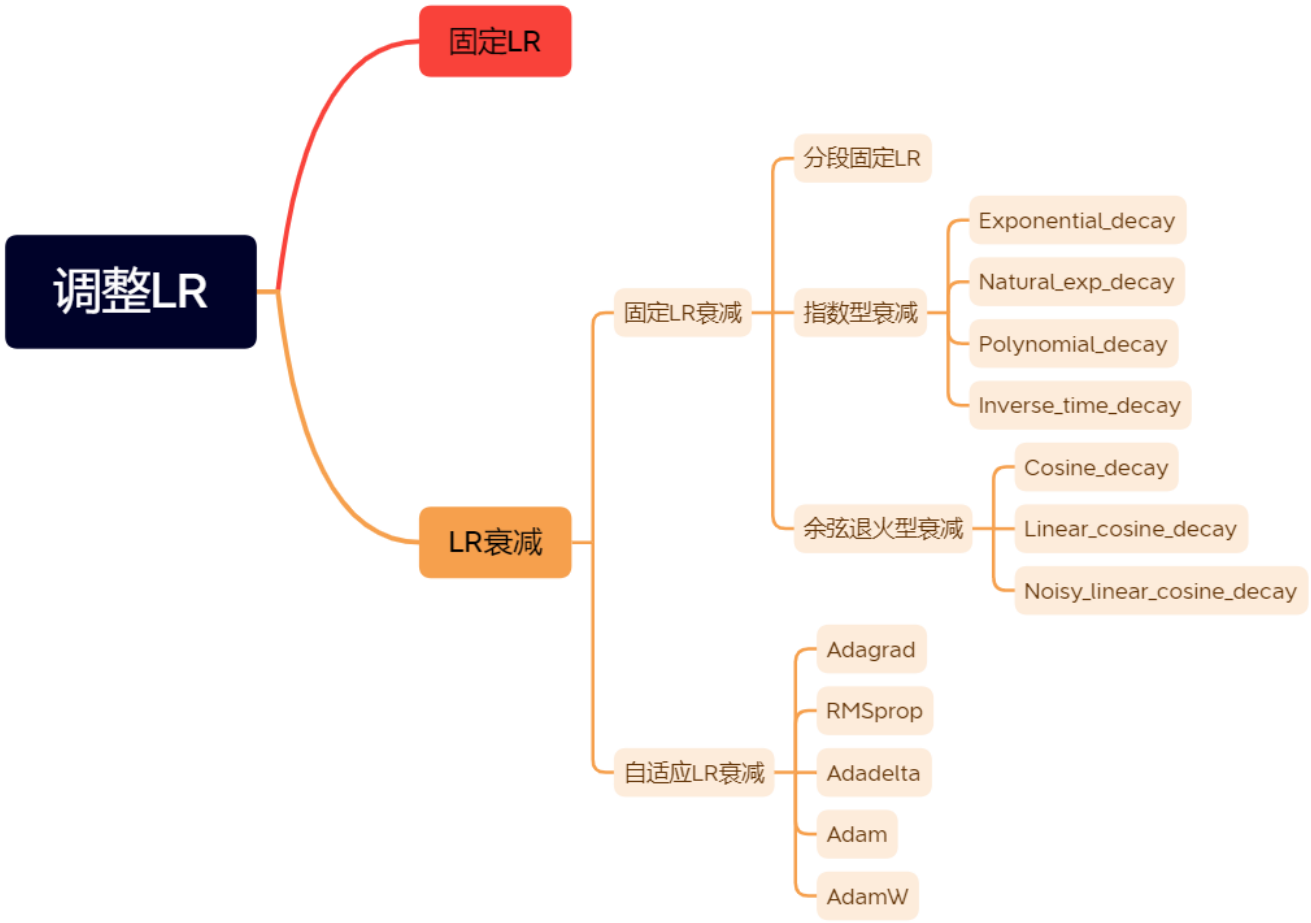
学习率（learning rate）

调整学习率的经典方法分类如下：

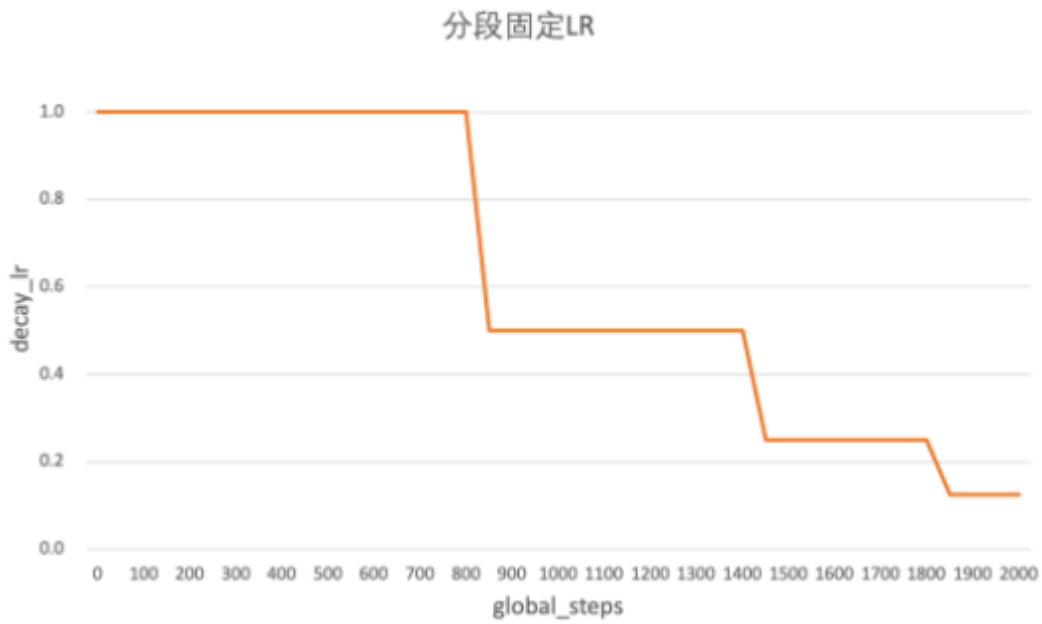


在使用过程中都是直接调包，但是了解其原理才能做出最优选择。

来源：泛函的范

1 固定学习率衰减

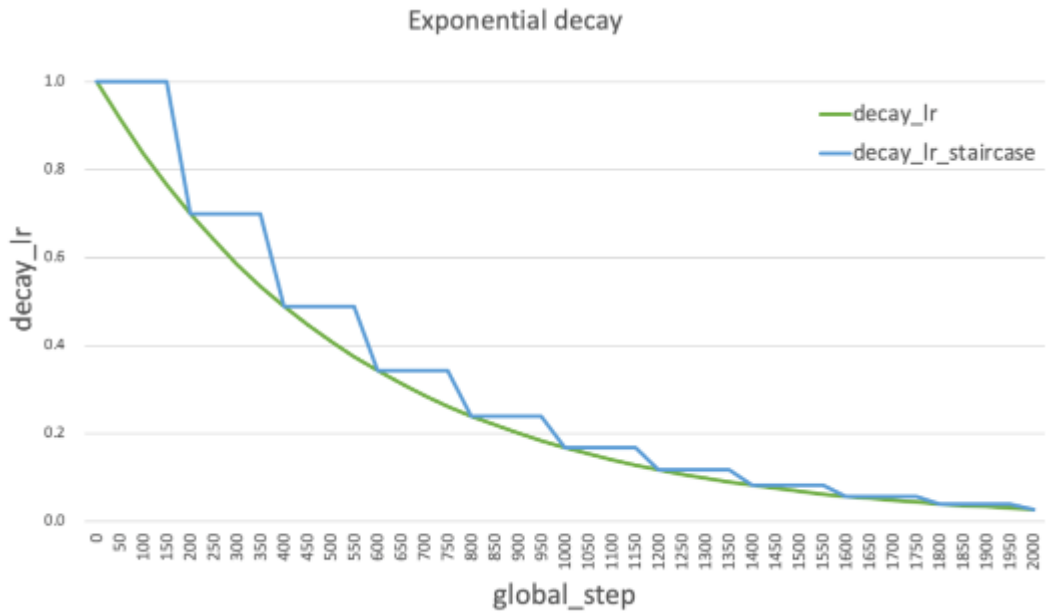
1.1 分段固定LR



对每个step设置固定的LR

