DA6401:Introduction to Deep Learning
Module 2B - Gradient Descent and its variants

Ganapathy Krishnamurthi

Department of Data Science and Artificial Intelligence , IIT Madras

The Learning Algorithm

What is a Learning Algorithm?

- A systematic procedure to adjust model parameters to minimize a loss function
- Also known as **optimizers** in deep learning

The Learning Algorithm

What is a Learning Algorithm?

- A systematic procedure to adjust model parameters to minimize a loss function
- Also known as **optimizers** in deep learning

Mathematical Objective:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta) \tag{1}$$

where:

- θ represents all learnable parameters (weights and biases)
- $\mathcal{L}(\theta)$ is the loss function measuring prediction error
- θ^* is the optimal parameter set

The Learning Algorithm

What is a Learning Algorithm?

- A systematic procedure to adjust model parameters to minimize a loss function
- Also known as **optimizers** in deep learning

Mathematical Objective:

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta) \tag{1}$$

where:

- θ represents all learnable parameters (weights and biases)
- $\mathcal{L}(\theta)$ is the loss function measuring prediction error
- θ^* is the optimal parameter set

Common Learning Algorithms:

- Gradient Descent (GD)
- Stochastic Gradient Descent (SGD)
- Adam, RMSprop, AdaGrad, etc.

Parameter Update

Learnable Parameters:

- Weights: $W^{(1)}, W^{(2)}, \ldots, W^{(L)}$
- Biases: $b^{(1)}, b^{(2)}, \ldots, b^{(L)}$
- Collectively denoted as: $\theta = \{W^{(1)}, b^{(1)}, \ldots, W^{(L)}, b^{(L)}\}$

Parameter Update

Learnable Parameters:

- Weights: $W^{(1)}, W^{(2)}, \ldots, W^{(L)}$
- Biases: $b^{(1)}, b^{(2)}, \ldots, b^{(L)}$
- Collectively denoted as: $\theta = \{W^{(1)}, b^{(1)}, \ldots, W^{(L)}, b^{(L)}\}$

Parameter Update:

$$\theta_{\text{new}} = \theta_{\text{old}} + \Delta\theta \tag{2}$$

Parameter Update

Learnable Parameters:

- Weights: $W^{(1)}, W^{(2)}, \ldots, W^{(L)}$
- Biases: $b^{(1)}, b^{(2)}, \ldots, b^{(L)}$
- Collectively denoted as: $\theta = \{W^{(1)}, b^{(1)}, \ldots, W^{(L)}, b^{(L)}\}$

Parameter Update:

$$\theta_{\text{new}} = \theta_{\text{old}} + \Delta\theta \tag{2}$$

Problem: Without proper scaling, we have no control over:

- How much the parameters change
- How fast the network learns
- Stability of training

Parameter Update

Learnable Parameters:

- Weights: $W^{(1)}, W^{(2)}, \ldots, W^{(L)}$
- Biases: $b^{(1)}, b^{(2)}, \ldots, b^{(L)}$
- Collectively denoted as: $\theta = \{W^{(1)}, b^{(1)}, \ldots, W^{(L)}, b^{(L)}\}$

Parameter Update:

$$\theta_{\text{new}} = \theta_{\text{old}} + \Delta\theta \tag{2}$$

Problem: Without proper scaling, we have no control over:

- How much the parameters change
- How fast the network learns
- Stability of training

Solution: Introduce a learning rate η

$$\theta_{\text{new}} = \theta_{\text{old}} + \eta \cdot \Delta\theta \tag{3}$$

Parameter Update: Vector Perspective

Setup:

Parameter vector:

$$\theta = [w, b]$$

Parameter Update: Vector Perspective

Setup:

Parameter vector:

$$\theta = [w, b]$$

Update vector:

$$\Delta\theta = [\Delta w, \Delta b]$$

Parameter Update: Vector Perspective

Setup:

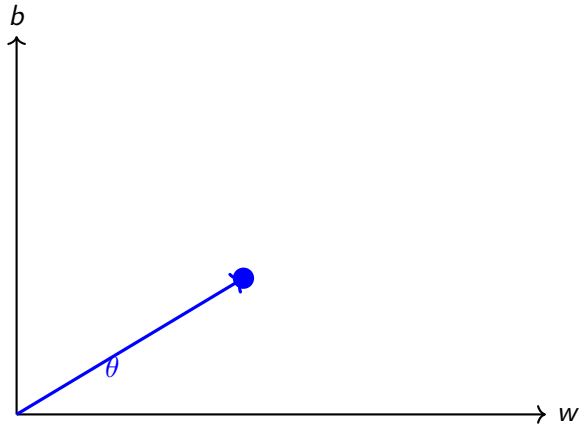
Parameter vector:

$$\theta = [w, b]$$

Update vector:

$$\Delta\theta = [\Delta w, \Delta b]$$

Problem: Directly adding $\Delta\theta$ to θ may cause unstable learning!



Parameter Update: Vector Perspective

Setup:

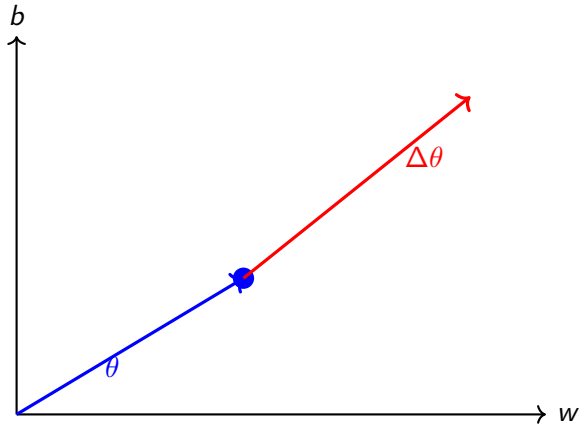
Parameter vector:

$$\theta = [w, b]$$

Update vector:

$$\Delta\theta = [\Delta w, \Delta b]$$

Problem: Directly adding $\Delta\theta$ to θ may cause unstable learning!



Parameter Update: Vector Perspective

Setup:

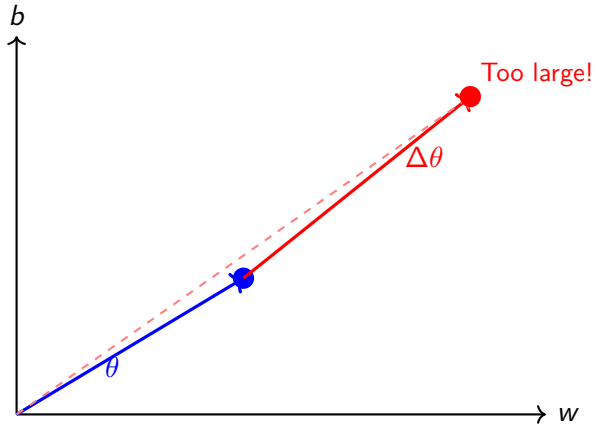
Parameter vector:

$$\theta = [w, b]$$

Update vector:

$$\Delta\theta = [\Delta w, \Delta b]$$

Problem: Directly adding $\Delta\theta$ to θ may cause unstable learning!



Parameter Update: Vector Perspective

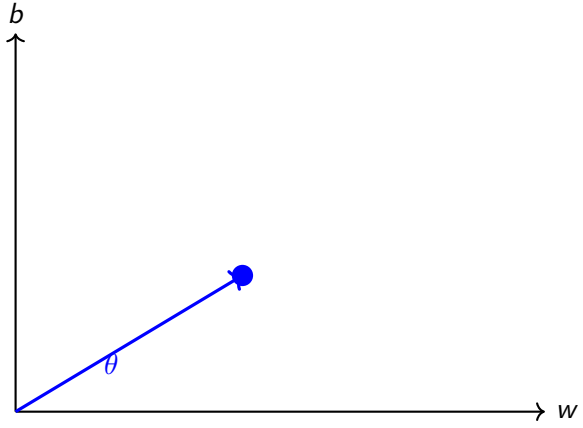
Solution:

Introduce learning rate η:

$$\theta_{\text{new}} = \theta + \eta \cdot \Delta\theta$$

where $0 < \eta < 1$
(typically $\eta \approx 0.001$ to 0.1)

This scales the update to a controlled step size



Controlled, stable learning!

