Pytorch官方英文文档：[https://pytorch.org/docs/stable/torch.html](https://pytorch.org/docs/stable/torch.html?)

Pytorch中文文档：<https://pytorch-cn.readthedocs.io/zh/latest/>

原文链接：<https://blog.csdn.net/wuzhongqiang/article/details/105612559>

**1、写在前面**

我们知道损失函数的作用是衡量模型输出与真实标签的差异，当我们有了这个loss之后，我们就可以通过**反向传播机制**得到**参数的梯度**，那么我们如何利用这个梯度进行更新参数使得模型的loss逐渐的降低呢？这个就是优化器干的活了。

首先学习优化器的概念，然后学习优化器的属性和方法，介绍常用的优化器的原理。 关于优化器，最重要的一个参数就是**学习率**，这个控制着参数更新的一个步伐， 在模型训练中，对于学习率的调整也是非常关键的，所以最后整理一些关于学习率调整的六种策略。

**大纲如下：**

优化器（什么是优化器，优化器运行机制、属性和方法，常用的优化器介绍）

学习率调整策略(为什么调整学习率， 六种学习率调整策略）

梳理总结

**2、优化器**

机器学习的五个步骤： 数据 -> 模型 -> 损失 -> 优化器 -> 迭代训练。 我们通过前向传播的过程，得到了模型输出与真实标签的差异，我们称之为损失， 有了损失，我们会进入反向传播过程得到参数的梯度，那么接下来就是优化器干活了，优化器要根据我们的这个梯度去更新参数，使得损失不断的降低。 那么优化器是怎么做到的呢？ 下面我们从三部分进行展开，首先是优化器的概念，然后是优化器的属性和方法，最后是常用的优化器。

**2.1 什么是优化器**

Pytorch的优化器： 管理并更新模型中**可学习参数的值**， 使得模型输出更接近真实标签。

我们在更新参数的时候一般使用梯度下降的方式去更新， 那么什么是梯度下降呢？ 说这个问题之前得先区分几个概念：

导数： 函数在指定坐标轴上的变化率

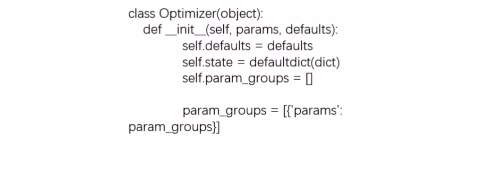
方向导数： 指定方向上的变化率

梯度： 一个向量，方向为 方向导数取得最大值 的方向

我们知道梯度是一个向量，它的方向是**导数取得最大值**的方向，也就是增长最快的方向，而梯度下降就是沿着梯度的负方向去变化，这样函数的下降也是最快的。所以我们往往采用梯度下降的方式去更新权值，使得函数的下降尽量的快。

**2.2 Optimizer的基本属性和方法**

**（1）优化器的基本属性：**



**defaults:** 优化器超参数，里面会存储一些学习率， momentum的值，衰减系数等

**state:** 参数的缓存，如momentum的缓存（使用前几次梯度进行平均）

**param\_groups:** 管理的参数组， 这是个列表，每一个元素是一个字典，在字典中有key，key里面的值才是我们真正的参数（这个很重要， 进行参数管理）

**\_step\_count:** 记录更新次数， 学习率调整中使用， 比如迭代100次之后更新学习率的时候，就得记录这里的100

**（2）优化器的基本方法：**

**zero\_grad()：** 清空所管理参数的梯度， 这里注意Pytorch有一个特性就是**张量梯度不自动清零**

**step():** 执行一步更新

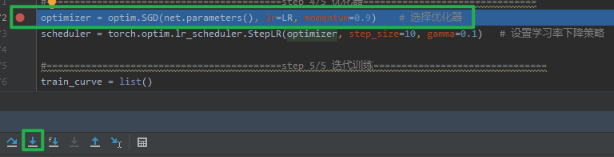
**add\_param\_group():** 添加参数组, 我们知道优化器管理很多参数，这些参数是可以分组的，我们对不同组的参数可以设置不同的超参数， 比如模型finetune中，我们希望前面特征提取的那些层学习率小一些，而后面我们新加的层学习率大一些更新快一点，就可以用这个方法。

**state\_dict():** 获取优化器当前状态信息字典

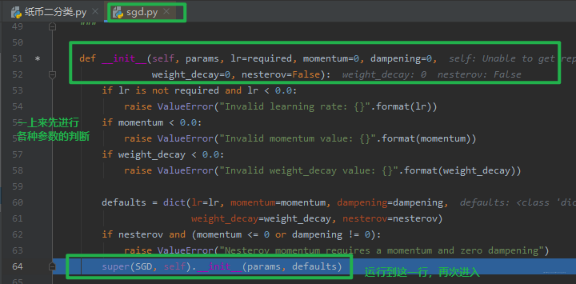
**load\_state\_dict():** 加载状态信息字典，这两个方法用于模型断点的一个续训练， 所以我们在模型训练的时候，一般多少个epoch之后就要保存当前的状态信息。

**2.3 优化器的运行机制**

了解了优化器的基本属性和方法之后，我们去代码中看看优化器的运行机制：



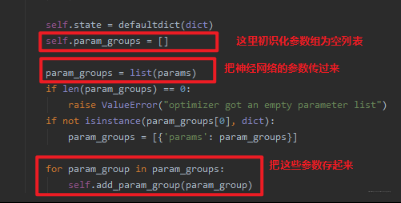
点击步入，进入sgd.py的SGD类：



SGD类是继承于optimizer的，所以我们将代码运行到父类初始化的这一行，点击步入，看看是如何初始化的：

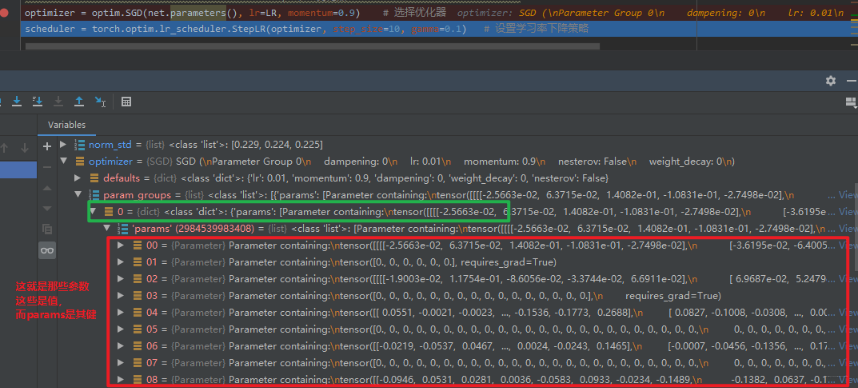


这里就是optimizer的\_\_init\_\_初始化部分了，可以看到上面介绍的那几个属性和它们的初始化方法，当然这里有个最重要的就是参数组的添加，我们看看是怎么添加的：

