

Optimization in Neural Networks

Tianxiang (Adam) Gao

School of Computing
DePaul University

Outline

- 1 Calculus Review: Second Derivatives
- 2 Convergence Issues
- 3 Advanced Optimization Algorithm

Recap: Neural Network Training

We use a **training process** to iteratively update the parameters in MLPs:

- MLPs are **parameterized** function f_{θ} , where $\theta = \{\mathbf{W}^{\ell}, \mathbf{b}^{\ell}\}$
- **Universal Approximation Theorem (UAT)**: MLPs can approximate “any” function f^* arbitrarily accurate, provided sufficient parameters (and training samples).
- Given a **training set** $\{\mathbf{x}_i, \mathbf{y}_i\}_{i=1}^{\ell}$ and a **loss function** ℓ , the training problem is:

$$\min_{\theta} \quad \mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n \ell(f_{\theta}(\mathbf{x}_i), \mathbf{y}_i)$$

- This optimization problem can be solved using **gradient descent**, which gradually reduces the cost \mathcal{L} along the *steepest descent direction*:

$$\theta^+ = \theta - \eta \nabla \mathcal{L}(\theta)$$

where $\eta > 0$ is the **learning rate**.

- The gradients in MLPs can be computed using the **chain rule** backward from the total cost.

Recap: Neural Network Training

- Using the **computational graph**, the gradients can be computed through **backpropagation**:

- Forward Propagation (biases omitted): Start with $\mathbf{x}^0 = \mathbf{x}$

$$\mathbf{z}^\ell = \mathbf{W}^\ell \mathbf{x}^{\ell-1}, \quad \forall \ell \in \{0, 1, 2, \dots, L\}$$

$$\mathbf{x}^\ell = \phi(\mathbf{z}^\ell),$$

- Backward Propagation (biases omitted): Start with $d\mathbf{z}^L = (\mathbf{x}^L - \mathbf{y}) \odot \phi'(\mathbf{z}^L)$

$$d\mathbf{z}^\ell = \left[(\mathbf{W}^{\ell+1})^\top d\mathbf{z}^{\ell+1} \right] \odot \phi'(\mathbf{z}^\ell), \quad \forall \ell \in \{1, 2, \dots, L-1\}$$

$$d\mathbf{W}^\ell = d\mathbf{z}^\ell \mathbf{x}^{(\ell-1)\top}$$

- To enable training, we use the **sigmoid** activation instead of the step function, as the step function has a zero derivative almost everywhere.

•

Questions

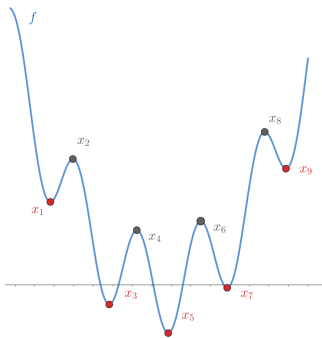
- What are other common activation functions?
- How do I select the learning rate, width, and depth of the network?
- Does gradient descent always converge? How can I speed up training?
- Does good training performance guarantee good test performance?

Outline

- 1 Calculus Review: Second Derivatives
- 2 Convergence Issues
- 3 Advanced Optimization Algorithm

Calculus Review: Extreme Values

Let $f(x)$ be a real-valued function, where $x \in \mathbb{R}$.



Local Min. x_1, x_3, x_5, x_7, x_9 ;

Local Max. x_2, x_4, x_6, x_8 ;

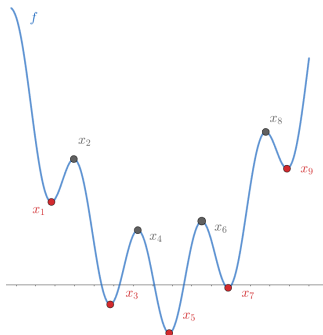
- The function f has an **local minimum** at point $x = a$ if $f(a) \leq f(x)$ when x is near a .
- The function f has an **local maximum** at point $x = a$ if $f(a) \geq f(x)$ when x is near a .
- The point a is a **global minimum** or **global maximum** if the above property holds for all x .
- **Fermat's Theorem:** If f has a local min or max at $x = a$, then $f'(a) = 0$, as $f'(a)$ points to the steepest *ascent* direction.
- A point $x = a$ is called **stationary** if $f'(a) = 0$.
- **Gradient descent** stops at *stationary points*:

$$\theta^+ = \theta - \eta \nabla_{\theta} \mathcal{L}(\theta).$$

Calculus Review: Curvature

Definition: The **second derivative** of a real-valued function $f(x)$ measure the rate of change of the first derivative $f'(x)$ at point x , e.g., the acceleration of an object's position w.r.t. time.

$$f''(a) \approx \frac{f'(x) - f'(a)}{x - a}$$



Concavity: the second derivative $f''(x)$ describes whether f is **concave up** or **concave down**

- If $f''(x) > 0$, then f is **concave up** at x .
- If $f''(x) < 0$, then f is **concave down** at x .

The Second Derivative Test:

- If $f'(a) = 0$ and $f''(a) \geq 0$, then a is a **local minimum**
- If $f'(a) = 0$ and $f''(a) \leq 0$, then a is a **local maximum**.

Conclusion

The goal of training in deep learning is to find a good **local minimum** that generalizes well.

Hessian Matrix

Let $f(x)$ be a **multivariate** real-valued function, where $x \in \mathbb{R}^n$.

