# 凸优化

## 14. 梯度下降算法的变形和可视化解释

李 力
清华大学
Email: li-li@tsinghua.edu.cn

2009-2021

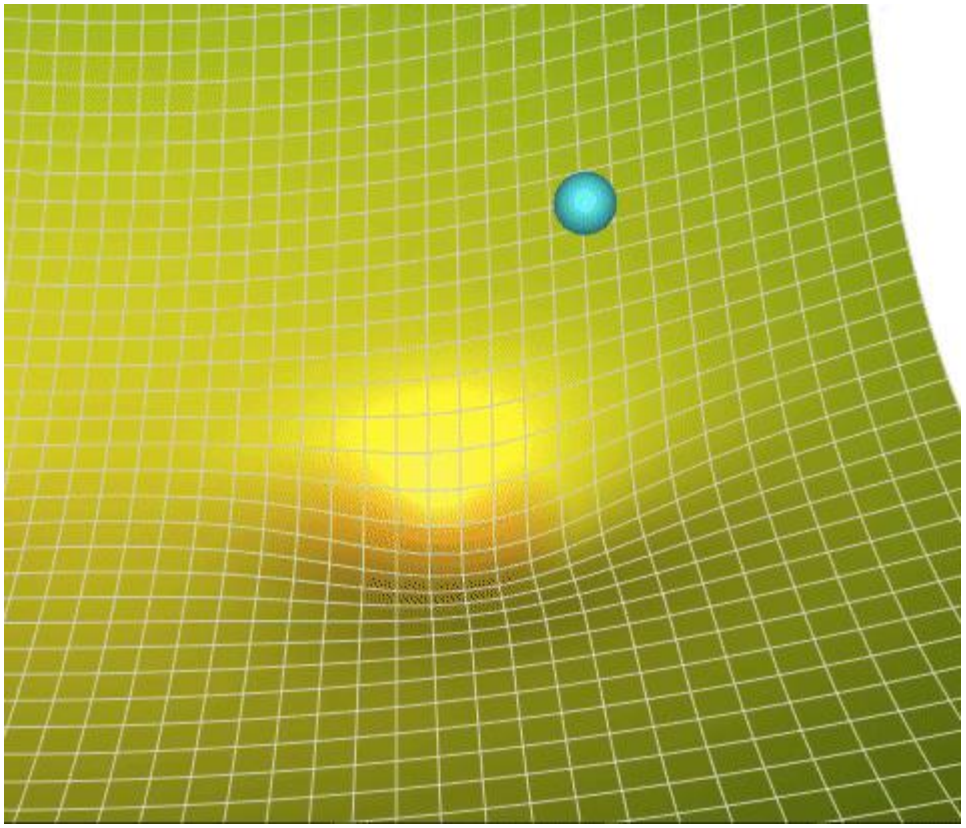
# 14.0. 提纲

# 14.1 Vanilla Gradient Descent

基本梯度下降算法遵循的思想是，梯度的相反方向指向较低的区域。 所以它在梯度的相反方向迭代。对于每个参数 theta，它做如下操作:

```
delta = - learning_rate * gradient
theta += delta
```

$$\theta \leftarrow \theta - \eta \cdot \nabla J(\theta)$$



# 14.2 Momentum

带有动量的梯度下降算法(简称动量)借鉴了物理学的思想。想象一下在无摩擦的碗里滚动一个球。没有在底部停止，而是积累的动量推动它前进，球继续前后滚动。在每个步骤中，除了常规的梯度之外，它还考虑了前一步中的移动。在数学上，它通常表示为:

```
delta = - learning_rate * gradient - previous_delta * decay_rate
theta += delta
```

$$m \leftarrow \gamma \cdot m + \eta \cdot \nabla J(\theta)$$
$$\theta \leftarrow \theta - m$$

或者进一步写成叠加模式

```
sum_of_gradient = gradient + previous_sum_of_gradient * decay_rate
delta = -learning_rate * sum_of_gradient
theta += delta
```