1.2 指数型衰减

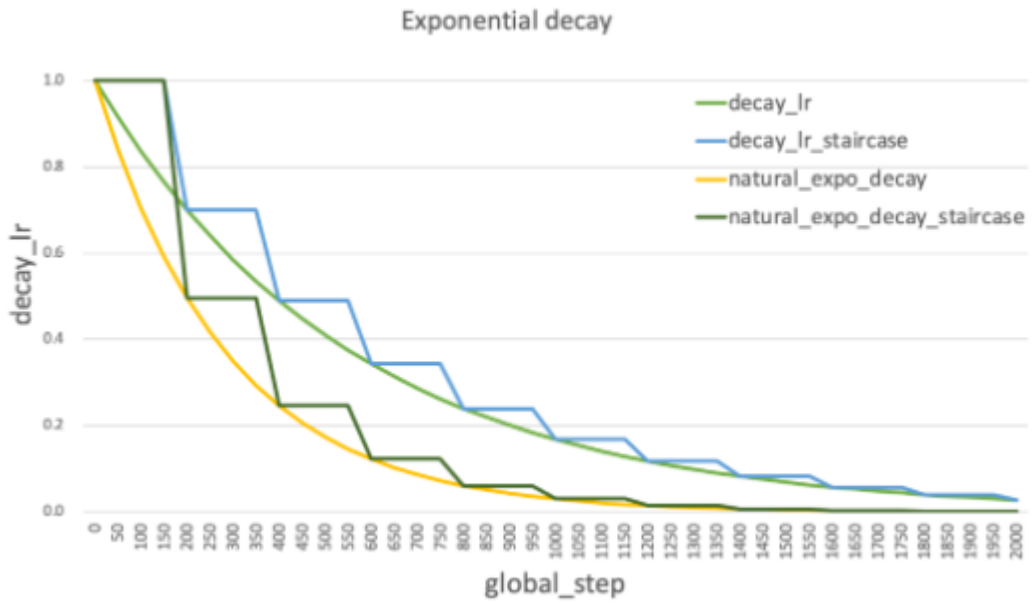
1.2.1 Exponential_decay



标准指数型衰减与阶梯式指数型衰减（为了在同一个衰减周期中保持相同的lr）

1.2.2 Natural_exp_decay

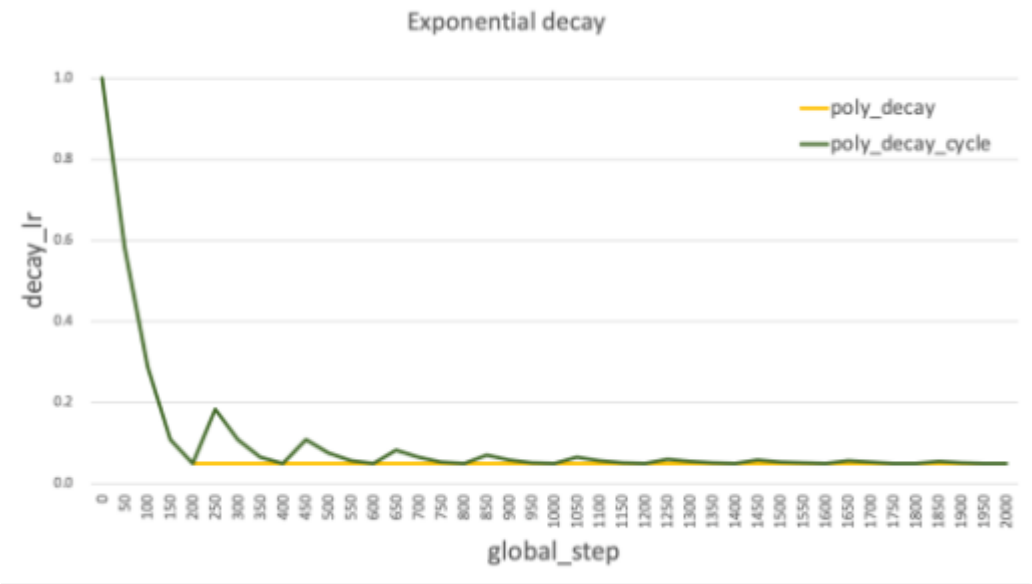
和上一个相比，使用了$\frac{1}{e}$作为底数