- A point $x = a$ is called **stationary point** if $\nabla f(a) = \mathbf{0}$, i.e.,

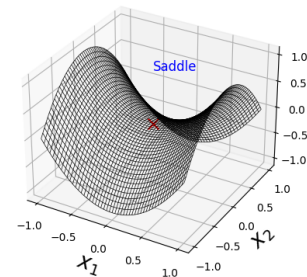
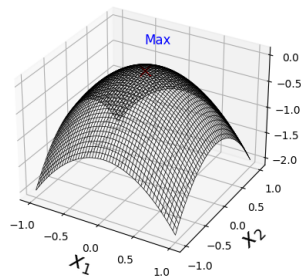
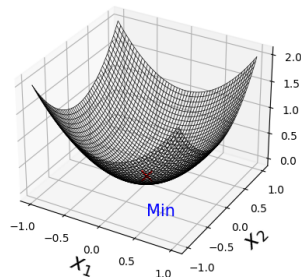
$$\nabla f(a) = \left[\frac{\partial f(a)}{\partial x_1} \quad \dots \quad \frac{\partial f(a)}{\partial x_n} \right]^\top = \mathbf{0}$$

- The **Hessian** matrix $H(w) \in \mathbb{R}^{n \times n}$ of f is the symmetric matrix of second-order partial derivatives:

$$\nabla^2 f(x) = H(x) = \left[\frac{\partial^2 f}{\partial x_i \partial x_j} \right]_{i,j \in [n]} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \dots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \dots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

- For the **second-order mixed partial derivative** $\frac{\partial^2 f}{\partial x \partial y}$ is the rate of change of $\frac{\partial f}{\partial x}$ w.r.t. y changes, holding x constant.

Significance of Hessian



Interpretation of the Hessian Matrix:

- The Hessian describes the **local curvature** of the function.
- **Positive** definite Hessian H implies a local minimum, *i.e.*, concave up in any direction.
- **Negative** definite Hessian implies a local maximum, *i.e.*, concave down in any direction.
- **Indefinite** Hessian implies a **saddle point**, *i.e.*, concave up in some directions and concave down in others.

Discussion Questions

Compute the gradients and Hessian of the following functions:

- $f(w) = \frac{1}{2}(xw - y)^2$
- $f(w) = \frac{1}{2}\|Xw - y\|^2$, where

$$w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}, \quad X = \begin{bmatrix} 3 & 1 \\ 1 & 1 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Hint: write $f(w) = f(w_1, w_2) = \frac{1}{2}(3w_1 - 1)^2 + \frac{1}{2}w_2^2$.

Instructions: Discuss these questions in small groups of 2-3 students.

Solutions to the Discussion Questions

Compute the gradients and Hessian of the following functions:

- $f(w) = \frac{1}{2}(xw - y)^2$, $f'(w) = x \cdot (xw - y)$, and $f''(w) = x^2$.
- $f(w) = \frac{1}{2}\|Xw - y\|^2$, where

$$w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}, \quad X = \begin{bmatrix} 3 & \\ & 1 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Hint: Write $f(w) = f(w_1, w_2) = \frac{1}{2}(3w_1 - 1)^2 + \frac{1}{2}w_2^2$. We have

$$\nabla f(w) = X^\top (Xw - y) = \begin{bmatrix} 3(3w_1 - 1) \\ w_2 \end{bmatrix} \quad \text{and} \quad H(w) = \begin{bmatrix} 9 & \\ & 1 \end{bmatrix},$$

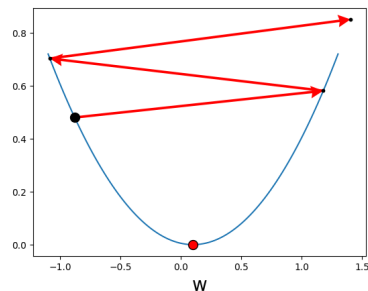
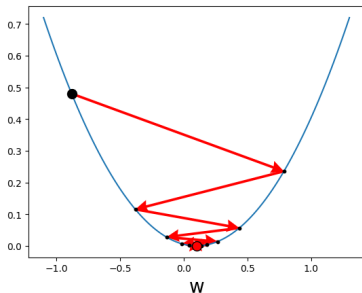
- Here 9 is the **largest eigenvalue** of H , 1 is the **smallest eigenvalue** of H , and their ratio is called **conditional number** $\kappa = 9$.

Outline

- ### 3 Advanced Optimization Algorithm

Learning Rate

Learning Rate



One-Dimensional Linear Regression

Consider a simple one-dimensional linear regression problem:

$$\min_w \mathcal{L}(w) = \ell(f_\theta(x), y) = \frac{1}{2}(wx - y)^2,$$

where $w, x, y \in \mathbb{R}$.

- The function $f_\theta(x) = wx$ is a perceptron with linear activation, without a bias term.
- With gradient $\nabla \mathcal{L}(w) = x(wx - y)$, the gradient descent update is:

$$w^+ = w - \eta \cdot x(wx - y),$$

where $\eta > 0$ is the learning rate.

- To find the **stationary point**:

$$\nabla \mathcal{L}(w) = 0 \implies x(wx - y) = 0 \implies w^* = \frac{y}{x}$$

- Second derivative test:

$$\nabla^2 \mathcal{L}(w^*) = x^2 > 0,$$

i.e., w^* is a local minimum (and also a global minimum since \mathcal{L} is concave up everywhere).

Residual Dynamics in Gradient Descent

The update rule for Gradient Descent on LSR is:

$$w^{k+1} = w^k - \eta \cdot x(w^k x - y).$$

From this, we can derive a recurrence relation for the residual or error ε^{k+1} :

$$\begin{aligned}\varepsilon^{k+1} &= w^{k+1}x - y \\ &= \left[w^k - \eta \cdot x(w^k x - y) \right] x - y \\ &= (1 - \eta x^2) \cdot \varepsilon^k \\ &= a \cdot \varepsilon^k,\end{aligned}$$

where $\boxed{a := 1 - \eta x^2}$ and $\varepsilon^k = w^k x - y$ is the error at step k . Repeating this relation, we obtain:

$$\boxed{\varepsilon^{k+1} = a^{k+1} \varepsilon^0},$$

where $\varepsilon^0 = w^0 x - y$ is the initial error.