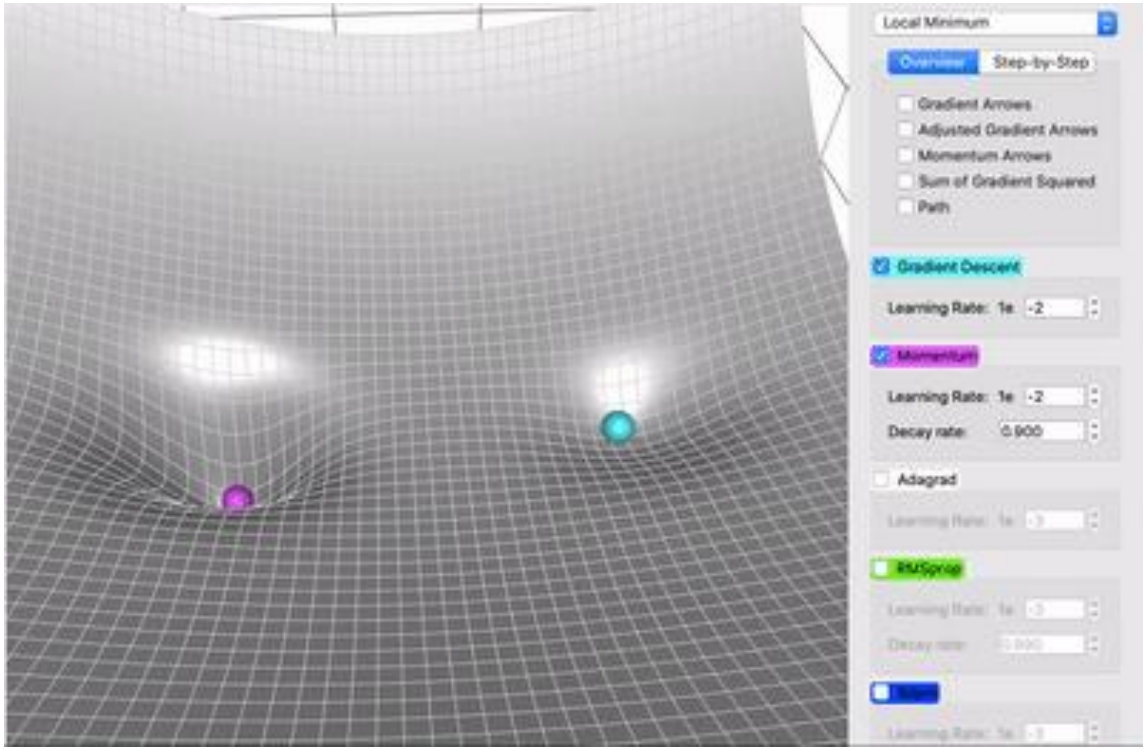
# 14.2 Momentum

动量比基本梯度下降有两个优点：

动量移动得更快(因为它积累的所有动量)

动量有机会逃脱局部极小值(因为动量可能推动它脱离局部极小值)。同样，它也将更好地通过高原区。

要不要加点摩檫力？



# 14.3 Nesterov Accelerated Gradient

NAG是momentum的改进，在梯度更新时做一个矫正：

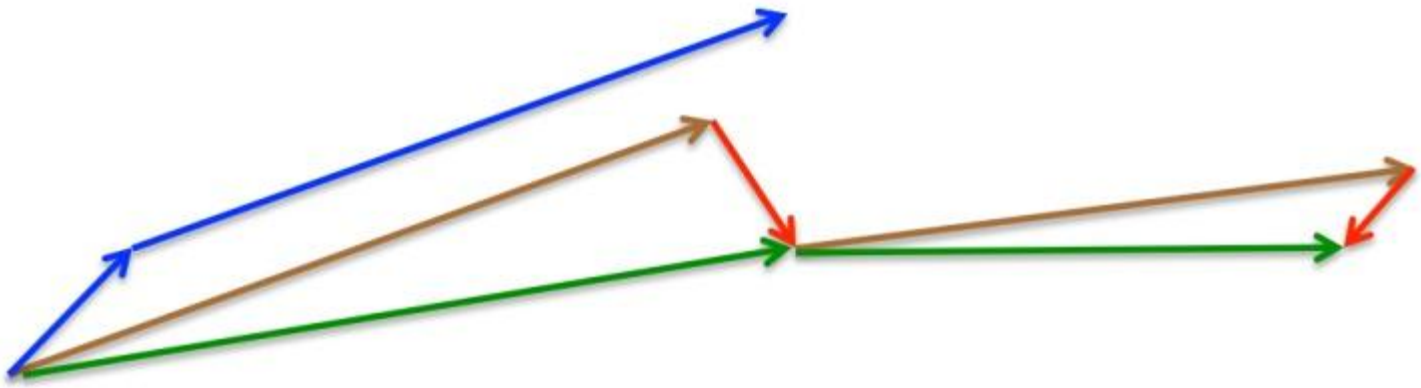
```
delta = - learning_rate * semi_gradient -
previous_delta * decay_rate
theta += delta
```

