模式识别与机器学习

Lecture Notes

笔记二

优化算法

目录

1 梯度的定义与梯度下降法

1.1 梯度的定义

梯度是多元函数在某一点处的一阶变化信息所组成的向量，通常记作 $\nabla f(x)$ 或 $\operatorname{grad} f(x)$。

对于函数：

$$f(x_0, x_1, \ldots, x_n),$$

其梯度可写为：

$$\operatorname{grad} f(x_0, x_1, \ldots, x_n) = \left(\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right).$$

如果把函数图像看作一座山，那么梯度 ∇f 指向“爬升最快”的方向，而负梯度 $-\nabla f$ 则指向“下降最快”的方向。

梯度可以从以下几个方面理解：

1. 梯度是一个向量，有方向也有大小；
2. 梯度的方向表示函数在该点处上升最快的方向；
3. 梯度的模表示该点处最大的变化率。

1.2 为什么需要梯度下降优化

在机器学习和优化问题中，常常需要最小化一个目标函数或损失函数，而在实际问题中，这个目标函数往往比较复杂，比如在多变量情况下解析求解会变得困难。因此，需要一种能够通过迭代逐步逼近最小值的方法，梯度下降法正是解决这类问题的基础方法之一。

由于目标是寻找函数最小值且解析解常不可得，梯度提供局部最快上升方向，因而沿负梯度更新参数能使函数值快速下降，通过迭代逐步逼近最优解。

1.3 梯度下降法的过程

梯度下降法的核心思想是：从一个初始点出发，反复沿着负梯度方向走一小步，不断减小目标函数值。

为了更直观地理解“沿着负梯度方向逼近最小值”，可以把二元函数看成一张三维曲面：

$$f(w_0, w_1) = w_0^2 + w_1^2$$

如图 1 所示，上述函数的图像是一个开口向上的“碗形曲面”，最低点位于原点附近，也就是函数的最小值点，若从曲面上的某个较高位置出发，每一步都沿负梯度方向向下移动，就会沿着红色路径不断靠近最低点，从而逐步逼近函数的最小值，最终达到或接近最优解。

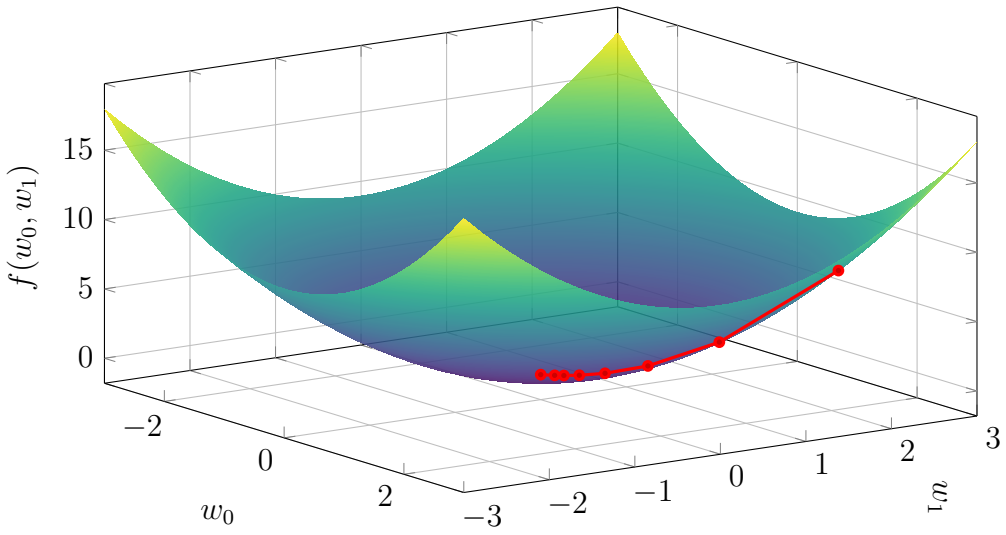

图 1: 损失曲面与梯度下降逼近最小值

在实际应用中，最常见更新公式为：

$$x_{k+1} = x_k - \eta \nabla f(x_k),$$

其中：

- x_k 表示第 k 次迭代时的参数；

- $\nabla f(x_k)$ 表示当前点的梯度；
- η 表示步长，也称学习率；
- 在机器学习中，也常写成：

$$w := w - \alpha \frac{\partial}{\partial w} \text{loss}(w),$$

其中 w 是模型参数，$\text{loss}(w)$ 是损失函数，α 是学习率。

对于一般的梯度下降法，可以通过以下步骤来实现：

1. 选择初始参数 x_0；
2. 计算当前点的梯度 $\nabla f(x_k)$；
3. 按照更新公式移动到下一点 x_{k+1}；
4. 重复上述过程，直到函数值变化很小或者梯度接近于 0；
5. 将最终结果视为函数的近似最优解。