Recursive Formula for Gradient Descent on LSR

- The update rule for Gradient Descent applied to linear regression is:

$$w^{k+1} = w^k - \eta \cdot x(w^k x - y) = (1 - \eta x^2)w^k + \eta xy := aw^k + b,$$

where $a := 1 - \eta x^2$ and $b := \eta xy$.

- Using this recurrence relation, w^{k+1} can be expanded as:

$$\begin{aligned} w^{k+1} &= aw^k + b \\ &= a(aw^{k-1} + b) + b \\ &= a^2w^{k-1} + ab + b \\ &= a^3w^{k-2} + a^2b + ab + b \\ &= a^{k+1}w^0 + b(a^k + a^{k-1} + \dots + a + 1) \\ &= a^{k+1}w^0 + b \frac{1 - a^{k+1}}{1 - a} \\ &= a^{k+1}(w^0 - w^*) + w^*, \end{aligned}$$

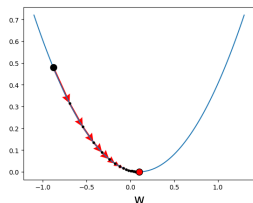
where we use the geometric series $\sum_{i=0}^k a^i = \frac{1 - a^{k+1}}{1 - a}$ and $w^* = \frac{y}{x}$.

Impact of Learning Rate on Convergence

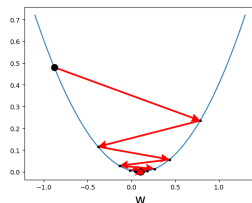
The recurrence relation:

$$\varepsilon^{k+1} = a^{k+1} \varepsilon^0, \quad \text{or} \quad w^{k+1} = a^{k+1}(w^0 - w^*) + w^*$$

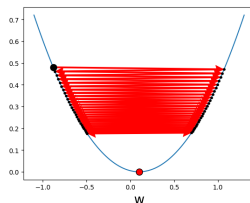
Here, the value of $a = 1 - \eta x^2$ leads to the following behaviors as $k \rightarrow \infty$:



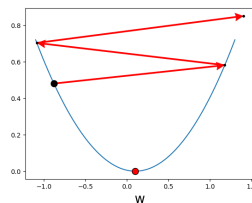
Slow



Just Right



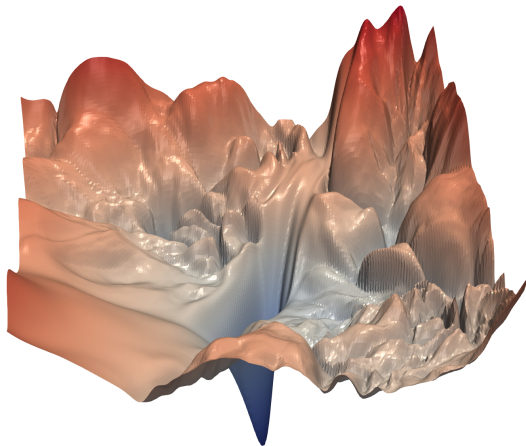
Oscillation



Divergence

- **Convergence:** If $\eta < 2/x^2$, then $|a| < 1$, so $a^k \rightarrow 0$, and w^k converges to the minimum w^* .
- **Oscillation:** If $\eta = 2/x^2$, then $a = -1$, and w^k oscillates around w^* with $w^{k+1} = (-1)^{k+1}(w^0 - w^*) + w^*$.
- **Divergence:** If $\eta > 2/x^2$, then $|a| > 1$, leading to $a^k \rightarrow \infty$, causing w^k to diverge.

Loss Landscape



Curse of Dimensionality in Optimization

- As the dimensionality of variables and the size of data increase, optimization becomes more challenging. For example, consider the following loss function:

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

- The recurrence relation for \mathbf{w}^{k+1} becomes:

$$\mathbf{w}^{k+1} = \left(\mathbf{I} - \frac{\eta}{n} \mathbf{X} \mathbf{X}^\top \right)^{k+1} (\mathbf{w}^0 - \mathbf{w}^*) + \mathbf{w}^* = \mathbf{A}^{k+1} (\mathbf{w}^0 - \mathbf{w}^*) + \mathbf{w}^*,$$

where $\mathbf{A} := \mathbf{I} - \frac{\eta}{n} \mathbf{X} \mathbf{X}^\top$.

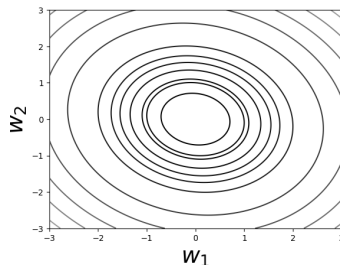
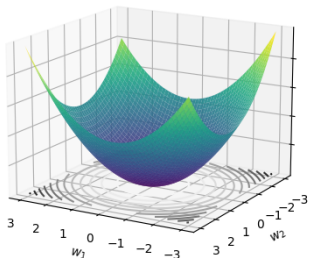
- Similarly, the dynamics of the residual $\mathbf{e}^k = \mathbf{X}\mathbf{w}^k - \mathbf{y}$ is given by:

$$\mathbf{e}^{k+1} = \mathbf{A} \mathbf{e}^k = \mathbf{A}^{k+1} \mathbf{e}^0$$

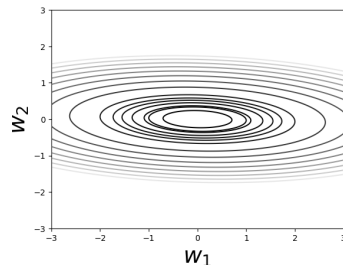
- The dynamics are governed by the matrix \mathbf{A} , rather than a scalar. In deep learning, this system becomes even more complex as \mathbf{A} can change during training, i.e., $\mathbf{A}(k)$.