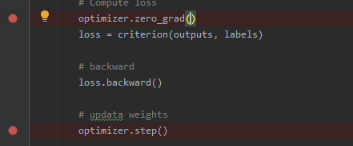


这里重点说一下这个，我们还记得初始化SGD的时候传入了一个形参：optim.SGD(net.parameters(), lr=LR, momentum=0.9)，这里的**net.parameters()** 就是**神经网络的每层的参数**，SGD在初始化的时候，会把这些参数**以参数组的方式再存起来**，上图中的params就是神经网络每一层的参数。

下面我们跳回去， 看看执行完这个 初始化参数 变成了什么样子：



这就是优化器的初始化工作了， 初始化完了之后， 我们就可以进行**梯度清空**，然后**更新梯度**即可：



这就是优化器的使用了。

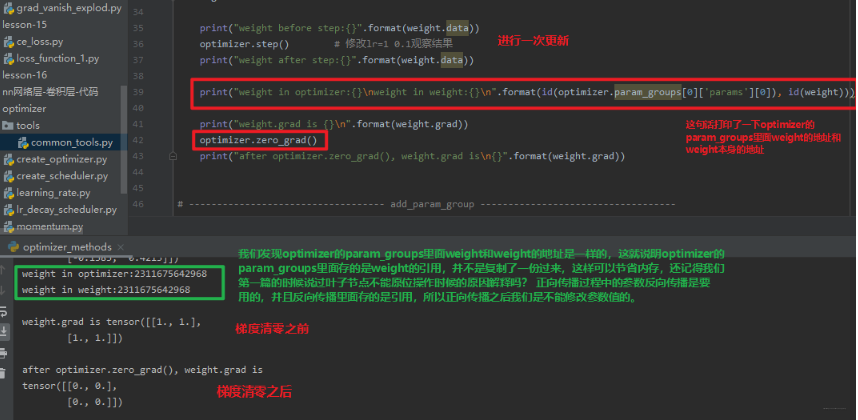
下面我们**学习优化器具体的方法：**

**2.4 优化器具体的方法**

1、step(): 一次梯度下降更新参数

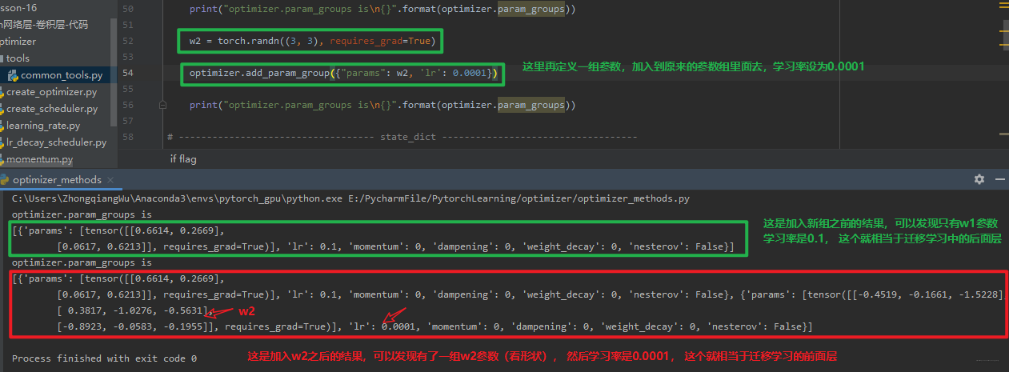


2、zero\_grad()： 将梯度清零



3、add\_param\_group(): 添加参数组

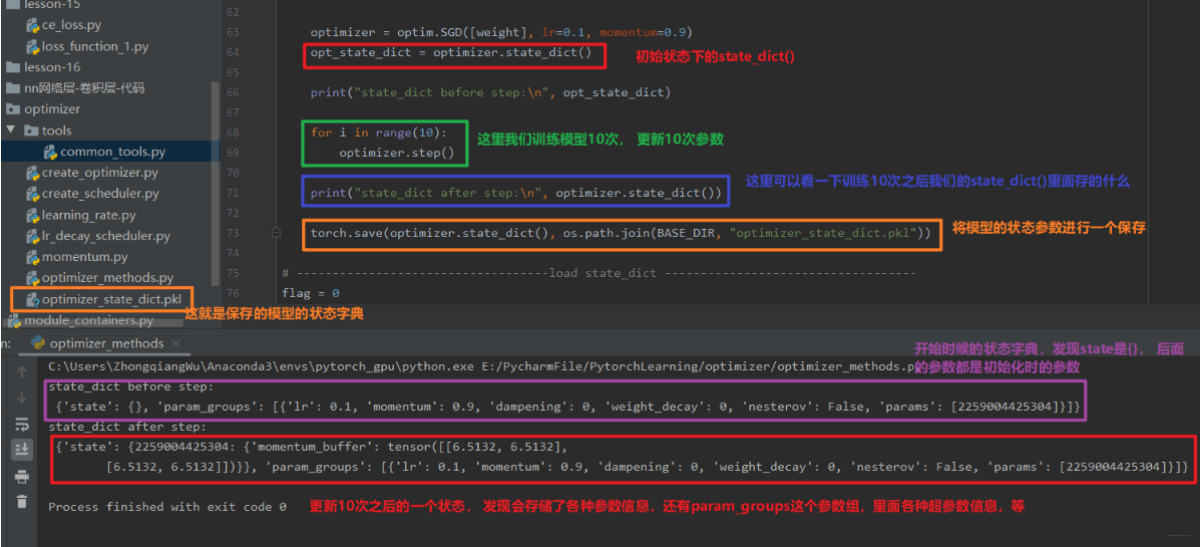
这个是在**模型的迁移学习中**非常实用的一个方法，我们看看怎么用：



5、state\_dict()和load\_state\_dict()

这两个方法用于保存和加载优化器的一个状态信息，通常用在**断点的续训练**， 比如我们训练一个模型，训练了10次停电了，那么再来电的时候我们就得需要从头开始训练，但是如果有了这两个方法，我们就可以再训练的时候接着上次的次数继续， 所以这两个也非常实用。