$$m \leftarrow \gamma \cdot m + \eta \cdot \nabla J(\theta - \gamma \cdot m)$$

$$\theta \leftarrow \theta - m$$

其中semi_gradient取的是(theta - decay_rate * old_delta)处的函数梯度。

也即momentum首先计算一个梯度(短的蓝色向量)，然后在加速更新梯度的方向进行一个大的跳跃(长的蓝色向量)，nesterov项首先在之前加速的梯度方向进行一个大的跳跃(棕色向量)，计算梯度然后进行校正(绿色梯向量)



# 14.3 Nesterov Accelerated Gradient

我们可以对NAG原来的更新公式进行变换，得到等效形式

$$d_i = \beta d_{i-1} + g(\theta_{i-1}) + \beta[g(\theta_{i-1}) - g(\theta_{i-2})]$$
$$\theta_i = \theta_{i-1} - \alpha d_i$$

这个NAG的等效形式与Momentum的区别在于，本次更新方向多加了一个 $\beta[g(\theta_{i-1}) - g(\theta_{i-2})]$ ，它的直观含义是：

能够让算法提前看到前方的地形梯度，如果前面的梯度比当前位置的梯度大，那我就可以把步子迈得比原来大一些，如果前面的梯度比现在的梯度小，那我就可以把步子迈得小一些。这个大一些、小一些，都是相对于原来不看前方梯度、只看当前位置梯度的情况来说的。

所以NAG本质上是多考虑了目标函数的二阶导信息，可以加速收敛！比喻起来是说“往前看”，数学本质上则是利用了目标函数的二阶导信息。

# 14.4 AdaGrad

Adaptive Gradient 算法，简称 AdaGrad，不是像动量一样跟踪梯度之和，而是跟踪梯度平方之和，并使用这种方法在不同的方向上调整梯度。对于每个维度：

```
sum_of_gradient_squared = previous_sum_of_gradient_squared + gradient²
delta = -learning_rate * gradient / sqrt(sum_of_gradient_squared)
theta += delta
```

