Machine Learning 101

Rajdeep Chatterjee, Ph.D.
Amygdala AI, Bhubaneswar, India *

February 2025

LMS to SGD

1 Introduction to Least Mean Square (LMS)

The Least Mean Square (LMS) algorithm is a stochastic gradient descent method used for adaptive filtering and machine learning. It minimizes the mean square error between the desired output and the actual output.

1.1 Mathematical Formulation

For a given input vector $\mathbf{x}(n)$ and desired output $d(n)$, the LMS algorithm updates the weight vector $\mathbf{w}(n)$ as:

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu e(n)\mathbf{x}(n) \tag{1}$$

where:

- μ is the learning rate
- $e(n) = d(n) - y(n)$ is the error
- $y(n) = \mathbf{w}^T(n)\mathbf{x}(n)$ is the output

2 Application to MLP Training

2.1 MLP Architecture

2.2 LMS for MLP Weight Update

For an MLP with L layers, the weight update rule becomes:

$$\mathbf{W}_l(n+1) = \mathbf{W}_l(n) + \mu \delta_l(n)\mathbf{a}_{l-1}^T(n) \tag{2}$$

where:

- $\delta_l(n)$ is the error gradient at layer l
- $\mathbf{a}_{l-1}(n)$ is the activation from previous layer

*Amygdala AI, is an international volunteer-run research group that advocates for *AI for a better tomorrow* `http://amygdalaai.org/`.

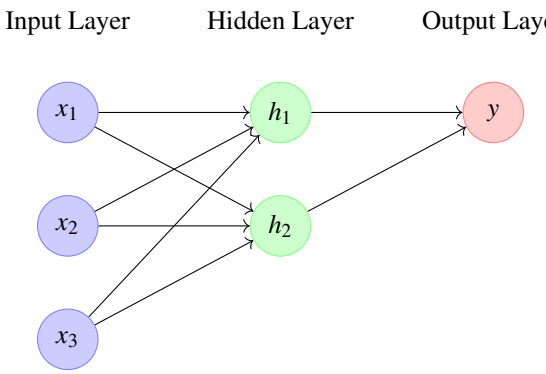

Figure 1: Multilayer Perceptron (MLP) Architecture

3 Advantages and Disadvantages

Table 1: Pros and Cons of LMS for MLP Training

Advantages	**Disadvantages**
Simple implementation	Slow convergence for complex problems
Low computational cost	Sensitive to learning rate choice
Online learning capability	May get stuck in local minima
Works well for linear problems	Not optimal for deep networks

4 Variants of LMS Algorithm

4.1 Normalized LMS (NLMS)

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \frac{\mu}{\|\mathbf{x}(n)\|^2 + \varepsilon} e(n)\mathbf{x}(n) \tag{3}$$

4.2 Leaky LMS

$$\mathbf{w}(n+1) = (1 - \mu\gamma)\mathbf{w}(n) + \mu e(n)\mathbf{x}(n) \tag{4}$$

4.3 Momentum LMS

$$\Delta\mathbf{w}(n+1) = \alpha\Delta\mathbf{w}(n) + \mu e(n)\mathbf{x}(n) \tag{5}$$

5 Example: XOR Problem

The LMS algorithm can train an MLP to solve the XOR problem by adjusting weights to create the necessary decision boundaries.

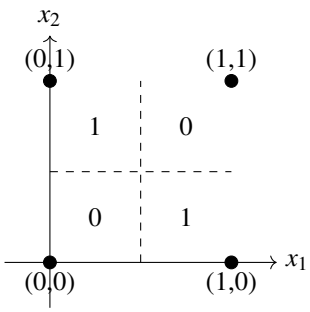

Figure 2: XOR Problem: A classic non-linear problem solvable by MLP with LMS

6 Convergence Analysis

The LMS algorithm converges when:

$$0 < \mu < \frac{2}{\lambda_{\max}} \tag{6}$$

where $\lambda_{\max}$ is the largest eigenvalue of the input autocorrelation matrix.

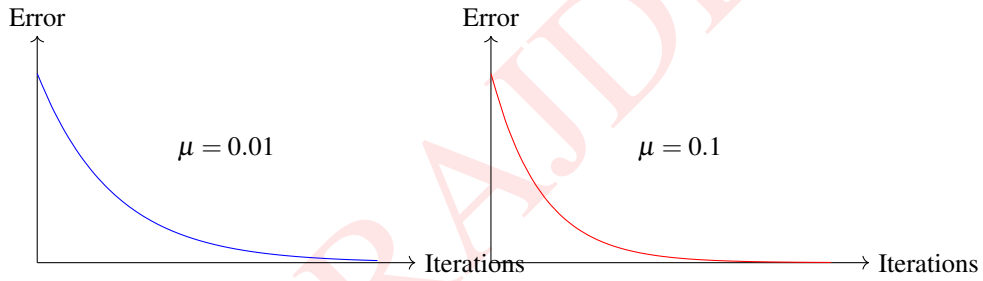

Figure 3: Error convergence for different learning rates

7 Gradient Descent

7.1 Definition

Gradient Descent is an optimization algorithm used to minimize a function by iteratively moving in the direction of steepest descent.

Given a cost function $J(\theta)$, gradient descent updates parameters as:

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \nabla J(\theta^{(t)})$$

where α is the learning rate.