3D Loss Landscape Visualization

Consider a case where $\mathbf{w} = (w_1, w_2)$. Below is the 3D contour of $\mathcal{L}(\mathbf{w})$:



Well-Conditioned



Ill-Conditioned

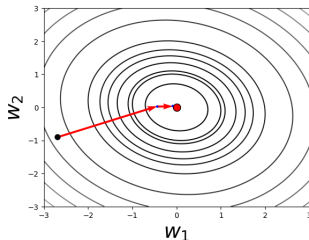
The loss landscape is not always smooth and easy to optimize:

$$\mathbf{X}_1 = \begin{bmatrix} 1 & 0.1 \\ 0 & 1 \end{bmatrix}, \quad \text{v.s.} \quad \mathbf{X}_2 = \begin{bmatrix} 3 & 0.1 \\ 0 & 1 \end{bmatrix}$$

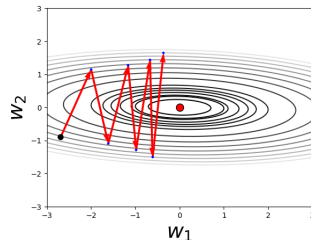
- **Well-Conditioned:** Contours are close to circular; curvature is similar in all directions
- **Ill-Conditioned:** Contours are elongated ellipses; curvatures are unequal across directions

Challenges in Gradient Descent: Zig-Zag Patterns

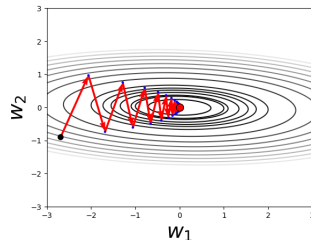
- In ill-conditioned systems, gradient descent can only progress with a **small** learning rate. The following examples illustrate different behaviors:



Fast Convergence ($\eta = 1.0$)



Divergence ($\eta = 0.23$)



Zig-Zag Pattern ($\eta = 0.22$)

Key Observations

- Ill-conditioned systems cannot tolerate large learning rates.
- Even with a small learning rate, gradient descent may exhibit a **zig-zag** pattern.

III-Conditioned Systems

Consider the recurrence relation for ill-conditioned systems:

$$\mathbf{e}^{k+1} = \left(\mathbf{I} - \frac{\eta}{n} \mathbf{X} \mathbf{X}^\top \right)^{k+1} \mathbf{e}^0 = \begin{bmatrix} 1 - \frac{9\eta}{2} & \\ & 1 - \frac{\eta}{2} \end{bmatrix}^{k+1} \mathbf{e}^0 = \begin{bmatrix} (1 - \frac{9\eta}{2})^{k+1} & \\ & (1 - \frac{\eta}{2})^{k+1} \end{bmatrix} \mathbf{e}^0.$$

where we use $n = 2$ and

$$\mathbf{X} = \begin{bmatrix} 3 & \\ & 1 \end{bmatrix}, \quad \text{and} \quad \mathbf{A} = \mathbf{X} \mathbf{X}^\top = \begin{bmatrix} 9 & \\ & 1 \end{bmatrix}$$

- From the first exponential, convergence requires $|1 - 9\eta/2| < 1$, i.e., $\eta < \frac{4}{9}$.
- From the second exponential, convergence requires $|1 - \eta/2| < 1$, i.e., $\eta < 4$.

Key Observations: Condition Number and Learning Rate

- To ensure convergence, we must choose $\eta < \frac{4}{9}$, which is much smaller than 4.
- One direction convergence may be slower than the other, leading to the zig-zag behavior.
- This occurs because the **condition number** κ of the Hessian $\mathbf{H}(\mathbf{w})$ is large, i.e., $\kappa = 9$.

Gradients Vanishing and Exploding

Gradients Vanishing and Exploding

Information Propagation in Deep Neural Networks

- **Forward Propagation (biases omitted):** Starting with $\mathbf{x}^0 = \mathbf{x}$,

$$\begin{aligned} \mathbf{z}^\ell &= \mathbf{W}^\ell \mathbf{x}^{\ell-1}, \quad \forall \ell \in \{0, 1, 2, \dots, L\}, \\ \mathbf{x}^\ell &= \phi(\mathbf{z}^\ell), \end{aligned}$$

where $\phi(\mathbf{z})$ is the activation function.

- Assuming a linear activation function $\phi(\mathbf{z}) = \mathbf{z}$ and tied weights $\mathbf{W}^\ell = \mathbf{W}$ for simplicity:

$$\mathbf{x}^\ell = \mathbf{W}^\ell \mathbf{x}^{\ell-1} = \begin{bmatrix} a & \\ & a \end{bmatrix} \mathbf{x}^{\ell-1} = \begin{bmatrix} a & \\ & a \end{bmatrix}^\ell \mathbf{x}^0, \quad \Rightarrow \quad \boxed{\mathbf{x}^\ell = a^\ell \mathbf{x}^0}.$$

As ℓ increases:

- If $a > 1$, then \mathbf{x}^ℓ grows exponentially (explodes).
- If $a < 1$, then \mathbf{x}^ℓ diminishes exponentially (vanishes).
- If $a \simeq 1$, then $\mathbf{x}^\ell \simeq \mathcal{O}(\mathbf{x}^0)$.

