Optimization Techniques for Machine Learning

AMLZC326 · #08 Continuous Optimization II

Anshid Aboobacker

Table of Contents

1. Limitations of Basic Gradient Descent
2. Stochastic Gradient Descent
3. AdaGrad
4. RMSProp
5. Adam Optimizer

Recap: Gradient Descent

We consider the unconstrained optimization problem $\min_x f(x)$

Gradient descent update: $x_{k+1} = x_k - \eta \nabla f(x_k)$

Interpretation

- $\nabla f(x_k)$ gives the direction of steepest increase
- Moving in the opposite direction reduces the function value
- $\eta > 0$ is the learning rate (step size)

Key Idea

Repeated steps in the negative gradient direction move the parameters toward a minimum.

Learning Objectives

By the end of this lecture you should be able to:

- Identify the three key limitations of vanilla gradient descent (cost, learning rate sensitivity, ill-conditioning)
- Implement SGD and mini-batch gradient descent and explain their trade-offs vs. full-batch GD
- Derive and apply the AdaGrad, RMSProp, and Adam update rules from the principle of adaptive learning rates
- Compare optimisers and select the appropriate one for a given problem

Limitation 1: Expensive Gradient Computation

In many machine learning problems the loss function has the form $L(w) = \frac{1}{n} \sum_{i=1}^{n} \ell_i(w)$

Example: Linear regression $\ell_i(w) = (x_i^T w - y_i)^2$

The gradient becomes $\nabla L(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla \ell_i(w)$

Issue

Each gradient step requires processing the entire dataset.

- Large datasets may contain millions of samples
- Each iteration becomes computationally expensive
- Training becomes slow

Limitation 2: Sensitivity to Learning Rate

Gradient descent update:

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

Performance depends strongly on the choice of η.

- If η is too small
  - Convergence becomes very slow
- If η is too large
  - The algorithm may oscillate
  - It may even diverge

Key Question

How should we choose an appropriate learning rate?

Limitation 3: Ill-Conditioned Optimization

Some objective functions have narrow curved valleys.
Example quadratic function:

$$f(x, y) = ax^2 + by^2$$

where $a \gg b$.

Behavior of Gradient Descent

- Gradient direction changes rapidly
- Updates zig-zag across the valley
- Convergence becomes very slow

Why Do We Need Better Algorithms?

Modern optimization methods address these issues by

- Using only a subset of data (stochastic methods)
- Adapting learning rates automatically
- Using historical gradient information

These ideas lead to algorithms such as

- Stochastic Gradient Descent (SGD)
- AdaGrad
- RMSProp
- Adam

Table of Contents

1. Limitations of Basic Gradient Descent
2. Stochastic Gradient Descent
3. AdaGrad
4. RMSProp
5. Adam Optimizer

Idea Behind Stochastic Gradient Descent

Earlier we saw that computing the full gradient

$$\nabla L(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla \ell_i(w)$$

requires processing the entire dataset.

Key Idea

Instead of computing the full gradient, we approximate it using a **single sample**.

Randomly choose a sample i and compute $\nabla \ell_i(w)$ as an estimate of the full gradient.

Intuition: SGD trades exact gradient computation for much cheaper updates.

Stochastic Gradient Descent Update

At iteration t:

1. Randomly select sample i
2. Compute sample gradient

$$g_t = \nabla \ell_i(w_t)$$

3. Update parameters

$$w_{t+1} = w_t - \eta g_t$$

Key Difference

- Gradient Descent: uses **all samples**
- SGD: uses **one sample per update**

Behavior and Advantages of SGD

Because gradients are estimated from individual samples, updates are **noisy**.
Typical behavior:

- Optimization path fluctuates around the minimum
- Convergence is less smooth
- Each update is computationally cheap

Advantages

- Faster iterations
- Scales to very large datasets
- Works well for streaming and online learning

Mini-Batch Gradient Descent

In practice we use **mini-batches** of samples.
Select batch B of size m.

$$\nabla L_B(w) = \frac{1}{m} \sum_{i \in B} \nabla \ell_i(w)$$

Update rule: $w_{t+1} = w_t - \eta \nabla L_B(w_t)$
Typical batch sizes: 32, 64, 128.

Cost per iteration:
Gradient Descent $\approx O(n)$
SGD $\approx O(1)$
Mini-batch $\approx O(m)$

Table of Contents

1. Limitations of Basic Gradient Descent
2. Stochastic Gradient Descent
3. AdaGrad
4. RMSProp
5. Adam Optimizer

Limitation of SGD

Stochastic Gradient Descent update:

$$w_{t+1} = w_t - \eta \nabla \ell_i(w_t)$$

Observation

All parameters use the **same learning rate** η.

However in many problems:

- Some parameters receive large gradients frequently
- Some parameters are updated rarely
- Different parameters may require different step sizes

Key Question

Can the learning rate adapt automatically during training?

Idea Behind AdaGrad

Instead of using a fixed learning rate, AdaGrad adjusts the step size using **past gradient magnitudes**.

Let $g_t = \nabla f(w_t)$ be the gradient at iteration t.
AdaGrad accumulates squared gradients

$$G_t = \sum_{i=1}^{t} g_i^2$$

G_t measures how large the gradients have been over time.

Large accumulated gradients indicate frequently updated parameters.

AdaGrad Update Rule

AdaGrad scales the learning rate using the accumulated gradient:

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla f(w_t)$$

where ϵ = small constant for numerical stability.
For each parameter w_j:

$$G_{t,j} = \sum_{i=1}^{t} g_{i,j}^2$$

$$w_{t+1,j} = w_{t,j} - \frac{\eta}{\sqrt{G_{t,j} + \epsilon}} g_{t,j}$$

Key Idea

Each parameter receives its own adaptive learning rate.