Parameter Update: Vector Perspective

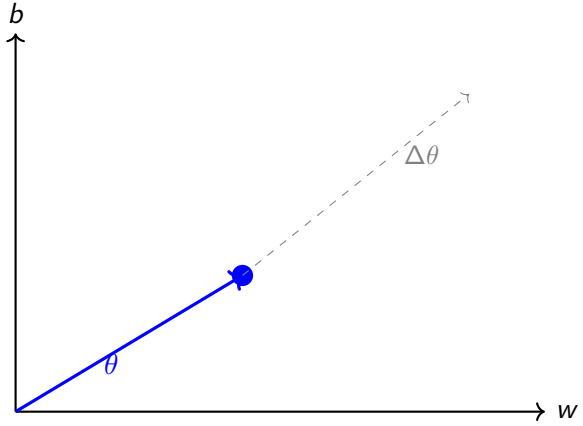
Solution:

Introduce learning rate η:

$$\theta_{\text{new}} = \theta + \eta \cdot \Delta\theta$$

where $0 < \eta < 1$
(typically $\eta \approx 0.001$ to 0.1)

This scales the update to a controlled step size



Controlled, stable learning!

Parameter Update: Vector Perspective

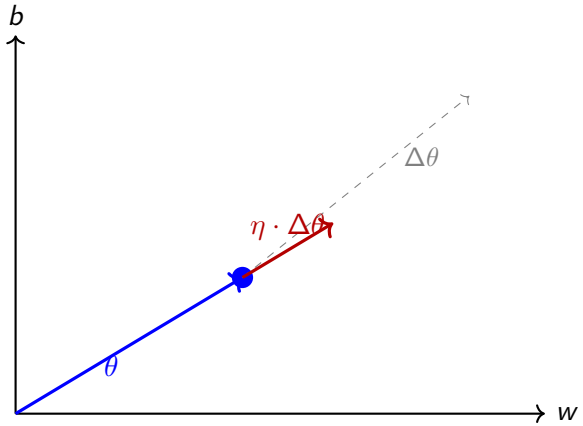
Solution:

Introduce learning rate η:

$$\theta_{\text{new}} = \theta + \eta \cdot \Delta\theta$$

where $0 < \eta < 1$
(typically $\eta \approx 0.001$ to 0.1)

This scales the update to a controlled step size



Controlled, stable learning!

Parameter Update: Vector Perspective

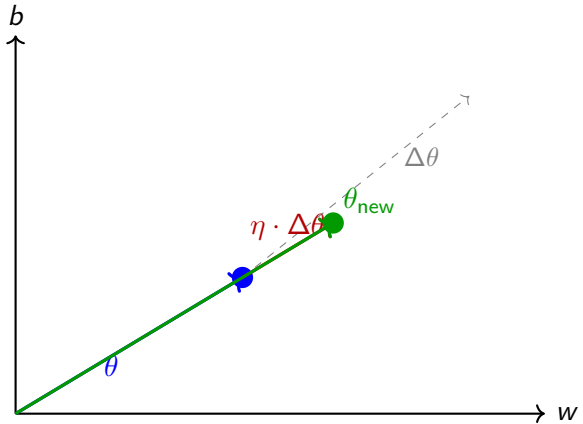
Solution:

Introduce learning rate η:

$$\theta_{\text{new}} = \theta + \eta \cdot \Delta\theta$$

where $0 < \eta < 1$
(typically $\eta \approx 0.001$ to 0.1)

This scales the update to a controlled step size



Controlled, stable learning!

Deriving the Incremental Update

Taylor Series Expansion:
For a function $f(x)$ around point a:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots \tag{4}$$

Deriving the Incremental Update

Taylor Series Expansion:
For a function $f(x)$ around point a:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \cdots \tag{4}$$

Applying to Loss Function:
Consider the loss after a parameter update: $\mathcal{L}(\theta + \eta \cdot \Delta\theta)$

Taylor expansion around θ:

$$\begin{aligned}\mathcal{L}(\theta + \eta \cdot \Delta\theta) = \mathcal{L}(\theta) + \nabla_\theta \mathcal{L}(\theta)^T \cdot (\eta \cdot \Delta\theta) \\ + \frac{1}{2}(\eta \cdot \Delta\theta)^T H(\theta)(\eta \cdot \Delta\theta) + \mathcal{O}(\eta^3)\end{aligned} \tag{5}$$

where $H(\theta)$ is the Hessian matrix (second derivatives)

Deriving the Incremental Update

First-Order Approximation:
Since η is typically small (e.g., 0.001), terms with $\eta^2, \eta^3, \ldots$ are negligible:

$$\mathcal{L}(\theta + \eta \cdot \Delta\theta) \approx \mathcal{L}(\theta) + \eta \cdot \nabla_\theta \mathcal{L}(\theta)^T \cdot \Delta\theta \tag{6}$$

Deriving the Incremental Update

First-Order Approximation:
Since η is typically small (e.g., 0.001), terms with $\eta^2, \eta^3, \ldots$ are negligible:

$$\mathcal{L}(\theta + \eta \cdot \Delta\theta) \approx \mathcal{L}(\theta) + \eta \cdot \nabla_\theta \mathcal{L}(\theta)^T \cdot \Delta\theta \tag{6}$$

Deriving the Update Rule:
To minimize the loss, we want:

$$\mathcal{L}(\theta + \eta \cdot \Delta\theta) < \mathcal{L}(\theta) \tag{7}$$

This requires:

$$\nabla_\theta \mathcal{L}(\theta)^T \cdot \Delta\theta < 0 \tag{8}$$

Optimal choice: Set $\Delta\theta = -\nabla_\theta \mathcal{L}(\theta)$
This gives the steepest descent direction!

The Gradient Descent Update Rule

Recall the Key Principle:

- The gradient $\nabla_\theta \mathcal{L}(\theta)$ points in the direction of steepest increase of the loss
- To minimize loss, we move in the opposite direction

The Gradient Descent Update Rule

Recall the Key Principle:

- The gradient $\nabla_\theta \mathcal{L}(\theta)$ points in the direction of steepest increase of the loss
- To minimize loss, we move in the opposite direction

Formal Update Rule:

$$\boxed{\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta \mathcal{L}(\theta_t)} \tag{9}$$

where:

- θ_t = parameters at iteration t
- η = learning rate (step size)
- $\nabla_\theta \mathcal{L}(\theta_t)$ = gradient of loss w.r.t. parameters
- The minus sign ensures we move downhill

The Gradient Descent Update Rule

Recall the Key Principle:

- The gradient $\nabla_\theta \mathcal{L}(\theta)$ points in the direction of steepest increase of the loss
- To minimize loss, we move in the opposite direction

Formal Update Rule:

$$\boxed{\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta \mathcal{L}(\theta_t)} \tag{9}$$

where:

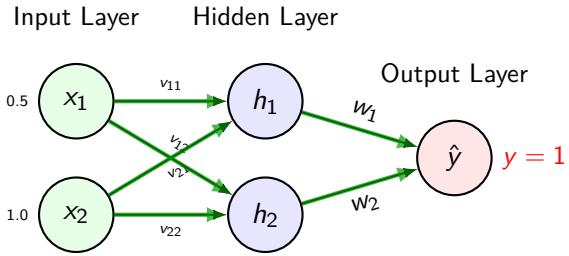
- θ_t = parameters at iteration t
- η = learning rate (step size)
- $\nabla_\theta \mathcal{L}(\theta_t)$ = gradient of loss w.r.t. parameters
- The minus sign ensures we move downhill

Component-wise:

$$W_{t+1}^{(l)} = W_t^{(l)} - \eta \cdot \frac{\partial \mathcal{L}}{\partial W^{(l)}} \tag{10}$$

$$b_{t+1}^{(l)} = b_t^{(l)} - \eta \cdot \frac{\partial \mathcal{L}}{\partial b^{(l)}} \tag{11}$$