Backward Propagation and Gradient Behavior

- **Backward Propagation (biases omitted):** Start with $d\mathbf{z}^L = (\mathbf{x}^L - \mathbf{y}) \odot \phi'(\mathbf{z}^L)$:

$$d\mathbf{z}^\ell = \left[(\mathbf{W}^{\ell+1})^\top d\mathbf{z}^{\ell+1} \right] \odot \phi'(\mathbf{z}^\ell), \quad \forall \ell \in \{1, 2, \dots, L-1\},$$
$$d\mathbf{W}^\ell = d\mathbf{z}^\ell \mathbf{x}^{(\ell-1)\top}.$$

- With linear activation, $\phi'(x) = 1$:

$$d\mathbf{z}^\ell = (\mathbf{W}^{\ell+1})^\top d\mathbf{z}^{\ell+1} = \begin{bmatrix} a & \\ & a \end{bmatrix} d\mathbf{z}^{\ell+1} = a^{L-\ell} d\mathbf{z}^L.$$

- As ℓ becomes far from L :
 - If $a > 1$, then $d\mathbf{z}^\ell$ grows rapidly (exploding gradients).
 - If $a < 1$, then $d\mathbf{z}^\ell$ diminishes rapidly (vanishing gradients).
 - If $a \simeq 1$, then $d\mathbf{z}^\ell \simeq \mathcal{O}(d\mathbf{z}^L)$.

Summary

Learning Rate:

- Small learning rates slow down the training.
- Large learning rates can cause oscillations or divergence.

Loss Landscape:

- The loss landscape is often ill-conditioned in DNNs, with local minima, maxima, and saddle points.
- Ill-conditioned local structure prevents using a large learning rate in gradient descent.

Gradient Vanishing and Exploding:

- Information propagation in DNNs can be unstable.
- Lower layers tend to have small gradient values due to vanishing gradients.
- Differing gradient scales can lead to ill-conditioned local structures.

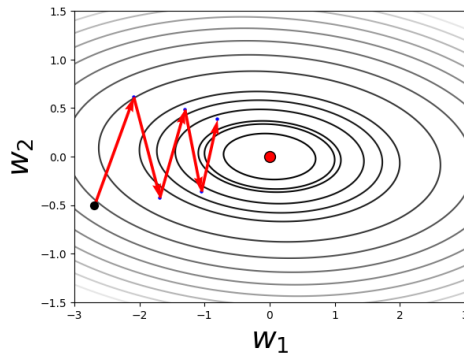
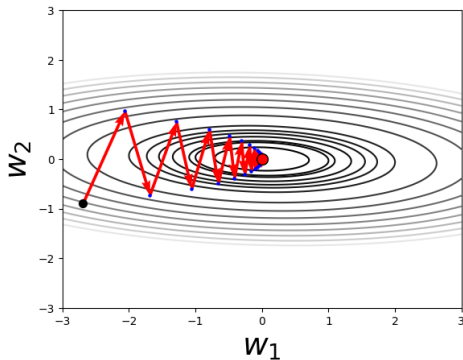
Outline

- 1 Calculus Review: Second Derivatives
- 2 Convergence Issues
- 3 Advanced Optimization Algorithm

Gradient Descent with Momentum

The Trajectory of Gradient Descent

Let us take a close look at the trajectory of gradient descent (GD):



- What happens if we decompose the update along each coordinate?

Average Search Direction

- The **general** iterative training process is defined as:

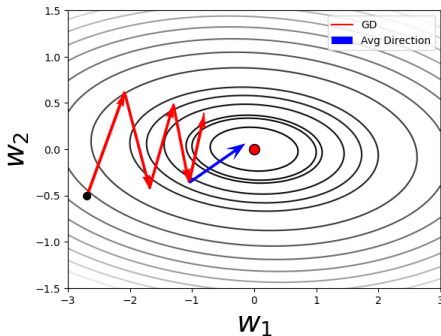
$$\mathbf{w}^+ = \mathbf{w} - \eta \cdot \mathbf{v},$$

where \mathbf{v} is the search direction, and η is the learning rate. We take $\mathbf{v} = \nabla \mathcal{L}(\mathbf{w})$ for GD.

- Given a trajectory of GD up to the k -th iteration, the sequence of gradient directions is:

$$\{\mathbf{g}^0, \mathbf{g}^1, \dots, \mathbf{g}^{k-1}\}, \quad \text{where } \mathbf{g}^i = \nabla \mathcal{L}(\mathbf{w}^i) \implies \boxed{\mathbf{v}^k = \frac{1}{k} \sum_{i=0}^{k-1} \mathbf{g}^i.}$$

- Smooth out noisy gradients and maintain a more stable descent trend over iterations



GD with Averaged Gradient Direction

By applying the idea of averaging the negative gradient direction, we have:

$$\begin{aligned} \mathbf{v}^{k+1} &= \frac{1}{k+1} \sum_{i=0}^k \mathbf{g}^i, \\ \mathbf{w}^{k+1} &= \mathbf{w}^k - \eta \cdot \mathbf{v}^{k+1}. \end{aligned}$$

- The **cumulative average** can be rewritten in a **running update** form:

$$\begin{aligned} \mathbf{v}^{k+1} &= \frac{1}{k+1} \left(\sum_{i=0}^{k-1} \mathbf{g}^i + \mathbf{g}^k \right) \\ &= \frac{k}{k+1} \cdot \frac{1}{k} \sum_{i=0}^{k-1} \mathbf{g}^i + \frac{1}{k+1} \mathbf{g}^k \\ &= \frac{k}{k+1} \mathbf{v}^k + \left(1 - \frac{k}{k+1} \right) \mathbf{g}^k. \end{aligned}$$

- With $\beta_k = \frac{k-1}{k}$, gradient descent with an averaged gradient direction is given by:

$$\begin{aligned} \mathbf{v}^{k+1} &= \beta_{k+1} \mathbf{v}^k + (1 - \beta_{k+1}) \mathbf{g}^k, \\ \mathbf{w}^{k+1} &= \mathbf{w}^k - \eta \cdot \mathbf{v}^{k+1}. \end{aligned}$$

- Only needs to store the most recent \mathbf{v}^k , instead of the entire $\{\mathbf{g}^0, \dots, \mathbf{g}^k\}$