`natural_expo_decay`的lr下降速度要比`exponential_decay`快得多。因此，`natural_expo_decay`相对适合容易训练的网络，使模型在训练初期就快速收敛。

1.2.3 Polynomial_decay

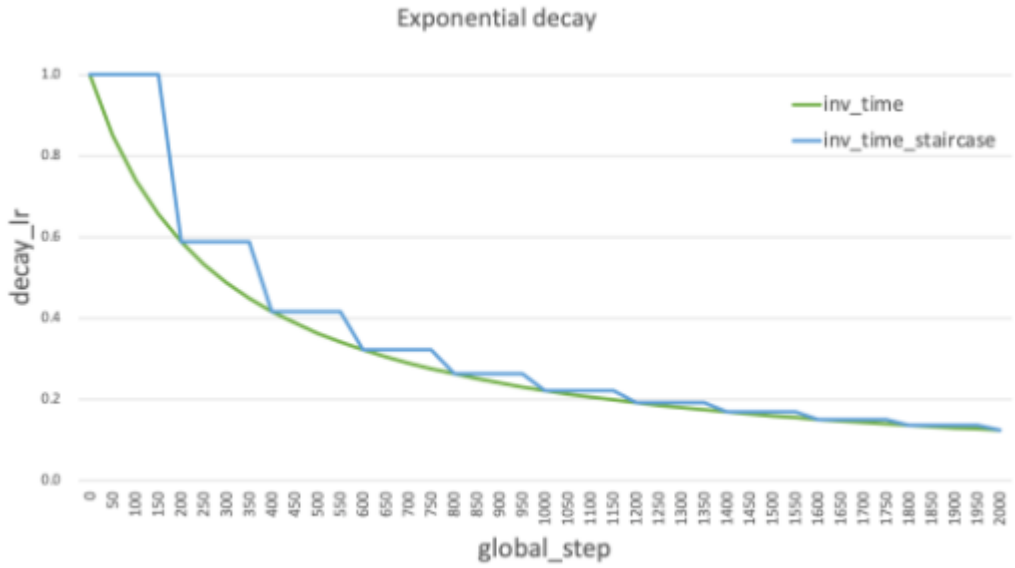
多项式衰减



调包时有一个参数`cycle`，用于决定lr是否在下降后重新上升，这是为了防止后期的lr非常小导致网络一直陷于某个局部极小点。通过突然调大lr可以帮助网络跳出该极小点，增加探索区域，从而找到更优的局部极小点。

