Machine Learning (CE 40717)

Fall 2024

Ali Sharifi-Zarchi

CE Department
Sharif University of Technology

December 17, 2025

Optimization Problem

- **Goal**: Find the value of x where $f(x)$ is at a **minimum** or **maximum**.
- In neural networks, we aim to minimize **prediction error** by finding the optimal weights w^*:

$$w^* = \arg\min_{w} J(w)$$

- Simply put: determine the **direction to step** that will quickly **reduce loss**.

Convexity and Optimization

- **Convex Functions**:
  - A function is **convex** if any line segment between points on the curve lies **above or on** the curve.
  - Convex functions are easier to optimize, as they have a single **global minimum**.
  - Numerical methods like **Gradient Descent** are guaranteed to reach the global minimum in convex functions.

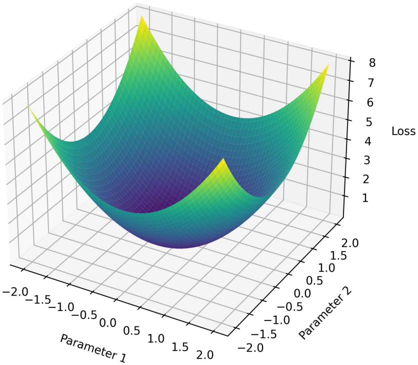

Figure 1: Example of convex function (bowl shape)

Non-Convex Functions and Challenges

- **Non-Convex Functions**:
  - Characterized by multiple **local minima** and **saddle points**.
  - **Global Minimum**: Overall lowest point.
  - **Local Minimum**: Lower than nearby points, but not the lowest overall.
  - **Saddle Points**: Regions where the gradient is close to zero but can increase or decrease in other directions.
- Finding the **global minimum** is more complex in non-convex functions.

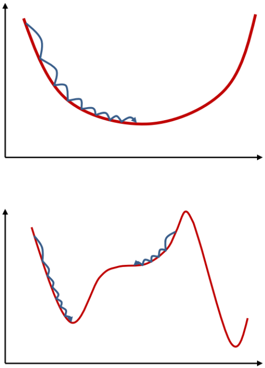

Figure 2: Convex (top) vs. Non-Convex (bottom) functions. Source: (CMU, 11-785)

Loss Surface Definition

- The **loss surface** shows how error changes based on network weights.
- For neural networks, the loss surface is typically **non-convex** due to multiple layers, nonlinear activations, and complex parameter interactions, resulting in **multiple local minima** and **saddle points**.
- In large networks, most local minima yield similar error values close to the **global minimum**; this is less true in smaller networks.

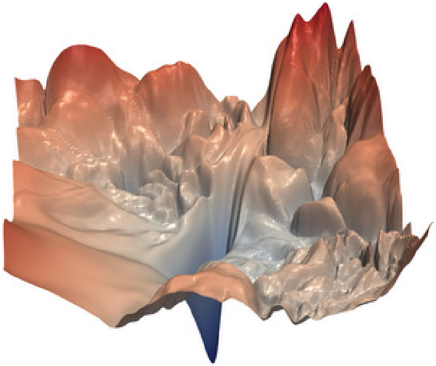

Figure 3: Loss surface of ResNet56. Source: GitHub: Loss Landscape

Loss Optimization

- **Goal**: How can we optimize a non-convex loss function effectively?
- **Gradient Descent**:
  - This method identifies the **steepest descent direction** to guide the optimization process.
- **Newton's Method**:
  - This method looks for **critical points** where the derivative $f'(x) = 0$, which may indicate minima, maxima, or saddle points.
  - Newton's Method uses the second derivative (Hessian) to adjust step sizes, which can lead to faster convergence compared to Gradient Descent.

Gradient Descent Overview

- **Gradient Descent**: As mentioned earlier in this course, Gradient Descent is an iterative method to minimize error by updating weights in the direction of the **negative gradient**:

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

  where η is the **learning rate**.