Gradient Descent with Momentum

- Fixing $\beta_k = \beta$ for $\beta \in (0, 1)$, i.e., $\beta = 0.9$, the update rule becomes:

$$\begin{aligned} \mathbf{v}^{k+1} &= \beta \mathbf{v}^k + (1 - \beta) \mathbf{g}^k, \\ \mathbf{w}^{k+1} &= \mathbf{w}^k - \eta \cdot \mathbf{v}^{k+1}, \end{aligned}$$

- Here, β balances the influence of past gradients \mathbf{v}^k and the current \mathbf{g}^k on the update.
- The value of β determines the **effect memory length** $n \approx \frac{1}{1-\beta}$, e.g., $\beta = 0.9$ corresponds to $n \approx 10$ and $\beta = 0.99$ corresponds to $n \approx 100$.
- This method is also referred to as **Gradient Descent (GD) with Momentum** or **accelerated GD**:

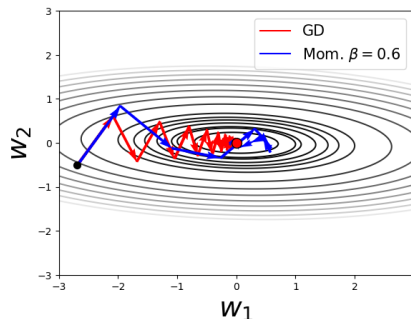
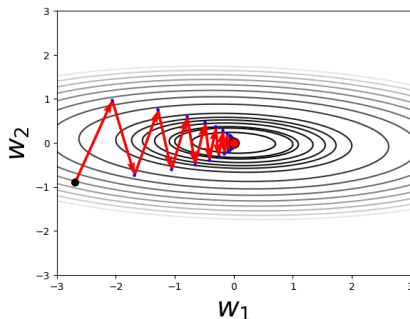
$$\begin{aligned} \mathbf{w}^{k+1} &= \mathbf{w}^k - \eta \cdot \mathbf{v}^{k+1} \\ &= \mathbf{w}^k - \eta \cdot [\beta \mathbf{v}^k + (1 - \beta) \mathbf{g}^k] \\ &= \mathbf{w}^k - \underbrace{\eta(1 - \beta)}_{:= \alpha} \cdot \mathbf{g}^k + \beta \cdot \underbrace{(\mathbf{w}^k - \mathbf{w}^{k-1})}_{\text{Momentum}}, \end{aligned}$$

The current update is influenced **both** by the latest gradient and the past movement (momentum).

Impact of Momentum in Gradient Descent

Gradient Descent with momentum:

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \eta(1 - \beta) \cdot \mathbf{g}^k + \beta \cdot (\mathbf{w}^k - \mathbf{w}^{k-1}).$$

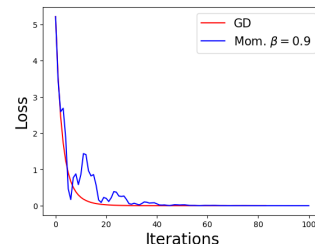
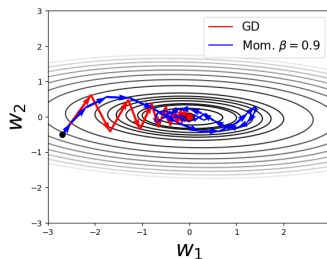
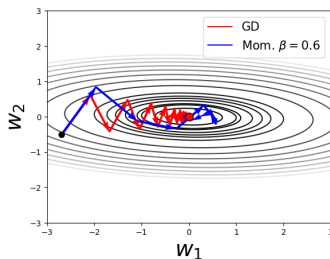


- Gradient Descent (GD) converges in 84 steps with $\eta = 0.22$, while GD with momentum converges in 36 steps with $\eta = 0.63$ and $\beta = 0.6$.
- For further reading, see this [illustration on the impact of momentum](#).

Damping in Gradient Descent with Momentum

Gradient Descent with momentum:

$$\mathbf{w}^{k+1} = \mathbf{w}^k - \alpha \cdot \mathbf{g}^k + \beta \cdot (\mathbf{w}^k - \mathbf{w}^{k-1}).$$



Key Observation

- A large momentum factor β can cause the loss to oscillate and not consistently decrease.
- This oscillation often occurs around the stationary point.

Summary

Gradient Descent with momentum:

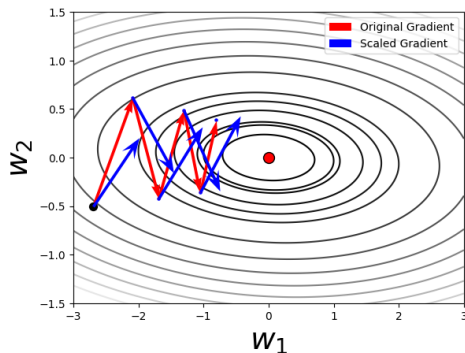
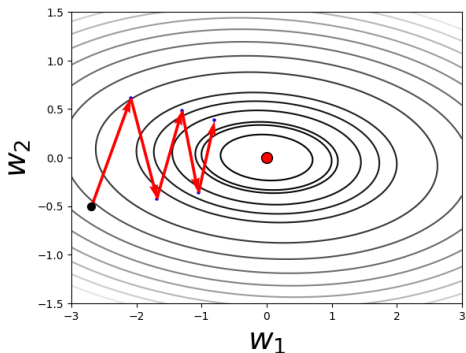
$$\mathbf{w}^{k+1} = \mathbf{w}^k - \alpha \cdot \mathbf{g}^k + \beta \cdot (\mathbf{w}^k - \mathbf{w}^{k-1}).$$

- The current update is influenced by **both** the most recent gradient and the past movement.
- The search direction in GD with momentum is a **running average** of past gradients.
- Momentum allows for **larger learning rates** and **faster** convergence.
- Too large a momentum factor β can cause **damping** in the loss and oscillation around the stationary point.