1.2.4 Inverse_time_decay

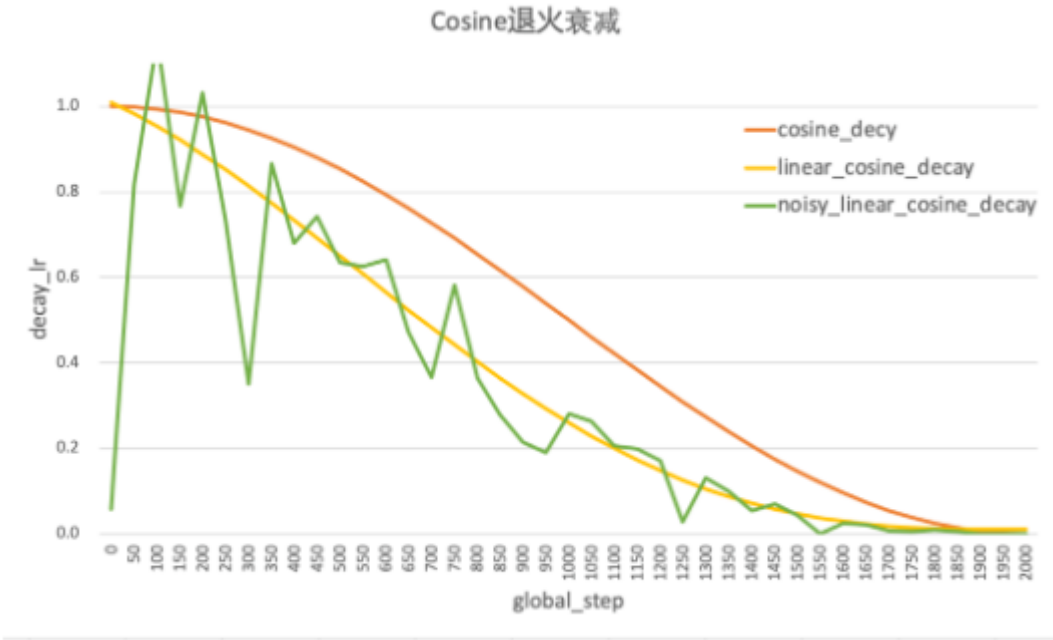
时间（迭代次数）倒数衰减



1.3 余弦退火衰减

近两年主流LLM都采用cosine decay的学习率策略

通过余弦函数来降低lr。如下图所示，lr首先缓慢下降，然后加速下降，再次缓慢下降。

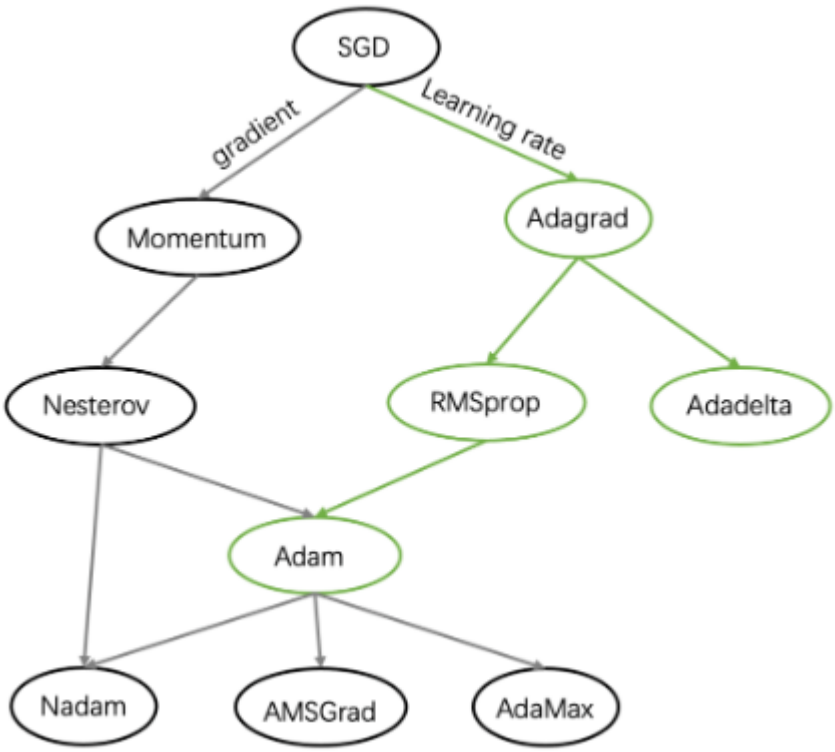


- **Cosine Decay**：仅采用余弦函数进行学习率衰减，使学习率在每个周期内逐渐减小。
- **Linear Cosine Decay**：在余弦衰减的基础上加入线性递减项，使学习率除了周期性振荡外，还能整体上缓慢下降。
- **Noisy Linear Cosine Decay**：在线性余弦衰减基础上添加随机噪声，使学习率在每次更新时有随机变化，从而避免陷入局部最优。