1.4 如何确定学习率

学习率决定了每次参数更新的步长，直接影响优化过程的快慢与稳定性，确定学习率时，应选择一个既能让损失函数明显下降，又不会引起震荡或发散的数值。

确定学习率时，可以按照下面的步骤来判断：

1. 先用较小的学习率作为起点，逐步调整；
2. 若损失上下波动或增大，说明步长太大，应减小；
3. 若损失持续下降但很慢，说明步长太小，应增大；
4. 损失稳定且在较少迭代内明显下降时，则说明学习率比较合适。

2 梯度下降的三种形式及其优缺点

2.1 批量梯度下降（Batch Gradient Descent, BGD）

2.1.1 基本思想

批量梯度下降的核心是每一次参数更新都使用**全部训练样本**来计算梯度。模型先对整个训练集完成一次前向计算与梯度累计，再统一更新一次参数。它反映的是全体样本上的平均下降方向，因此方向最稳定，但更新频率最低。

课件中的参数更新公式为：

$$w_j := w_j - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(\left(h\left(x^{(i)}\right) - y^{(i)} \right) \cdot x_j^{(i)} \right)$$

其中，同步更新 w_j，$(j = 0, 1, \ldots, n)$。

求和范围覆盖全部 m 个样本，因此该式得到的是训练集整体意义下的平均梯度。若训练集很小，这种全量更新能够给出较为平滑的下降路径，反之若训练集很大，则每更新一次参数都要等待较长时间。

2.1.2 优点分析

- **梯度方向稳定。**由于每一步都综合了全部样本的信息，单个异常样本对更新方向的影响会被平均掉。举例，如在房价预测任务中，若训练集只有几百条样本，其中某一条样本因录入错误使价格异常偏高，BGD 计算梯度时会把这条异常样本的影响分散到全体样本中，因此整体更新方向仍然较稳定。若只用单样本更新，该异常点就可能使参数突然偏移。

- **损失曲线平滑。** BGD 适合课堂推导和小规模实验。在教学演示中，BGD 的 loss 下降通常更规整，更容易看出何时接近极小值，也更便于解释梯度下降为何沿负梯度方向移动。对于样本量仅有几百到几千的数据集，遍历全体样本的成本通常不算高，此时 BGD 的可解释性优势比较明显。

- **对凸优化问题表现清晰。**在线性回归、逻辑回归等较典型的凸优化任务中，BGD 的更新方向接近真实全局下降方向，因此算法行为直观，便于分析收敛过程。

2.1.3 缺点分析

- **单次更新代价高。**若训练集包含一百万条样本，则 BGD 每更新一次参数都必须先完成一百万条样本的计算。此时虽然方向稳定，但模型迟迟得不到新参数，训练初期的推进速度会显得很慢。

- **参数响应不及时。**一个 epoch 中 BGD 只更新一次，与之相比 SGD 和 MBGD 会在同一个 epoch 内更新多次，因此能更快从较差的初始参数移动到较优区域。此时若使用 BGD 会遇到在相邻两次更新之间等待时间太长情况，但这并不意味着 BGD 最终得到的参数就不好。

- **在复杂非凸损失面前缺少随机扰动。**深度学习中的损失曲面通常包含鞍点、平台区和局部低谷。BGD 因为方向平稳而更容易在梯度很小的区域停滞较久。与之相比，带有随机性的 SGD 或 MBGD 有时反而更容易离开这些区域。

2.2 随机梯度下降（Stochastic Gradient Descent, SGD）

2.2.1 基本思想

随机梯度下降的核心是每次只使用**一个训练样本**计算梯度，并在计算后**立即更新参数**。它把一次大规模的全量更新拆成了许多单样本更新，因此单步代价最低，更新最频繁，同时也会引入最大的随机波动。

课件中的参数更新公式为：

$$w_j := w_j - \alpha \left(h\left(x^{(i)}\right) - y^{(i)} \right) x_j^{(i)}$$

其中，同步更新 w_j，$(j = 0, 1, \ldots, n)$。

这里的关键在于，梯度仅由第 i 个样本决定，因此它只能近似反映整体下降方向。

这个近似有时偏差较大，但换来的好处是每处理一个样本就能立刻修正参数。

2.2.2 优点分析

- **单次更新快。**对于一万个样本的任务，SGD 在一个 epoch 内可更新一万次，而 BGD 只更新一次。若模型初始参数较差，SGD 常常能更快把参数从明显不合理的位置推向较优区域。
- **适合在线学习和流式数据。**在广告点击率预测、日志流分析、传感器连续采样等场景中，数据并不是一次性给出，而是不断到来。SGD 可以做到样本到达后立即学习，因此更适合实时更新模型。
- **存在随机扰动。**若损失面存在鞍点或较浅局部低谷，BGD 可能因整体梯度很小而停滞；SGD 由于每一步都含有样本噪声，路径会产生抖动，这种抖动有时能推动参数离开这些不良区域。课件中所说可以从局部最小值区域跳出来，本质上就是这种随机扰动带来的搜索能力。
- **内存需求最低。**因为一次只处理一个样本，所以特别适合内存有限或必须实时更新的场景。

