PyTorch中的 SGD with momentum 已经在optim.SGD中的参数momentum中实现，顺便提醒一下PyTorch中的momentum实现机制和其他框架略有不同

### 1.3 Nesterov Momentum

Nesterov Momentum实际上是拿着上一步的速度先走一小步，再看当前的梯度然后再走一步。Nesterov Momentum 与 普通Momentum 的区别:




### 1.4 AdaGrad

之前的方法是对所有的参数都是一个学习率，现在对不同的参数有不同的学习率

```
torch.optim.Adagrad(params, lr=0.01, lr_decay=0, weight_decay=0,
initial_accumulator_value=0)
```

- params (iterable) – iterable of parameters to optimize or dicts defining parameter groups
- lr (float, optional) – learning rate (default: 1e-2)
- lr_decay (float, optional) – learning rate decay (default: 0)
- weight_decay (float, optional) – weight decay (L2 penalty) (default: 0)

### 1.5 RMSProp

RMSProp简单修改了Adagrad方法，它做了一个梯度平方的滑动平均（it uses a moving average of squared gradients instead）.

，RMSProp相较于Adagrad的优点是在鞍点等地方，它在鞍点呆的越久，学习率会越大。
PyTorch中的用法：

```
torch.optim.RMSprop(params, lr=0.01, alpha=0.99, eps=1e-08,
 weight_decay=0, momentum=0, centered=False)
```

- params (iterable) – iterable of parameters to optimize or dicts defining parameter groups
- lr (float, optional) – learning rate (default: 1e-2)
- momentum (float, optional) – momentum factor (default: 0)
- alpha (float, optional) – smoothing constant (default: 0.99)
- eps (float, optional) – term added to the denominator to improve numerical stability (default: 1e-8)
- centered (bool, optional) – if True, compute the centered RMSProp, the gradient is normalized by an estimation of its variance
- weight_decay (float, optional) – weight decay (L2 penalty) (default: 0)

1.6 Adam

在实际操作中，我们推荐Adam作为默认的算法，一般而言跑起来比RMSProp要好一点。但是也可以试试SGD+Nesterov动量。

```
torch.optim.Adam(params, lr=0.001, betas=(0.9, 0.999), eps=1e-08,
weight_decay=0, amsgrad=False)
```

- params (iterable) – iterable of parameters to optimize or dicts defining parameter groups
- lr (float, optional) – learning rate (default: 1e-3)
- betas (Tuple[float, float], optional) – coefficients used for computing running averages of gradient and its square (default: (0.9, 0.999))
- eps (float, optional) – term added to the denominator to improve numerical stability (default: 1e-8)
- weight_decay (float, optional) – weight decay (L2 penalty) (default: 0) amsgrad (boolean, optional) – whether to use the AMSGrad variant of this algorithm from the paper On the Convergence of Adam and Beyond (default: False)

## 2.7 损失函数

### 2.7.1 nn.MSELoss()

该函数叫做平均平方误差，简称均方误差。它的英文名是mean squared error，该损失函数是挨个元素计算的。该元素的公式如下：

```
loss= torch.nn.MSELoss(reduce=True, size_average=True)
```

1、 如果reduce = False，返回向量形式的 loss

2、如果reduce = True， 返回标量形式的loss

3、如果size_average = True，返回 loss.mean();

4、如果 size_average = False，返回 loss.sum()

默认情况下：两个参数都为True.

模型的预测值与标签的L2距离。一般用于回归问题。之所以不用于分类问题，可能原因为：使用sigmoid之后，函数形式不是凸函数 ， 不容易求解 ， 容易进入局部最优。

### 2.7.2 nn.BCELoss()

交叉熵损失函数，衡量两个分布之间的差异，一般用于分类问题。输入的x值需先经过sigmoid压缩到（0,1）之间。标签形式为[0, 0, 1], [0, 1, 1]等，各个类别预测概率独立，类与类之间不互斥，

可见不仅能用于二分类问题，也能用于多标签分类问题。

y是真实标签，x是预测值

$$\mathrm{Loss} = -\frac{1}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{N}} \left[ \mathrm{y_i} \log\left(\mathrm{p_i}\right) + \left(1 - \mathrm{y_i}\right) \log\left(1 - \mathrm{p_i}\right) \right]$$