2 自适应学习率衰减

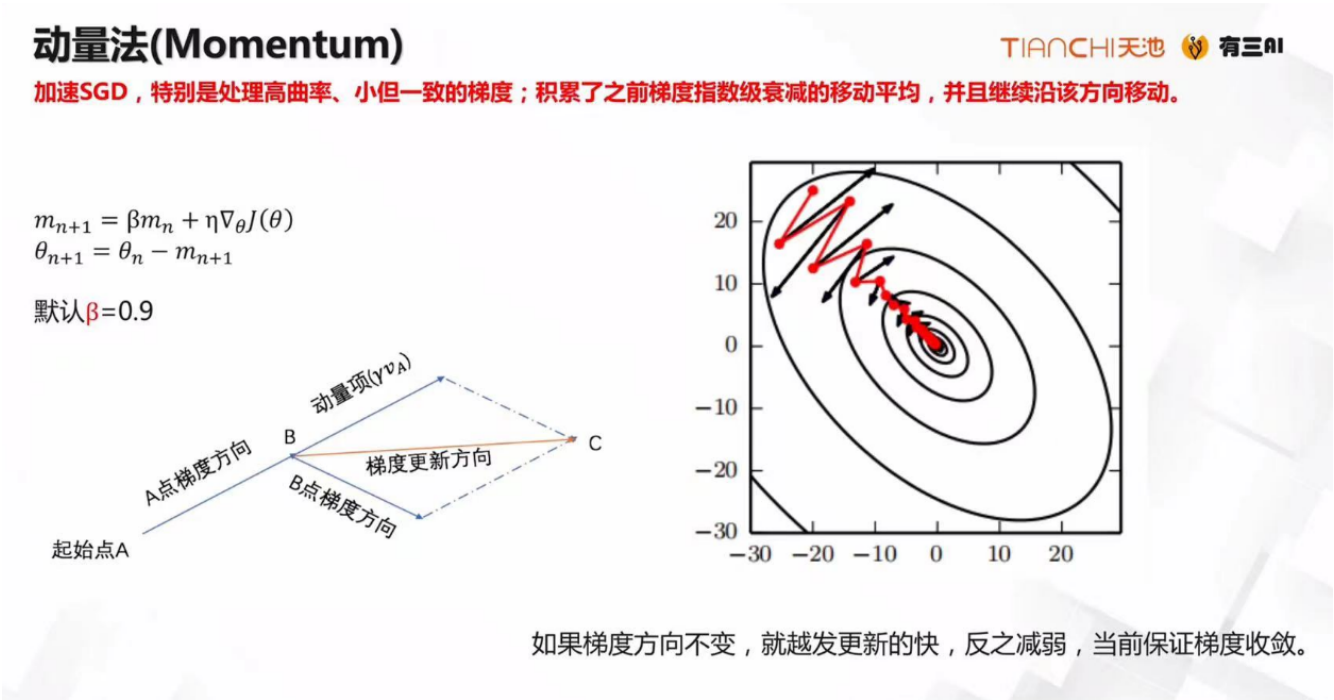
自适应学习率方法是梯度下降（`Gradient Descent`）法的一种。

下图给展示了优化算法的发展关系：



从图中可以看到，从SGD分别从梯度和学习率两个分支进行了发展。

来源：有三AI

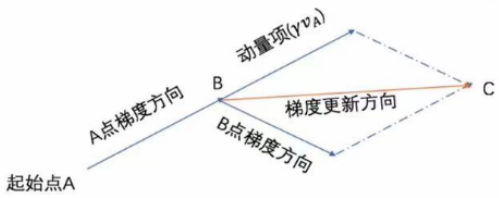


Nesterov accelerated gradient法

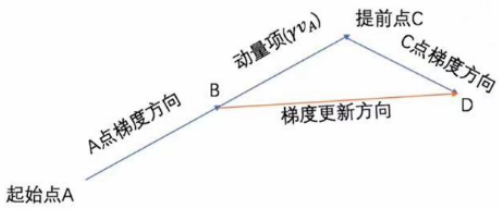
TIANCHI天池 有三AI

在标准动量方法中添加了一个校正因子

动量法 $$m_{n+1} = \beta m_n + \eta \nabla_\theta J(\theta)$$ $$\theta_{n+1} = \theta_n - m_{n+1}$$



NAG法 $$m_{n+1} = \beta m_n + \eta \nabla_\theta J(\theta - \beta m_n)$$ $$\theta_{n+1} = \theta_n - m_{n+1}$$



要求梯度下降更快，更加智能，直接先按照前一次梯度方向更新一步将它作为当前的梯度

自适应学习率方法是对SGD在学习率分支上的优化，常见的主要有五种：Adagrad、Adadelta、RMSprop、Adam、AdamW。

- **Adagrad**：使用梯度平方和来对学习率进行修正，核心思想是使用梯度累积信息来适应每个参数的学习率。这意味着，如果一个参数的梯度变化较大，其学习率会逐渐减小，而梯度变化较小的参数学习率则相对较大。但学习率会持续衰减最终趋于0。

Adagrad法

TIANCHI天池 有三AI

自适应地为各个维度的参数分配不同的学习率

g_t是当前的梯度，η是初始学习率，G_t是梯度平方累计值，ϵ是一个比较小的数。

$$G_t = G_{t-1} + g_t^2$$

$$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

- 优点：
  - g_t较小的时候，能够放大梯度，g_t较大的时候，能够约束梯度（激励+惩罚）。
- 缺点：
  - 梯度累积导致学习率单调递减，后期学习率非常小。
  - 需要设置一个合适的全局初始学习率 。

- **Adadelta**：**Adadelta** 是为了解决 Adagrad 学习率不断衰减的问题设计的，它通过限制累积梯度的窗口来取代全局累积的方法，使得过去一段时间内的梯度对当前更新产生影响。**不需要全局学习率（lr）**。
- **RMSprop**：结合 Adadelta 的思想，通过指数加权移动平均来调整学习率，从而适应不同的梯度情况。