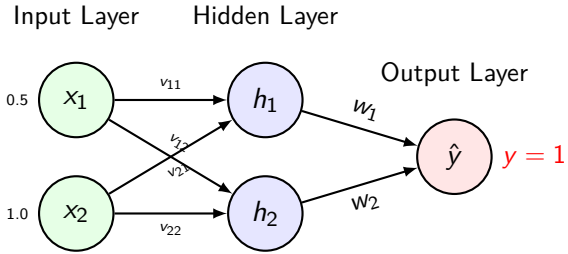
The Complete Learning Process



Step 1: Forward Pass

$$
\begin{aligned}
h_1 &= \sigma(v_{11}x_1 + v_{21}x_2 + b_1) \\
h_2 &= \sigma(v_{12}x_1 + v_{22}x_2 + b_2) \\
\hat{y} &= \sigma(w_1 h_1 + w_2 h_2 + b_3)
\end{aligned}
$$

The Complete Learning Process

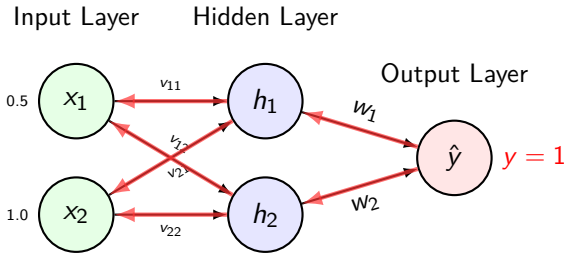


Step 2: Compute Loss

$$\mathcal{L} = \frac{1}{2}(\hat{y} - y)^2$$

For this example: $\mathcal{L} = \frac{1}{2}(\hat{y} - 1)^2$

The Complete Learning Process

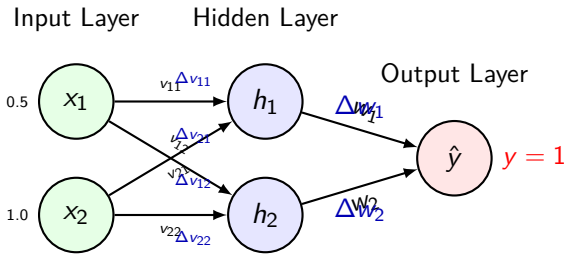


Step 3: Backward Pass (Backpropagation)

Compute: $\frac{\partial \mathcal{L}}{\partial w_1}, \frac{\partial \mathcal{L}}{\partial w_2}, \frac{\partial \mathcal{L}}{\partial v_{11}}, \frac{\partial \mathcal{L}}{\partial v_{12}}, \frac{\partial \mathcal{L}}{\partial v_{21}}, \frac{\partial \mathcal{L}}{\partial v_{22}}$

Gradients flow backward through the network

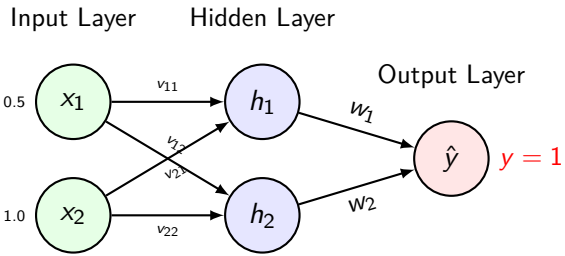
The Complete Learning Process



Step 4: Update Parameters

$$w_1 \leftarrow w_1 - \eta \frac{\partial \mathcal{L}}{\partial w_1}, \quad w_2 \leftarrow w_2 - \eta \frac{\partial \mathcal{L}}{\partial w_2}$$
$$v_{ij} \leftarrow v_{ij} - \eta \frac{\partial \mathcal{L}}{\partial v_{ij}} \quad \text{for all } i, j$$

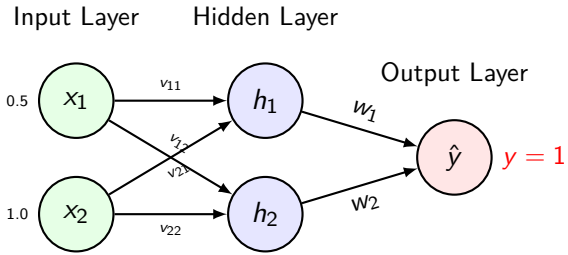
The Complete Learning Process



Step 5: Evaluate on Validation Set

Run forward pass on validation data (x_j, y_j)
Compute validation loss $\mathcal{L}_{\text{val}}$
Check for convergence or overfitting

The Complete Learning Process



Repeat: Iterate Until Convergence

For each epoch:
Repeat Steps 1-4 for all training samples
Evaluate on validation set (Step 5)
Stop when: Loss converges or max epochs reached

The Complete Learning Process: Algorithm

Training Algorithm

Input: $\mathcal{D}_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{N}$, learning rate η, epochs E

Initialize: $\theta_0 = \{v_{ij}, w_k, b_l\}$ randomly

For epoch $e = 1$ to E:

  For each data sample (x_i, y_i) in $\mathcal{D}_{\text{train}}$:

    1. **Forward Pass:** $\hat{y}_i = f(x_i; \theta)$
    2. **Compute Loss:** $\mathcal{L} = \frac{1}{|\text{batch}|} \sum \ell(\hat{y}, y)$
    3. **Backward Pass:** $\nabla_\theta \mathcal{L}$ via backpropagation
    4. **Update:** $\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}$

  End For

  5. **Evaluate:** Compute $\mathcal{L}_{\text{val}}$ on validation set
  6. **If** converged or performance degrades: **break**

End For

Return: Optimized parameters θ^*

Stochastic and Mini-Batch Gradient Descent

Vanilla Gradient Descent (Batch Gradient Descent)

Definition: Compute the gradient using the entire training dataset in each update step.

Vanilla Gradient Descent (Batch Gradient Descent)

Definition: Compute the gradient using the entire training dataset in each update step.

Mathematical Formulation:
Given training dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$ with N samples:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(x^{(i)}; \theta), y^{(i)}) \tag{12}$$

Vanilla Gradient Descent (Batch Gradient Descent)

Definition: Compute the gradient using the entire training dataset in each update step.

Mathematical Formulation:
Given training dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$ with N samples:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(x^{(i)}; \theta), y^{(i)}) \tag{12}$$

Gradient Computation:

$$\nabla_\theta \mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \ell(f(x^{(i)}; \theta), y^{(i)}) \tag{13}$$

Vanilla Gradient Descent (Batch Gradient Descent)

Definition: Compute the gradient using the entire training dataset in each update step.

Mathematical Formulation:
Given training dataset $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$ with N samples:

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f(x^{(i)}; \theta), y^{(i)}) \tag{12}$$

Gradient Computation:

$$\nabla_\theta \mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \ell(f(x^{(i)}; \theta), y^{(i)}) \tag{13}$$

Parameter Update:

$$\boxed{\theta_{t+1} = \theta_t - \eta \cdot \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \ell(f(x^{(i)}; \theta_t), y^{(i)})} \tag{14}$$

Vanilla Gradient Descent: Properties

Advantages:

- Guaranteed convergence to global minimum for convex functions
- Stable gradient estimates (low variance)
- Smooth convergence trajectory
- Deterministic updates

Vanilla Gradient Descent: Properties

Advantages:

- Guaranteed convergence to global minimum for convex functions
- Stable gradient estimates (low variance)
- Smooth convergence trajectory
- Deterministic updates

Disadvantages:

- Computationally expensive for large datasets
- Requires entire dataset to fit in memory
- Slow updates (one update per full pass through data)
- Cannot handle online learning scenarios

Vanilla Gradient Descent: Properties

Advantages:

- Guaranteed convergence to global minimum for convex functions
- Stable gradient estimates (low variance)
- Smooth convergence trajectory
- Deterministic updates

Disadvantages:

- Computationally expensive for large datasets
- Requires entire dataset to fit in memory
- Slow updates (one update per full pass through data)
- Cannot handle online learning scenarios

Computational Complexity:

- Time per update: $\mathcal{O}(N \cdot d)$ where d = number of parameters
- Memory: $\mathcal{O}(N)$ to store all training samples

Stochastic Gradient Descent (SGD)

Definition: Compute the gradient using one random sample at a time.

Stochastic Gradient Descent (SGD)

Definition: Compute the gradient using one random sample at a time.