2.2.3 缺点分析

- **更新轨迹抖动明显。**一个样本只能反映局部信息，不能代表整体数据分布。例如在二分类问题中，若当前样本恰好位于决策边界附近或本身是少量异常点，SGD 就可能朝一个与整体趋势偏差较大的方向走一步。
- **接近极小值后难以稳定停留。**即使模型已经接近最优点，单样本噪声仍然存在，因此 SGD 常在最优点附近来回摆动，表现为 loss 曲线震荡较强。若用学习率衰减加以配合，后期往往难获得很平滑的收敛过程。
- **对样本顺序敏感。**若训练样本按类别集中排列，例如前半段几乎全是正类、后半段几乎全是负类，那么 SGD 会在训练初期持续向某一类偏移，再在后期大幅纠正，整个过程不稳定。解决方法是实际训练中在每个 epoch 前打乱数据。

- **学习率设置更敏感**。学习率过大时，SGD 的震荡会进一步放大，严重时甚至发散，而学习率过小时，又会削弱它在前期快速推进的优势。因此 SGD 常需要精心设置学习率。

2.3 小批量梯度下降（Mini-Batch Gradient Descent, MBGD）

2.3.1 基本思想

小批量梯度下降位于 BGD 与 SGD 之间。它每次使用**一定批量的训练样本**来计算梯度，再更新一次参数。它不是看完整个训练集，也不是只看一个样本，而是每次看一个规模适中的样本子集。

课件中的参数更新公式为：

$$w_j := w_j - \alpha \frac{1}{b} \sum_{k=i}^{i+b-1} \left(h\left(x^{(k)}\right) - y^{(k)} \right) x_j^{(k)}$$

其中，同步更新 w_j，$(j = 0, 1, \ldots, n)$。

课件中的对应关系为：

$$b = 1 \Rightarrow \text{随机梯度下降，SGD}$$

$$b = m \Rightarrow \text{批量梯度下降，BGD}$$

$b = batch\_size \Rightarrow$ 通常是 2 的指数倍，常见 32，64，128 等。 $\Rightarrow$ 小批量梯度下降，MBGD

从这个关系可以看出，MBGD 本质上是由批量大小 b 调节出来的一类方法。当 b 很小时接近 SGD ，b 很大时接近 BGD。

2.3.2 优点分析

- **兼顾稳定性与效率**。一个 mini-batch 中包含多个样本，因此求出的梯度比 SGD 更能代表整体趋势，但又不需要扫描完整数据集，所以单次更新开销远小于 BGD。以图像分类为例，若每次取 64 张图片计算梯度，得到的方向通常比单张图片稳定

得多，同时每个 epoch 又能更新多次。

- **能够高效利用向量化与 GPU 并行计算。**目前的深度学习框架擅长把一批样本组织成矩阵后统一计算。若一次只输入一个样本，GPU 的并行能力往往利用不足；mini-batch 则正好能让卷积、矩阵乘法等操作以更高吞吐量运行，因此在工程实践中最为常见。

- **更新频率与噪声水平较平衡。**相比 BGD，MBGD 在同一个 epoch 内会更新多次，因此参数能较快响应，而相比 SGD，MBGD 每一步由一批样本平均决定，因此噪声更小、路径更平滑。

- **工程可操作性最好。**绝大多数神经网络训练代码，包括卷积神经网络、循环神经网络、Transformer 等，几乎都默认按 mini-batch 加载数据并更新参数，MBGD 是实践中最常见的训练方式。

2.3.3 缺点分析

- **需要调参。**如前述，若 b 太小，算法行为会越来越接近 SGD，噪声较大，而若 b 太大，又会越来越接近 BGD，更新变慢且显存占用增加。因此 batch size 本身就是一个重要超参数，需要专门调参。

- **随机波动不能被完全消除。**虽然 MBGD 比 SGD 更稳定，但它使用的仍然只是局部样本，不是全部样本，因此梯度依旧只是近似值，不可能像 BGD 那样平滑。

- **资源问题。**在深度学习中，过大的 batch size 会显著增加显存占用，同时也可能使优化过程过于确定化，减弱随机扰动对泛化的帮助。因此批量更大并不必然更好。

2.4 三种方法对比

2.4.1 核心差异

比较维度	BGD	SGD	MBGD	对复习应重点记住的结论
单次更新使用样本数	全部 m 个样本	1 个样本	b 个样本	三者区别就是每次用多少样本算梯度
单次更新计算代价	高	低	中等	样本越多，单步代价越高
参数更新频率	低	高	较高	在同一个 epoch 内，SGD 更新最多，BGD 更新最少
梯度稳定性	高	低	较高	全量平均最稳定，单样本噪声最大
下降轨迹	平滑	抖动明显	较平滑	轨迹平滑程度与梯度方差直接相关
并行计算友好性	一般	较弱	强	MBGD 最适合现代 GPU 训练
跳出局部不良区域能力	较弱	较强	中等	随机性越强，越可能摆脱鞍点和浅局部低谷
实际应用频率	较少	特定场景常用	最常用	深度学习训练通常默认采用 MBGD