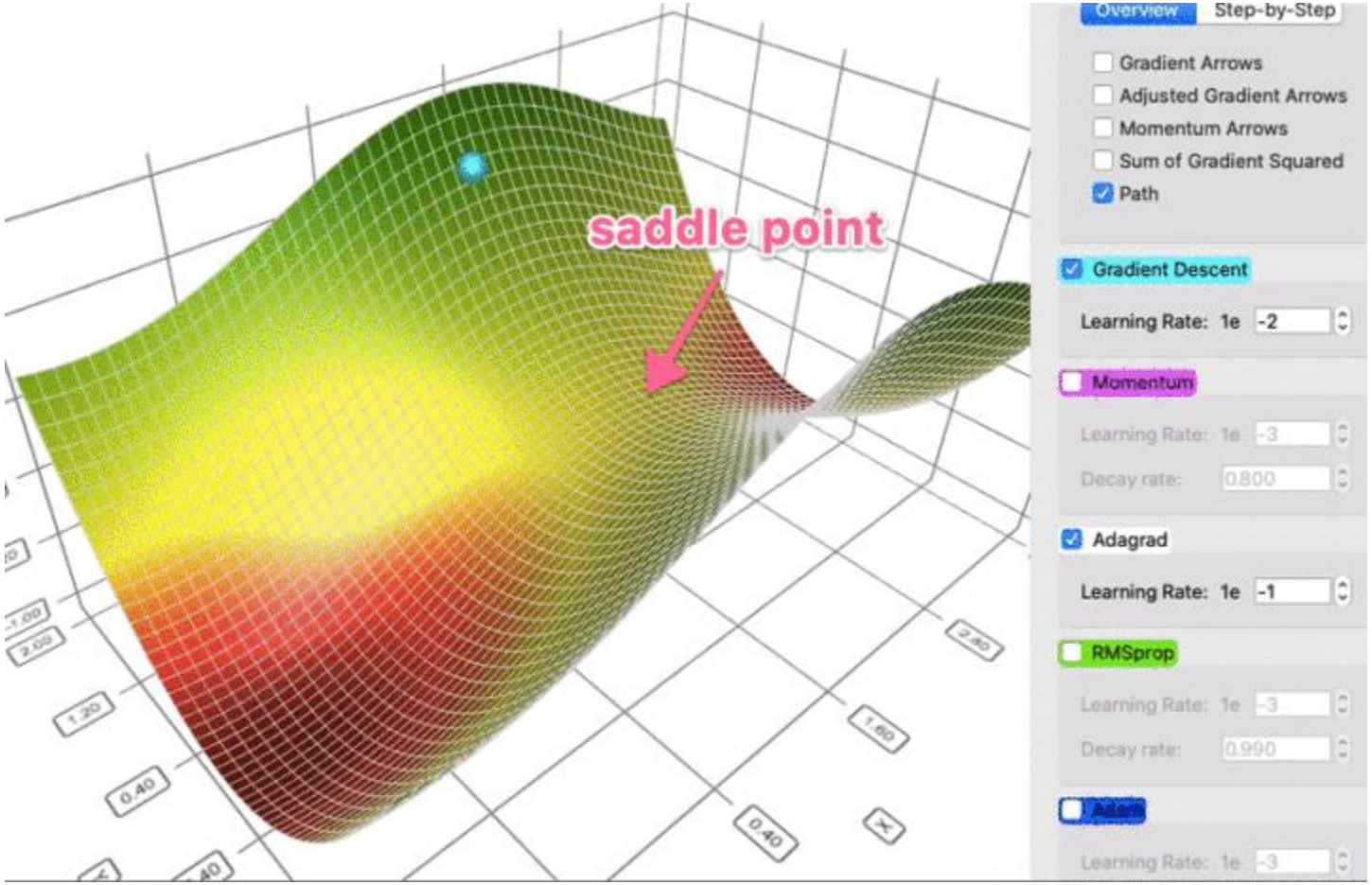
$$s \leftarrow s + \nabla J(\theta) \odot \nabla J(\theta)$$

$$\theta \leftarrow \theta - \frac{\eta}{\sqrt{s+\varepsilon}} \odot \nabla J(\theta)$$

在机器学习优化中，一些特征是非常稀疏的。稀疏特征的平均梯度通常很小，所以这些特征的训练速度要慢得多。解决这个问题的一种方法是为每个特征设置不同的学习率，但这很快就会变得混乱。Adagrad 解决这个问题的思路是：你已经更新的特征越多，你将来更新的就越少，这样就有机会让其它特征（例如稀疏特征）赶上来。