Adadelta与Rmsprop法

TIANCHI天池 有三AI

Adadelta，Rmsprop与Adagrad不同，只累加了一个窗口的梯度，使用动量平均计算

RMSprop方法

$G_t = \beta G_{t-1} + (1-\beta) g_t^2$ ，默认β=0.9

$$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} g_t$$

Adadelta方法，与RMSprop方法不同在于学习率是动态计算的

$$\mathrm{E}[\Delta\theta^2]_t = \beta \mathrm{E}[\Delta\theta^2]_{t-1} + (1-\beta)\Delta\theta_t^2$$

$$\eta = \sqrt{\mathrm{E}[\Delta\theta^2]_{t-1} + \epsilon}$$

缺点：训练后期反复在局部最小值附近抖动。

- **Adam**：结合了 RMSprop 和 Momentum 的优点，使用动量的思想来计算梯度的一阶和二阶矩估计，然后对这些矩进行偏差校正从而提高训练效果。

Adam法

TIANCHI天池 有三AI

同时包含了动量更新与学习率调整，使用梯度的一阶矩估计和二阶矩估计来动态调整每个参数的学习率，momentum和RMSProp结合。

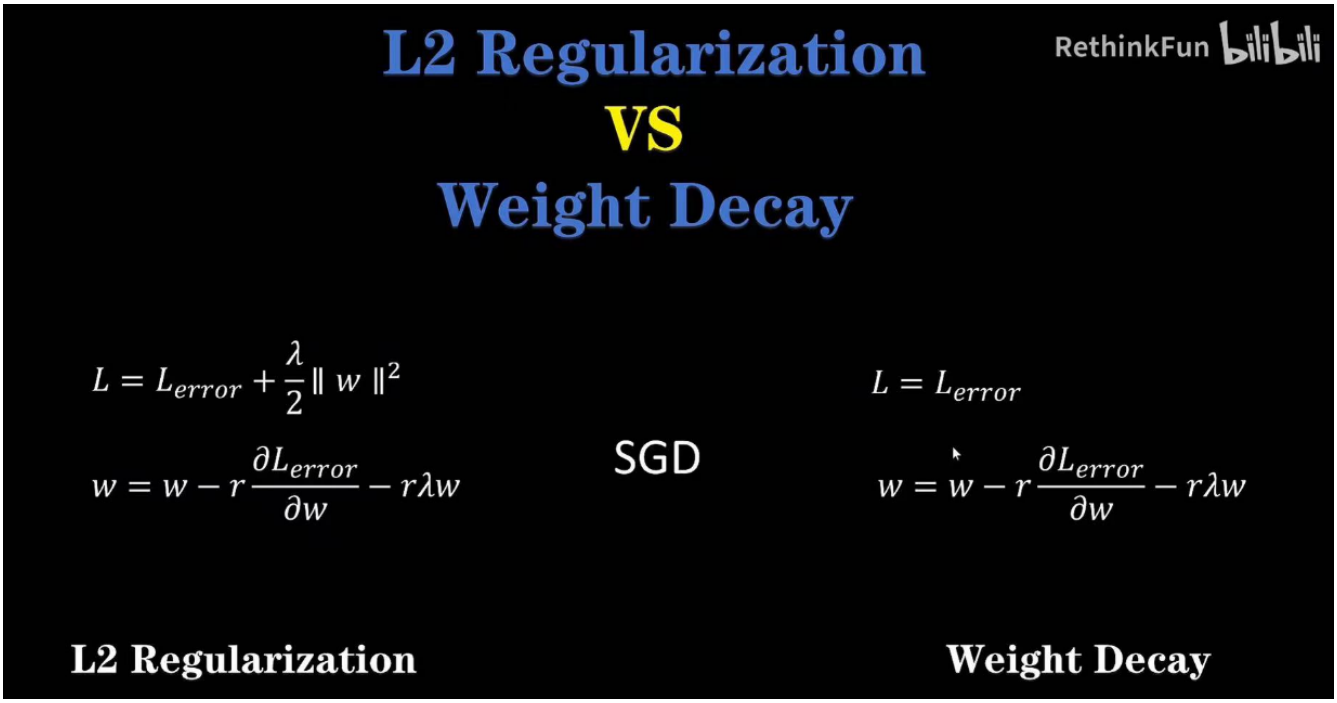
$$m_t = \beta_1 * m_{t-1} + (1-\beta_1) * g_t$$
$$n_t = \beta_2 * n_{t-1} + (1-\beta_2) * g_t^2$$

$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t}$$
$$\widehat{n_t} = \frac{n_t}{1-\beta_2^t}$$
$$\Delta\theta_t = -\frac{\widehat{m_t}}{\sqrt{\widehat{n_t} + \epsilon}} * \eta$$