Mathematical Formulation:
At iteration t, randomly select one sample $(x^{(i)}, y^{(i)})$ from $\mathcal{D}$:

$$\ell^{(i)}(\theta) = \ell(f(x^{(i)}; \theta), y^{(i)}) \tag{15}$$

Stochastic Gradient Descent (SGD)

Definition: Compute the gradient using one random sample at a time.

Mathematical Formulation:
At iteration t, randomly select one sample $(x^{(i)}, y^{(i)})$ from $\mathcal{D}$:

$$\ell^{(i)}(\theta) = \ell(f(x^{(i)}; \theta), y^{(i)}) \tag{15}$$

Gradient Computation:

$$\nabla_\theta \ell^{(i)}(\theta) = \nabla_\theta \ell(f(x^{(i)}; \theta), y^{(i)}) \tag{16}$$

Stochastic Gradient Descent (SGD)

Definition: Compute the gradient using one random sample at a time.

Mathematical Formulation:
At iteration t, randomly select one sample $(x^{(i)}, y^{(i)})$ from $\mathcal{D}$:

$$\ell^{(i)}(\theta) = \ell(f(x^{(i)}; \theta), y^{(i)}) \tag{15}$$

Gradient Computation:

$$\nabla_\theta \ell^{(i)}(\theta) = \nabla_\theta \ell(f(x^{(i)}; \theta), y^{(i)}) \tag{16}$$

Parameter Update:

$$\boxed{\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta \ell(f(x^{(i)}; \theta_t), y^{(i)})} \tag{17}$$

Stochastic Gradient Descent (SGD)

Definition: Compute the gradient using one random sample at a time.

Mathematical Formulation:
At iteration t, randomly select one sample $(x^{(i)}, y^{(i)})$ from $\mathcal{D}$:

$$\ell^{(i)}(\theta) = \ell(f(x^{(i)}; \theta), y^{(i)}) \tag{15}$$

Gradient Computation:

$$\nabla_\theta \ell^{(i)}(\theta) = \nabla_\theta \ell(f(x^{(i)}; \theta), y^{(i)}) \tag{16}$$

Parameter Update:

$$\boxed{\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta \ell(f(x^{(i)}; \theta_t), y^{(i)})} \tag{17}$$

Key Property: The gradient from a single sample is an unbiased estimator of the true gradient:

$$\mathbb{E}_{i \sim \text{Uniform}(1,N)}[\nabla_\theta \ell^{(i)}(\theta)] = \nabla_\theta \mathcal{L}(\theta) \tag{18}$$

Stochastic Gradient Descent: Properties

Advantages:

- Fast updates - one update per sample
- Low memory requirements
- Can escape shallow local minima due to noise
- Suitable for online learning
- Can handle large datasets that don't fit in memory

Stochastic Gradient Descent: Properties

Advantages:

- Fast updates - one update per sample
- Low memory requirements
- Can escape shallow local minima due to noise
- Suitable for online learning
- Can handle large datasets that don't fit in memory

Disadvantages:

- High variance in gradient estimates
- Noisy convergence trajectory
- May not converge exactly to minimum (oscillates around it)
- Requires careful learning rate tuning

Stochastic Gradient Descent: Properties

Advantages:

- Fast updates - one update per sample
- Low memory requirements
- Can escape shallow local minima due to noise
- Suitable for online learning
- Can handle large datasets that don't fit in memory

Disadvantages:

- High variance in gradient estimates
- Noisy convergence trajectory
- May not converge exactly to minimum (oscillates around it)
- Requires careful learning rate tuning

Computational Complexity:

- Time per update: $\mathcal{O}(d)$ where d = number of parameters
- Memory: $\mathcal{O}(1)$ only one sample at a time
- Total updates per epoch: N (one per sample)

Mini-batch Gradient Descent

Definition: Compute the gradient using a small batch of samples (compromise between Batch GD and SGD).

Mini-batch Gradient Descent

Definition: Compute the gradient using a small batch of samples (compromise between Batch GD and SGD).

Mathematical Formulation:

At iteration t, randomly select a mini-batch $\mathcal{B}_t \subset \mathcal{D}$ with $|\mathcal{B}_t| = b$ samples:

$$\mathcal{B}_t = \{(x^{(i)}, y^{(i)})\}_{i \in I_t}, \quad |I_t| = b \tag{19}$$

Mini-batch Gradient Descent

Definition: Compute the gradient using a small batch of samples (compromise between Batch GD and SGD).

Mathematical Formulation:

At iteration t, randomly select a mini-batch $\mathcal{B}_t \subset \mathcal{D}$ with $|\mathcal{B}_t| = b$ samples:

$$\mathcal{B}_t = \{(x^{(i)}, y^{(i)})\}_{i \in I_t}, \quad |I_t| = b \tag{19}$$

Gradient Computation:

$$\nabla_\theta \mathcal{L}_{\mathcal{B}_t}(\theta) = \frac{1}{b} \sum_{i \in I_t} \nabla_\theta \ell(f(x^{(i)}; \theta), y^{(i)}) \tag{20}$$

Mini-batch Gradient Descent

Definition: Compute the gradient using a small batch of samples (compromise between Batch GD and SGD).

Mathematical Formulation:

At iteration t, randomly select a mini-batch $\mathcal{B}_t \subset \mathcal{D}$ with $|\mathcal{B}_t| = b$ samples:

$$\mathcal{B}_t = \{(x^{(i)}, y^{(i)})\}_{i \in I_t}, \quad |I_t| = b \tag{19}$$

Gradient Computation:

$$\nabla_\theta \mathcal{L}_{\mathcal{B}_t}(\theta) = \frac{1}{b} \sum_{i \in I_t} \nabla_\theta \ell(f(x^{(i)}; \theta), y^{(i)}) \tag{20}$$

Parameter Update:

$$\boxed{\theta_{t+1} = \theta_t - \eta \cdot \frac{1}{b} \sum_{i \in I_t} \nabla_\theta \ell(f(x^{(i)}; \theta_t), y^{(i)})} \tag{21}$$

Common batch sizes: $b \in \{16, 32, 64, 128, 256, 512\}$

Mini-batch Gradient Descent: Properties

Advantages:

- Best of both worlds: balance between stability and speed
- Reduced variance compared to SGD
- Efficient use of vectorized operations (GPU acceleration)
- More stable convergence than SGD
- Faster than Batch GD

Mini-batch Gradient Descent: Properties

Advantages:

- Best of both worlds: balance between stability and speed
- Reduced variance compared to SGD
- Efficient use of vectorized operations (GPU acceleration)
- More stable convergence than SGD
- Faster than Batch GD

Disadvantages:

- Requires tuning of batch size hyperparameter
- Still has some noise (though less than SGD)

Mini-batch Gradient Descent: Properties

Advantages:

- Best of both worlds: balance between stability and speed
- Reduced variance compared to SGD
- Efficient use of vectorized operations (GPU acceleration)
- More stable convergence than SGD
- Faster than Batch GD

Disadvantages:

- Requires tuning of batch size hyperparameter
- Still has some noise (though less than SGD)

Computational Complexity:

- Time per update: $\mathcal{O}(b \cdot d)$ where b = batch size, d = parameters
- Memory: $\mathcal{O}(b)$ to store mini-batch
- Updates per epoch: $\lceil N/b \rceil$

Comparison: Gradient Descent Variants

Property	Batch GD	SGD	Mini-batch GD
Samples per update	N	1	b
Updates per epoch	1	N	$\lceil N/b \rceil$
Gradient variance	Low	High	Medium
Convergence speed	Slow	Fast	Fast
Memory requirement	High	Low	Medium
Stability	High	Low	Medium
GPU efficiency	Low	Low	High

Gradient Estimate Variance:

$$\text{Var}[\nabla_\theta \mathcal{L}] = \begin{cases} 0 & \text{Batch GD} \\ \sigma^2 & \text{SGD} \\ \sigma^2/b & \text{Mini-batch GD} \end{cases} \tag{22}$$

where σ^2 is the variance of individual sample gradients.