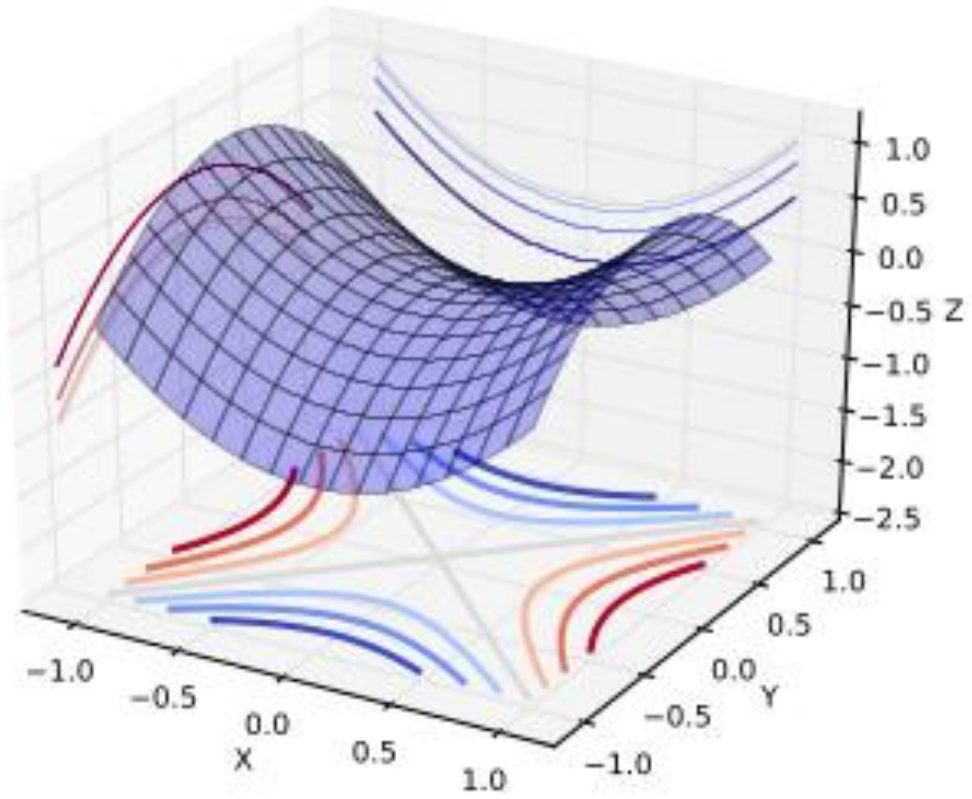
# 14.4 AdaGrad

用可视化的术语来说，更新这个特征的程度即在这个维度中移动了多少，这个概念由梯度平方的累积和表达。下面的图里面，基本梯度让球走直角移动， 如果Adagrad调整合适，球会沿对角线方向移动



# 14.4 AdaGrad

这个属性让 AdaGrad(以及其它类似的基于梯度平方的方法，如 RMSProp和Adam)更好地避开鞍点。Adagrad将采取直线路径，而梯度下降(或相关的动量)采取的方法是“让我先滑下陡峭的斜坡，然后才可能担心较慢的方向”。有时候，原版梯度下降可能非常满足的仅仅停留在鞍点，那里两个方向的梯度都是0。



# 14.5 Rmsprop

然而，AdaGrad的问题在于它非常慢。这是因为梯度的平方和只会增加而不会减小。Rmsprop(Root Mean Square Propagation)通过添加衰减因子来修复这个问题:

```
sum_of_gradient_squared =
previous_sum_of_gradient_squared * decay_rate+
gradient² * (1- decay_rate)

delta = -learning_rate * gradient /
sqrt(sum_of_gradient_squared)

theta += delta
```

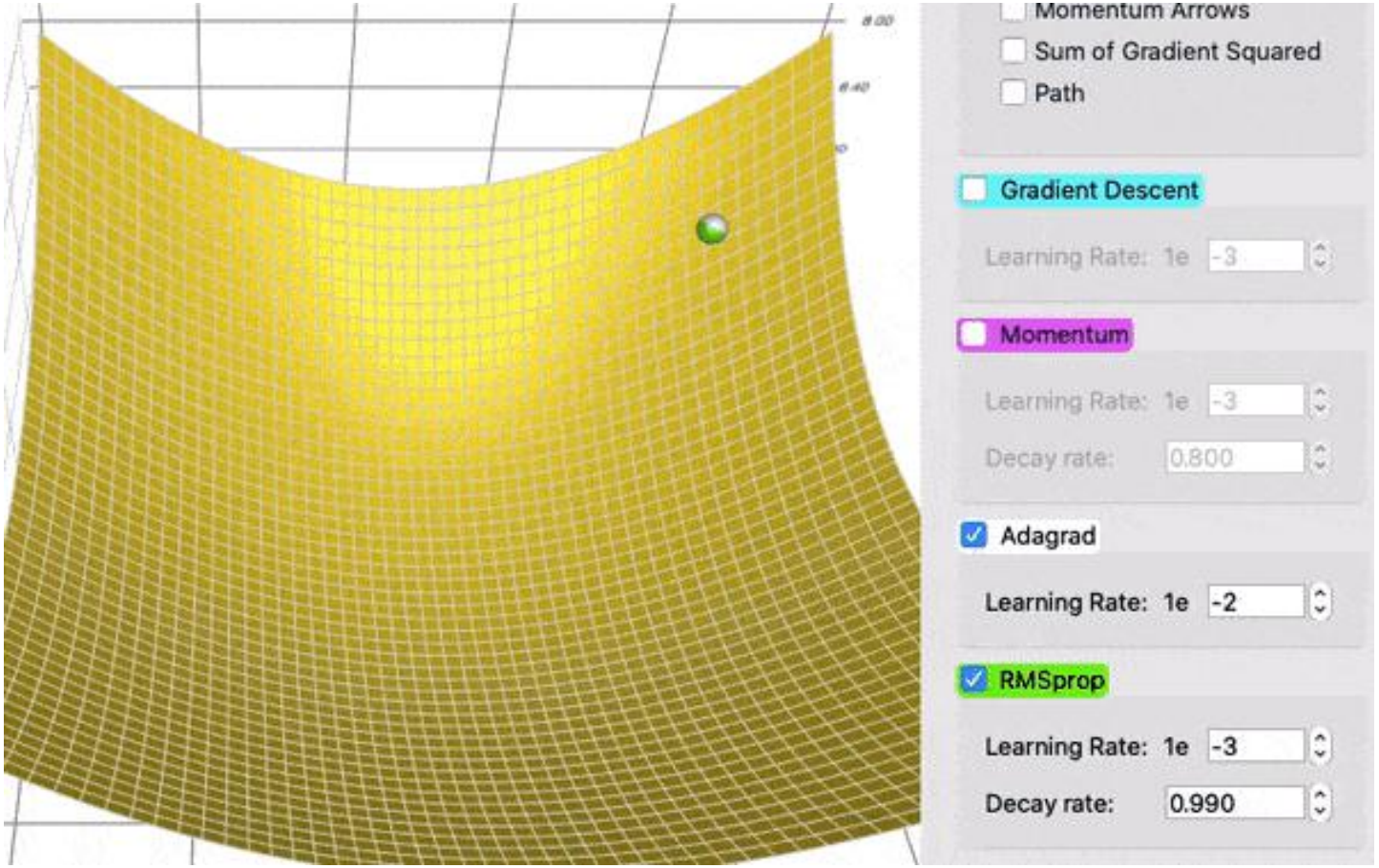
$$s \leftarrow \gamma \cdot s + (1-\gamma) \cdot \nabla J(\theta) \odot \nabla J(\theta)$$

