摘要

在计算机飞速发展的21世纪，人类不再满足计算机的程序化计算，越来越希望及其能够自动学习从而进行自我更新。由此，机器学习应运而生。为了将机器学习过程中的损失函数降到最小，各种优化算法层出不穷。其中，梯度下降法作为最普遍、最受欢迎的优化算法，在神经网络中应用广泛。

本文从传统的梯度下降法的实现和各自的利弊入手，根据其样本容量分为**批量梯度下降法(BGD)**、**随机梯度下降法(SGD)**和**小批量梯度下降法(MBGD)**。然后介绍了它们的改进算法：**动量(Momentum)算法**、**自适应梯度(AdaGrad)下降法**以及AdaGrad的优化算法——**RMSProp算法**。接着，介绍了集Momentum算法和RMSProp算法优点于一体的**Adam算法**，它不仅可以**收敛到最小值**，还能进行**快速迭代**。根据实践，也证明了Adam算法的优越性和快速性。最后，对各种梯度下降法的优缺点和应用场景进行对比，并分析了Adam算法的应用前景。

关键词：梯度下降法、损失函数、最小值、优化、收敛、快速、Adam算法

目录

引言

在计算机飞速发展的 21 世纪，我们对计算机给予了越来越高的期望，希望通过大量样本数据的训练得到一个稳定的模型可以像人类一样进行学习与工作，比如图像识别、文字判断。受益于人类神经传递信息的启示，神经网络应运而生。然而在机器学习的过程中，常常要用到优化模型对样本像素数据提取、寻找规律并找到其最小值损失函数的对应参数。梯度下降法作为一种普遍的优化方法，在神经网络中应用相当广泛。

本文从传统的梯度下降法的实现和各自的利弊开始介绍，根据其样本容量分为批量梯度下降法(BGD)、随机梯度下降法(SGD)和小批量梯度下降法(MBGD)。接着介绍了它们的改进算法，动量(Momentum)算法、自适应梯度(AdaGrad)下降法以及 AdaGrad 的优化算法——RMSProp 算法。最后，介绍了集 Momentum 算法和 RMSProp 算法优点于一体的 Adam 算法。

1. 传统梯度下降法

假设这样一个场景：一名游客被困在山上，需要从山上下到山谷(即找到山的最低点)，但此时山上的浓雾很大，导致可视度很低；因此，下山的路径就无法确定，必须利用自己周围的信息一步一步找到下山的路。这个时候，便可以利用梯度下降法来帮助自己下山。具体做法是：首先以他当前的所处的位置为基准，寻找这个位置最陡峭的地方，然后朝着下降方向走一步，然后又继续以当前位置为基准，再找最陡峭的地方，再走直到最后到达最低处。