Key Terminologies

1. **Iteration:**
   - One forward pass + backward pass + parameter update
   - One step of gradient descent

Key Terminologies

1. **Iteration:**
   - One forward pass + backward pass + parameter update
   - One step of gradient descent
2. **Epoch:**
   - One complete pass through the entire training dataset
   - All N training samples have been seen once by the network

Key Terminologies

1. **Iteration:**
   - One forward pass + backward pass + parameter update
   - One step of gradient descent
2. **Epoch:**
   - One complete pass through the entire training dataset
   - All N training samples have been seen once by the network
3. **Batch Size (b):**
   - Number of samples processed before one parameter update
   - Determines memory usage and gradient variance

Key Terminologies

1. **Iteration:**
   - One forward pass + backward pass + parameter update
   - One step of gradient descent
2. **Epoch:**
   - One complete pass through the entire training dataset
   - All N training samples have been seen once by the network
3. **Batch Size (b):**
   - Number of samples processed before one parameter update
   - Determines memory usage and gradient variance
4. **When does the network learn?**
   - The network learns (parameters update) at each iteration
   - Learning = parameter update via gradient descent

Key Terminologies: Iterations per Epoch

Number of iterations in one epoch:

Let N = total number of training samples

1. **Batch Gradient Descent:**

$$\text{Iterations per epoch} = 1 \tag{23}$$

   Uses all N samples in one iteration

2. **Stochastic Gradient Descent:**

$$\text{Iterations per epoch} = N \tag{24}$$

   Uses 1 sample per iteration, needs N iterations to see all data

3. **Mini-batch Gradient Descent:**

$$\text{Iterations per epoch} = \left\lceil \frac{N}{b} \right\rceil \tag{25}$$

   Uses b samples per iteration, needs $\lceil N/b \rceil$ iterations

Key Terminologies: Concrete Example

Example: $N = 1000$ training samples

Algorithm	**Batch Size**	**Iterations/Epoch**	**Updates/Epoch**
Batch GD	$b = 1000$	1	1
Mini-batch GD	$b = 100$	10	10
Mini-batch GD	$b = 50$	20	20
Mini-batch GD	$b = 32$	32	32
SGD	$b = 1$	1000	1000

Key Observation:

- More iterations per epoch $\Rightarrow$ more frequent updates
- SGD: 1000 updates per epoch vs Batch GD: 1 update per epoch
- Mini-batch strikes a balance

Refined Learning Algorithm: Batch Gradient Descent

Batch Gradient Descent Algorithm

Input: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, learning rate η, epochs E

Initialize: θ_0 randomly

For epoch $e = 1$ to E:

// One iteration per epoch

1. **Forward Pass:** Compute $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ for all $i = 1, \ldots, N$
2. **Compute Loss:** $\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(\hat{y}^{(i)}, y^{(i)})$
3. **Backward Pass:** $\nabla_\theta \mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \nabla_\theta \ell(\hat{y}^{(i)}, y^{(i)})$
4. **Update:** $\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}(\theta)$
5. **Evaluate:** Compute $\mathcal{L}_{\text{val}}$ on validation set

End For

Return: θ^*

Refined Learning Algorithm: Stochastic Gradient Descent

Stochastic Gradient Descent Algorithm

Input: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, learning rate η, epochs E

Initialize: θ_0 randomly

For epoch $e = 1$ to E:

Randomly shuffle $\mathcal{D}$

For $i = 1$ to N: // N iterations per epoch

1. **Forward Pass:** $\hat{y}^{(i)} = f(x^{(i)}; \theta)$
2. **Compute Loss:** $\ell^{(i)} = \ell(\hat{y}^{(i)}, y^{(i)})$
3. **Backward Pass:** $\nabla_\theta \ell^{(i)} = \nabla_\theta \ell(\hat{y}^{(i)}, y^{(i)})$
4. **Update:** $\theta \leftarrow \theta - \eta \cdot \nabla_\theta \ell^{(i)}$

End For

5. **Evaluate:** Compute $\mathcal{L}_{\text{val}}$ on validation set

End For

Return: θ^*

Refined Learning Algorithm: Mini-batch Gradient Descent

Mini-batch Gradient Descent Algorithm

Input: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, learning rate η, batch size b, epochs E

Initialize: θ_0 randomly
For epoch $e = 1$ to E:
Randomly shuffle
For $t = 1$ to $\lceil N/b \rceil$: // $\lceil N/b \rceil$ iterations per epoch
$\mathcal{B}_t = \{(x^{(i)}, y^{(i)})\}_{i=(t-1)b+1}^{\min(tb,N)}$ // Create mini-batch
1. **Forward Pass:** $\hat{y}^{(i)} = f(x^{(i)}; \theta)$ for all $(x^{(i)}, y^{(i)}) \in \mathcal{B}_t$
2. **Compute Loss:** $\mathcal{L}_{\mathcal{B}_t} = \frac{1}{|\mathcal{B}_t|} \sum_{(x,y)\in\mathcal{B}_t} \ell(\hat{y}, y)$
3. **Backward Pass:** $\nabla_\theta \mathcal{L}_{\mathcal{B}_t} = \frac{1}{|\mathcal{B}_t|} \sum_{(x,y)\in\mathcal{B}_t} \nabla_\theta \ell(\hat{y}, y)$
4. **Update:** $\theta \leftarrow \theta - \eta \cdot \nabla_\theta \mathcal{L}_{\mathcal{B}_t}$

End For
5. **Evaluate:** Compute $\mathcal{L}_{\text{val}}$ on validation set
End For
Return: θ^*

Momentum based Gradient Descent

Limitations of Vanilla Gradient Descent

Problem 1: Slow Convergence in Ravines

- Ravines: areas where surface curves more steeply in one dimension than another
- Vanilla GD oscillates across narrow ravine while making slow progress along the bottom

Limitations of Vanilla Gradient Descent

Problem 1: Slow Convergence in Ravines

- Ravines: areas where surface curves more steeply in one dimension than another
- Vanilla GD oscillates across narrow ravine while making slow progress along the bottom

Problem 2: Oscillations Around Minima

- GD can oscillate when gradient directions change rapidly
- Wastes computation on back-and-forth movement

Limitations of Vanilla Gradient Descent

Problem 1: Slow Convergence in Ravines

- Ravines: areas where surface curves more steeply in one dimension than another
- Vanilla GD oscillates across narrow ravine while making slow progress along the bottom

Problem 2: Oscillations Around Minima

- GD can oscillate when gradient directions change rapidly
- Wastes computation on back-and-forth movement

Problem 3: No Acceleration Mechanism

- Each update is independent - no "memory" of previous directions
- Cannot speed up in directions of consistent gradient
- Update magnitude solely determined by current gradient: $\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t)$

Limitations of Vanilla Gradient Descent

Problem 1: Slow Convergence in Ravines

- Ravines: areas where surface curves more steeply in one dimension than another
- Vanilla GD oscillates across narrow ravine while making slow progress along the bottom

Problem 2: Oscillations Around Minima

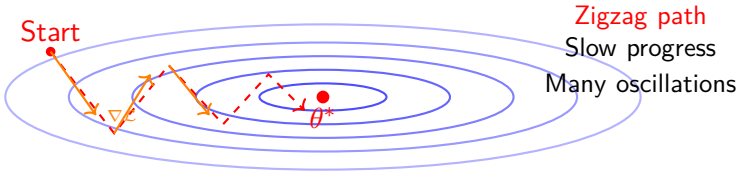
- GD can oscillate when gradient directions change rapidly
- Wastes computation on back-and-forth movement

Problem 3: No Acceleration Mechanism

- Each update is independent - no "memory" of previous directions
- Cannot speed up in directions of consistent gradient
- Update magnitude solely determined by current gradient: $\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t)$

Key Limitation: No way to accumulate velocity in favorable directions or dampen oscillations in unfavorable ones!

Visualizing the Problem: Vanilla GD in Ravines

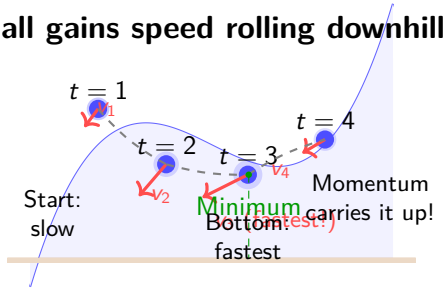


Observation: Vanilla GD makes slow progress toward optimum, wasting steps on oscillations perpendicular to the optimal direction.

