Optimization Algorithms in Deep Learning

by Dr. Rishikesh Yadav

September 29, 2025

Assistant Professor, School of Mathematical and Statistical Sciences, IIT Mandi, India

Table of Contents

Recap: Regression and Classification

Supervised Learning: General Framework

- **Goal:** Learn (estimate) a function $f(\mathbf{x}; \mathbf{w})$ that maps several input features $\mathbf{x}$ to an output variable y.

$$y \approx f(\mathbf{x}; \mathbf{w}) \quad \text{(there will be some error terms as well)}$$

Supervised Learning: General Framework

- **Goal:** Learn (estimate) a function $f(\mathbf{x}; \mathbf{w})$ that maps several input features $\mathbf{x}$ to an output variable y.

$$y \approx f(\mathbf{x}; \mathbf{w}) \quad \text{(there will be some error terms as well)}$$

- **Data:** A collection of input–output pairs

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$$

Supervised Learning: General Framework

- **Goal:** Learn (estimate) a function $f(\mathbf{x}; \mathbf{w})$ that maps several input features $\mathbf{x}$ to an output variable y.

$$y \approx f(\mathbf{x}; \mathbf{w}) \quad \text{(there will be some error terms as well)}$$

- **Data:** A collection of input–output pairs

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$$

  - Inputs (features): $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^\top$

Supervised Learning: General Framework

- **Goal:** Learn (estimate) a function $f(\mathbf{x}; \mathbf{w})$ that maps several input features $\mathbf{x}$ to an output variable y.

$$y \approx f(\mathbf{x}; \mathbf{w}) \quad \text{(there will be some error terms as well)}$$

- **Data:** A collection of input–output pairs

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$$

  - Inputs (features): $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^\top$
  - Outputs (targets): y_i (numeric or categorical)

Supervised Learning: General Framework

- **Goal:** Learn (estimate) a function $f(\mathbf{x}; \mathbf{w})$ that maps several input features $\mathbf{x}$ to an output variable y.

$$y \approx f(\mathbf{x}; \mathbf{w}) \quad \text{(there will be some error terms as well)}$$

- **Data:** A collection of input–output pairs

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$$

  - Inputs (features): $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^\top$
  - Outputs (targets): y_i (numeric or categorical)

- **Learning:** Estimate the unknown parameters (weights in machine learning) $\mathbf{w}$ by minimizing a suitable **loss function** $\mathcal{L}(\mathbf{w})$.

Supervised Learning: General Framework

- **Goal:** Learn (estimate) a function $f(\mathbf{x}; \mathbf{w})$ that maps several input features $\mathbf{x}$ to an output variable y.

$$y \approx f(\mathbf{x}; \mathbf{w}) \quad \text{(there will be some error terms as well)}$$

- **Data:** A collection of input–output pairs

$$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^{n}$$

  - Inputs (features): $\mathbf{x}_i = (x_{i1}, x_{i2}, \ldots, x_{ip})^\top$
  - Outputs (targets): y_i (numeric or categorical)
- **Learning:** Estimate the unknown parameters (weights in machine learning) $\mathbf{w}$ by minimizing a suitable **loss function** $\mathcal{L}(\mathbf{w})$.
- **Prediction:** Once $\hat{\mathbf{w}}$ is obtained, use

$$\hat{y} = f(\mathbf{x}; \hat{\mathbf{w}})$$

for new input $\mathbf{x}$.

Regression Models

- **Task:** Predict a continuous output $y \in \mathbb{R}$.

Regression Models

- **Task:** Predict a continuous output $y \in \mathbb{R}$.
- **One simpler model (Assume f is linear in w):** Multiple Linear Regression (MLR)

$$y_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + \cdots + w_p x_{ip} + \varepsilon_i$$

Regression Models

- **Task:** Predict a continuous output $y \in \mathbb{R}$.
- **One simpler model (Assume f is linear in w):** Multiple Linear Regression (MLR)

$$y_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + \cdots + w_p x_{ip} + \varepsilon_i$$

- **Loss function (Mean Squared Error):**

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \mathbf{w}^\top \mathbf{x}_i)^2$$