首先是state\_dict()



我们可以看到，state\_dict()方法里面保存了我们**优化器的各种状态信息**，我们通过torch.save就可以**保存这些状态到文件(.pkl)**， 这样假设此时停电了。我们就可以通过load\_state\_dict()来导入这个状态信息，让优化器在这个基础上进行训练，看看是怎么做的？



这就是优化器的初始化和优化器的5个方法的使用了。了解了这些知识之后，我们就知道了优化器的运行机制，管理和更新模型的可学习参数（管理是通过各种属性，尤其是param\_groups这个重要的属性，而更新是通过各种方法，主要是step()方法进行更新）。

那么究竟有哪些常用的优化器呢？ 它们又用于什么场景呢？ 下面我们就来看看：

**2.5 常用的优化器**

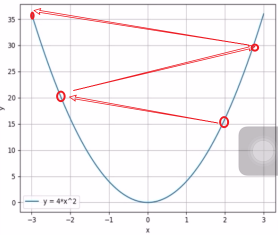
这次我们会学习Pytorch中的10种优化器，但是在介绍这些优化器之前，得先学习两个非常重要的概念， 那就是**学习率和动量**。 我们先从学习率开始：

**2.5.1 学习率**

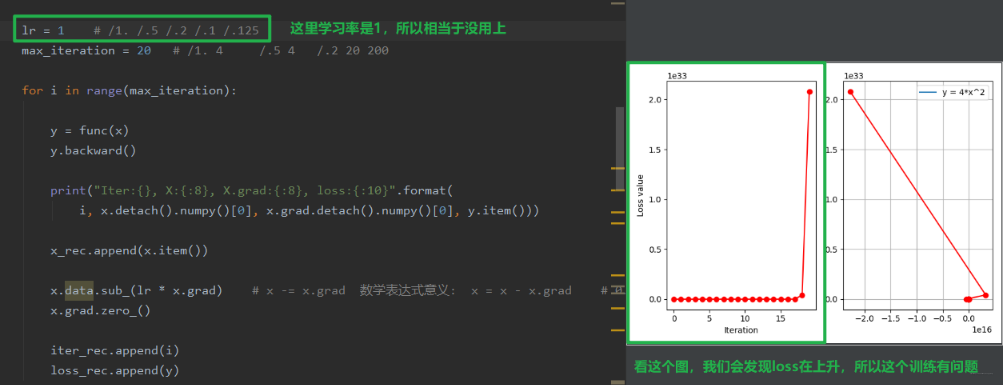
在梯度下降过程中，学习率起到了控制参数更新的一个步伐的作用，参数更新公式我们都知道：

截图.png

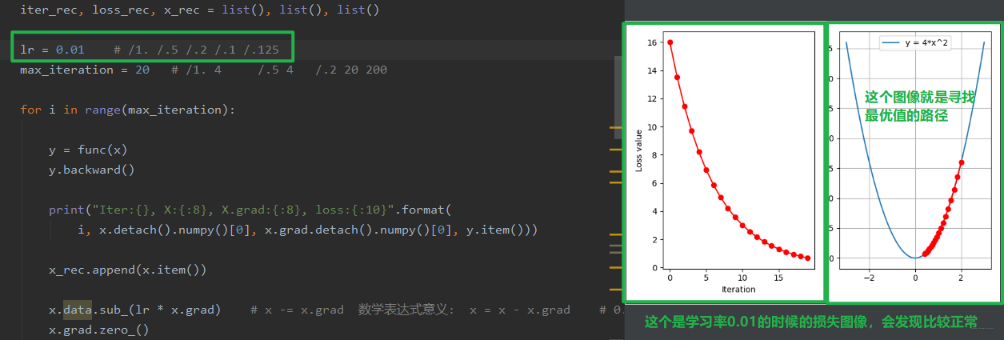
如果没有这个学习率LR的话，往往有可能由于梯度过大而错过我们的最优值，就是下面这种感觉：



随着迭代次数的增加，反而越增越大， 就是因为这个步子太大了，跳过了我们的最优值。所以这时候我们想让他这个跨度小一些，就得需要一个参数来控制我们的这个跨度，这个就是学习率。 这样说起来，有点抽象，我们还是从代码中看看吧：

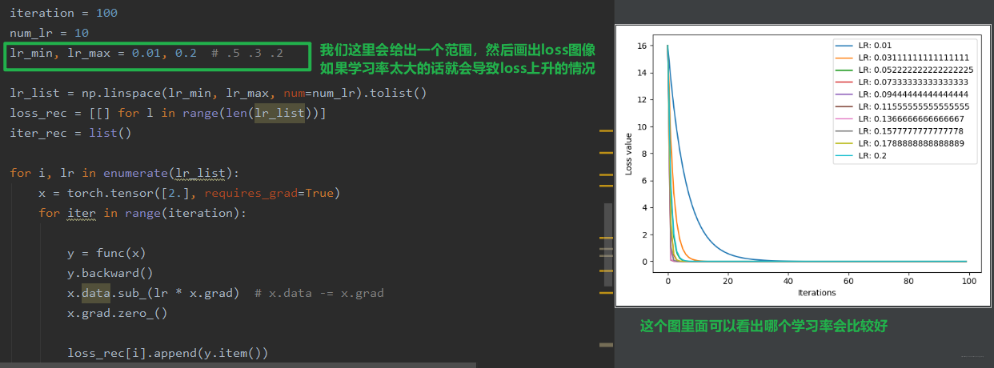


我们可以看一下上面的图像，loss是不断上升的，这说明这个跨度是有问题的，所以下面我们尝试改小一点学习率，我们就可以发现区别了：



我们发现，当loss上升不降的时候，有可能是学习率的问题，所以我们一般会尝试一个小的学习率。 慢慢的去进行优化。

学习率一般是我们需要调的一个非常重要的超参数， 我们一般是给定一个范围，然后画出loss的变化曲线，看看哪学习率比较好，当然下面也会重点学习学习率的调整策略。

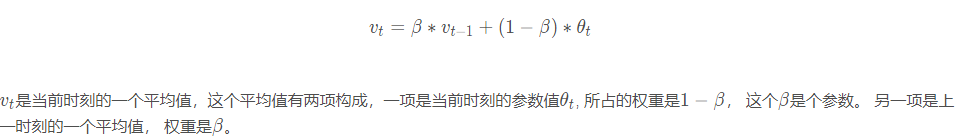


**2.5.2 动量**

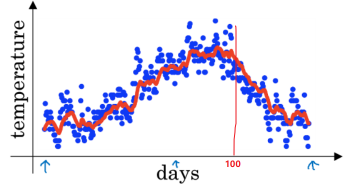
Momentum：结合当前梯度与上一次更新信息， 用于当前更新。这么说可能有点抽象， 那么我们可以举个比较形象的例子：