- **Types of Gradient Descent**:
  - **Batch**: Full dataset for stable but slow updates.
  - **Stochastic (SGD)**: One data point for fast, noisy updates.
  - **Mini-Batch**: Small batches, balancing speed and stability.

Problems with Gradient Descent

- **High Variability (SGD)**: Quick in steep directions but slow in shallow ones, causing **jitter and slow progress**.
- **Local Minima and Saddle Points**: Risk of **sub-optimal solutions** or long convergence times in flat regions.
- **Noisy Updates**: Using individual points or mini-batches introduces noise, affecting stable convergence.

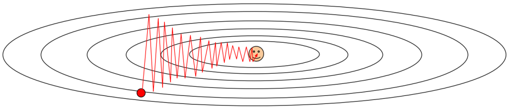

Figure 4: SGD Variability (CS231n, Stanford)

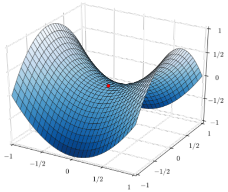

Figure 5: Saddle Point. Source: Wikipedia

Problem Definition

- **Objective**: **Enhance** the vanilla Gradient Descent algorithm to improve convergence and stability.
- **Challenges**:
  - Selecting **an appropriate learning rate** is crucial to avoid slow convergence and getting stuck in local minima.
- **Proposed Solution**:
  - Instead of testing multiple learning rates, incorporate **Momentum** to adaptively adjust the learning rate based on oscillations:
    - Increase steps in stable directions.
    - Decrease steps in oscillating directions.

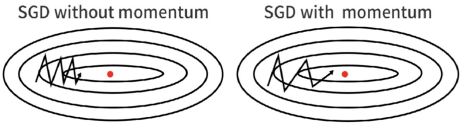

Figure 6: Momentum smooths oscillations and accelerates progress. Source: Papers with Code

Introduction to Momentum in Optimization

- **Origin of Momentum**:
  - Inspired by Newtonian physics, momentum in optimization uses **the concept of velocity in motion**, accumulating gradient history to smooth the learning trajectory, akin to an object moving based on past inertia.
  - Initially introduced to tackle challenges in gradient descent, where **inconsistent gradients or noisy updates** lead to erratic and slow convergence.
- **Purpose of Momentum**:
  - **Dampens Oscillations**: Utilizes prior gradients to minimize oscillations along steep or erratic regions, resulting in a smoother and more stable path.
  - **Speeds Up Convergence**: Particularly effective in narrow valleys or flat regions, where standard gradient descent may struggle or oscillate, causing slow progress.

First Moment (Momentum)

- **Definition**: The first moment, m_t, represents a moving average of past gradients. It builds "velocity" that propels learning in a consistent direction.
- **Update Rule**:

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1)\nabla_w J(w_t)$$

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

  where:
  - β_1: Decay rate, usually 0.9 or 0.99, which controls the weight of past gradients.
  - η: Learning rate.
- **Why Use First Momentum?**
  - Inspired by the idea of rolling momentum, it smooths and accelerates learning by sustaining direction from prior gradients.
  - This type of momentum is ideal for traversing narrow valleys or regions where standard gradient descent would oscillate.

Example of First Moment

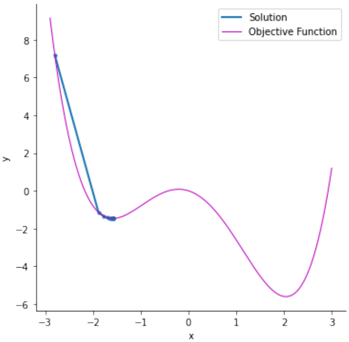

Figure 7: Stochastic gradient descent without momentum stops at a local minimum. Source: Akash Ajagekar (SYSEN 6800 Fall 2021)