Regression Models

- **Task:** Predict a continuous output $y \in \mathbb{R}$.
- **One simpler model (Assume f is linear in w):** Multiple Linear Regression (MLR)

$$y_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + \cdots + w_p x_{ip} + \varepsilon_i$$

- **Loss function (Mean Squared Error):**

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \mathbf{w}^\top \mathbf{x}_i)^2$$

- **Ordinary Least Squares (OLS) solution:**

$$\hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

where

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Regression Models

- **Task:** Predict a continuous output $y \in \mathbb{R}$.
- **One simpler model (Assume f is linear in w):** Multiple Linear Regression (MLR)

$$y_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + \cdots + w_p x_{ip} + \varepsilon_i$$

- **Loss function (Mean Squared Error):**

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \mathbf{w}^\top \mathbf{x}_i)^2$$

- **Ordinary Least Squares (OLS) solution:**

$$\hat{\mathbf{w}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

where

$$\mathbf{X} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1p} \\ 1 & x_{21} & x_{22} & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & x_{n2} & \cdots & x_{np} \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

- **Important Note**: For all regression types of problems, we might not get the closed-form expressions of $\hat{\mathbf{w}}$.

Classification Models

- **Task:** Predict a categorical output $y \in \{1, 2, \ldots, K\}$.

Classification Models

- **Task:** Predict a categorical output $y \in \{1, 2, \ldots, K\}$.
- **Model:** Logistic regression (binary or multinomial)

$$P(Y = k \mid \mathbf{x}, \mathbf{w}) = \frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^\top \mathbf{x})}$$

Classification Models

- **Task:** Predict a categorical output $y \in \{1, 2, \dots, K\}$.
- **Model:** Logistic regression (binary or multinomial)

$$P(Y = k \mid \mathbf{x}, \mathbf{w}) = \frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^\top \mathbf{x})}$$

- **Loss function (Cross-Entropy / Log-Loss):**

$$\mathcal{L}(\mathbf{w}) = -\sum_{i=1}^{n} \sum_{k=1}^{K} \mathbf{1}\{y_i = k\} \log P(Y = k \mid \mathbf{x}_i, \mathbf{w})$$

Classification Models

- **Task:** Predict a categorical output $y \in \{1, 2, \ldots, K\}$.
- **Model:** Logistic regression (binary or multinomial)

$$P(Y = k \mid \mathbf{x}, \mathbf{w}) = \frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^\top \mathbf{x})}$$

- **Loss function (Cross-Entropy / Log-Loss):**

$$\mathcal{L}(\mathbf{w}) = -\sum_{i=1}^{n} \sum_{k=1}^{K} \mathbf{1}\{y_i = k\} \log P(Y = k \mid \mathbf{x}_i, \mathbf{w})$$

- **Estimation:** No closed-form solution; parameters **w** are found using iterative optimization methods (e.g., gradient descent, Adam, Newton–Raphson etc.)

Classification Models

- **Task:** Predict a categorical output $y \in \{1, 2, \dots, K\}$.
- **Model:** Logistic regression (binary or multinomial)

$$P(Y = k \mid \mathbf{x}, \mathbf{w}) = \frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^\top \mathbf{x})}$$

- **Loss function (Cross-Entropy / Log-Loss):**

$$\mathcal{L}(\mathbf{w}) = -\sum_{i=1}^{n} \sum_{k=1}^{K} \mathbf{1}\{y_i = k\} \log P(Y = k \mid \mathbf{x}_i, \mathbf{w})$$

- **Estimation:** No closed-form solution; parameters $\mathbf{w}$ are found using iterative optimization methods (e.g., gradient descent, Adam, Newton–Raphson etc.)
- **Prediction:** Assign the class with the highest predicted probability

$$\hat{y}_i = \arg \max_{k \in \{1, \dots, K\}} P(Y = k \mid \mathbf{x}_i, \mathbf{w})$$

Role of Gradients and Hessians in Optimization

Role of Gradients in Optimization

- The **gradient** $\nabla f(x)$ points in the direction of the steepest increase of the function.