这个函数就是CrossEntropyLoss的当类别数N=2时候的特例。因为类别数为2，属于第一类的概率为y，那么属于第二类的概率自然就是(1-y)。因此套用与CrossEntropy损失的计算方法，用对应的标签乘以对应的预测值再求和，就得到了最终的损失。

N是总样本数，$y_i$是第i个样本的所属类别，$p_i$是第i个样本的预测值，一般来说，它是一个概率值。

### 2.7.3 nn.BCEWithLogitsLoss()

交叉熵损失函数， 与nn.BCELoss()不同的是网络的输出无需用sigmoid压缩，函数内部整合了nn.sigmoid()和nn.BCELoss(),并且使用log-sum-exp trick提高了数值稳定性。同样可用于二分类及多标签分类。 这里简单介绍一下log-sum-exp trick： 原始的log-sum-exp公式为：

$$\mathrm{y} = \log \sum_{\mathrm{i}} \mathrm{e}^{\mathrm{x_i}}$$

在$\mathrm{x_i}$都很小时，$\sum_{\mathrm{i}} \mathrm{e}^{\mathrm{x_i}}$趋近于0，导致数值计算问题。
而如果$\mathrm{x_i}$很大，$\sum_{\mathrm{i}} \mathrm{e}^{\mathrm{x_i}}$也会很大，同样会导致数值计算问题。
而log-sum-exp trick的计算公式为：

$$\mathrm{a} = \max(\mathrm{x_i})$$

$$\mathrm{y} = \mathrm{a} + \log \sum_{\mathrm{i}} \mathrm{e}^{\mathrm{x_i} - \mathrm{a}}$$

其中$\mathrm{e}^{\mathrm{x_i} - \mathrm{a}} \leq 1$,使$\sum_{\mathrm{i}} \mathrm{e}^{\mathrm{x_i}}$既不会趋于0也不会很大，避免数值溢出。