表 1: 三种方法核心差异对比

2.4.2 适用场景

- **BGD：**当样本量较小、主要希望观察平滑 loss 曲线、或者课程讲解需要展示标准梯度下降过程时，BGD 更合适。例如线性回归课堂实验、小规模逻辑回归训练、需要严格分析收敛过程的教学型实验。
- **SGD：**当数据以流式形式持续到来，或者模型需要收到新样本后立刻更新时，SGD 更合适。例如在线广告点击率预测、连续日志分析、传感器实时监测数据建模等。

- **MBGD**：当训练数据较大、需要调用 GPU 并行计算，同时又希望优化过程比 SGD 更稳定时，MBGD 更合适。例如图像分类、语音识别中的神经网络训练。

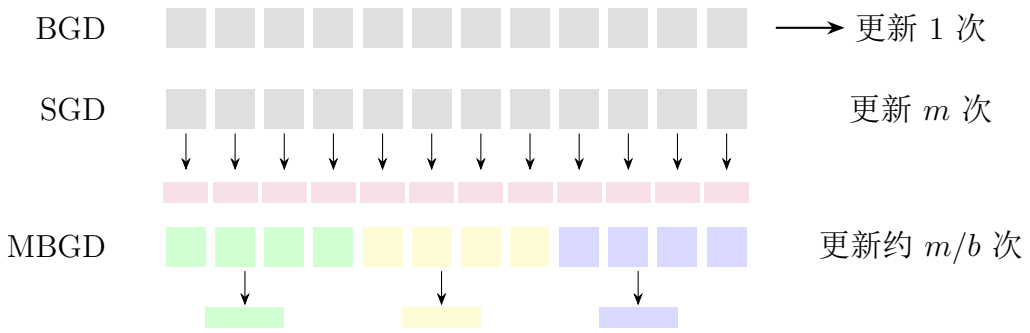

图 2: 同一 epoch 内三种方法的参数更新次数对比

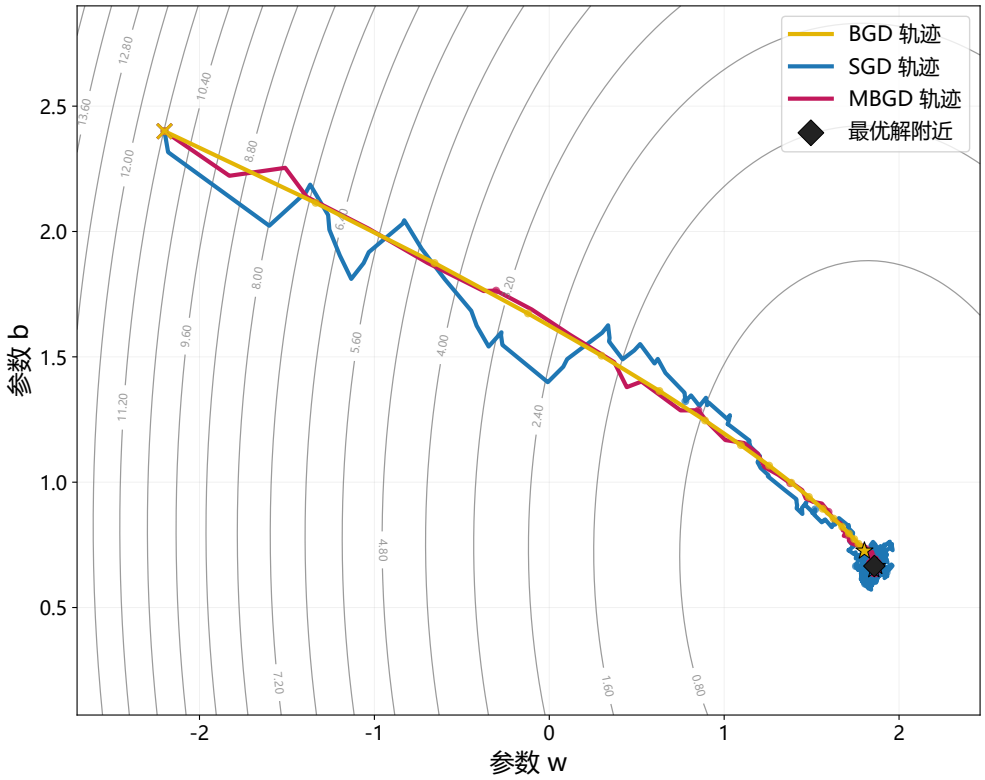

图 3: BGD、SGD 与 MBGD 在同一损失面上的下降轨迹

2.5 总结

BGD 用全部样本求平均梯度，方向最稳定，曲线最平滑，但更新最慢；SGD 每次只用一个样本，更新最快但波动最大；MBGD 用一小批样本求近似平均梯度，在稳定性和效率之间取得平衡，实际训练中最常用。