Role of Gradients in Optimization

- The **gradient** $\nabla f(x)$ points in the direction of the steepest increase of the function.
- Optimization algorithms use the **negative gradient** to move towards a minimum.

Role of Gradients in Optimization

- The **gradient** $\nabla f(x)$ points in the direction of the steepest increase of the function.
- Optimization algorithms use the **negative gradient** to move towards a minimum.
- **Intuition**:

Role of Gradients in Optimization

- The **gradient** $\nabla f(x)$ points in the direction of the steepest increase of the function.
- Optimization algorithms use the **negative gradient** to move towards a minimum.
- **Intuition**:
    - If $\nabla f(\boldsymbol{x}) = 0$, we are at a **stationary point**.

Role of Gradients in Optimization

- The **gradient** $\nabla f(x)$ points in the direction of the steepest increase of the function.
- Optimization algorithms use the **negative gradient** to move towards a minimum.
- **Intuition**:
  - If $\nabla f(\boldsymbol{x}) = 0$, we are at a **stationary point**.
  - Gradient magnitude $\|\nabla f(\boldsymbol{x})\|$ indicates how steep the surface is.

Role of Gradients in Optimization

- The **gradient** $\nabla f(x)$ points in the direction of the steepest increase of the function.
- Optimization algorithms use the **negative gradient** to move towards a minimum.
- **Intuition**:
  - If $\nabla f(\boldsymbol{x}) = 0$, we are at a **stationary point**.
  - Gradient magnitude $\|\nabla f(\boldsymbol{x})\|$ indicates how steep the surface is.
- **Example**: Gradient Descent updates

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta \nabla f(\boldsymbol{x}_k)$$

Role of Gradients in Optimization

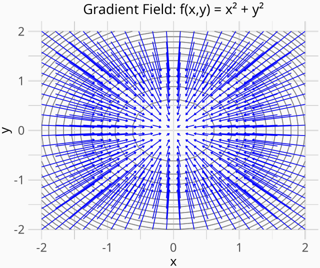
- The **gradient** $\nabla f(x)$ points in the direction of the steepest increase of the function.
- Optimization algorithms use the **negative gradient** to move towards a minimum.
- **Intuition**:
  - If $\nabla f(\boldsymbol{x}) = 0$, we are at a **stationary point**.
  - Gradient magnitude $\|\nabla f(\boldsymbol{x})\|$ indicates how steep the surface is.
- **Example**: Gradient Descent updates

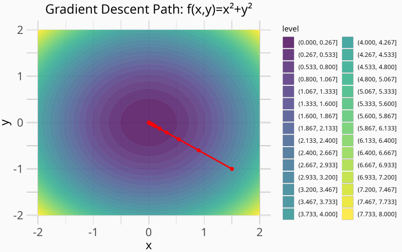
$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta \nabla f(\boldsymbol{x}_k)$$

Role of Gradients in Optimization

- The **gradient** $\nabla f(x)$ points in the direction of the steepest increase of the function.
- Optimization algorithms use the **negative gradient** to move towards a minimum.
- **Intuition**:
    - If $\nabla f(\boldsymbol{x}) = 0$, we are at a **stationary point**.
    - Gradient magnitude $\|\nabla f(\boldsymbol{x})\|$ indicates how steep the surface is.
- **Example**: Gradient Descent updates

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \eta \nabla f(\boldsymbol{x}_k)$$





Role of Hessians in Optimization

- The **Hessian matrix** $H(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x})$ captures the **curvature** of the function.

Role of Hessians in Optimization

- The **Hessian matrix** $H(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x})$ captures the **curvature** of the function.
- **Key roles:**

Role of Hessians in Optimization

- The **Hessian matrix** $H(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x})$ captures the **curvature** of the function.
- **Key roles:**
    - Determines if a stationary point is a minimum, maximum, or saddle.

Role of Hessians in Optimization

- The **Hessian matrix** $H(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x})$ captures the **curvature** of the function.
- **Key roles:**
  - Determines if a stationary point is a minimum, maximum, or saddle.
  - Guides second-order methods (e.g., Newton's method).