$$\theta \leftarrow \theta - \frac{\eta}{\sqrt{s+\varepsilon}} \odot \nabla J(\theta)$$

衰减率表明的是只是最近的梯度平方有意义，而很久以前的梯度基本上会被遗忘。与我们在动量中看到的衰减率不同，除了衰减之外，这里的衰减率还有一个缩放效应：它以一个因子(1 - 衰减率)向下缩放整个项。如果衰减率设置为0.99，除了衰减之外，梯度的平方和将是 sqrt (1-0.99) = 0.1，因此对于相同的学习率，这一步大10倍。

# 14.5 Rmsprop

下图对比中，AdaGrad(白色)最初与RMSProp(绿色)差不多，正如调整学习率和衰减率的预期。 但是AdaGrad的梯度平方和累计得非常快，以至于它们很快变得非常巨大。



# 14.6 Adam

Adam(Adaptive Moment Estimation)同时兼顾了动量和RMSProp的优点。Adam在实践中效果很好，因此在最近几年，它是深度学习问题的常用选择:

```
sum_of_gradient = previous_sum_of_gradient * beta1 + gradient * (1 - beta1) [Momentum]
sum_of_gradient_squared = previous_sum_of_gradient_squared * beta2 + gradient² * (1- beta2) [RMSProp]
delta = -learning_rate * sum_of_gradient / sqrt(sum_of_gradient_squared)
theta += delta
```

Beta1是一阶矩梯度之和(动量之和)的衰减率，通常设置为0.9。 Beta2是二阶矩梯度平方和的衰减率，通常设置为0.999。

# 14.6 Adam

可以看到前两项和Momentum和RMSprop是非常一致的， 由于和的初始值一般设置为0，在训练初期其可能较小，第三和第四项主要是为了放大它们。最后一项是参数更新。其中超参数eta的建议值是1e-8。

超参越来越多，研究近似炼丹

$$m \leftarrow \beta_1 \cdot m + (1 - \beta_1) \cdot \nabla J(\theta)$$

$$s \leftarrow \beta_2 \cdot s + (1 - \beta_2) \cdot \nabla J(\theta) \odot \nabla J(\theta)$$