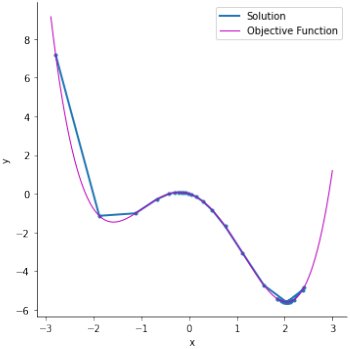

Figure 8: Stochastic gradient descent with momentum stops at the global minimum. Source: Akash Ajagekar (SYSEN 6800 Fall 2021)

Second Moment (Variance)

- **Definition**: The second moment, v_t, represents the moving average of squared gradients. It measures the gradient magnitude over time.
- **Update Rule**:

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

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_{t+1} + \epsilon}} m_{t+1}$$

  where:
  - β_2: Decay rate for variance (usually 0.99 or 0.999).
  - ϵ: Small constant to prevent division by zero.
- **Why Use Second Momentum?**
  - Adjusts step size based on gradient magnitude, preventing large steps when gradients are large and accelerating learning when they are small.

Moment Bias Correction

- **Problem**: When we start training, both m_t and v_t are initialized to zero, causing their estimates to be **biased toward zero in the early steps**, especially when gradients are small.
- **Solution**: We use bias-corrected versions of m_t and v_t to address this:

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

- These corrections compensate for the bias by scaling m_t and v_t upward, especially in the early steps when t is small, ensuring more accurate estimates of the moments.

Introduction to Adam Optimizer

- **Origin and Purpose**:
  - Proposed in 2014 by Diederik Kingma and Jimmy Ba, Adam (Adaptive Moment Estimation) addresses key limitations in earlier optimization methods by combining aspects of **momentum** and **adaptive learning rates**.
  - Adam is designed to handle sparse gradients and noisy updates by adjusting the learning rate for each parameter based on historical gradients.
- **Core Idea**:
  - Adam optimizes by maintaining two moving averages — the **first moment (mean of gradients)** and the **second moment (variance of gradients)** — allowing it to **adapt learning rates for each parameter individually**.

Adam's Adaptive Learning Rate Mechanism

- **Why Adaptive Rates?**
  - Unlike traditional SGD, Adam adapts the learning rate for **each parameter** based on recent gradient magnitudes.
  - **Large gradients** lead to **reduced** update sizes, while **smaller gradients** allow **larger** updates, balancing convergence speed and stability.
- **Moment Tracking**
  - The **first moment** (m_t) tracks the mean of gradients to provide momentum.
  - The **second moment** (v_t) tracks squared gradients, enabling Adam to normalize updates and prevent sudden changes in direction.

Mathematical Formulation of Adam

- **Adam Update Rules**:
  - First moment estimate:

$$m_{t+1} = \beta_1 m_t + (1 - \beta_1) \nabla_w J(w_t)$$

  - Second moment estimate:

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

  - Bias-corrected moments to address initialization bias:

$$\hat{m}_{t+1} = \frac{m_{t+1}}{1 - \beta_1^{t+1}}, \quad \hat{v}_{t+1} = \frac{v_{t+1}}{1 - \beta_2^{t+1}}$$

  - Update step for parameter w_t:

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

Adam Pseudo-code

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while θ_t not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return θ_t (Resulting parameters)

Figure 9: Adam Pseudo-code. Source: kingma2014adam

Adam Visualization

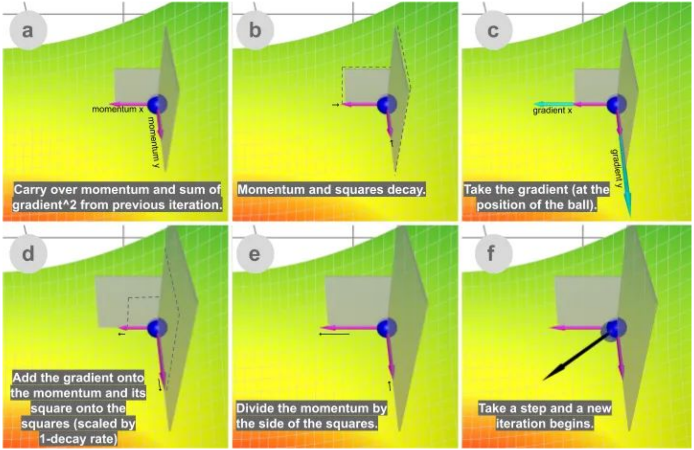

Figure 10: Step-by-step illustration of Adam descent. Source: Towards Data Science