7.2 Types of Gradient Descent

- **Batch Gradient Descent**: Uses entire dataset to compute gradient
- **Stochastic Gradient Descent (SGD)**: Uses one sample per iteration
- **Mini-batch Gradient Descent**: Uses a small subset of data per iteration

7.3 Gradient Descent for OLS

For OLS, the cost function is:

$$J(\beta) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - x_i^T\beta)^2$$

The gradient is:

$$\nabla J(\beta) = -\frac{1}{n}X^T(y - X\beta)$$

7.4 Numerical Example of Gradient Descent

Using the same toy dataset, let's perform one iteration of gradient descent with $\beta^{(0)} = (0,0)^T$ and $\alpha = 0.1$.

Initial cost:

$$J(\beta^{(0)}) = \frac{1}{8}[(1-0)^2 + (3-0)^2 + (3-0)^2 + (5-0)^2] = 5.5$$

Gradient calculation:

$$\nabla J(\beta^{(0)}) = -\frac{1}{4}\begin{pmatrix}1 & 1 & 1 & 1\\ 1 & 2 & 3 & 4\end{pmatrix}\begin{pmatrix}1\\3\\3\\5\end{pmatrix} = -\frac{1}{4}\begin{pmatrix}12\\34\end{pmatrix} = \begin{pmatrix}-3\\-8.5\end{pmatrix}$$

Parameter update:

$$\beta^{(1)} = \begin{pmatrix}0\\0\end{pmatrix} - 0.1\begin{pmatrix}-3\\-8.5\end{pmatrix} = \begin{pmatrix}0.3\\0.85\end{pmatrix}$$

New cost:

$$J(\beta^{(1)}) \approx 2.23$$

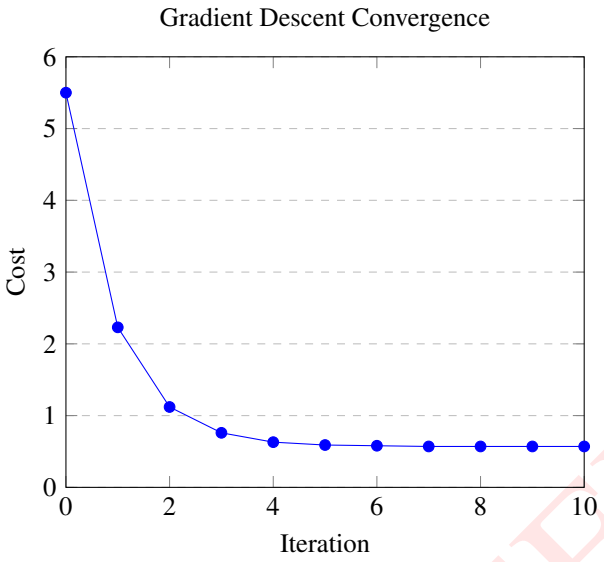

Figure 4: Cost function values during gradient descent optimization

8 Comparison of Gradient Descent (GD) and Least Mean Squares (LMS)

Both **Gradient Descent (GD)** and **Least Mean Squares (LMS)** are optimization methods, but GD is more general and flexible. Below are key advantages of GD over LMS:

8.1 1. Generalizability

- **GD** can optimize any differentiable loss function:

$$\mathscr{L}(\theta) \to \min_{\theta}$$

- **LMS** is restricted to Mean Squared Error (MSE) in linear regression:

$$\mathscr{L}(\theta) = \frac{1}{2N}\sum_{i=1}^{N}(y_i - \theta^T x_i)^2$$

8.2 2. Learning Rate Flexibility

- **GD** supports adaptive learning rates (e.g., Adam, AdaGrad):

$$\theta_{t+1} = \theta_t - \eta_t \nabla \mathscr{L}(\theta_t)$$

- **LMS** uses a fixed step size μ:

$$\theta_{t+1} = \theta_t + \mu(y_i - \theta_t^T x_i)x_i$$

8.3 3. Update Variants

- **GD** has:
  - Batch GD (full dataset)
  - Stochastic GD (one sample)
  - Mini-batch GD (small batches)
- **LMS** is equivalent to SGD with fixed μ.

8.4 4. Scalability

- **Mini-batch GD** is efficient for large datasets.
- **LMS** processes one sample at a time (slow for big data).

8.5 5. Non-linear Models

- **GD** works with neural networks, SVM, etc.
- **LMS** is limited to linear models.

8.6 6. Regularization Support

- **GD** easily incorporates L_1/L_2 penalties:

$$\mathscr{L}_{\text{reg}}(\theta) = \mathscr{L}(\theta) + \lambda \|\theta\|^2$$

- **LMS** requires manual modification for regularization.

Gradient Descent is **more general, scalable, and flexible** than LMS, making it suitable for a wider range of machine learning problems.

Table 2: Benefits of Gradient Descent (GD) Over Least Mean Squares (LMS)

Criteria	Gradient Descent (GD)	Least Mean Squares (LMS)
General Applicability	Works for any differentiable loss function (e.g., MSE, cross-entropy).	Limited to linear regression with Mean Squared Error (MSE).
Learning Rate	Supports adaptive learning rates (e.g., Adam, RMSprop).	Uses a fixed step size (μ), which may require careful tuning.
Update Variants	• Batch GD (full dataset) • Stochastic GD (per-sample) • Mini-batch GD (small batches)	Only processes one sample at a time (like SGD).
Scalability	Efficient for large datasets (especially mini-batch GD).	Less efficient for big data due to per-sample updates.
Non-linear Models	Applicable to neural networks, SVMs, and other non-linear models.	Only works for linear models.
Regularization	Easily incorporates L_1/L_2 penalties and constraints.	Requires explicit modification for regularization.
Convergence Control	Supports momentum, Nesterov acceleration, and other optimizations.	No built-in acceleration mechanisms.