The General Intuition behind Momentum

Physical Analogy: Ball rolling down a hill

- Ball accumulates velocity as it rolls downhill
- Momentum helps it roll through small bumps and saddle points
- Ball doesn't stop immediately when gradient becomes zero



Key Idea: Maintain a velocity vector that accumulates gradients over time, allowing faster movement in consistent directions and dampening oscillations.

The General Intuition behind Momentum

Mathematical Intuition:

Vanilla GD:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t) \tag{26}$$

- Update depends only on current gradient
- No memory of past directions

The General Intuition behind Momentum

Mathematical Intuition:

Vanilla GD:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t) \tag{26}$$

- Update depends only on current gradient
- No memory of past directions

Momentum-based GD:

$$v_{t+1} = \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta_t) \tag{27}$$

$$\theta_{t+1} = \theta_t - v_{t+1} \tag{28}$$

- Update depends on accumulated velocity v_t
- γ (typically 0.9) controls how much past gradients influence current update
- Acts like a moving average of gradients

Momentum: Dampening Oscillations

Consider oscillating gradients:

Suppose gradients alternate: $g_1 = +5$, $g_2 = -4$, $g_3 = +5$, $g_4 = -4$, ...

Momentum: Dampening Oscillations

Consider oscillating gradients:

Suppose gradients alternate: $g_1 = +5$, $g_2 = -4$, $g_3 = +5$, $g_4 = -4$, ...

Vanilla GD: (with $\eta = 0.1$)

$$\Delta\theta_1 = -0.5, \quad \Delta\theta_2 = +0.4, \quad \Delta\theta_3 = -0.5, \quad \Delta\theta_4 = +0.4 \tag{29}$$

Large oscillations persist!

Momentum: Dampening Oscillations

Consider oscillating gradients:

Suppose gradients alternate: $g_1 = +5$, $g_2 = -4$, $g_3 = +5$, $g_4 = -4$, ...

Vanilla GD: (with $\eta = 0.1$)

$$\Delta\theta_1 = -0.5, \quad \Delta\theta_2 = +0.4, \quad \Delta\theta_3 = -0.5, \quad \Delta\theta_4 = +0.4 \tag{29}$$

Large oscillations persist!

Momentum GD: (with $\eta = 0.1$, $\gamma = 0.9$)

$$v_1 = 0.9(0) + 0.1(5) = 0.5 \tag{30}$$
$$v_2 = 0.9(0.5) + 0.1(-4) = 0.45 - 0.4 = 0.05 \tag{31}$$
$$v_3 = 0.9(0.05) + 0.1(5) = 0.045 + 0.5 = 0.545 \tag{32}$$
$$v_4 = 0.9(0.545) + 0.1(-4) = 0.49 - 0.4 = 0.09 \tag{33}$$

Oscillations dampened! Velocity stabilizes around 0.

Update Rule for Momentum-based Gradient Descent

Momentum Update Rule:

Momentum Gradient Descent

Initialize velocity: $v_0 = 0$

At each iteration t:

$$v_{t+1} = \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta_t) \tag{34}$$

$$\theta_{t+1} = \theta_t - v_{t+1} \tag{35}$$

Update Rule for Momentum-based Gradient Descent

Momentum Update Rule:

Momentum Gradient Descent

Initialize velocity: $v_0 = 0$

At each iteration t:

$$v_{t+1} = \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta_t) \tag{34}$$

$$\theta_{t+1} = \theta_t - v_{t+1} \tag{35}$$

Parameters:

- $\gamma \in [0, 1)$: Momentum coefficient (typically 0.9 or 0.99)
- $\eta > 0$: Learning rate
- v_t: Velocity (accumulated gradient)

Update Rule for Momentum-based Gradient Descent

Momentum Update Rule:

Momentum Gradient Descent

Initialize velocity: $v_0 = 0$

At each iteration t:

$$v_{t+1} = \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta_t) \tag{34}$$

$$\theta_{t+1} = \theta_t - v_{t+1} \tag{35}$$

Parameters:

- $\gamma \in [0, 1)$: Momentum coefficient (typically 0.9 or 0.99)
- $\eta > 0$: Learning rate
- v_t: Velocity (accumulated gradient)

Special Cases:

- $\gamma = 0$: Reduces to vanilla gradient descent
- $\gamma \to 1$: Maximum momentum, very smooth trajectory

Understanding Momentum: Exponential Moving Average

Claim: Velocity v_t is an exponential moving average (EMA) of all past gradients.

Understanding Momentum: Exponential Moving Average

Claim: Velocity v_t is an exponential moving average (EMA) of all past gradients.

Proof: Expand the recursion $v_{t+1} = \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta_t)$
Starting from $v_0 = 0$:

$$v_1 = \gamma v_0 + \eta g_0 = \eta g_0 \tag{36}$$

$$v_2 = \gamma v_1 + \eta g_1 = \gamma \eta g_0 + \eta g_1 \tag{37}$$

$$\vdots \tag{38}$$

$$v_t = \eta \sum_{i=0}^{t-1} \gamma^{t-1-i} g_i \tag{39}$$

where $g_i = \nabla_\theta \mathcal{L}(\theta_i)$
General Form:

$$\boxed{v_t = \eta \sum_{i=0}^{t-1} \gamma^{t-1-i} \nabla_\theta \mathcal{L}(\theta_i)} \tag{40}$$

Understanding Momentum: Exponential Moving Average

Claim: Velocity v_t is an exponential moving average (EMA) of all past gradients.

Proof: Expand the recursion $v_{t+1} = \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta_t)$
Starting from $v_0 = 0$:

$$v_1 = \gamma v_0 + \eta g_0 = \eta g_0 \tag{36}$$

$$\vdots \tag{38}$$

$$v_t = \eta \sum_{i=0}^{t-1} \gamma^{t-1-i} g_i \tag{39}$$

where $g_i = \nabla_\theta \mathcal{L}(\theta_i)$
General Form:

$$\boxed{v_t = \eta \sum_{i=0}^{t-1} \gamma^{t-1-i} \nabla_\theta \mathcal{L}(\theta_i)} \tag{40}$$

Understanding Momentum: Exponential Moving Average

Claim: Velocity v_t is an exponential moving average (EMA) of all past gradients.

Proof: Expand the recursion $v_{t+1} = \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta_t)$
Starting from $v_0 = 0$:

$$v_1 = \gamma v_0 + \eta g_0 = \eta g_0 \tag{36}$$

$$v_t = \eta \sum_{i=0}^{t-1} \gamma^{t-1-i} g_i \tag{39}$$

where $g_i = \nabla_\theta \mathcal{L}(\theta_i)$
General Form:

$$\boxed{v_t = \eta \sum_{i=0}^{t-1} \gamma^{t-1-i} \nabla_\theta \mathcal{L}(\theta_i)} \tag{40}$$

Understanding Momentum: Exponential Moving Average

Claim: Velocity v_t is an exponential moving average (EMA) of all past gradients.

Proof: Expand the recursion $v_{t+1} = \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta_t)$
Starting from $v_0 = 0$:

$$v_1 = \gamma v_0 + \eta g_0 = \eta g_0 \tag{36}$$

$$v_2 = \gamma v_1 + \eta g_1 = \gamma \eta g_0 + \eta g_1 \tag{37}$$

$$\vdots \tag{38}$$

$$v_t = \eta \sum_{i=0}^{t-1} \gamma^{t-1-i} g_i \tag{39}$$

where $g_i = \nabla_\theta \mathcal{L}(\theta_i)$

Understanding Momentum: Exponential Moving Average

Claim: Velocity v_t is an exponential moving average (EMA) of all past gradients.