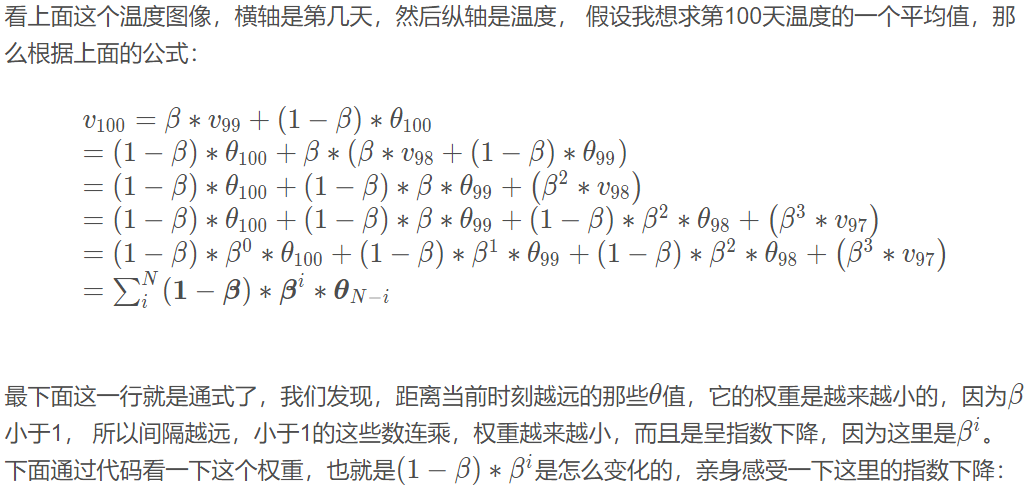
那么这个动量是怎么作用于我们的更新的呢？ 在这之前，我们得先学习一个概念叫做**指数加权平均**， 指数加权平均**在时间序列中经常用于求取平均值的一个方法**，它的思想是这样，我们要求取当前时刻的平均值，距离当前时刻越近的那些参数值，它的参考性越大，所占的权重就越大，这个权重是**随时间间隔的增大呈指数下降**，所以叫做**指数滑动平均**。公式如下：

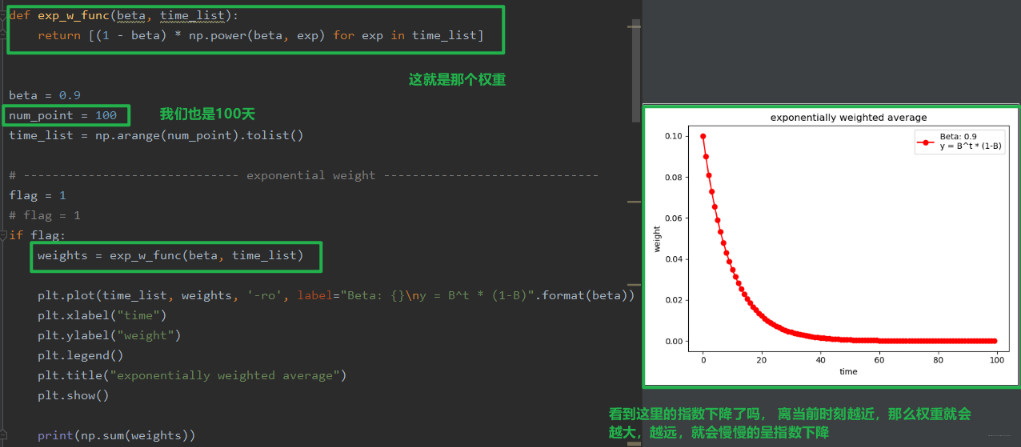


当然这个公式看起来还是很抽象，丝毫没有看出点指数滑动的意思， 那么还是用吴恩达老师PPT里的一个例子解释一下吧：



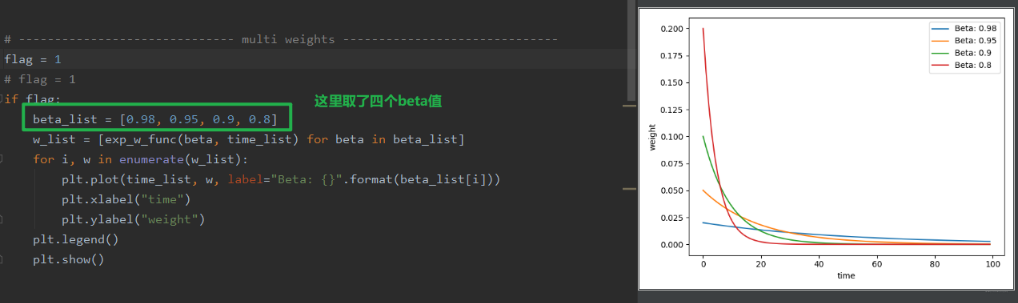
看上面这个温度图像，横轴是第几天，然后纵轴是温度， 假设我想求第100天温度的一个平均值，那么根据上面的公式：