3 高级优化算法：Momentum、RMSprop 与 Adam

3.1 Momentum 优化算法

3.1.1 物理直觉与核心定义

Momentum（动量梯度下降法）模拟了物理世界中球体沿斜坡滚下的过程。在标准的随机梯度下降（SGD）中，参数更新完全由当前的瞬时梯度 g_t 决定，这容易导致在峡谷状的损失函数表面产生剧烈震荡。

Momentum 通过引入变量 v_t（速度，Velocity）来累积历史梯度的信息。其数学定义基于指数加权移动平均（Exponentially Weighted Moving Averages）：

$$v_t = \beta v_{t-1} + (1-\beta) g_t \tag{1}$$

其中：

- $g_t = \nabla_w J(w_t)$ 为当前时刻的梯度。
- $\beta \in [0, 1)$ 为超参数（动量因子/衰减率），代表对历史速度的保留比例。通常经验值设为 0.9。

3.1.2 数学迭代推导

为了深入理解 v_t 如何平滑优化路径，我们将递归公式进行迭代展开。假设初始速度 $v_0 = 0$，则各时刻的速度序列为：

- $v_1 = (1-\beta) g_1$
- $v_2 = \beta v_1 + (1-\beta) g_2 = \beta(1-\beta) g_1 + (1-\beta) g_2$
- $v_3 = \beta v_2 + (1-\beta) g_3 = \beta^2(1-\beta) g_1 + \beta(1-\beta) g_2 + (1-\beta) g_3$

推广到任意时刻 t，其通项公式可表示为：

$$v_t = (1-\beta)\sum_{i=1}^{t}\beta^{t-i}g_i \tag{2}$$

从该推导中可以看出两个关键点：

1. **权重衰减效应**：历史梯度 g_i 的权重随时间距离以指数级 β^{t-i} 衰减。这意味着算法在参考历史趋势的同时，更侧重于近期的梯度方向。

2. **方向抵消与加强**：当连续时刻的梯度方向一致时，v_t 会迅速增大，形成“惯性”加速；而当梯度在某个维度（如垂直于前进方向）往复跳变时，正负项在求和过程中会相互抵消，从而有效抑制震荡。

3.1.3 权重更新

最终，参数 w 的更新不再直接依赖当前梯度，而是沿着融合了惯性的速度方向进行：

$$w_{t+1} = w_t - \alpha v_t \tag{3}$$

其中 α 为学习率。这种机制使得模型能够更快地跨过平坦区域（鞍点），并在震荡较大的区域保持稳定的前进方向。

3.1.4 算法性能对比分析

为了验证 Momentum 算法的优越性，我们在 MATLAB 中模拟了一个“窄长峡谷”地形的损失函数。图 **??** 展示了标准 SGD 与 Momentum 算法在相同起点和学习率下的优化路径对比。

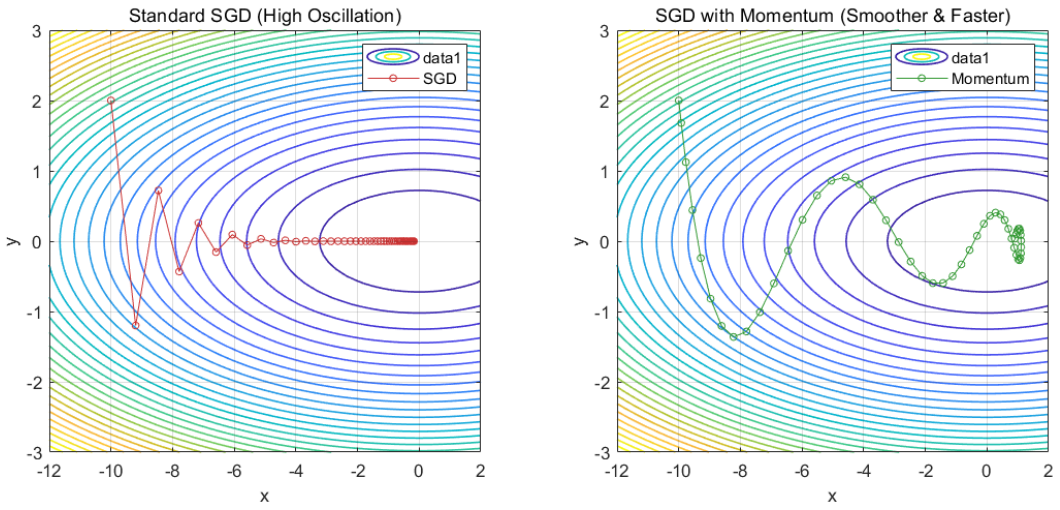

图 4: 标准 SGD 与带有动量的 SGD 优化路径对比图

通过对比可以看出：

- **标准 SGD（左图）**：在垂直方向（y 轴）产生了剧烈的“之”字形震荡，导致在目标方向（x 轴）的进展十分缓慢。
- **Momentum（右图）**：路径更加平滑，通过累积历史梯度，抵消了无效的垂直震荡，并在水平方向实现了快速加速，收敛效率显著提升。