Comparison of Momentum Methods

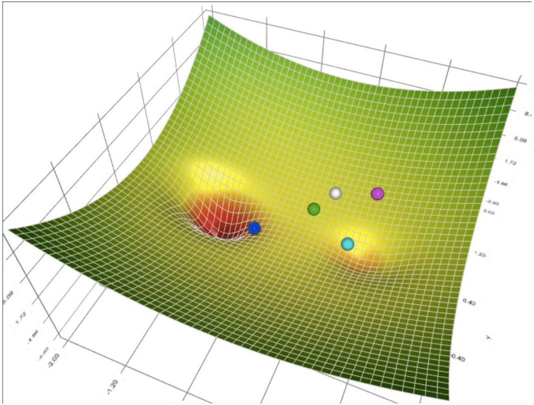

Figure 11: Comparison of 5 gradient descent methods on a surface: gradient descent (cyan), momentum (magenta), AdaGrad (white), RMSProp (green), Adam (blue). Left well is the global minimum; right well is a local minimum. Source: Towards Data Science

Newton's Method

- Newton method is originally intended to **find the root(s)** of an equation.
- **Example:** for the equation $x^2 - 1 = 0$, we can find the roots by decomposing $(x-1)(x+1) = 0$ which gives $x = 1, x = -1$
- **But, what about complex equations?**
  - We can use **numerical method** to find the root of an equation, one of them is by using **Newton's method**

Definition

- **Objective:** Derive Newton's method by finding the tangent line of $f(x)$ at x_0.
- **Tangent Line Equation:** Given a point x_0 where $f(x_0) \neq 0$, the tangent line at x_0 is:

$$y = mx_0 + c$$

- **Gradient:** The slope m matches the derivative of $f(x)$ at x_0:

$$m = f'(x_0)$$

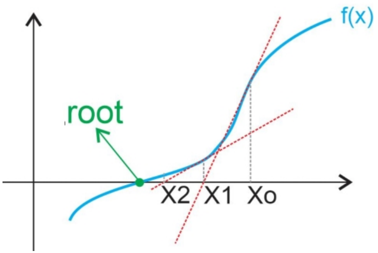

Figure 12: Finding root location by using Newton's method. Source: Ardian Umam's Blog

Formulating the Tangent Line

- **Finding c :** Substitute $(x_0, f(x_0))$ into $y = mx + c$, where $y = f(x_0)$ and $m = f'(x_0)$:

$$f(x_0) = f'(x_0)x_0 + c \Rightarrow c = f(x_0) - f'(x_0)x_0$$

- **Tangent Line Equation:** Substitute $m = f'(x_0)$ and c back:

$$y = f'(x_0)x + f(x_0) - f'(x_0)x_0$$

- Simplify to get:

$$y = f(x_0) + f'(x_0)(x - x_0)$$

Newton's Iterative Step

- To approximate the root, set $y = 0$ in the tangent equation:

$$0 = f(x_0) + f'(x_0)(x_1 - x_0)$$

- Rearrange to solve for x_1:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

- **Iteration:** Repeat this step to approximate the root.