Adaptive Gradient Descent

Divergent Gradient Scaling

During the GD, the **magnitudes** of the gradient coordinates can vary significantly. One approach is to **scale** the magnitudes so that each gradient coordinate has an order of $\mathcal{O}(1)$ magnitude.



RMSProp

- By applying the idea of a *running average* on the **gradient magnitudes**, the scaling factors are:

$$\begin{aligned}s^+ &= \beta s + (1 - \beta)g^2, \\ w^+ &= w - \eta \cdot \frac{g}{\sqrt{s^+ + \varepsilon}},\end{aligned}$$

where all operations including x^2 , \sqrt{x} , and x/y are taken **element-wise**, and ε is a small value (e.g., $\varepsilon = 10^{-8}$) preventing dividing by zero.

- This method is called **root mean squared propagation (RMSP)**.
- RMSProp is effectively an **adaptive learning rate** algorithm:

$$w_i^+ = w_i - \eta_i \cdot g_i,$$

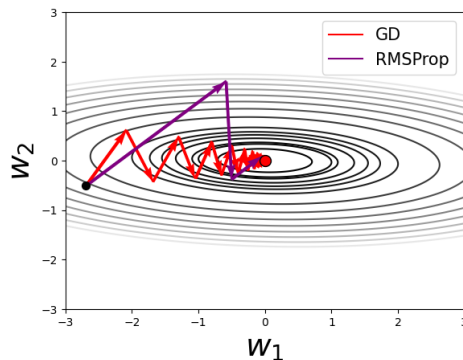
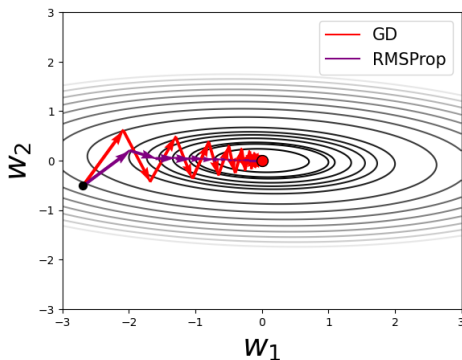
where $\eta_i = \eta / \sqrt{s_i^+}$ is the **adaptive learning rate**.

- Each gradient coordinate has a unique, adaptive learning rate.

Performance of RMSProp

RMSProp:

$$\mathbf{s}^+ = \beta \mathbf{s} + (1 - \beta) \mathbf{g}^2,$$
$$\mathbf{w}^+ = \mathbf{w} - \eta \cdot \frac{\mathbf{g}}{\sqrt{\mathbf{s}^+ + \varepsilon}}.$$



- GD converges in 84 steps with $\eta = 0.22$.
- RMSProp converges in 43 steps with $\eta = 0.07$ and in 10 steps with $\eta = 0.22$.
- **Note:** RMSProp may **not** perform well with large learning rates.

Adam

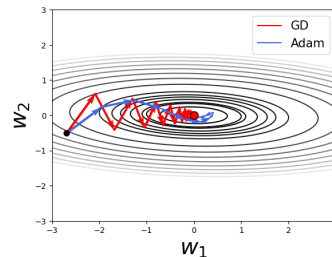
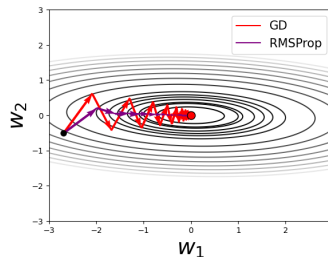
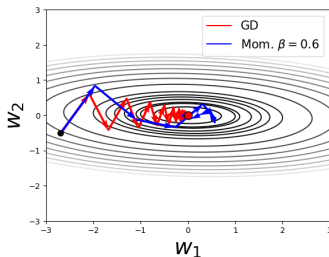
The **Adaptive Moment Estimation (Adam)** algorithm combines the advantages of GD with momentum and RMSProp:

$$\mathbf{v}^+ = \beta_1 \mathbf{v} + (1 - \beta_1) \mathbf{g},$$

$$\mathbf{s}^+ = \beta_2 \mathbf{s} + (1 - \beta_2) \mathbf{g}^2,$$

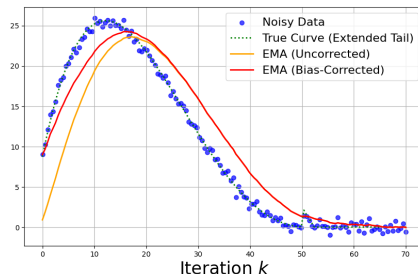
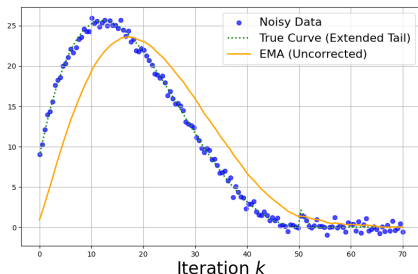
$$\mathbf{w}^+ = \mathbf{w} - \eta \cdot \frac{\mathbf{v}^+}{\sqrt{\mathbf{s}^+ + \varepsilon}},$$

where typical values in training DNNs are $\beta_1 = 0.9$ and $\beta_2 = 0.99$.



- GD converges in 84 steps with $\eta = 0.22$.
- GD with momentum converges in 35 steps with $\eta = 0.63$ and $\beta = 0.6$.
- RMSProp converges in 43 steps with $\eta = 0.07$.
- Adam converges in 32 steps with $\eta = 0.74$.
- $\mathbf{v}^0 = \mathbf{0}$.