### 2.7.4 nn.LogSoftmax()

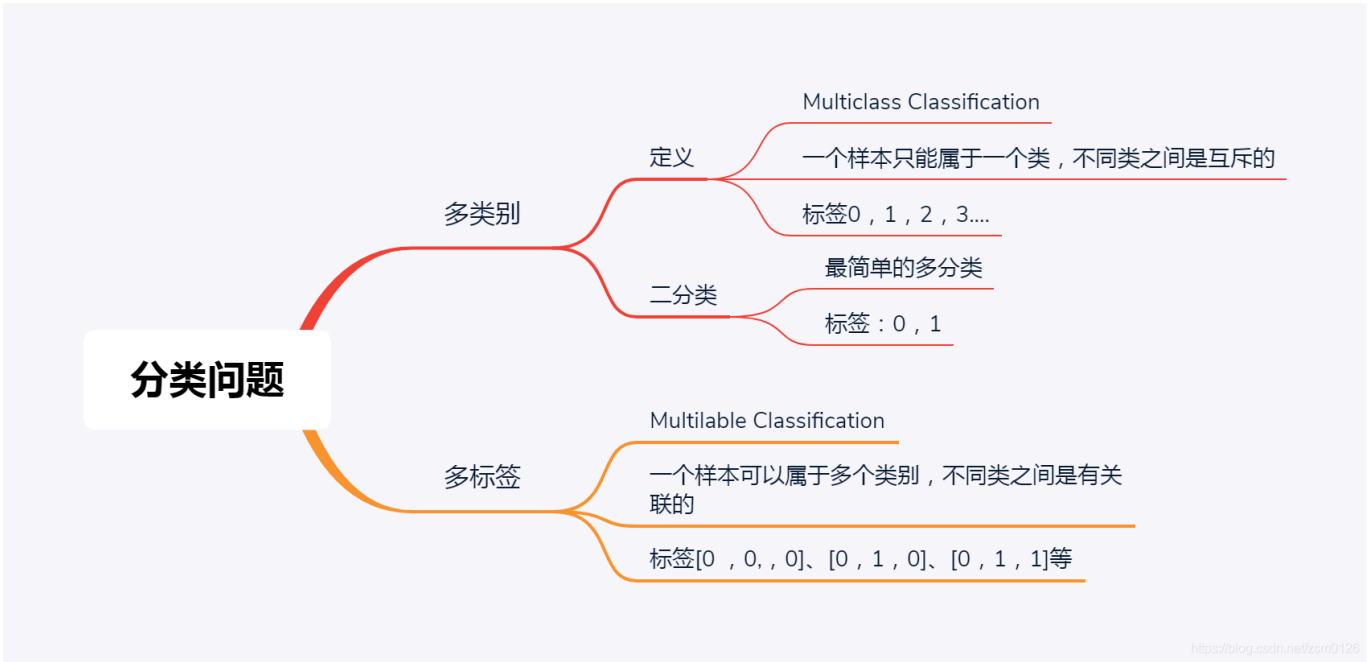
将输入softmax后取对数

### 2.7.5 nn.NLLLoss()

NLLLoss需要配合nn.LogSoftmax()使用，网络输出的值经过nn.LogSoftmax()后传入NLLLoss()。其标签实际上就是网络输出里对应数值的索引，通过这个索引取得对应的值并去掉负号。如果是一批次的话，还需要求均值。由于网络输出需先经过softmax，各个类别预测概率之和为1，所以类与类之间互斥，用于多类别问题。

### 2.7.6 nn.CrossEntropyLoss()

nn.CrossEntropyLoss()是nn.logSoftmax()和nn.NLLLoss()的整合,适用于多类别问题。




## 2.8 激活函数

### Linear常用激活函数

```
#Linear常用激活函数
torch.nn.ReLU()
torch.nn.RReLU()
torch.nn.LeakyReLU()
torch.nn.PReLU()
torch.nn.Softplus()
torch.nn.ELU()
torch.nn.CELU()
torch.nn.SELU()
torch.nn.GELU()
torch.nn.ReLU6()
torch.nn.Sigmoid()
torch.nn.Tanh()
torch.nn.Softsign()
torch.nn.Hardtanh()
torch.nn.Threshold()
torch.nn.Tanhshrink()
torch.nn.Softshrink()
torch.nn.Hardshrink()
torch.nn.LogSigmoid()
torch.nn.Softmin()
torch.nn.Softmax()
torch.nn.LogSoftmax()
```

## 三、d2l

### 3.1 d2l.load_array（）函数

```
train_iter = d2l.load_array((train_features, train_labels), batch_size)
```

这行代码的作用将训结数据集train_features,train_labels流加载到内存中并将其转换为一个可以迭代的数据集，每个迭代器会返回一 个大小为batch_size的小批量数据
具体来说，load_array函数总d2l(dive into deep leaming)库中的一个函数，它的作用是将数据集转换为一个迭代器。这个迭代器可以用来对数据集进行迭代，每次返回一个大小为batch_size的小批量数据。其中，batch_size是一个指定的批量大小，用于控制每次迭代返回的数据量大小
因此，train_iter是一个迭代器，每次迭代返回一个大小为batch_size的小批量数据。这些数据可以用于模型的训练

# 其他

### 1.assert

使用assert是学习python的一个非常好的习惯，在没完善一个程序之前，我们不知道程序在哪里会出错，与其让它在运行时崩溃，不如在出现错误条件时就崩溃。

```
assert a > 0,"a超出范围"    #这句的意思：如果a确实大于0，程序正常往下运行
```

## 2.Python语言中/与//的区别

在Python中/表示浮点整除法，返回浮点结果，也就是结果为浮点数;而//在Python中表示整数除法，返回大于结果的一个最大的整数，意思就是除法结果向下取整。

## 3.slice

slice 是 Python 中的一个内置函数，用于创建切片对象，表示一系列连续的索引。slice 对象通常用于从序列类型（如列表、元组、字符串、NumPy数组等）中提取指定范围的元素。

```
slice(start, stop, step)
```

start：切片开始的索引（包含），默认值为 0。 stop：切片结束的索引（不包含），默认值为序列的结尾。 step：切片的步长，表示从开始索引到结束索引的间隔，如 step 为 2，则提取从开始索引到结束索引的偶数位置的元素。默认值为 1。

## 4.介绍和*

解包操作可以应用于元组、列表、集合、字典。

*args：用于列表、元组、集合 **kwargs：用于字典

```
li = list(range(7))
tu = tuple(range(7))
se = set(range(7))

print(*li,*tu,*se)
```

```
0 1 2 3 4 5 6 0 1 2 3 4 5 6 0 1 2 3 4 5 6
```

## 5._的用法

忽略特定的值 下划线还用于忽略特定的值。如果不需要特定的值，或者不使用这些值，只需将这些值赋给下划线即可。_