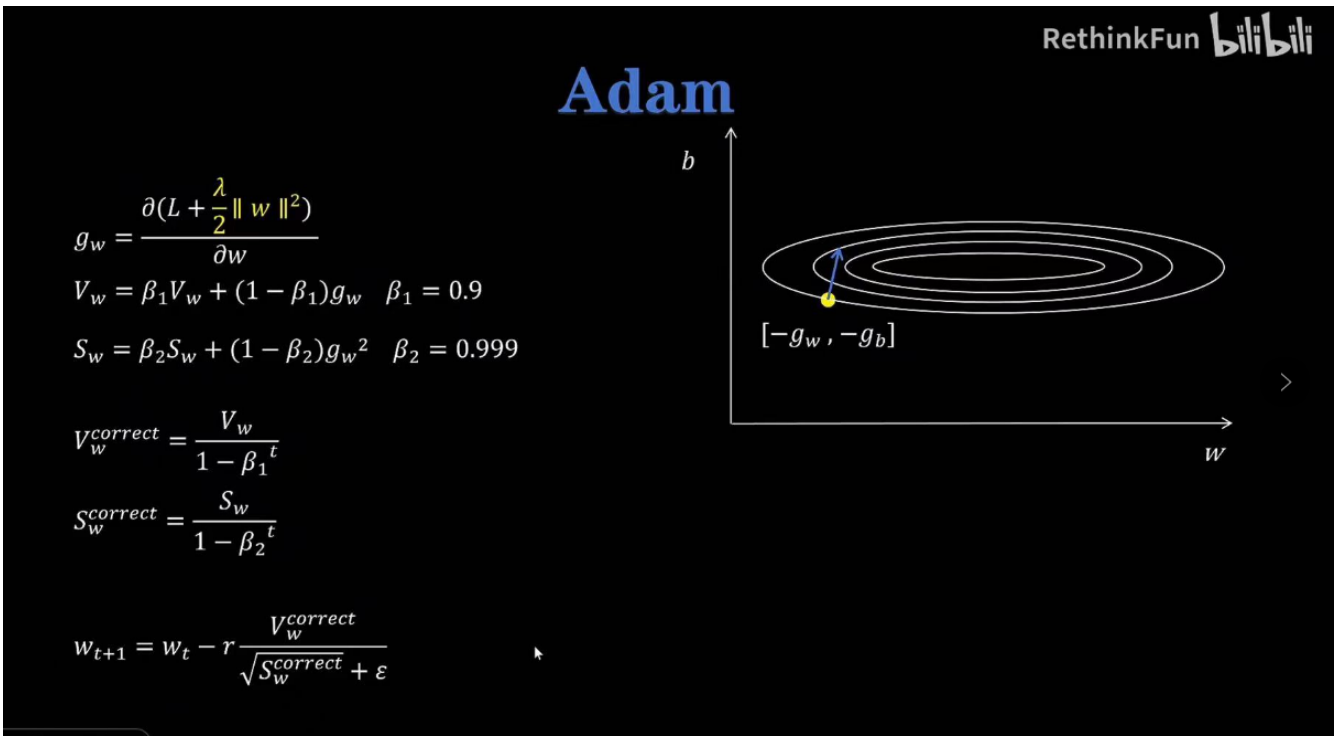
默认β_1 =0.9，β_2 =0.99

- 优点：
  - 对学习率没有那么敏感，学习步长有一个确定的范围，参数更新比较稳。
- 缺点：
  - 学习率在训练的后期仍然可能不稳定，导致无法收敛到足够好的值，泛化能力较差。