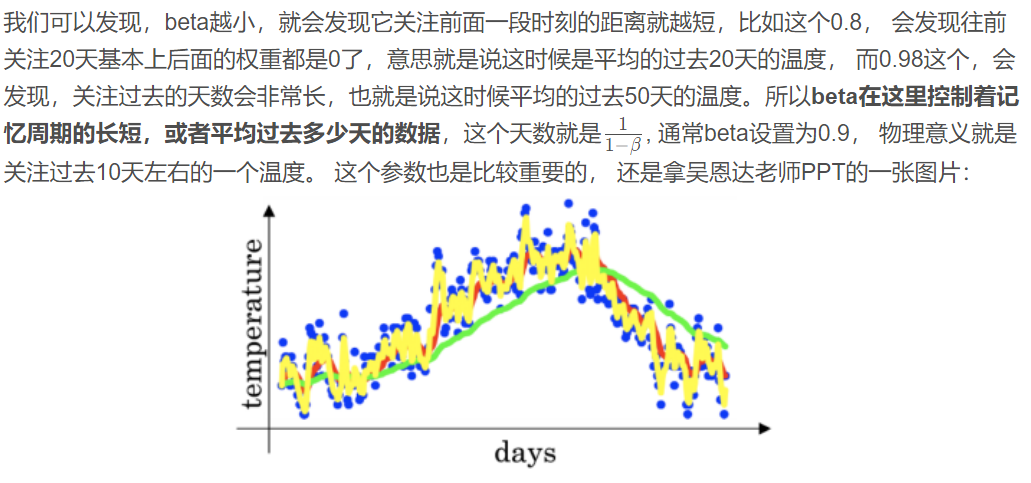




距离当前时刻越远，对当前时刻的一个平均值影响就越小。距离当前时刻越近，对当前时刻的一个平均值影响越大，这就是指数加权平均的思想了。

这里我们发现，有个超参数β, 这个到底是干嘛的？ 我们先来观察一个图， 还是上面的代码，我们设置不同的β来观察一下这个**权重的变化曲线**：





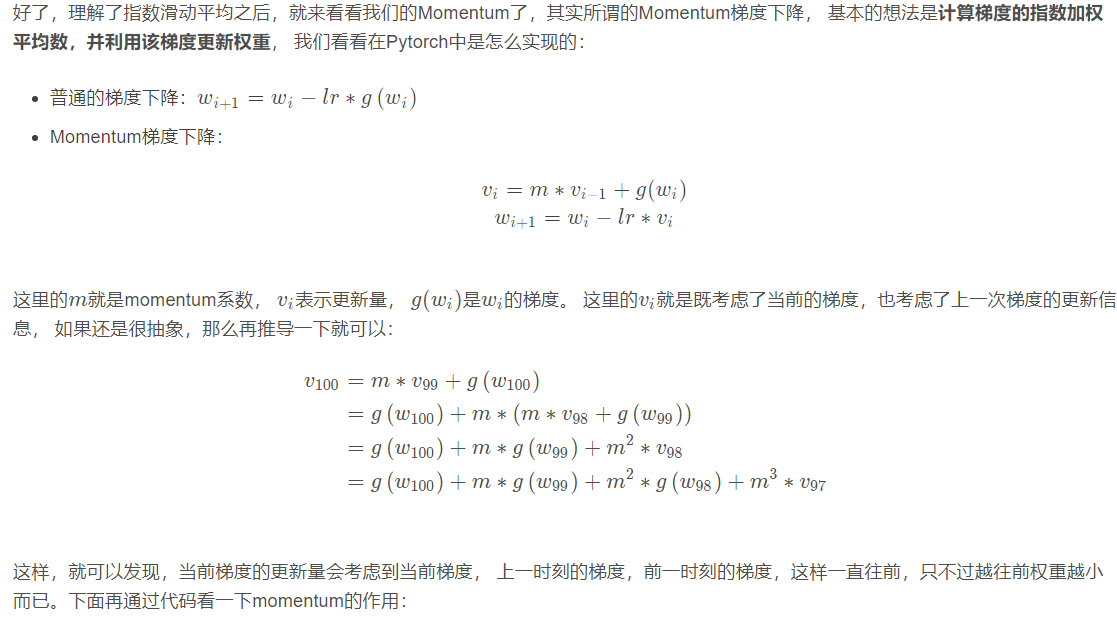
看上图，是不同beta下得到的一个温度变化曲线

红色的那条，是beta=0.9， 也就是过去10天温度的平均值

绿色的那条，是beta=0.98， 也就是过去50天温度的平均值

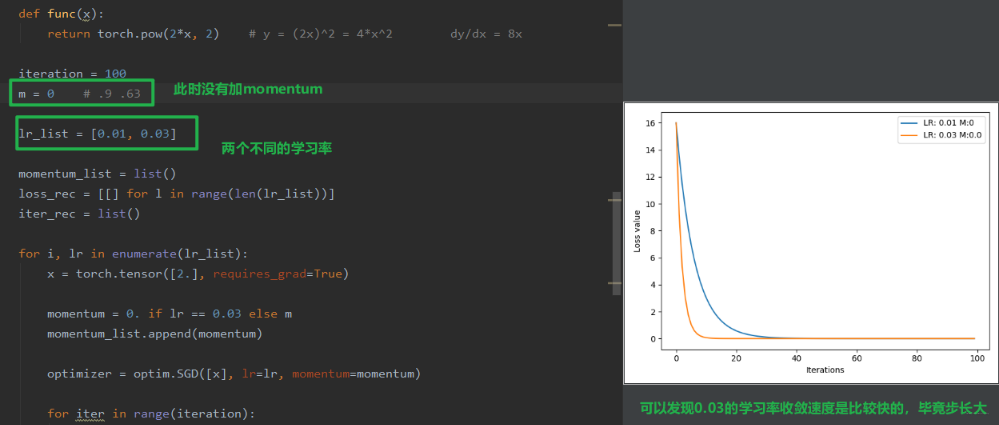
黄色的那条，beta=0.5， 也就是过去2天的温度的平均





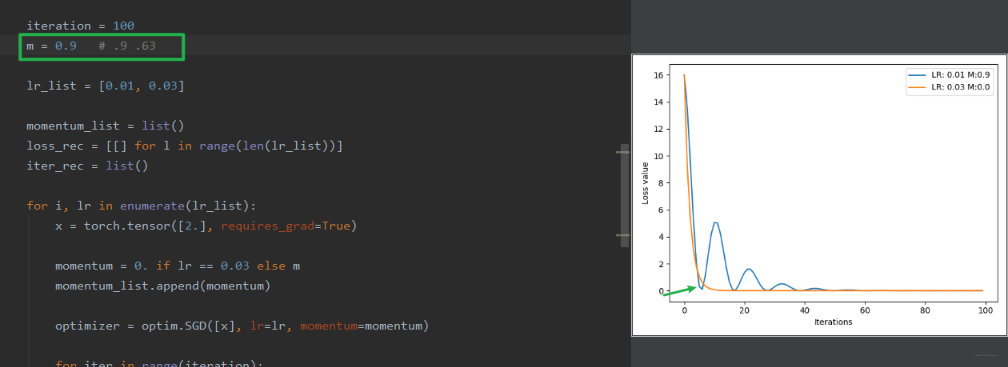
我们用0.01和0.03两个学习率，训练模型，我们看看loss的变化曲线：

**结果：0.03收敛速度比较快**

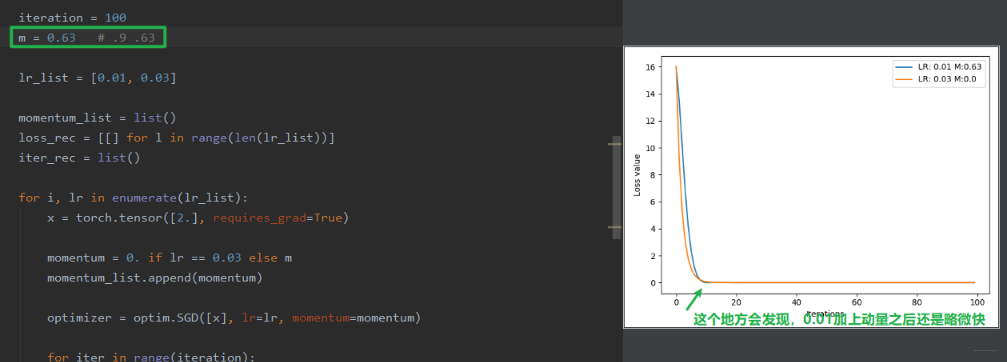


现在，我们给学习率0.01的这个加一个动量momentum，再看看效果：

**结果：可以看到加上动量的0.01收敛的速度快了，但是前面会有震荡**， 这是因为这里的m太大了，当日温度的权重太小，所以前面梯度一旦大小变化，这里就会震荡，当然会发现震荡会越来越小最后趋于平缓，这是因为不断平均的梯度越来越多。



这时候假设我们减少动量m， 效果会好一些，比如0.63



好了，学习率和动量解释清楚了，下面就看看常用的优化器了。

**2.5.3 常用优化器介绍**

**1、optim.SGD**

param: 管理的参数组

lr: 初始学习率

momentum：动量系数，beta

weight\_decay: L2正则化系数

nesterov: 是否采用NAG

这个优化器是非常常用的。

然后下面列出10款优化器，具体的不去介绍，可以大体了解有哪些优化器可用：

optim.SGD: 随机梯度下降法

optim.Adagrad: 自适应学习率梯度下降法

optim.RMSprop: Adagrad的改进

optim.Adadelta: Adagrad的改进

optim.Adam: RMSprop结合Momentum

optim.Adamax: Adam增加学习率上限

optim.SparseAdam: 稀疏版的Adam

optim.ASGD: 随机平均梯度下降

optim.Rprop: 弹性反向传播

optim.LBFGS: BFGS的改进

**3、学习率调整策略**

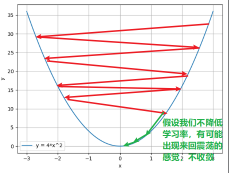