Role of Hessians in Optimization

- The **Hessian matrix** $H(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x})$ captures the **curvature** of the function.
- **Key roles:**
  - Determines if a stationary point is a minimum, maximum, or saddle.
  - Guides second-order methods (e.g., Newton's method).

  - Positive definite Hessian
    $\Rightarrow$ local minimum.

Role of Hessians in Optimization

- The **Hessian matrix** $H(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x})$ captures the **curvature** of the function.
- **Key roles:**
  - Determines if a stationary point is a minimum, maximum, or saddle.
  - Guides second-order methods (e.g., Newton's method).

  - Positive definite Hessian $\Rightarrow$ local minimum.
  - Negative definite Hessian $\Rightarrow$ local maximum.

Role of Hessians in Optimization

- The **Hessian matrix** $H(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x})$ captures the **curvature** of the function.
- **Key roles:**
  - Determines if a stationary point is a minimum, maximum, or saddle.
  - Guides second-order methods (e.g., Newton's method).

  - Positive definite Hessian $\Rightarrow$ local minimum.
  - Negative definite Hessian $\Rightarrow$ local maximum.
  - Indefinite Hessian $\Rightarrow$ saddle point.

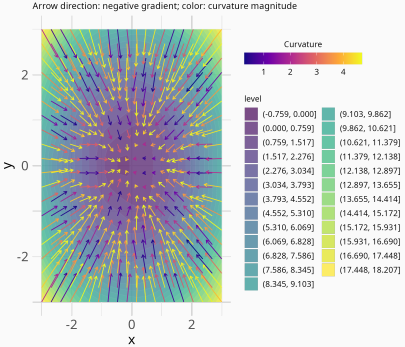
Role of Hessians in Optimization

- The **Hessian matrix** $H(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x})$ captures the **curvature** of the function.
- **Key roles:**
  - Determines if a stationary point is a minimum, maximum, or saddle.
  - Guides second-order methods (e.g., Newton's method).

  - Positive definite Hessian $\Rightarrow$ local minimum.
  - Negative definite Hessian $\Rightarrow$ local maximum.
  - Indefinite Hessian $\Rightarrow$ saddle point.

Role of Hessians in Optimization

- The **Hessian matrix** $H(\boldsymbol{x}) = \nabla^2 f(\boldsymbol{x})$ captures the **curvature** of the function.
- **Key roles:**
  - Determines if a stationary point is a minimum, maximum, or saddle.
  - Guides second-order methods (e.g., Newton's method).
  - Positive definite Hessian $\Rightarrow$ local minimum.
  - Negative definite Hessian $\Rightarrow$ local maximum.
  - Indefinite Hessian $\Rightarrow$ saddle point.



- **Example:** Newton-Raphson update

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - H(\boldsymbol{x}_k)^{-1} \nabla f(\boldsymbol{x}_k)$$

Famous Optimization Algorithms Used in Deep Learning

Agenda

Newton-Raphson Method

- **What is Newton–Raphson?** An iterative optimization method using both the **gradient** and the **Hessian**:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \left[\nabla_w^2 L(\boldsymbol{w}_t)\right]^{-1} \nabla_w L(\boldsymbol{w}_t)$$

Newton-Raphson Method

- **What is Newton–Raphson?** An iterative optimization method using both the **gradient** and the **Hessian**:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \left[\nabla_w^2 L(\boldsymbol{w}_t)\right]^{-1} \nabla_w L(\boldsymbol{w}_t)$$

- Uses curvature information (Hessian) for optimal step size and direction

Newton-Raphson Method

- **What is Newton–Raphson?** An iterative optimization method using both the **gradient** and the **Hessian**:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \left[\nabla_w^2 L(\boldsymbol{w}_t)\right]^{-1} \nabla_w L(\boldsymbol{w}_t)$$

- Uses curvature information (Hessian) for optimal step size and direction
- **Application Example:** classical Logistic regression with small, well-conditioned datasets

Newton-Raphson Method

- **What is Newton–Raphson?** An iterative optimization method using both the **gradient** and the **Hessian**:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \left[\nabla_w^2 L(\boldsymbol{w}_t)\right]^{-1} \nabla_w L(\boldsymbol{w}_t)$$