梯度下降法模拟过程如下，分别为二维和三维：

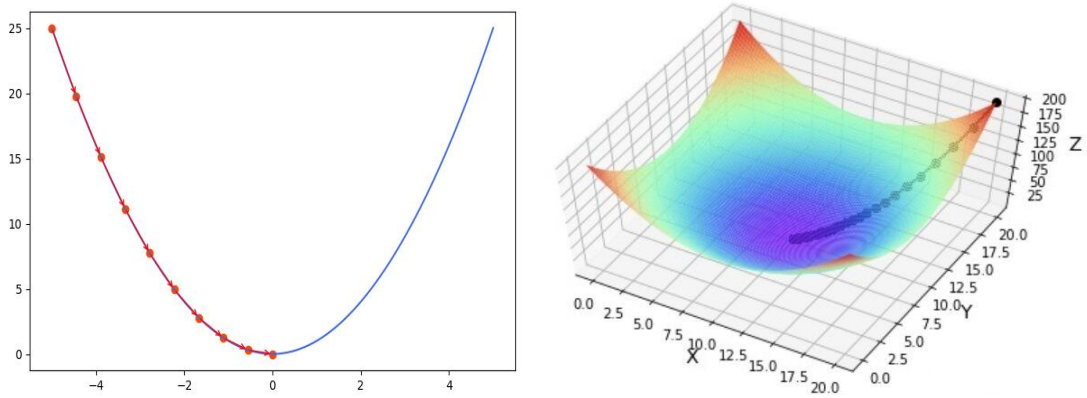

图 1 梯度下降法过程模拟

以二元函数为例，为了求 J(Θ)的最小值，利用迭代公式：

$$\Theta_{n+1} = \Theta_n - \alpha \nabla J(\Theta) \tag{1}$$

不断迭代，直至Θ满足收敛条件，J(Θ)达到最小值。

对于一般的线性回归问题，设(x_i, y_i)为已知样本点，h(x)为目标函数，m 为样本容量，则损失函数：

$$J(\Theta) = \frac{1}{2m}\sum_{i=1}^{m}(h(xi) - yi)^2 \tag{2}$$

通过迭代公式(1)可求得最优参数值将损失函数降到最小。

但对于大量样本，每次迭代更新参数的过程会用到所有样本，计算相当复杂。并且学习率α难以确定，过长可能造成越过最小值，结果不收敛；过小使得收敛缓慢，计算耗时太长。于是在训练方式上根据样本数量不同可分为**批量梯度下降法(BGD)**、**随机梯度下降法(SGD)**和**小批量梯度下降法(MBGD)**。

1.1 批量梯度下降(BGD)

如上文所述，虽然批量梯度下降法最终可以找到全局最优解，但在大量样本计算上存在较大的不足，并且批量数据的最适学习率也不尽相同，因此出现了将样本分批计算的改进方法。

1.2 随机梯度下降(SGD)

在迭代更新参数时，SGN 每次只使用单个训练样本来更新Θ参数，依次遍历训练集，而不是一次更新中考虑所有的样本。对于上面的例子，每次计算一次，更新一次Θ，直到Θ收敛或者达到后期更新幅度已经小于我们设置的阀值。

该方法优化的不是在全部训练数据上的损失函数，而是在每轮迭代中，随机优化某一条训练数据上的损失函数，这样可以使得每一轮参数的更新速度大大加快。但是在某一条数据上损失函数更小并不代表在全部数据上的损失函数更小，于是使用随机梯度下降优化得到的神经网络甚至可能无法达到全局最优。

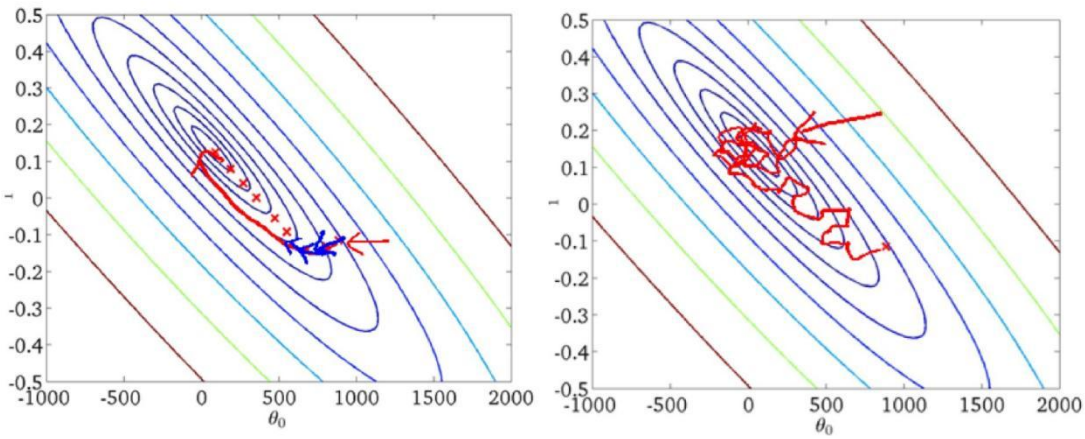

图 2 BGN 和 SGN 效果对比图

1.3 小批量梯度下降(MBGD)

为了综合批量梯度下降算法和随机梯度下降算法的优缺点，在实际应用中一般采用这两个算法的折中——每次计算一小部分训练数据的损失函数，即小批量梯度下降法。MBGD 在深度学习很多算法的反向传播算法中非常常用，这一小部分训练数据也称为一个 batch，因此也引入了 batch_size 的概念，即用来度量每一个 batch 中实例的个数。