3.2 RMSprop 优化算法

3.2.1 核心动机与背景

在深度学习的优化过程中，损失函数曲面往往表现出极强的各向异性。某些维度的梯度可能非常大（导致严重的震荡），而某些维度的梯度则非常小（导致学习极其缓慢）。如果全局使用统一的学习率 α，很难同时兼顾不同维度的需求。

RMSprop（Root Mean Square Prop）的直觉是：**观察历史梯度的波动情况，如果某个维度最近波动很大，就通过缩放因子调小其更新步长；反之，则保持或放大其步长。**

3.2.2 基于二阶矩的数学推导

与 Momentum 累积梯度的一阶矩不同，RMSprop 计算的是**梯度平方**的指数加权移动平均（即二阶矩）：

$$s_t = \beta s_{t-1} + (1-\beta) g_t^2 \tag{4}$$

其中：

- g_t^2 表示对梯度的每个元素进行逐元素平方。
- s_t 是对历史梯度平方的累积，反映了每个维度在最近一段时间内的波动强度。
- β 为衰减率，经验值通常设为 0.99。

3.2.3 自适应步长更新

在参数更新阶段，RMSprop 利用 s_t 对原始梯度进行“归一化”处理：

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{s_t + \epsilon}} \odot g_t \tag{5}$$

这里的 $\odot$ 表示逐元素相乘，ϵ 为防止分母为零的微小常数（如 10^{-8}）。该公式体现了 RMSprop 的自适应机制：

1. **震荡抑制**：当某一维度的梯度持续较大（震荡严重）时，s_t 迅速增大，导致分母 $\sqrt{s_t + \epsilon}$ 变大。这会自动减小该方向的有效学习率，从而“抹平”剧烈的摆动。
2. **加速平缓区**：在梯度较小的平缓地带，s_t 较小，分母随之减小。这相对放大了更新步长，确保模型在坡度较小的区域仍能保持有效的进度。

3.2.4 算法优势分析

相比于标准 SGD，RMSprop 的优势在于其能够为每个参数独立定制“个性化”的学习率。这种机制使得它在处理非平稳目标函数（如循环神经网络 RNN 的训练）以及具有复杂曲率的损失函数时，表现出极强的稳健性和收敛效率。

3.2.5 算法性能对比分析

为了直观验证 RMSprop 的自适应特性，我们构建了一个极具挑战性的各向异性损失函数曲面（类似窄长峡谷地形），其中 y 轴方向的曲率远大于 x 轴。我们在相同的起点和基础学习率下，对比了标准 SGD 与 RMSprop 的优化路径，结果如图 5 所示。

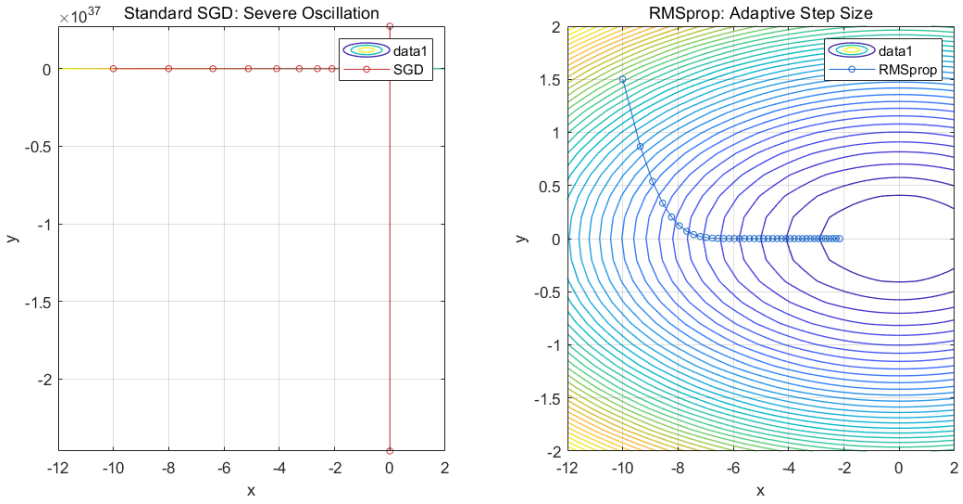

图 5: 标准 SGD 与 RMSprop 在高曲率地形下的路径对比图

实验结果分析：

- **标准 SGD（左图）：**表现出了极大的不稳定性。由于 y 轴方向梯度过大，更新步长迅速失控，导致数值发生爆炸（量级达到 10^{37}），算法崩溃，无法收敛。
- **RMSprop（右图）：**展示了卓越的自适应能力。通过对二阶矩 s_t 的实时监测，算法自动识别并大幅压制了 y 轴方向的过大更新，使得路径能够顺滑地转向最优解，证明了其在复杂地形下的鲁棒性。

3.3 Adam 优化算法

3.3.1 核心动机与设计理念

Adam (Adaptive Moment Estimation) 算法是目前深度学习领域应用最广泛的优化器。它实质上是将 **Momentum** 的动量机制与 **RMSprop** 的自适应学习率机制结合在一起。其核心设计理念在于：利用梯度的一阶矩（方向平均）来控制前进的惯性，同时利用梯度的二阶矩（波动平均）来自适应地缩放更新的步长。