上面我们已经学习了优化器，在优化器当中有很多超参数，例如学习率，动量系数等，这里面最重要的一个参数就是学习率。它直接控制了参数更新步伐的大小，整个训练当中，学习率也不是一成不变的，也可以调整和变化。 所以下面整理学习率的调整策略，首先是为什么要调整学习率，然后是Pytorch的六种学习率调整策略，最后是小结一下：

**3.1 为什么要调整学习率**

学习率是可以控制更新的步伐的。 我们在训练模型的时候，一般开始的时候学习率会比较大，这样可以以一个比较快的速度到达最优点的附近，然后再把学习率降下来， 缓慢的去收敛到最优值。 这样说可能比较抽象，玩过高尔夫球吗？ 我们可以看一个例子：

开始的时候，一般是大力把球打到洞口的旁边，然后再把力度降下来，一步步的把球打到洞口，这里的学习率调整也差不多是这个感觉。

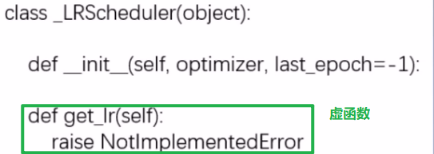
当然，再看一个函数的例子也行：



所以，在模型的训练过程中，调整学习率也是非常重要的，**学习率前期要大，后期要小**。Pytorch中提供了一个很好的学习率的调整方法，下面我们就来具体学习，学习率该如何进行调整。

**3.2 Pytorch的学习率调整策略**

在学习 学习率调整策略 之前，得先学习一个基类， 因为后面的六种学习率调整策略都是继承于这个类的，所以得先明白这个类的原理：



**主要属性：**

optimizer: 关联的优化器， 得需要先关联一个优化器，然后再去改动学习率

last\_epoch: 记录epoch数， 学习率调整以epoch为周期

base\_lrs: 记录初始学习率

**主要方法：**

step(): 更新下一个epoch的学习率， 这个是和用户对接

get\_lr(): 虚函数，计算下一个epoch的学习率， 这是更新过程中的一个步骤

下面关于**优化器的定义和使用的内部运行原理**就可以稍微总结了：

首先我们在定义优化器的时候，这时候会完成优化器的初始化工作，主要有关联优化器(self.optimizer属性), 然后初始化last\_epoch和base\_lrs(记录原始的学习率，后面get\_lr方法会用到)。

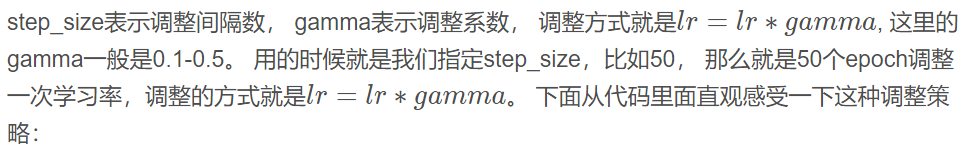
然后就是用Scheduler，我们是直接用的step()方法进行更新下一个epoch的学习率（这个千万要注意**放到epoch的for循环里面** 而不要放到batch的循环里面 ），而这个内部是在\_Scheduler类的step()方法里面调用了get\_lr()方法，而这个方法需要我们写Scheduler的时候自己覆盖，告诉程序按照什么样的方式去更新学习率，这样程序根据方式去计算出下一个epoch的学习率，然后直接更新进优化器的\_param\_groups()里面去。