Proof: Expand the recursion $v_{t+1} = \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta_t)$
Starting from $v_0 = 0$:

$$v_1 = \gamma v_0 + \eta g_0 = \eta g_0 \tag{36}$$

$$v_2 = \gamma v_1 + \eta g_1 = \gamma \eta g_0 + \eta g_1 \tag{37}$$

$$\vdots \tag{38}$$

$$v_t = \eta \sum_{i=0}^{t-1} \gamma^{t-1-i} g_i \tag{39}$$

where $g_i = \nabla_\theta \mathcal{L}(\theta_i)$

General Form:

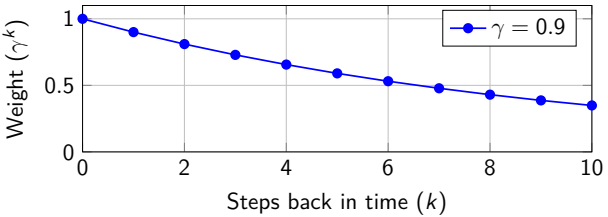
$$\boxed{v_t = \eta \sum_{i=0}^{t-1} \gamma^{t-1-i} \nabla_\theta \mathcal{L}(\theta_i)} \tag{40}$$

Understanding Momentum: Weighted Sum of Gradients

$$v_t = \eta \left(g_{t-1} + \gamma g_{t-2} + \gamma^2 g_{t-3} + \gamma^3 g_{t-4} + \cdots \right) \tag{41}$$

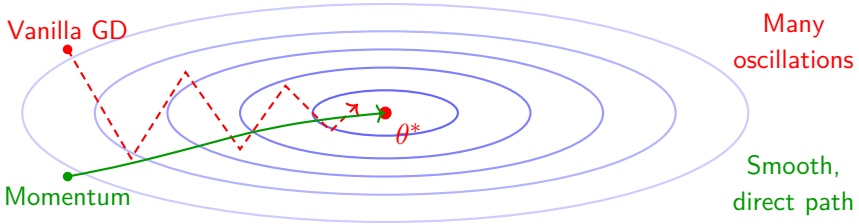
Interpretation:

- Recent gradients have weight $\gamma^0 = 1$ (most influence)
- Gradients decay exponentially with age: $\gamma^1, \gamma^2, \gamma^3, \ldots$
- With $\gamma = 0.9$:
  - g_{t-1}: weight $= 1.0$
  - g_{t-2}: weight $= 0.9$
  - g_{t-3}: weight $= 0.81$
  - g_{t-10}: weight $= 0.35$



Conclusion: Momentum smooths out gradient noise by averaging over history!

Momentum vs Vanilla GD: Visual Comparison



Observation:

- Vanilla GD: Zigzag pattern, slow convergence
- Momentum GD: Smooth trajectory, faster convergence

Momentum-based Gradient Descent: Complete Algorithm

Momentum Gradient Descent

Input: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, learning rate η, momentum γ, epochs E, batch size b

Initialize: θ_0 randomly, $v_0 = 0$

For epoch $e = 1$ to E:

Shuffle $\mathcal{D}$

For each mini-batch $\mathcal{B}_t$:

1. **Forward:** $\hat{y} = f(x; \theta)$ for $(x, y) \in \mathcal{B}_t$
2. **Loss:** $\mathcal{L}_{\mathcal{B}_t} = \frac{1}{|\mathcal{B}_t|} \sum_{(x,y) \in \mathcal{B}_t} \ell(\hat{y}, y)$
3. **Backward:** $g_t = \nabla_\theta \mathcal{L}_{\mathcal{B}_t}$
4. **Update velocity:** $v_{t+1} = \gamma v_t + \eta g_t$
5. **Update parameters:** $\theta \leftarrow \theta - v_{t+1}$

End For

Evaluate on validation set

End For

Return: θ^*

Practical Considerations

Choosing Momentum Coefficient γ:

- $\gamma = 0.9$: Standard choice, good for most problems
- $\gamma = 0.95$ or 0.99: More aggressive, useful for very noisy gradients
- $\gamma < 0.9$: Less common, reduces momentum effect

Learning Rate with Momentum:

- May need to reduce learning rate compared to vanilla GD
- Momentum amplifies the effective step size
- Rule of thumb: If using η with vanilla GD, try $\eta/2$ or $\eta/3$ with momentum

When to Use Momentum:

- Always recommended for deep neural networks
- Particularly effective for:
  - High-dimensional optimization
  - Noisy gradients (mini-batch SGD)
  - Ill-conditioned loss surfaces (ravines, plateaus)

Summary: Momentum-based Gradient Descent

Key Ideas

1. **Accumulates velocity**: $v_{t+1} = \gamma v_t + \eta g_t$
2. **Exponential moving average** of gradients
3. **Dampens oscillations**, accelerates in consistent directions
4. Converges faster than vanilla GD
5. **Note:** Both Stochastic and Mini-batch variants of momentum GD exist

Mathematical Insight:

$$v_t = \eta \sum_{i=0}^{t-1} \gamma^{t-1-i} g_i \quad \text{(weighted sum of all past gradients)} \tag{42}$$

Update Rule:

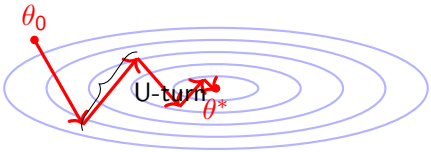
$$v_{t+1} = \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta_t) \tag{43}$$

$$\theta_{t+1} = \theta_t - v_{t+1} \tag{44}$$

Nesterov Accelerated Gradient Descent

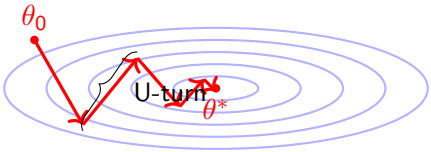
Limitations of Momentum-based GD

Problem: Excessive Overshooting



Limitations of Momentum-based GD

Problem: Excessive Overshooting



Issue: Momentum builds up velocity but cannot anticipate the turn, leading to:

- Excessive overshooting past the minimum
- Unnecessary oscillations (U-turns)
- Slower convergence near the minimum

Limitations of Momentum-based GD (continued)

Mathematical Analysis of the Problem:
At iteration t, momentum GD computes:

$$v_{t+1} = \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta_t) \tag{45}$$

Limitations of Momentum-based GD (continued)

Mathematical Analysis of the Problem:
At iteration t, momentum GD computes:

$$v_{t+1} = \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta_t) \tag{45}$$

The Issue:

- Gradient $\nabla_\theta \mathcal{L}(\theta_t)$ is evaluated at *current* position θ_t
- But the update moves in direction v_{t+1} which includes momentum from v_t
- No "look-ahead" to see where we're actually going

Limitations of Momentum-based GD (continued)

Mathematical Analysis of the Problem:
At iteration t, momentum GD computes:

$$v_{t+1} = \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta_t) \tag{45}$$

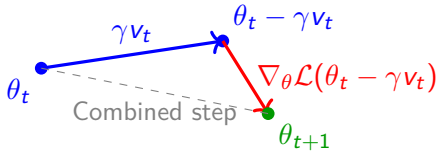
The Issue:

- Gradient $\nabla_\theta \mathcal{L}(\theta_t)$ is evaluated at *current* position θ_t
- But the update moves in direction v_{t+1} which includes momentum from v_t
- No "look-ahead" to see where we're actually going

Consequence: When approaching minimum from one direction with high velocity, the algorithm overshoots and must correct course in the next iteration.

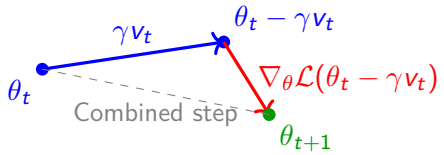
Intuition behind NAG

Key Idea: "Look ahead" before computing the gradient



Intuition behind NAG

Key Idea: "Look ahead" before computing the gradient

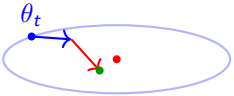


NAG Philosophy:

1. First, take a step in the direction of accumulated momentum: $\theta_t - \gamma v_t$
2. Then, compute gradient at this "look-ahead" position
3. Make a correction based on this anticipated gradient

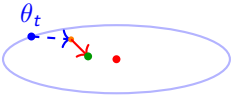
Intuition: Momentum GD vs NAG

Momentum GD



Gradient at θ_t
(blind to future)

Nesterov AG

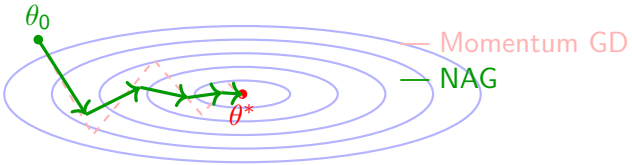


Gradient at look-ahead
(anticipates future)

Result: NAG corrects the trajectory *before* overshooting, leading to:

- Fewer oscillations
- Smoother convergence
- Better performance near the minimum

NAG: Improved Convergence Path



Observation: NAG path shows reduced oscillations and more direct convergence.