对于传统SGD算法来说，权值衰减正好跟l2正则是等价的，因此，在大多数优化机器学习库中，l2正则被作为权值衰减来实现。

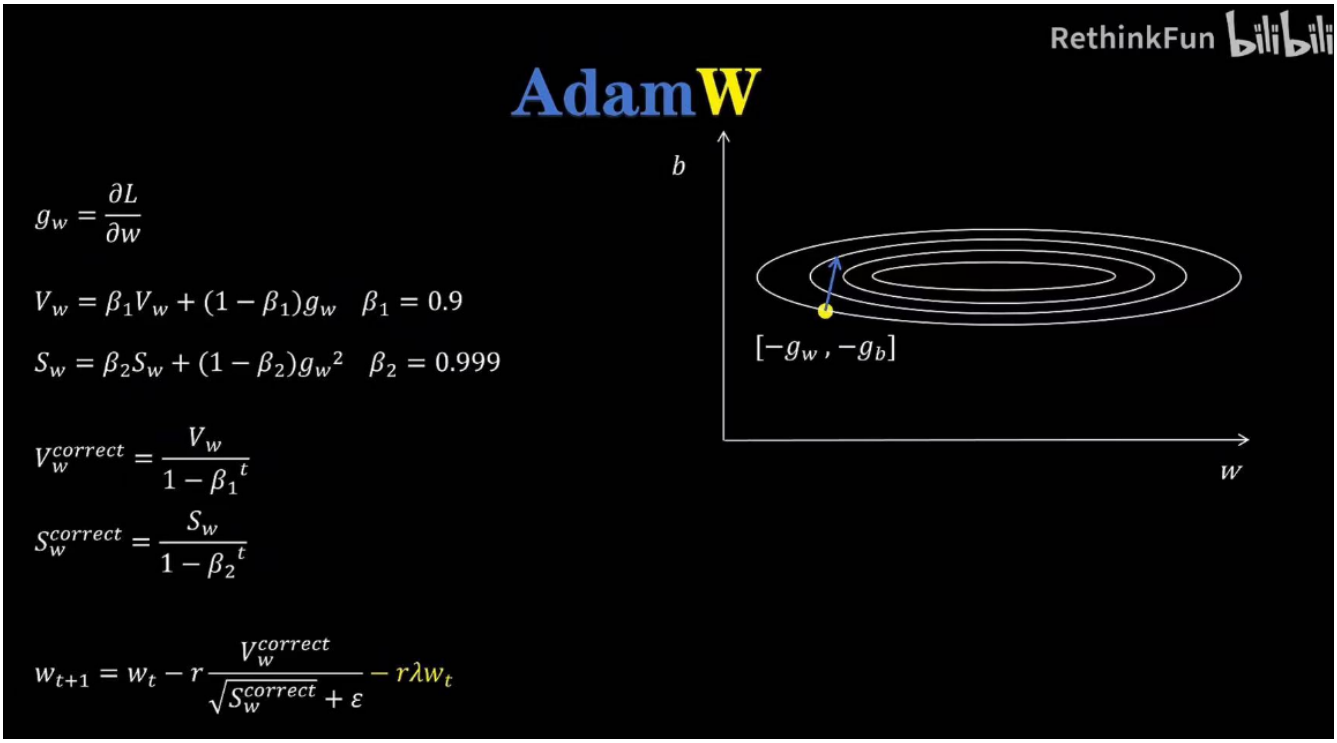


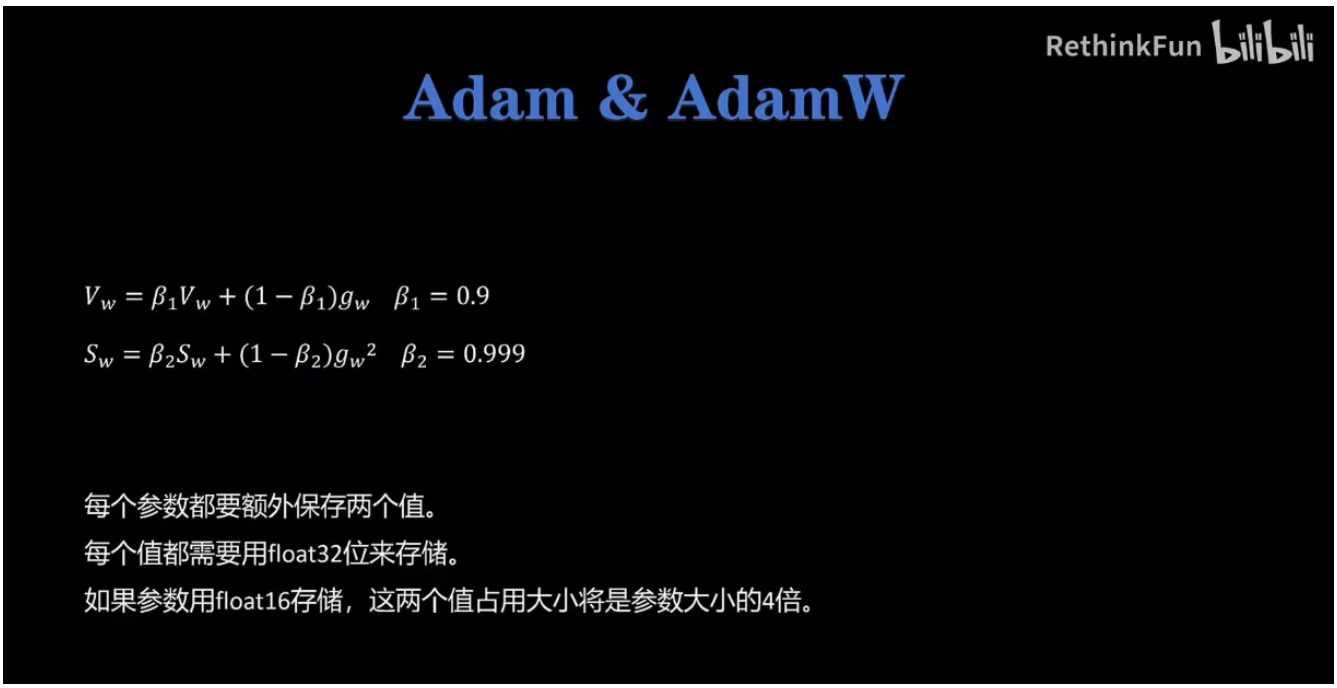
但是，l2正则和权值衰减的等价关系在自适应学习率衰减中并不成立，因为自适应学习率衰减方法会根据历史梯度动态调整每个参数的学习率。这些方法对梯度进行更复杂的处理，不再是简单的梯度相加减。以Adam为例：



为了解决这个问题，通过将权值衰减和梯度更新进行解耦，提出了AdamW：

- **AdamW**：作为 Adam 的增强版，通过分离权重衰减对梯度更新的影响，提供了更加有效的正则化策略，特别适用于深度学习中的大模型训练场景。





使用：

```
from keras import optimizers
from tensorflow_addons.optimizers import AdamW
# SGD
optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)
# Adagrad
optimizers.Adagrad(lr=0.01, epsilon=None, decay=0.0)
# RMSprop
optimizers.RMSprop(lr=0.001, rho=0.9, epsilon=None, decay=0.0)
# Adadelta
optimizers.Adadelta(lr=1.0, rho=0.95, epsilon=None, decay=0.0)
# Adam
```

```
optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
# AdamW
adamw_optimizer = AdamW(learning_rate=0.001, weight_decay=1e-5, beta_1=0.9, beta_2=0.999,
epsilon=1e-7)
```