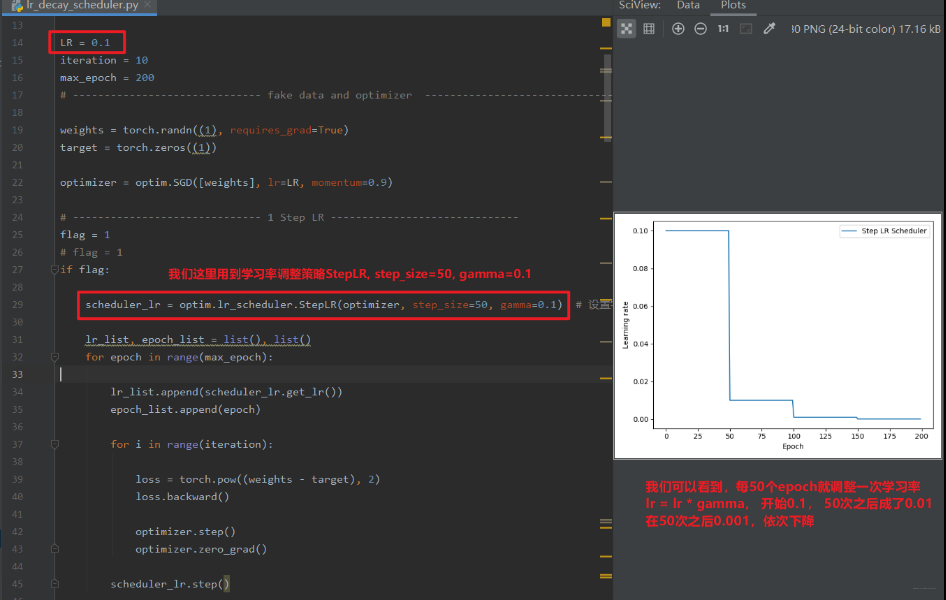
**3.3 Pytorch提供的六种学习率调整策略**

**1、StepLR**

功能： 等间隔调整学习率

截图.png

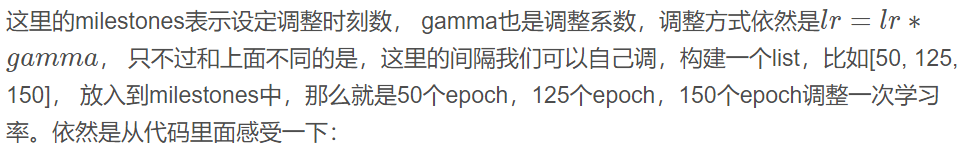


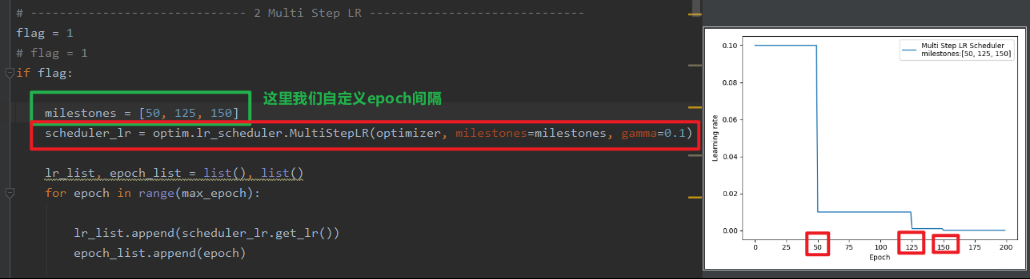


2、MultiStepLR

功能： 按给定间隔调整学习率

截图.png





3、ExponentialLR

功能：按指数衰减调整学习率

截图.png

截图.png



4、CosineAnnealingLR

功能：余弦周期调整学习率

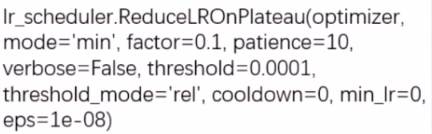
截图.png





5、ReduceLROnPlateau

功能: 监控指标， 当指标不再变化则调整， **这个非常实用**。可以监控loss或者准确率，当不在变化的时候，我们再去调整。



**主要参数：**

mode: min/max两种模式（min就是监控指标不下降就调整，比如loss，max是监控指标不上升就调整， 比如acc）

factor: 调整系数，类似上面的gamma

patience: “耐心”, 接受几次不变化， 这一定要是连续多少次不发生变化

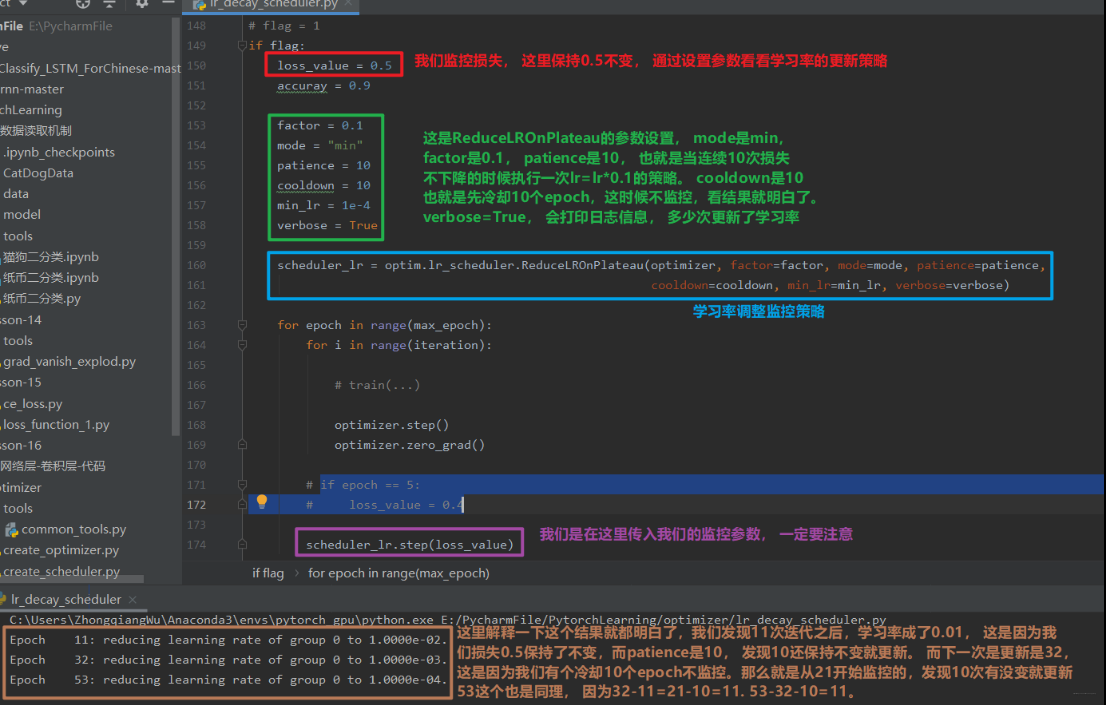
cooldown: “冷却时间”, 停止监控一段时间

verbose： 是否打印日志， 也就是什么时候更新了我们的学习率

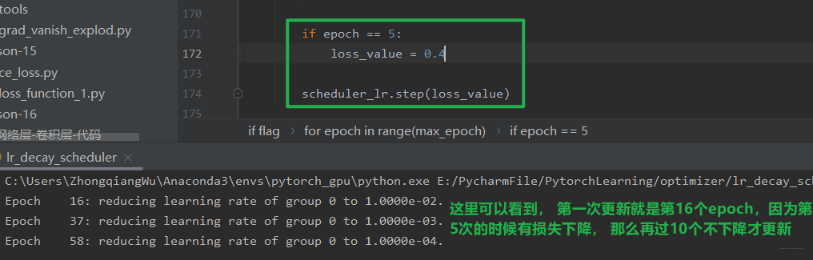
min\_lr: 学习率下限

eps: 学习率衰减最小值

下面我们直接从代码中学习这个学习率调整策略的使用：



上面是学习率一直保持不变，如果我们在第5个epoch更新一下子，那么这个更新策略会成什么样呢？



6、LambdaLR

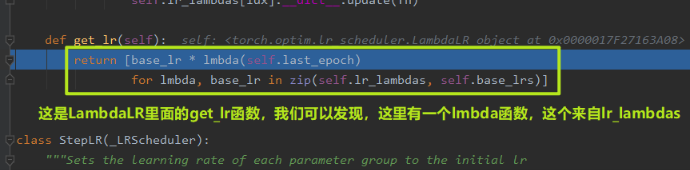
功能： 自定义调整策略，这个也比较实用，可以自定义我们的学习率更新策略，这个就是真的告诉程序我们想怎么改变学习率了。并且**还可以对不同的参数组设置不同的学习率调整方法，所以在模型的finetune中非常实用**。

截图.png