- Uses curvature information (Hessian) for optimal step size and direction
- **Application Example:** classical Logistic regression with small, well-conditioned datasets

Newton-Raphson Method

- **What is Newton–Raphson?** An iterative optimization method using both the **gradient** and the **Hessian**:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \left[\nabla_w^2 L(\boldsymbol{w}_t)\right]^{-1} \nabla_w L(\boldsymbol{w}_t)$$

- Uses curvature information (Hessian) for optimal step size and direction
- **Application Example:** classical Logistic regression with small, well-conditioned datasets

Advantages over Gradient Descent:

✓ **Quadratic convergence** near optimum (vs. linear)

Newton-Raphson Method

- **What is Newton–Raphson?** An iterative optimization method using both the **gradient** and the **Hessian**:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \left[\nabla_w^2 L(\boldsymbol{w}_t)\right]^{-1} \nabla_w L(\boldsymbol{w}_t)$$

- Uses curvature information (Hessian) for optimal step size and direction
- **Application Example:** classical Logistic regression with small, well-conditioned datasets

Advantages over Gradient Descent:

✓ **Quadratic convergence** near optimum (vs. linear)

✓ **No learning rate** η to tune - step size is adaptive

Newton-Raphson Method

- **What is Newton–Raphson?** An iterative optimization method using both the **gradient** and the **Hessian**:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \left[\nabla_w^2 L(\boldsymbol{w}_t)\right]^{-1} \nabla_w L(\boldsymbol{w}_t)$$

- Uses curvature information (Hessian) for optimal step size and direction
- **Application Example:** classical Logistic regression with small, well-conditioned datasets

Advantages over Gradient Descent:

✓ **Quadratic convergence** near optimum (vs. linear)

✓ **No learning rate** η to tune - step size is adaptive

✓ **Accounts for curvature** - follows natural shape of loss landscape

From Newton-Raphson to Gradient Descent

- **Hessian Computation:** $\nabla_w^2 L(\boldsymbol{w})$ requires $O(p^2)$ memory

From Newton-Raphson to Gradient Descent

- **Hessian Computation:** $\nabla_w^2 L(\boldsymbol{w})$ requires $O(p^2)$ memory
    - Modern DL: millions of parameters $\rightarrow$ infeasible storage

From Newton-Raphson to Gradient Descent

- **Hessian Computation:** $\nabla_w^2 L(\boldsymbol{w})$ requires $O(p^2)$ memory
  - Modern DL: millions of parameters $\rightarrow$ infeasible storage
- **Matrix Inversion:** $[\nabla_w^2 L(\boldsymbol{w})]^{-1}$ costs $O(p^3)$ computationally

From Newton-Raphson to Gradient Descent

- **Hessian Computation:** $\nabla_w^2 L(\boldsymbol{w})$ requires $O(p^2)$ memory
  - Modern DL: millions of parameters $\rightarrow$ infeasible storage
- **Matrix Inversion:** $[\nabla_w^2 L(\boldsymbol{w})]^{-1}$ costs $O(p^3)$ computationally
- **Non-Convexity:** Hessian may not be positive definite $\rightarrow$ convergence issues

From Newton-Raphson to Gradient Descent

- **Hessian Computation:** $\nabla_w^2 L(\boldsymbol{w})$ requires $O(p^2)$ memory
  - Modern DL: millions of parameters $\rightarrow$ infeasible storage
- **Matrix Inversion:** $[\nabla_w^2 L(\boldsymbol{w})]^{-1}$ costs $O(p^3)$ computationally
- **Non-Convexity:** Hessian may not be positive definite $\rightarrow$ convergence issues
- **Sensitivity:** Requires careful initialization and well-conditioned problems

From Newton-Raphson to Gradient Descent

- **Hessian Computation:** $\nabla_w^2 L(\boldsymbol{w})$ requires $O(p^2)$ memory
  - Modern DL: millions of parameters $\rightarrow$ infeasible storage
- **Matrix Inversion:** $[\nabla_w^2 L(\boldsymbol{w})]^{-1}$ costs $O(p^3)$ computationally
- **Non-Convexity:** Hessian may not be positive definite $\rightarrow$ convergence issues
- **Sensitivity:** Requires careful initialization and well-conditioned problems