3.3.2 数学推导过程

Adam 的更新逻辑可以细分为以下三个关键步骤：

一阶与二阶矩的估计算法首先利用指数加权移动平均（EWMA）计算梯度的动量估计与波动估计：

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{6}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{7}$$

其中，m_t 捕捉梯度的均值信息（一阶矩），v_t 捕捉梯度的未中心化方差信息（二阶矩）。根据工程经验，通常设置 $\beta_1 = 0.9$ 和 $\beta_2 = 0.999$。

偏差修正机制 (Bias Correction) 由于 m_t 和 v_t 初始化为零，在迭代初期，矩估计值会表现出向零点的严重偏差。为了使估计更加准确，Adam 引入了修正项：

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{8}$$

随着迭代次数 t 的增大，修正因子 $1-\beta^t$ 迅速逼近 1。这一机制确保了算法在训练全周期（尤其是初始阶段）都能保持数值稳定性和有效的步长。

参数自适应更新最终的参数更新公式为：

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{9}$$

其中 α 为学习率，ϵ 为平滑项（通常取 10^{-8}）。该公式揭示了 Adam 的本质：更新方向

由平滑后的动量决定，而步长由梯度的二阶矩动态缩放。

3.3.3 算法优势分析

Adam 算法在处理稀疏梯度、非平稳目标以及具有复杂曲率的损失函数时，展现了极强的鲁棒性。它是 Momentum 的平稳特性与 RMSprop 的自适应特性的完美融合，通过偏差修正克服了初期收敛缓慢的问题，因而在大规模深度学习任务中表现卓越。

3.3.4 算法性能对比分析

为了进一步验证各优化算法在极端地形下的表现，本实验模拟了一个各向异性的椭圆峡谷损失函数。实验结果（见图 6）展现了自适应算法与传统算法在路径规划上的本质差异。

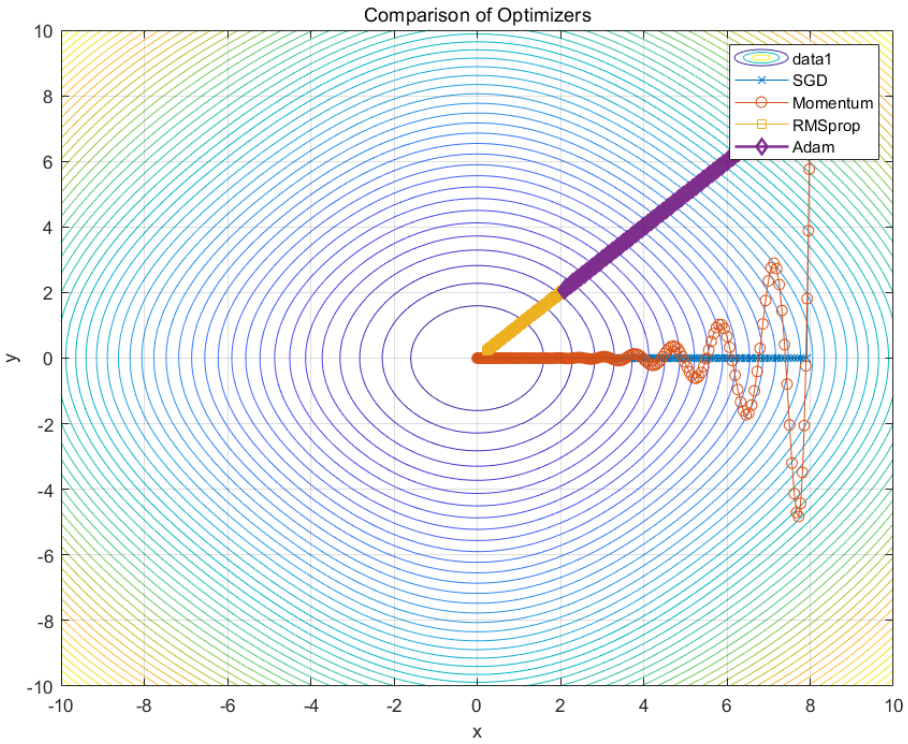

图 6: 各优化算法在各向异性地形下的收敛路径对比

实验结果深度分析：

1. **路径各向异性与坐标重缩放：**如图所示，**Adam** 与 **RMSprop** 的轨迹呈现出极高的直线度。这证明了自适应算法通过梯度的二阶矩估计，有效地对不同维度的步长进行了“归一化”处理。在 y 轴高梯度方向施加阻尼，在 x 轴低梯度方向补偿增益，从而使算法能够无视地形扭曲，沿最短欧几里得距离趋近最优解。

2. **震荡抑制与数值稳定性：**相比之下，标准 **SGD** 与 **Momentum** 在垂直方向（高曲率方向）发生了剧烈的震荡。这是由于固定学习率无法适应不同方向的梯度量级差异导致的。虽然 Momentum 凭借动量机制加快了向左侧移动的速度，但其路径的摆动显著增加了计算开销并降低了收敛稳定性。