Update Rule for NAG

Nesterov Accelerated Gradient (Original Formulation):

$$v_{t+1} = \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta_t - \gamma v_t) \tag{46}$$

$$\theta_{t+1} = \theta_t - v_{t+1} \tag{47}$$

where:

- $\theta_t - \gamma v_t$ is the "look-ahead" position
- $\gamma \in [0, 1)$ is the momentum coefficient (typically 0.9)
- $\eta > 0$ is the learning rate

Update Rule for NAG

Nesterov Accelerated Gradient (Original Formulation):

$$v_{t+1} = \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta_t - \gamma v_t) \tag{46}$$
$$\theta_{t+1} = \theta_t - v_{t+1} \tag{47}$$

where:

- $\theta_t - \gamma v_t$ is the "look-ahead" position
- $\gamma \in [0, 1)$ is the momentum coefficient (typically 0.9)
- $\eta > 0$ is the learning rate

Key Difference from Momentum GD:

Momentum:	$\nabla_\theta \mathcal{L}(\theta_t)$	(gradient at current position)
NAG:	$\nabla_\theta \mathcal{L}(\theta_t - \gamma v_t)$	(gradient at look-ahead)

Mathematical Comparison

Method	Update Rule
Vanilla GD	$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t)$
Momentum GD	$v_{t+1} = \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta_t)$ $\theta_{t+1} = \theta_t - v_{t+1}$
NAG	$v_{t+1} = \gamma v_t + \eta \nabla_\theta \mathcal{L}(\theta_t - \gamma v_t)$ $\theta_{t+1} = \theta_t - v_{t+1}$

Computational Cost:

- Same as momentum GD: one gradient evaluation per iteration
- Slight overhead: computing $\theta_t - \gamma v_t$ before gradient
- Negligible compared to gradient computation cost

Advantages of NAG

Compared to Vanilla GD:

- **Faster convergence:** $O(1/t^2)$ vs $O(1/t)$
- **Better handling of ill-conditioned problems:** Accumulates velocity in consistent directions
- **Reduced sensitivity to learning rate:** Momentum provides stability

Compared to Momentum GD:

- **Reduced oscillations:** Look-ahead mechanism anticipates turns
- **Better convergence near minimum:** Less overshooting
- **Theoretically optimal:** Proven optimal convergence rate for first-order methods
- **More responsive:** Adapts to changing gradient directions faster

Advantages of NAG (continued)

Practical Benefits:

- Same computational cost as momentum GD
- Simple to implement (one-line change from momentum)
- Works well in practice for deep learning
- Particularly effective for:
  - Problems with elongated valleys
  - High condition number optimization landscapes
  - Training deep neural networks

Empirical Observation:

In machine learning applications, NAG often converges in 30-50% fewer iterations than standard momentum GD while using the same hyperparameters.

Algorithm: Nesterov Accelerated Gradient (NAG)

Nesterov Accelerated Gradient

Input: $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{N}$, learning rate η, momentum γ, epochs E, batch size b

Initialize: θ_0 randomly, $v_0 = 0$

For epoch $e = 1$ to E:

Shuffle $\mathcal{D}$

For each mini-batch $\mathcal{B}_t$:

1. **Look-ahead:** $\tilde{\theta} = \theta - \gamma v_t$
2. **Forward:** $\hat{y} = f(x; \tilde{\theta})$ for $(x, y) \in \mathcal{B}_t$
3. **Loss:** $\mathcal{L}_{\mathcal{B}_t} = \frac{1}{|\mathcal{B}_t|} \sum_{(x,y)\in\mathcal{B}_t} \ell(\hat{y}, y)$
4. **Backward:** $g_t = \nabla_{\tilde{\theta}} \mathcal{L}_{\mathcal{B}_t}$
5. **Update velocity:** $v_{t+1} = \gamma v_t + \eta g_t$
6. **Update parameters:** $\theta \leftarrow \theta - v_{t+1}$

End For

Evaluate on validation set

End For

Practical Considerations

When to use NAG:

- Training deep neural networks
- Optimization problems with ravines or valleys
- When momentum GD shows oscillatory behavior
- When faster convergence is desired with minimal tuning

Practical Considerations

When to use NAG:

- Training deep neural networks
- Optimization problems with ravines or valleys
- When momentum GD shows oscillatory behavior
- When faster convergence is desired with minimal tuning

Hyperparameter tuning:

- Start with $\gamma = 0.9$, $\eta = 0.01$
- Increase γ (up to 0.99) for smoother optimization surfaces
- Decrease γ if oscillations persist
- Use learning rate schedules for best performance

Practical Considerations

When to use NAG:

- Training deep neural networks
- Optimization problems with ravines or valleys
- When momentum GD shows oscillatory behavior
- When faster convergence is desired with minimal tuning

Hyperparameter tuning:

- Start with $\gamma = 0.9$, $\eta = 0.01$
- Increase γ (up to 0.99) for smoother optimization surfaces
- Decrease γ if oscillations persist
- Use learning rate schedules for best performance

Limitation:

- Still requires careful learning rate selection
- May not adapt well to sparse gradients (use Adam/RMSprop instead)

Summary

Key Takeaways

1. NAG improves momentum GD by computing gradients at "look-ahead" positions
2. Achieves optimal $O(1/t^2)$ convergence for smooth convex functions
3. Reduces oscillations and overshooting near minima
4. Same computational cost as momentum GD
5. Widely used in deep learning and remains relevant today
6. **Note:** Both Stochastic and Mini-batch variants of momentum GD exist

The NAG Principle:

"Anticipate where you're going, then correct course accordingly"

The Learning Rate Dilemma

We have calculated the gradients ∇w using Backpropagation.

Standard Update Rule (SGD):

$$w_{new} = w_{old} - \eta \cdot \frac{\partial \mathcal{L}}{\partial w}$$

The Learning Rate Dilemma

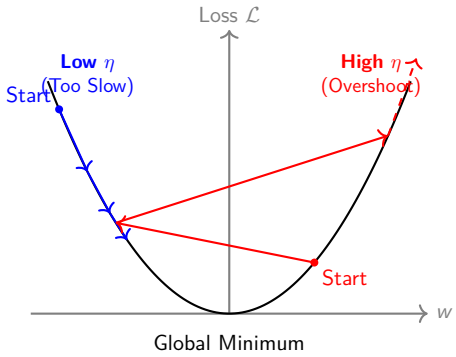
We have calculated the gradients ∇w using Backpropagation.

Standard Update Rule (SGD):

$$w_{new} = w_{old} - \eta \cdot \frac{\partial \mathcal{L}}{\partial w}$$

The Problem with Fixed η:

1. **High η:** The model overshoots the minimum and may diverge (explode).
2. **Low η:** The model learns painfully slowly and gets stuck in local minima.



Why Fixed Learning Rates Fail

1. One Size Does Not Fit All

- We apply the **same** scalar η to every single weight in the network.
- *Reality:* Some features are sparse (rare) and need large updates; others are dense and need small updates.

Why Fixed Learning Rates Fail

1. One Size Does Not Fit All

- We apply the **same** scalar η to every single weight in the network.
- *Reality:* Some features are sparse (rare) and need large updates; others are dense and need small updates.

2. The Landscape Changes

- The loss surface is not a perfect bowl. It has cliffs, valleys, and plateaus.
- A rate that works at the start (steep area) might be terrible at the end (flat area).

Next Steps: Adaptive Algorithms

We need algorithms that **adapt** the learning rate automatically for each parameter.

RMSProp Adjusts rate based on recent gradient magnitude.

Adam Combines both Momentum and RMSProp.

(Coming up in the next lecture...)