From Newton-Raphson to Gradient Descent

- **Hessian Computation:** $\nabla_w^2 L(\boldsymbol{w})$ requires $O(p^2)$ memory
  - Modern DL: millions of parameters $\rightarrow$ infeasible storage
- **Matrix Inversion:** $[\nabla_w^2 L(\boldsymbol{w})]^{-1}$ costs $O(p^3)$ computationally
- **Non-Convexity:** Hessian may not be positive definite $\rightarrow$ convergence issues
- **Sensitivity:** Requires careful initialization and well-conditioned problems

Gradient Descent: The Practical Choice

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla_w L(\boldsymbol{w}_t)$$

- **Memory:** Stores only gradient ($O(p)$) vs. Hessian ($O(p^2)$)

From Newton-Raphson to Gradient Descent

- **Hessian Computation:** $\nabla^2_w L(\boldsymbol{w})$ requires $O(p^2)$ memory
  - Modern DL: millions of parameters $\rightarrow$ infeasible storage
- **Matrix Inversion:** $[\nabla^2_w L(\boldsymbol{w})]^{-1}$ costs $O(p^3)$ computationally
- **Non-Convexity:** Hessian may not be positive definite $\rightarrow$ convergence issues
- **Sensitivity:** Requires careful initialization and well-conditioned problems

Gradient Descent: The Practical Choice

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla_w L(\boldsymbol{w}_t)$$

- **Memory:** Stores only gradient ($O(p)$) vs. Hessian ($O(p^2)$)
- **Speed:** $O(p)$ per update vs. Newton's $O(p^3)$

From Newton-Raphson to Gradient Descent

- **Hessian Computation:** $\nabla_w^2 L(\boldsymbol{w})$ requires $O(p^2)$ memory
  - Modern DL: millions of parameters $\rightarrow$ infeasible storage
- **Matrix Inversion:** $[\nabla_w^2 L(\boldsymbol{w})]^{-1}$ costs $O(p^3)$ computationally
- **Non-Convexity:** Hessian may not be positive definite $\rightarrow$ convergence issues
- **Sensitivity:** Requires careful initialization and well-conditioned problems

Gradient Descent: The Practical Choice

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla_w L(\boldsymbol{w}_t)$$

- **Memory:** Stores only gradient ($O(p)$) vs. Hessian ($O(p^2)$)
- **Speed:** $O(p)$ per update vs. Newton's $O(p^3)$
- **Robustness:** Works reliably even in non-convex landscapes

From Newton-Raphson to Gradient Descent

- **Hessian Computation:** $\nabla^2_w L(\boldsymbol{w})$ requires $O(p^2)$ memory
  - Modern DL: millions of parameters $\rightarrow$ infeasible storage
- **Matrix Inversion:** $[\nabla^2_w L(\boldsymbol{w})]^{-1}$ costs $O(p^3)$ computationally
- **Non-Convexity:** Hessian may not be positive definite $\rightarrow$ convergence issues
- **Sensitivity:** Requires careful initialization and well-conditioned problems

Gradient Descent: The Practical Choice

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla_w L(\boldsymbol{w}_t)$$

- **Memory:** Stores only gradient ($O(p)$) vs. Hessian ($O(p^2)$)
- **Speed:** $O(p)$ per update vs. Newton's $O(p^3)$
- **Robustness:** Works reliably even in non-convex landscapes
- **Scalability:** Mini-batch version enables training on massive datasets

Agenda

Gradient Descent: The Core Idea

- **Goal:** Minimize a loss function $L(w)$ to find the best model parameters w for regression/classification.
- **Intuition:** Find the lowest point in a valley by always walking downhill.

Gradient Descent: The Core Idea

- **Goal:** Minimize a loss function $L(w)$ to find the best model parameters w for regression/classification.
- **Intuition:** Find the lowest point in a valley by always walking downhill.

The Update Rule:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla_w \mathcal{L}(\boldsymbol{w}_t)$$

- $\boldsymbol{w}$: Model parameters (e.g., weights)

Gradient Descent: The Core Idea

- **Goal:** Minimize a loss function $L(w)$ to find the best model parameters w for regression/classification.
- **Intuition:** Find the lowest point in a valley by always walking downhill.