这里的lr\_lambda表示function或者是list。 这个我们从代码中进行学习：



但这个过程到底是怎么实现的呢？ 我们依然可以debug看一下过程，依然是调用get\_lr()函数，但是我们这里看看这里面到底是怎么实现自定义的：



我们再这里再次stepinto ,就会发现跳到了我们自定义的两个更新策略上来：



好了，**六种学习率调整策略已经整理完毕**，下面小结一下：

**有序调整：** Step、MultiStep、 Exponential和CosineAnnealing， 这些得事先知道学习率大体需要在多少个epoch之后调整的时候用

**自适应调整：** ReduceLROnPleateau， 这个非常实用，可以监控某个参数，根据参数的变化情况自适应调整

**自定义调整：**Lambda， 这个在模型的迁移中或者多个参数组不同学习策略的时候实用

调整策略就基本完了，那么我们得先有个初始的学习率啊， 下面介绍两种学习率初始化的方式：

设置较小数：0.01， 0.001， 0.0001

搜索最大学习率： 看论文《Cyclical Learning Rates for Training Neural Networks》, 这个就是先让学习率从0开始慢慢的增大，然后观察acc， 看看啥时候训练准确率开始下降了，就把初始学习率定为那个数。



**4、总结**

今天的内容就是这些了，还是有点多的，依然是快速梳理一遍：

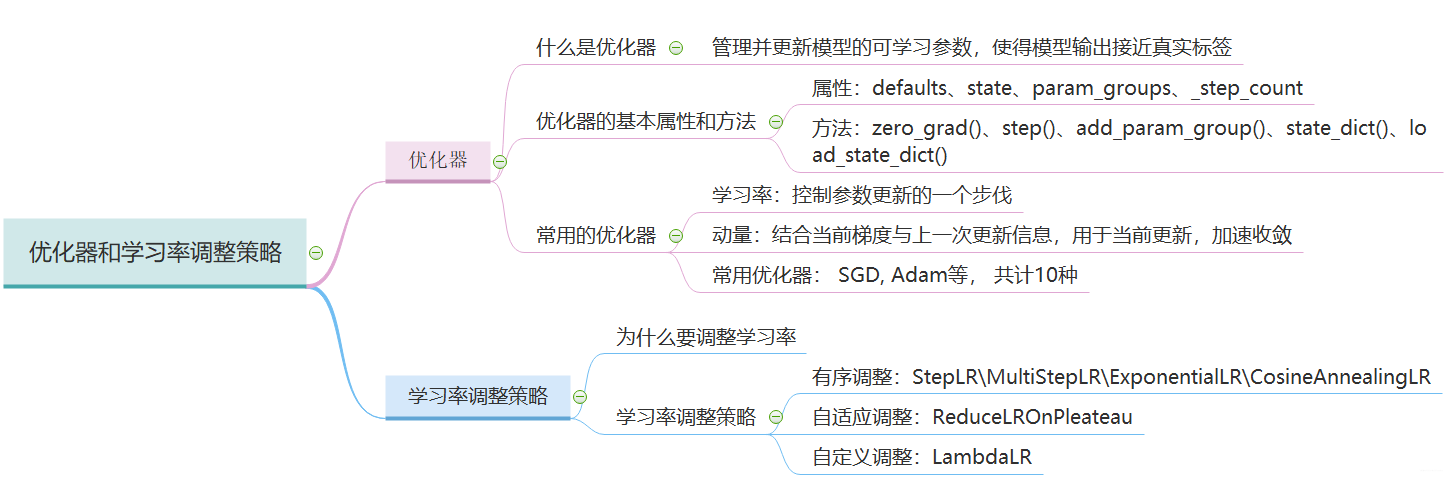
我们今天开始的优化器模块，优化器管理更新参数，不断降低损失。

首先从优化器本身开始学习，学习了优化器的基本属性和方法，并通过代码调试的方式了解了优化器的初始化和使用原理。

然后学习了常用的优化器，介绍了两个非常关键的概念学习率和动量，学习了SGD优化器。 优化器中非常重要的一个参数就是学习率，在模型的训练过程中，对学习率调整非常关键，

所以最后又学习了学习率的6种调整策略，从三个维度进行总结。

下面依然是一张导图把这次的知识拎起来，方便以后查阅：



好了， 优化器模块介绍完毕， 我们再回忆一下机器模型训练的五个步骤， 数据模块 -> 模型模块 -> 损失函数模块 -> 优化器模块 -> 迭代训练。 前四个模块我们都一一介绍完毕，下面我们进入最后一个模块的学习，这里面首先就是迭代训练过程中的可视化模块TensorBoard