Bias-Corrected Adam



The **bias-corrected Adam** adjusts the moving averages to account for their **initial bias toward zero**:

$$\begin{aligned} \mathbf{v}^{k+1} &= \beta_1 \mathbf{v}^k + (1 - \beta_1) \mathbf{g}^k, & \hat{\mathbf{v}}^{k+1} &= \frac{\mathbf{v}^{k+1}}{1 - \beta_1^k}, \\ \mathbf{s}^{k+1} &= \beta_2 \mathbf{s}^k + (1 - \beta_2) (\mathbf{g}^k)^2, & \hat{\mathbf{s}}^{k+1} &= \frac{\mathbf{s}^{k+1}}{1 - \beta_2^k}, \\ \mathbf{w}^{k+1} &= \mathbf{w}^k - \eta \cdot \frac{\hat{\mathbf{v}}^{k+1}}{\sqrt{\hat{\mathbf{s}}^{k+1} + \epsilon}}, \end{aligned}$$

- The **bias correction** improves accuracy, especially during the early training steps.

Summary

- Adaptive gradient descent (AdaGrad) scales each gradient coordinate to have the same $\mathcal{O}(1)$ magnitudes.
- Adaptive methods provide an **adaptive learning rate** for each gradient coordinate.
- Typically, adaptive methods do not use large learning rates.
- Adam combines momentum-based and adaptive scaling techniques, balancing fast convergence with gradient smoothing.
- Adam applies bias correction to compensate for the initial bias of moving averages toward zero.

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) Overview

Recap: Training deep neural networks as an optimization problem over parameters θ :

$$\min_{\theta} \mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^n \ell(f_{\theta}(x_i), y_i)$$

- The gradient descent (GD) update rule is:

$$\theta^+ = \theta - \eta \nabla_{\theta} \mathcal{L}(\theta) = \theta - \eta \cdot \frac{1}{n} \sum_{i=1}^n \nabla_{\theta} \ell_i(\theta),$$

where $\ell_i(\theta) := \ell(f_{\theta}(x_i), y_i)$ is the loss for sample i .

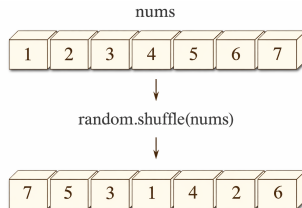
- In practice, the number of training samples n can be extremely large (millions or even billions). Computing the gradient over all samples becomes computationally expensive.
- **Stochastic Gradient Descent (SGD)**: Instead of computing the gradient over the full dataset, we randomly select a smaller batch \mathcal{B} of samples (called a **mini-batch**):

$$\theta^+ = \theta - \eta \cdot \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \nabla_{\theta} \ell_i(\theta)$$

- The size of the mini-batch $|\mathcal{B}|$ can vary. If $|\mathcal{B}| = 1$, it is called **SGD**. Otherwise, it is called **mini-batch SGD**.

Mini-batch SGD and Epochs

- In mini-batch SGD, the entire dataset is typically divided into several mini-batches of a fixed size b .
- The mini-batches are often selected by random shuffling (or **permutation**), and the model is updated iteratively for each mini-batch.



- After processing all mini-batches once, we complete an **epoch**, and the process can be repeated for multiple epochs until convergence.
- **Efficiency:** Mini-batch SGD can be computationally efficient because each update is based on a subset of data, reducing the cost per iteration.
- **Advanced Techniques:** Mini-batch SGD can be combined with other optimization techniques, such as momentum, RMSProp, and Adam.

SGD vs. Full Batch Gradient Descent

- **Stochastic Behavior:** Unlike full-batch gradient descent, the loss function in SGD does **not** always decrease at every step due to the randomness of mini-batches. This can cause oscillations.
- **Convergence Speed:** Although SGD may take more iterations to converge in theory, it often converges faster in terms of wall-clock time due to its lower per-iteration computational cost.
- **Trade-off:** Full-batch GD ensures a consistent reduction in loss at each step, but the cost per iteration is high, especially for large datasets. SGD trades off some accuracy for faster convergence.

Summary

Calculus Review:

- A point \mathbf{a} is a **local minimum** of $f(\mathbf{x})$ if $f(\mathbf{a}) \leq f(\mathbf{x})$ for all \mathbf{x} near \mathbf{a} .
- A point \mathbf{a} is **stationary** if $\nabla f(\mathbf{a}) = \mathbf{0}$, and gradient descent stops at \mathbf{a} .
- A stationary point \mathbf{a} is a local minimum if the Hessian $\mathbf{H}(\mathbf{a}) \succeq 0$, i.e., the function is concave up.

Convergence Issues:

- Small learning rates lead to **slow** convergence.
- Large learning rates cause **oscillations** or **divergence**.
- DNN loss landscapes are complex, with high and varied **condition numbers** κ .
- Ill-conditioned loss landscapes cause **zig-zag patterns** in gradient descent.
- Unstable information propagation in DNNs leads to vanishing or exploding gradients.

Advanced Optimizers

- Averaging gradients leads to a smoother descent direction.
- Gradient descent with averaged search directions is equivalent to **GD with momentum**.
- Momentum allows **faster convergence** with **larger learning rates**.
- Too large a learning rate may cause **damping** or oscillations during training.
- Adaptive methods like RMSProp **scale** gradients to ensure consistent $\mathcal{O}(1)$ magnitudes.
- Adaptive optimizers provide an **adaptive learning rate** for each gradient coordinate.
- Adam applies bias correction to counteract the initial bias in moving averages.
- Mini-batch SGD is computationally efficient, updating weights using subsets of data to accelerate training.
- SGD can be combined with advanced optimizers (e.g., momentum, RMSProp, Adam).

Questions

- What are common activation functions beyond sigmoid and ReLU?
- How should I choose learning rate, width, and depth for my network?
- Does gradient descent always converge? How can I speed up training?
- Does good training performance guarantee good test performance?