The Update Rule:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla_w \mathcal{L}(\boldsymbol{w}_t)$$

- $\boldsymbol{w}$: Model parameters (e.g., weights)
- $\mathcal{L}(\boldsymbol{w})$: Loss function (e.g., Mean Squared Error, Log Loss)

Gradient Descent: The Core Idea

- **Goal:** Minimize a loss function $L(w)$ to find the best model parameters w for regression/classification.
- **Intuition:** Find the lowest point in a valley by always walking downhill.

The Update Rule:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla_w \mathcal{L}(\boldsymbol{w}_t)$$

- $\boldsymbol{w}$: Model parameters (e.g., weights)
- $\mathcal{L}(\boldsymbol{w})$: Loss function (e.g., Mean Squared Error, Log Loss)
- η: Learning rate (step size)

Gradient Descent: The Core Idea

- **Goal:** Minimize a loss function $L(w)$ to find the best model parameters w for regression/classification.
- **Intuition:** Find the lowest point in a valley by always walking downhill.

The Update Rule:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla_w \mathcal{L}(\boldsymbol{w}_t)$$

- $\boldsymbol{w}$: Model parameters (e.g., weights)
- $\mathcal{L}(\boldsymbol{w})$: Loss function (e.g., Mean Squared Error, Log Loss)
- η: Learning rate (step size)
- $\nabla_w \mathcal{L}(\boldsymbol{w})$: Gradient (direction of steepest ascent)

Gradient Descent: The Core Idea

- **Goal:** Minimize a loss function $L(w)$ to find the best model parameters w for regression/classification.
- **Intuition:** Find the lowest point in a valley by always walking downhill.

The Update Rule:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla_w \mathcal{L}(\boldsymbol{w}_t)$$

- $\boldsymbol{w}$: Model parameters (e.g., weights)
- $\mathcal{L}(\boldsymbol{w})$: Loss function (e.g., Mean Squared Error, Log Loss)
- η: Learning rate (step size)
- $\nabla_w \mathcal{L}(\boldsymbol{w})$: Gradient (direction of steepest ascent)

Gradient Descent: The Core Idea

- **Goal:** Minimize a loss function $L(w)$ to find the best model parameters w for regression/classification.
- **Intuition:** Find the lowest point in a valley by always walking downhill.

The Update Rule:

$$\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \eta \nabla_w \mathcal{L}(\boldsymbol{w}_t)$$

- $\boldsymbol{w}$: Model parameters (e.g., weights)
- $\mathcal{L}(\boldsymbol{w})$: Loss function (e.g., Mean Squared Error, Log Loss)
- η: Learning rate (step size)
- $\nabla_w \mathcal{L}(\boldsymbol{w})$: Gradient (direction of steepest ascent)

Motivation: Fundamental algorithm for training models like Linear Regression and Logistic Regression.

Batch Gradient Descent: The Standard Version

Algorithm:

1. Initialize parameters $\boldsymbol{w}$ randomly.
2. **Compute Gradient** over the **entire dataset**:

$$\nabla_w \mathcal{L}(\boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_w \text{Loss}(f_w(\boldsymbol{x}^{(i)}), y^{(i)})$$

3. **Update:** $\boldsymbol{w} = \boldsymbol{w} - \eta \nabla_w \mathcal{L}(\boldsymbol{w})$
4. Repeat until convergence.

Batch Gradient Descent: The Standard Version

Algorithm:

1. Initialize parameters $\boldsymbol{w}$ randomly.
2. **Compute Gradient** over the **entire dataset**:

$$\nabla_w \mathcal{L}(\boldsymbol{w}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_w \mathrm{Loss}(f_w(\boldsymbol{x}^{(i)}), y^{(i)})$$

3. **Update:** $\boldsymbol{w} = \boldsymbol{w} - \eta \nabla_w \mathcal{L}(\boldsymbol{w})$
4. Repeat until convergence.