Newton's Method for Optimization

- Newton's method for finding roots is based on a first-order approximation (tangent line).
- For optimization, we use a second-order Taylor approximation to find the minimum.
- **Second-order Taylor expansion** of $f(x)$ around $x = x_0$:

$$f(x) \approx f(x_0) + f'(x_0)(x - x_0) + \frac{f''(x_0)}{2}(x - x_0)^2$$

- Rearranged for minimal value location:

$$f(x) \approx \frac{1}{2}f''(x_0)x^2 + [f'(x_0) - f''(x_0)x_0]x + [f(x_0) - f'(x_0)x_0 + \frac{1}{2}f''(x_0)x_0^2]$$

Deriving the Update Formula for Minimization

- To locate the minimum, take the derivative with respect to x and set it to zero:

$$\frac{d}{dx}f(x) \approx f''(x_0)x + [f'(x_0) - f''(x_0)x_0] = 0$$

- Solving for x yields:

$$x = x_0 - \frac{f'(x_0)}{f''(x_0)}$$

- This is the update step for Newton's method in optimization, guiding us to the minimum. The general update rule is:

$$x_{t+1} = x_t - H^{-1}\nabla_x f(x_t)$$

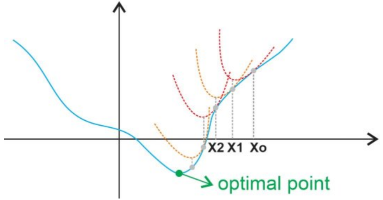

Figure 13: Finding root using Taylor's expansion and Newton's method. Source: Ardian Umam's Blog

Hessian Matrix and Newton's Method for Optimization

- The Hessian matrix, $H(\theta)$, is a square matrix of second-order partial derivatives of a scalar-valued function $f(\theta)$:

$$H(\theta) = \begin{bmatrix} \frac{\partial^2 f}{\partial \theta_1^2} & \frac{\partial^2 f}{\partial \theta_1 \partial \theta_2} & \cdots & \frac{\partial^2 f}{\partial \theta_1 \partial \theta_n} \\ \frac{\partial^2 f}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 f}{\partial \theta_2^2} & \cdots & \frac{\partial^2 f}{\partial \theta_2 \partial \theta_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial \theta_n \partial \theta_1} & \frac{\partial^2 f}{\partial \theta_n \partial \theta_2} & \cdots & \frac{\partial^2 f}{\partial \theta_n^2} \end{bmatrix}$$

- In Newton's method for optimization, the update rule for parameters θ is:

$$\theta_{t+1} = \theta_t - H^{-1}(\theta_t)\nabla f(\theta_t)$$

- **Example:**

$$f(\theta_1, \theta_2) = \theta_1^2 + 2\theta_2^2$$

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial \theta_1} \\ \frac{\partial f}{\partial \theta_2} \end{bmatrix} = \begin{bmatrix} 2\theta_1 \\ 4\theta_2 \end{bmatrix}$$

$$H(\theta) = \begin{bmatrix} \frac{\partial^2 f}{\partial \theta_1^2} & \frac{\partial^2 f}{\partial \theta_1 \partial \theta_2} \\ \frac{\partial^2 f}{\partial \theta_2 \partial \theta_1} & \frac{\partial^2 f}{\partial \theta_2^2} \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 4 \end{bmatrix}$$

$$H^{-1} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/4 \end{bmatrix}$$

$$\theta_{t+1} = \theta_t - H^{-1}\nabla f(\theta_t)$$

Newton's Method: Advantages and Disadvantages

- Newton's method offers various benefits but also has limitations, especially in large-scale machine learning. Below is a summary:

Advantages	Disadvantages
Faster Convergence Quadratic convergence enables reaching minima faster in convex problems.	**Computationally Expensive** Requires Hessian calculation, making it costly in high-dimensional models.
Adaptive Step Sizes Curvature-based step adjustment avoids slow progress in shallow regions.	**Memory Intensive** Storing the Hessian matrix is memory-intensive for models with millions of parameters.
Reduced Oscillations Curvature information stabilizes paths in oscillatory regions.	**Convergence Challenges** May converge to saddle points in non-convex functions common in machine learning.

Table 1: Advantages and Disadvantages of Newton's Method

Contribution

- **These slides were prepared with contributions from:**
  - Alireza Sabounchi
  - Sina Daneshgar