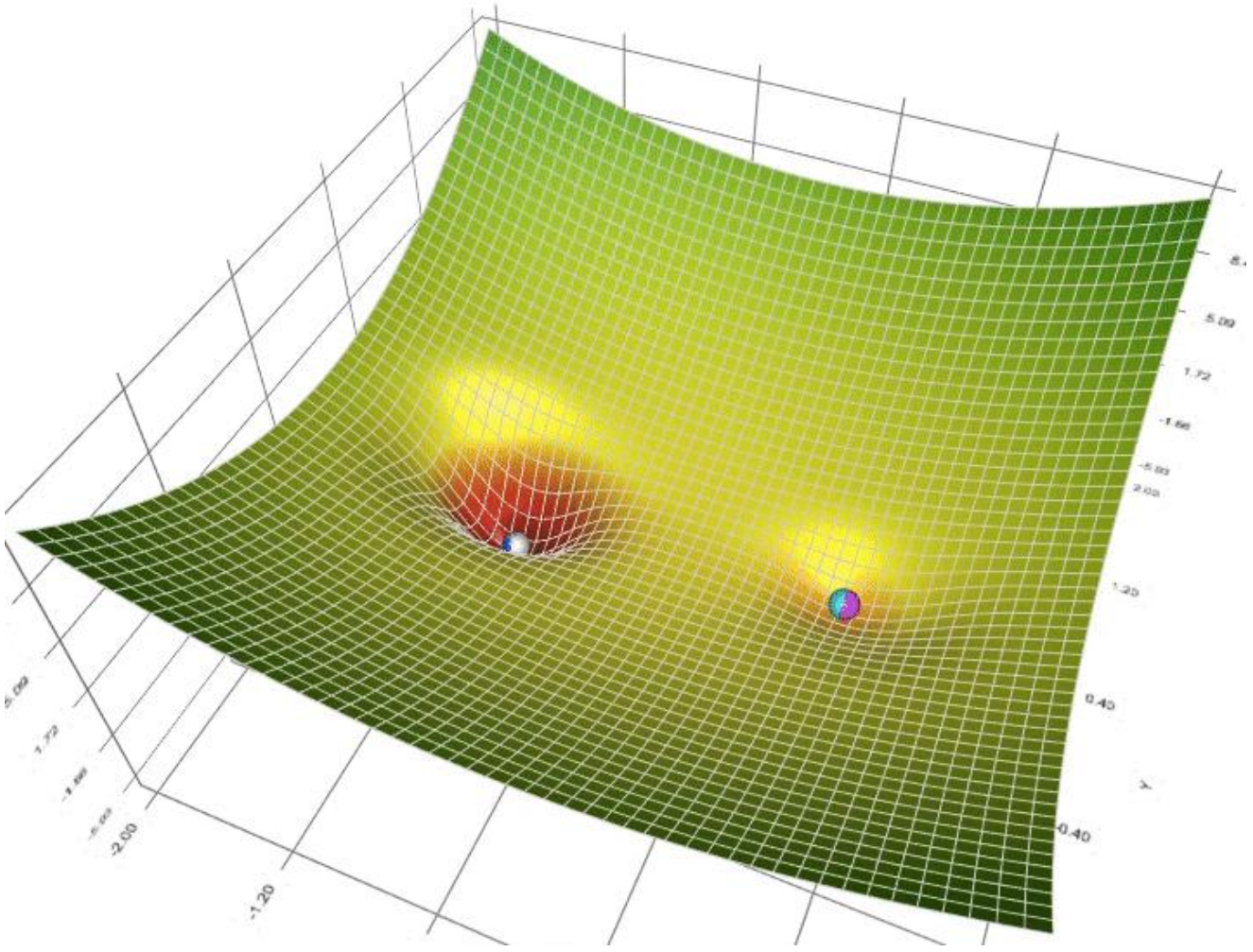
$$m \leftarrow \frac{m}{1 - \beta_1^t}$$