Intuition and Applications

If a parameter receives large gradients frequently

$$G_t \uparrow$$

the effective learning rate $\frac{\eta}{\sqrt{G_t}}$ becomes smaller.
If a parameter receives small or rare gradients

$$G_t \text{ remains small}$$

the learning rate stays larger.

Where AdaGrad Works Well

- Sparse data
- Natural language processing
- High-dimensional feature spaces

Rare features automatically receive larger learning rates.

Table of Contents

1. Limitations of Basic Gradient Descent
2. Stochastic Gradient Descent
3. AdaGrad
4. RMSProp
5. Adam Optimizer

Limitation of AdaGrad

Since $G_t = \sum_{i=1}^{t} g_i^2$ the accumulated value keeps increasing.

The effective learning rate $\frac{\eta}{\sqrt{G_t}}$ continually decreases.

Result

The learning rate can become extremely small, causing training to stop making progress.

Next Idea

Instead of accumulating all past gradients, use a **moving average**. Older gradients should have less influence than recent gradients.

This leads to the **RMSProp algorithm**.

Idea Behind RMSProp

RMSProp tracks a moving average of squared gradients.
Let

$$g_t = \nabla f(w_t)$$

be the gradient at iteration t.
The exponential moving average is

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1 - \beta) g_t^2$$

where β is the decay rate.

Interpretation

Recent gradients have more influence than older gradients.

This prevents the learning rate from shrinking indefinitely.

RMSProp Update Rule

Using the moving average of squared gradients, the update becomes

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

For each parameter w_j

$$E[g_j^2]_t = \beta E[g_j^2]_{t-1} + (1 - \beta) g_{t,j}^2$$

$$w_{t+1,j} = w_{t,j} - \frac{\eta}{\sqrt{E[g_j^2]_t + \epsilon}} g_{t,j}$$

Key Idea

Each parameter receives its own adaptive learning rate.

Intuition and Comparison with AdaGrad

If recent gradients are large $E[g^2]_t \uparrow$ step size decreases.

If recent gradients are small $E[g^2]_t \downarrow$ step size increases.

Comparison

AdaGrad

- Uses cumulative gradient history
- Learning rate shrinks indefinitely

RMSProp

- Uses exponential moving average
- Focuses on recent gradients
- Prevents learning rate collapse

RMSProp in Practice

Typical hyperparameters

learning rate $\eta \approx 0.001$ decay rate $\beta \approx 0.9$ $\epsilon \approx 10^{-8}$

RMSProp works well when

- gradients change rapidly
- optimization landscape is noisy
- training deep neural networks

Next Step

Combine

- momentum (first moment)
- RMSProp scaling (second moment)

to obtain the **Adam optimizer**.

Table of Contents

1. Limitations of Basic Gradient Descent
2. Stochastic Gradient Descent
3. AdaGrad
4. RMSProp
5. Adam Optimizer

Momentum in Optimization

Gradient descent updates can oscillate in narrow valleys.
Momentum smooths the update direction by accumulating past gradients.
Update rule:

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

$$w_{t+1} = w_t - \eta v_t$$

Idea

Past gradients influence the current update, leading to smoother optimization trajectories.

Motivation for Adam

We have seen two important ideas:

Momentum

Uses past gradients to smooth the update direction.

RMSProp

Adapts learning rates using the magnitude of recent gradients.

Key Idea

Can we combine both ideas?

The result is the **Adam optimizer**.

Adam = Adaptive Moment Estimation

Moments of the Gradient

Let $g_t = \nabla f(w_t)$ be the gradient at iteration t.
Adam maintains two exponential moving averages:

First Moment (Mean of Gradients)

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

Captures the average direction of recent gradients.

Second Moment (Mean of Squared Gradients)

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

Captures the magnitude of recent gradients.

β controls the decay rate with typical values $\beta_1 = 0.9$, $\beta_2 = 0.999$

Bias Correction

At early iterations the moving averages are biased toward zero. Adam corrects this using

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Interpretation

These corrected estimates provide more accurate statistics of gradients during early training.

Adam Update Rule

Using the corrected moment estimates, parameters are updated as

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

where

- η = learning rate
- ϵ = small constant for numerical stability

Key Idea

Adam combines momentum with adaptive learning rate scaling.

Adam Algorithm

Initialize $w_0, \quad m_0 = 0, \quad v_0 = 0$

For each iteration t:

1. Compute gradient $g_t = \nabla f(w_t)$
2. Update moments

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \qquad v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

3. Bias correction

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \qquad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

4. Update parameters

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

Why Adam Is Popular

Adam provides several advantages:

- Momentum stabilizes updates
- Adaptive learning rates handle scaling
- Works well for noisy gradients
- Requires little hyperparameter tuning

Typical default hyperparameters:

$$\eta = 0.001, \quad \beta_1 = 0.9, \quad \beta_2 = 0.999, \quad \epsilon = 10^{-8}$$

Adam is widely used for training deep neural networks.

Summary of Optimization Methods

Gradient Descent	Uses full dataset
SGD	Uses single samples
Mini-batch SGD	Uses a batch of samples
AdaGrad	Adaptive learning rates
RMSProp	Moving average of squared gradients
Adam	Momentum + RMSProp

Key Takeaway

Modern machine learning relies on adaptive stochastic optimization methods.

Thank you :)