引入 batch 可以使得每次在一个 batch 上优化参数并不会比单个数据慢太多，并且每次使用一个 batch 可以大大减小收敛所需要的迭代次数，同时可以使收敛到的结果更加接近梯度下降的效果。但是 MBGD 的难点在于难以选取适中的 batch_size。如下图：蓝色为 batch_size=m 时的优化曲线，即退化成 BGD；紫色为 batch_size=1 时的优化曲线，即退化成 SGD；绿色为 MBGN，效果明显。

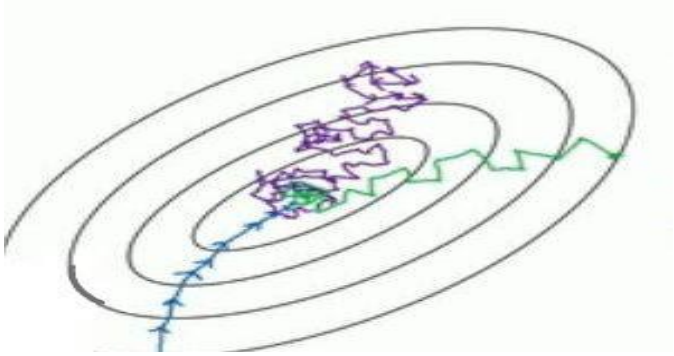

图 3 MBGD 效果图

但由于样本分布的随机性，MBGD 无法较好的选取 batch_size，这有时会使得计算结果不符合预期。

2. 改进梯度下降法

由于传统的梯度下降法存在着相当的无法克服的不足与先天的弊端，经过研究者们的不断探索，陆续提出了**动量(Momentum)算法**、**自适应梯度(AdaGrad)下降法**、**RMSProp 算法**、**Adam 算法**等诸多基于最速梯度下降法的改进办法。下面，我将介绍 Adam 算法，在此之前，需要提前了解 Momentum 算法、AdaGrad 下降法和 RMSProp 算法。

2.1 Momentum 算法

带有动量的梯度下降算法借鉴了物理学中动量的思想：在无摩擦的碗里有一个滚动的球，根据能量守恒定律，它不会在底部停止，而是由积累的动量推动它前进，球继续前后滚动。我们将动量的概念应用到传统梯度下降法中：在每次迭代时，除了常规的梯度之外，它还增加了前一步的移动。递推关系式通常可以表示为:

$$\Delta\Theta_{n+1} = \rho\Delta\Theta_n - \alpha\nabla J(\Theta) \tag{3}$$

其中$\Delta\Theta_{n+1} = \Theta_{n+1} - \Theta_n$，$\rho$称为衰减率，显然当$\rho=0$ 时该算法退化成传统梯度下降法，$\rho=1$ 时算法在一定轨迹内陷入死循环而不收敛，都难以得到预期的结果。因此ρ的取值在 0 到 1 之间，一般情况下衰减率取 0.8~0.9。它就像一个有一点摩擦的表面，所以它最终会减慢并停止以达到预期的结果。