$$s \leftarrow \frac{s}{1 - \beta_2^t}$$

$$\theta \leftarrow \theta - \frac{\eta}{\sqrt{s + \varepsilon}} \odot m$$

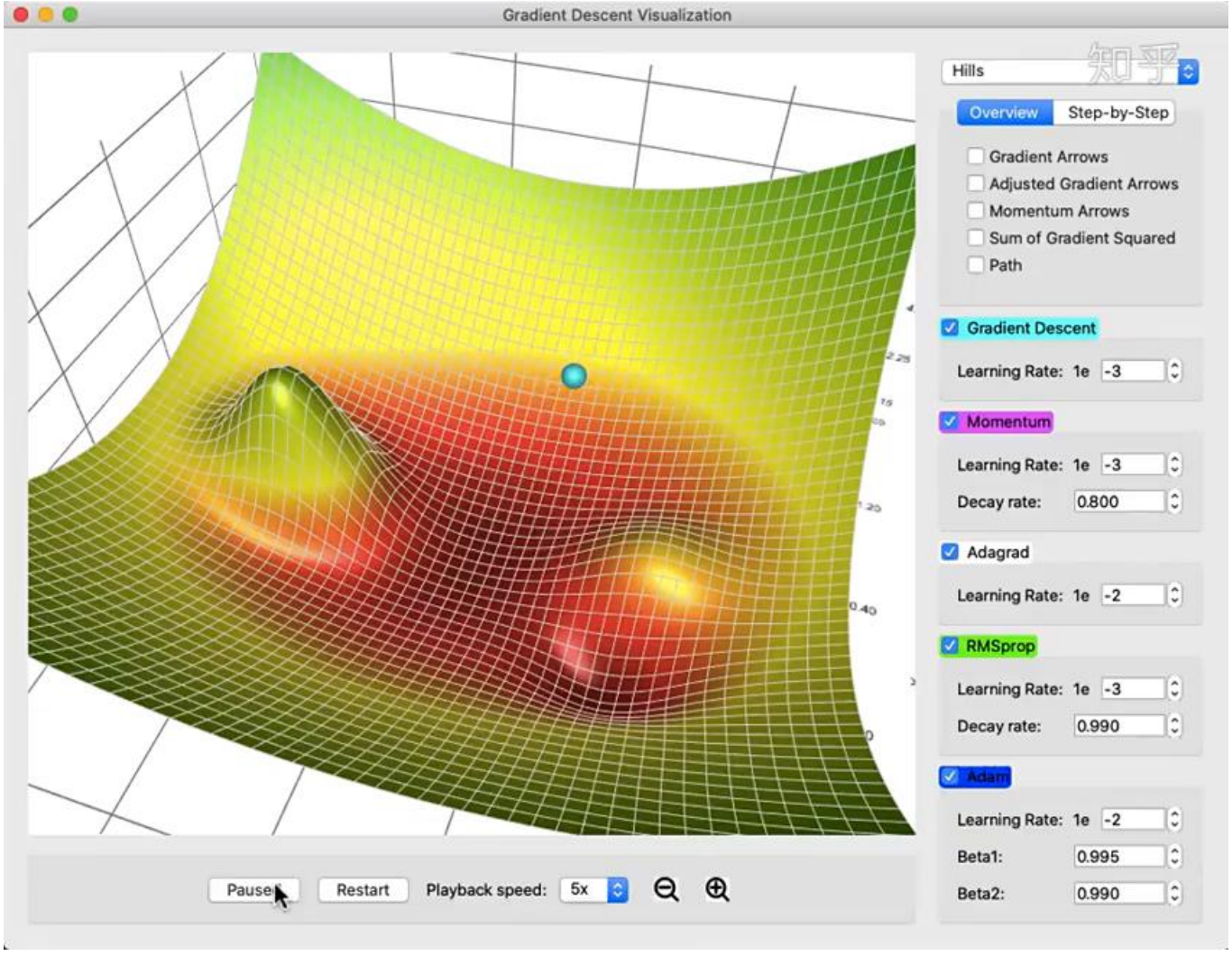
# 14.6 Adam

在一个表面上动画演示5个梯度下降法：梯度下降(青色)，momentum(洋红色)，AdaGrad(白色），RMSProp(绿色)，Adam(蓝色)。左坑是全局极小值，右坑是局部极小值



# 14.6 Adam

在下面图中有两座小山阻挡了通往全局极小值的道路。Adam是上述算法中，唯一能够找到通往全局极小值的算法。



# 14.7 参考文献


- Lili Jiang, A visual explanation of gradient descent methods (Momentum, AdaGrad, RMSProp, Adam). https://zhuanlan.zhihu.com/p/147275344
- Sebastian Ruder, An overview of gradient descent optimization algorithms. https://ruder.io/optimizing-gradient-descent/index.html https://arxiv.org/abs/1609.04747
- 覃含章, Nesterov's accelerated method: gradient descent & mirror descent的线性耦合. https://zhuanlan.zhihu.com/p/35692553
- https://zhuanlan.zhihu.com/p/57860231