3. **收敛进度与学习率保守性：**值得注意的是，在相同的迭代步数下，Adam 与 RMSprop 尚未完全抵达全局最低点。这揭示了自适应算法的一种“谨慎”更新机制：在感知到复杂的梯度环境时，算法会优先保证数值稳定性，通过压低步长来换取路径的平滑性。在实际工程应用中，这通常意味着 Adam 需要配合更多的迭代次数或学习率调度策略以实现最终的精准落位。

4 拉格朗日乘子法与 KKT 条件

4.1 拉格朗日乘子法

4.1.1 核心定义

拉格朗日乘子法是一种寻找多元函数在一组约束下的极值的方法。通过引入拉格朗日乘子，可将有 d 个变量与 k 个约束条件的最优化问题转化为具有 $d+k$ 个变量的无约束优化问题求解。

4.1.2 约束优化问题的数学表达

在机器学习和数学规划中，我们通常面临如下形式的约束优化问题：

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{s.t.} \quad & h_i(\boldsymbol{x}) = 0 \quad (i = 1, \ldots, m) \\ & g_j(\boldsymbol{x}) \leqslant 0 \quad (j = 1, \ldots, n) \end{aligned} \tag{10}$$

其中 $f(\boldsymbol{x})$ 为目标函数，$h_i(\boldsymbol{x}) = 0$ 为等式约束，$g_j(\boldsymbol{x}) \leqslant 0$ 为不等式约束。

4.2 拉格朗日函数与 KKT 条件

4.2.1 构建拉格朗日函数

引入拉格朗日乘子向量：等式约束乘子 $\boldsymbol{\lambda} = (\lambda_1, \ldots, \lambda_m)^T$，不等式约束乘子 $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_n)^T$。拉格朗日函数定义为：

$$L(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i h_i(\boldsymbol{x}) + \sum_{j=1}^{n} \mu_j g_j(\boldsymbol{x}) \tag{11}$$

4.2.2 KKT 条件

对于包含不等式约束的优化问题，最优解必须满足 Karush-Kuhn-Tucker (KKT) 条件：

$$\begin{cases} \nabla_{\boldsymbol{x}} L = 0 & \text{（梯度为零条件）} \\ h_k(\boldsymbol{x}) = 0 & (k = 1, \ldots, m) \\ g_j(\boldsymbol{x}) \leqslant 0 & (j = 1, \ldots, n) \\ \mu_j \geqslant 0 & (j = 1, \ldots, n) \\ \mu_j g_j(\boldsymbol{x}) = 0 & (j = 1, \ldots, n) \quad \text{（互补松弛条件）} \end{cases} \tag{12}$$

互补松弛条件说明：

- 若 $\mu_j > 0$，则 $g_j(\boldsymbol{x}) = 0$（约束有效，解在边界上）。

- 若 $g_j(\boldsymbol{x}) < 0$，则 $\mu_j = 0$（约束无效，解在内部）。

4.3 应用实例：带不等式约束的最小化问题

考虑问题：

$$\min_{x_1,x_2} f(x_1, x_2) = (x_1 - 3)^2 + (x_2 - 3)^2, \quad \text{s.t. } g(x_1, x_2) = x_1 + x_2 - 4 \leqslant 0 \tag{13}$$

4.3.1 第一步：构建拉格朗日函数

$$L(x_1, x_2, \mu) = (x_1 - 3)^2 + (x_2 - 3)^2 + \mu(x_1 + x_2 - 4)$$

4.3.2 第二步：列出 KKT 条件

1. 梯度为零：$\frac{\partial L}{\partial x_1} = 2(x_1 - 3) + \mu = 0$，$\frac{\partial L}{\partial x_2} = 2(x_2 - 3) + \mu = 0$。
2. 原始可行性：$x_1 + x_2 - 4 \leqslant 0$。
3. 对偶可行性：$\mu \geqslant 0$。
4. 互补松弛：$\mu(x_1 + x_2 - 4) = 0$。

4.3.3 第三步：分情况求解

- **情况 A**：假设 $\mu = 0$，解得 $x_1 = x_2 = 3$，但代入约束得 $2 > 0$，违反可行性，舍去。
- **情况 B**：假设 $x_1 + x_2 - 4 = 0$，结合梯度条件得 $x_1 = x_2 = 2$，$\mu = 2$，满足所有 KKT 条件。

4.3.4 第四步：最终答案

最优解为 $x_1 = 2, x_2 = 2$，目标函数最小值 $f(2, 2) = 2$。

4.4 直观理解与说明

- 引入乘子 μ 相当于一个“惩罚项”，将无约束极值点 $(3,3)$ 拉回到可行域边界 $(2,2)$。
- 互补松弛条件自动判断解在内部还是边界上。
- 对偶可行性 $\mu \geqslant 0$ 保证了梯度方向的正确性。