对方程(3)稍加修改，跟踪(衰减的)累积梯度之和,记 P_n 表示当前的梯度累积之和，带入(3)式可得：

$$P_{n+1}=\rho*P_n+\nabla J(\Theta) \tag{4}$$

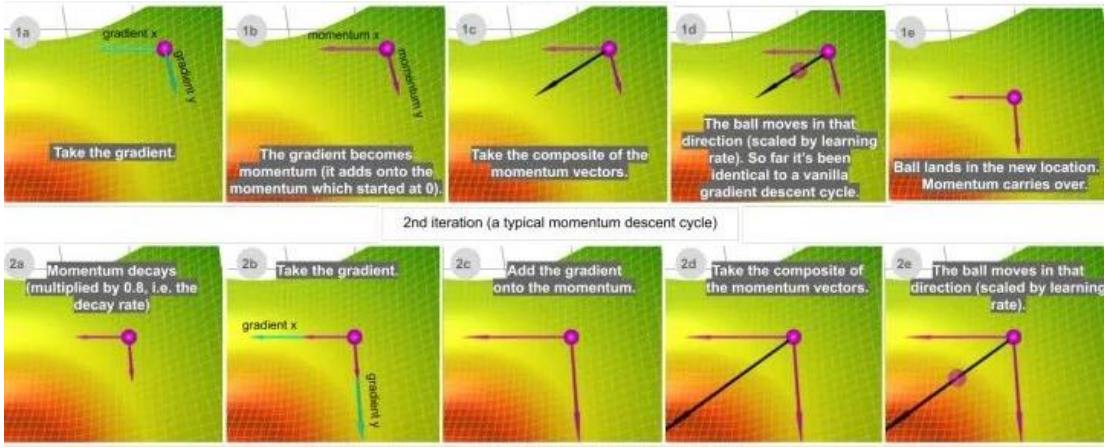

图 5 Momentum 算法下降过程

在上面的比较中可以发现，相比 BGD 的梯度下降，动量移动得更快，因为它积累了所有动量。并且因为动量可能推动下降点脱离局部极小值，Momentum 算法不会陷入局部极小值。同样，它也能更好地通过高原区。

2.2 AdaGrad 下降法

AdaGrad 算法(Adaptive Gradient)，并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，记 S_n 表示梯度的平方和，则有：

$$S_{n+1}=S_n+\nabla J(\Theta)^2 \tag{5}$$

$$\Delta\Theta n+1=-\frac{\alpha\nabla J(\Theta)}{\sqrt{Sn}} \tag{6}$$

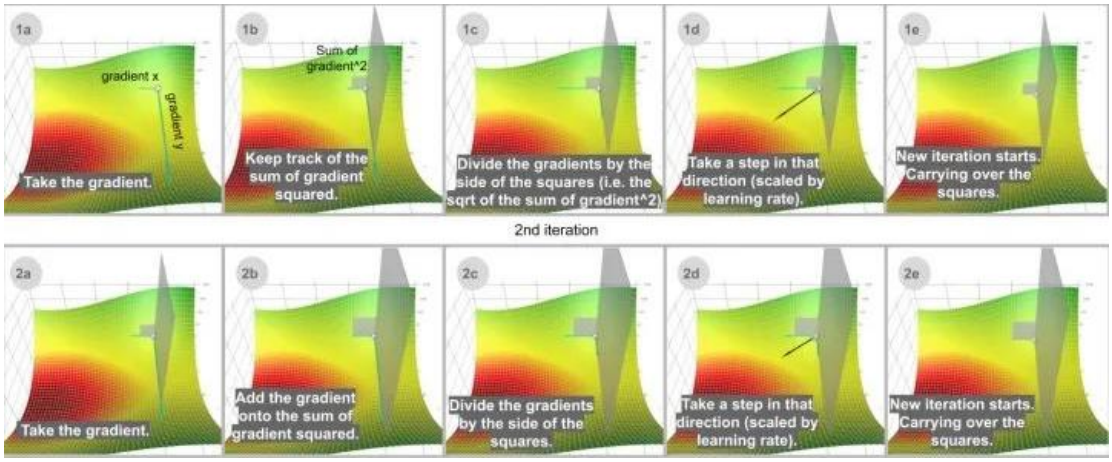

图 6 AdaGrad 算法下降过程

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：已经更新的特征越多，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。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。

这个属性让 AdaGrad 及其衍生算法更好地避开了鞍点。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“先滑下陡峭的斜坡，然后才可能担心较慢的方向”。有时候，传统梯度下降法仅仅停留在鞍点，因为那里两个方向的梯度都是 0，而 AdaGrad 可以最终到达最小值点。

2.3 RMSProp 算法

然而，AdaGrad 的缺陷在于它非常慢。这是因为梯度的平方和只会增加而不会减小。由此提出了 Rmsprop(Root Mean Square Propagation)算法，通过添加衰减因子ρ来修复这个问题。(6)式不变，(5)式修改为：

$$S_{n+1}=S_n*\rho+\nabla J(\Theta)^2*(1-\rho) \tag{7}$$

更准确地说，梯度的平方和实际上是梯度平方的衰减和。衰减率ρ只是表示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。此处的衰减率与 Momentum 算法中的衰减率不同，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-ρ)向下缩放整个项。比如衰减率设置为 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。