Characteristics:

- ✓ **Pros:** Stable convergence. Guaranteed for convex functions.
- × **Cons:** Very slow for large datasets. One update requires a full data pass.

Stochastic Gradient Descent (SGD)

Core Idea: Use a **single, random** training example $(\boldsymbol{x}^{(i)}, y^{(i)})$ to compute a noisy gradient.

Stochastic Gradient Descent (SGD)

Core Idea: Use a **single, random** training example $(\boldsymbol{x}^{(i)}, y^{(i)})$ to compute a noisy gradient.

Algorithm per Epoch (one complete pass through data):

1. Shuffle the entire dataset.
2. For each example (input) i in the dataset:
   2.1 Compute gradient for one example: $\nabla_w \mathcal{L}(\boldsymbol{w}; \boldsymbol{x}^{(i)}, y^{(i)})$
   2.2 Update immediately: $\boldsymbol{w} = \boldsymbol{w} - \eta \nabla_w \mathcal{L}(\boldsymbol{w}; \boldsymbol{x}^{(i)}, y^{(i)})$

Stochastic Gradient Descent (SGD)

Core Idea: Use a **single, random** training example $(\boldsymbol{x}^{(i)}, y^{(i)})$ to compute a noisy gradient.

Algorithm per Epoch (one complete pass through data):

1. Shuffle the entire dataset.
2. For each example (input) i in the dataset:
   2.1 Compute gradient for one example: $\nabla_w \mathcal{L}(\boldsymbol{w}; \boldsymbol{x}^{(i)}, y^{(i)})$
   2.2 Update immediately: $\boldsymbol{w} = \boldsymbol{w} - \eta \nabla_w \mathcal{L}(\boldsymbol{w}; \boldsymbol{x}^{(i)}, y^{(i)})$

Key Properties:

- ✓ **Pros:** Extremely fast per update. Can escape local minima due to noise.
- × **Cons:** Very noisy path. Loss may fluctuate heavily. Harder to converge precisely.

Mini-batch Gradient Descent

Core Idea: The best compromise. Use a small random subset (a **mini-batch**) of size b to compute the gradient.

Mini-batch Gradient Descent

Core Idea: The best compromise. Use a small random subset (a **mini-batch**) of size b to compute the gradient.

Algorithm per Epoch:

1. Shuffle the dataset.
2. For each batch of b examples:
   2.1 Compute gradient for the batch: $\nabla_w \mathcal{L}(\boldsymbol{w}) = \frac{1}{b} \sum_{k=1}^{b} \nabla_w \mathrm{Loss}(f_w(\boldsymbol{x}^{(k)}), y^{(k)})$
   2.2 Update parameters: $\boldsymbol{w} = \boldsymbol{w} - \eta \nabla_w \mathcal{L}(\boldsymbol{w})$

Mini-batch Gradient Descent

Core Idea: The best compromise. Use a small random subset (a **mini-batch**) of size b to compute the gradient.

Algorithm per Epoch:

1. Shuffle the dataset.
2. For each batch of b examples:
   2.1 Compute gradient for the batch: $\nabla_w \mathcal{L}(\boldsymbol{w}) = \frac{1}{b} \sum_{k=1}^{b} \nabla_w \text{Loss}(f_w(\boldsymbol{x}^{(k)}), y^{(k)})$
   2.2 Update parameters: $\boldsymbol{w} = \boldsymbol{w} - \eta \nabla_w \mathcal{L}(\boldsymbol{w})$

Key Properties:

- ✓ **Pros:** Efficient and leverages GPU parallelism. More stable than SGD.
- × **Cons:** Introduces the batch size b as a new hyperparameter to tune.

Comparison: GD, SGD, and Mini-batch GD

Criterion	Batch GD	Stochastic GD	Mini-batch GD
Gradient	Full dataset	Single example	Small batch (b)
Speed/Update	Slow	Very Fast	Fast
Stability	Smooth	Noisy	Moderate
Memory	High	Low	Medium
Parallelization	Difficult	No	Excellent
Use Case	Small datasets	Large datasets	Deep Learning

Conclusion: For most modern machine learning tasks, especially deep learning, Mini-batch Gradient Descent is the preferred algorithm.