2.4 Adam 算法

Adam 算法(Adaptive Moment Estimation)同时兼顾了动量和 RMSProp 的优点：较快的移动速度和准确地移动到最小值点。Adam 算法在实践中效果很好，因此在最近几年，它是深度学习问题的常用选择。接下来介绍其工作原理：

$$P_{n+1}=P_n*\beta1+\nabla J(\Theta)*(1-\beta1) \tag{8}$$

$$S_{n+1}=S_n*\beta2+\nabla J(\Theta)^2*(1-\beta2) \tag{9}$$

$$\Theta_{n+1}=\Theta_n-\alpha\frac{Pn}{\sqrt{Sn}} \tag{10}$$

β1 是一阶矩梯度之和(动量之和)的衰减率，通常设置为 0.9。β2 是二阶矩梯度平方和的衰减率，通常设置为 0.999。显然(8)式为 Momentum 算法，(9)式为 RMSProp 算法。

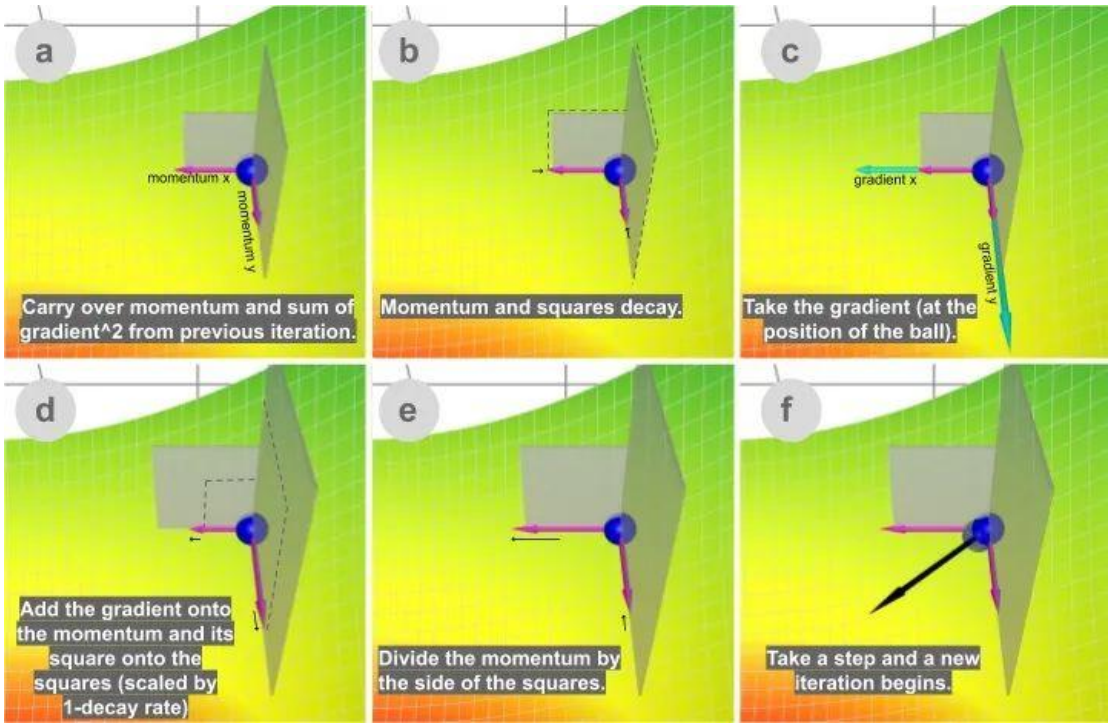

图 7 Adam 算法下降过程

Adam 的速度来自于动量和 RMSProp 适应不同方向的梯度的能力。这两者的结合使它变得更强大。

3. 代码实现

语言：python
编译软件：VS Code
参数设定：β1=0.9，β2=0.999，α=0.01

以 $y=x_1+2*x_2$ 生成的一组样本点为例，拟合函数参数，代码如下：

```
import math
import numpy as np
import time
import random

n=100000
x=[]
y=[]
for i in range(n):
    r1=random.uniform(1.1,10.4)
    r2=random.uniform(1.1,10.4)
    r3=float(r1+2*r2)
    tup=(r1,r2)
    x.append(tup)
```

```
15.         y.append(r3)
16.     x=np.array(x)
17.     y=np.array(y)
18.
19.     def adam():
20.         # 训练集，每个样本有三个分量
21.         # x=np.array([(1,1),(1,2),(2,2),(3,1),(1,3),(2,4),(2,3),(3,3)])
22.         # y=np.array([3,5,6,5,7,10,8,9])
23.
24.         m,dim = x.shape
25.         theta = np.zeros(dim)  #参数
26.         alpha = 0.01          #学习率
27.         momentum = 0.1        #冲量
28.         threshold = 0.0001    #迭代的终止条件阈值
29.         iterations = 3000     #迭代次数
30.         error = 0             #损失
31.
32.         b1 = 0.9              #beita1 默认值 0.9
33.         b2 = 0.999            #beita2 默认值 0.999
34.         e = 0.00000001        #算法作者建议的默认值
35.         mt = np.zeros(dim)
36.         vt = np.zeros(dim)
37.
38.         for i in range(iterations):
39.             j = i % m
40.             error=1/(2*m)*np.dot((np.dot(x, theta)-y).T,(np.dot(x,theta)-y))
41.             if abs(error) <= threshold:
42.                 break
43.
44.             gradient=x[j]*(np.dot(x[j], theta)-y[j])
45.             mt=b1*mt+(1-b1)*gradient
46.             vt=b2*vt+(1-b2)*(gradient**2)
47.             mtt=mt/(1-(b1**(i+1)))
48.             vtt=vt/(1-(b2**(i+1)))
49.             vtt_sqrt=np.array([math.sqrt(vtt[0]),math.sqrt(vtt[1])])   #只能对标量进
    行开方
50.             theta=theta-alpha*mtt/(vtt_sqrt+e)
51.
52.         print('迭代次数：%d'%(i+1),'theta:',theta,'error:%f'%error)
53.
54.     if __name__ == '__main__':
55.         print('n=',n)
56.         time_start = time.time()
57.         adam()
```

```
58.     time_end = time.time()
59.     time_c= time_end - time_start
60.     print('time cost', time_c, 's')
```

4. 结果与算法分析

对比传统梯度下降法和 Adam 算法在不同样本规模情况下求解上述问题的迭代次数和运行时间。

Adam 算法：

```
n= 100
迭代次数: 1624 theta: [1.00315894 1.9968982 ] error:0.000100
time cost 0.02892303466796875 s
```

```
n= 1000
迭代次数: 2009 theta: [1.00382256 1.99645333] error:0.000100
time cost 0.04487895965576172 s
```

```
n= 10000
迭代次数: 2002 theta: [1.00381709 1.99642938] error:0.000098
time cost 0.26828694343566895 s
```

```
n= 100000
迭代次数: 1921 theta: [1.00350136 1.99615697] error:0.000100
time cost 1.8981308937072754 s
```

传统梯度下降法：

```
n= 100
迭代次数: 144 theta: [1.000008851274473, 1.9999917474118443]
time cost 0.009972810745239258 s
```

```
n= 1000
迭代次数: 152 theta: [1.0000087595013814, 1.9999909603419472]
time cost 0.11070418357849121 s
```

```
n= 10000
迭代次数: 145 theta: [1.0000085444463114, 1.9999913958709508]
time cost 1.0462079048156738 s
```

```
n= 100000
迭代次数: 147 theta: [1.0000086372762123, 1.9999913377985286]
time cost 10.860235929489136 s
```

由运行结果显然可以看到，两种算法在计算结果上都比较完美。在数据量较小的情况下，两种算法所用时间也比较接近；当样本规模达到 10^5 时，Adam 算法所用的时间仅为传统梯度下降法的 $\frac{1}{10}$ 。虽然相比较传统梯度下降法，Adam 算法迭代次数虽然大大增加，但是其迭代速度大大提升。

在实际应用中，Adam 算法有以下几处优点：

·**惯性保持**：Adam 算法记录了梯度的一阶矩，即过往所有梯度与当前梯度的平均，使得每一次更新时，上一次更新的梯度与当前更新的梯度不会相差太大，即梯度平滑、稳定的过渡，可以适应不稳定的目标函数。

·**环境感知**:Adam 记录了梯度的二阶矩，即过往梯度平方与当前梯度平方的平均，这体现了环境感知能力，为不同参数产生自适应的学习速率。

·**超参数**，即α、β1、β2、ε具有很好的解释性，且通常无需调整或仅需很少的微调。

5. 应用前景

先来对比一下各种梯度下降算法的优缺点和适用情景：

算法	优点	缺点	适用情景
BGD	目标函数为凸函数时，可以找到全局最优值	收敛速度慢，需要用到全部数据，内存消耗大	不适用于大数据集，不能在线更新模型
SGD	避免冗余数据的干扰，收敛速度加快，能够在线学习	更新值的方差较大，收敛过程会产生波动，可能落入极小值（卡在鞍点），选择合适的学习率比较困难（需要不断减小学习率）	适用于需要在线更新的模型，适用于大规模训练样本情况
Momentum	能够在相关方向加速 SGD，抑制振荡，从而加快收敛	需要人工设定学习率	适用于有可靠的初始化参数
Adam	速度快，对内存需求较小，为不同的参数计算不同的自适应学习率	在局部最小值附近震荡，可能不收敛	需要快速收敛，训练复杂网络时；善于处理稀疏梯度和非平稳目标的优点，也适用于大多非凸优化和大数据集和高维空间

现在机器学习中用的最多的算法是 SGD 和 Adam，两者各有好坏。SGD 能够到达全局最优解，而且训练的最佳精度也要高于其他优化算法，但它对学习率的调节要求非常严格，而且容易停在鞍点；Adam 很容易的跳过鞍点，而且不需要

人为的干预学习率的调节，但是它很容易在局部最小值处震荡，存在在特殊的数据集下出现学习率突然上升，造成不收敛的情况。

Adam 是一种可以替代传统随机梯度下降过程的一阶优化算法，它能基于训练数据迭代地更新神经网络权重。Adam 在深度学习领域内是十分流行的算法，因为它能很快地实现优良的结果。经验性结果证明 Adam 算法在实践中性能优异，相对于其他种类的随机优化算法具有很大的优势。在原论文中，作者经验性地证明了 Adam 算法的收敛性符合理论性的分析。Adam 算法可以在 MNIST 手写字符识别和 IMDB 情感分析数据集上应用优化 logistic 回归算法，也可以在 MNIST 数据集上应用于多层感知机算法和在 CIFAR-10 图像识别数据集上应用于卷积神经网络。他们总结道：“在使用大型模型和数据集的情况下，我们证明了 Adam 优化算法在解决局部深度学习问题上的高效性。”

结语

梯度下降法算法是一类通过梯度来寻找函数最小点的算法。传统梯度下降只遵循寻找梯度方向按学习速率不断下降寻找最小值。改善的梯度下降法的两个常用工具是梯度之和(一阶矩)和梯度平方之和(二阶矩)。动量利用一阶矩的衰减率来获得速度。Adagrad 使用没有衰减的二阶矩来处理稀疏特征。Rmsprop 使用二阶矩的衰减率来加速 AdaGrad。Adam 同时使用一阶矩和二阶矩，通常是最好的选择。

事实上，Insofar、RMSprop、Adadelta 和 Adam 算法都是比较类似的优化算法，他们都在类似的情景下都可以执行地非常好。但是 Adam 算法的偏差修正令其在梯度变得稀疏时要比其他算法更快速和优秀。

参考文献

[1] An overview of gradient descent optimization algorithms-Sebastian Ruder

[2] 机器学习入门：线性回归及梯度下降

[3] 一文看懂各种神经网络优化算法：从梯度下降到 Adam 方法

[4] 随机梯度下降、动量、RmsProp、AdamDelta、Adam 方法优化

[5] https://web.stanford.edu/class/msande311/lecture13.pdf

[6] http://cs231n.github.io/neural-networks-3/

[7] ADAM: A method for stochastic optimization. Diederik P.Kingma . Jimmy Lei Ba

[8] https://blog.csdn.net/qq_30614451/article/details/93753322

[9] https://blog.csdn.net/tsyccnh/article/details/76270707