Optimization in Neural Networks

Tianxiang (Adam) Gao

Jan 23, 2025

Outline

Recap: Neural Network Training

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

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

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

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

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

  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 $\boldsymbol{x}^0 = \boldsymbol{x}$

$$\boldsymbol{z}^\ell = \boldsymbol{W}^\ell \boldsymbol{x}^{\ell-1}, \quad \forall \ell \in \{0, 1, 2, \ldots, L\}$$
$$\boldsymbol{x}^\ell = \phi(\boldsymbol{z}^\ell),$$

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

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

- **Random initialization** is preferred over zero initialization to avoid the issue of *symmetric patterns*.

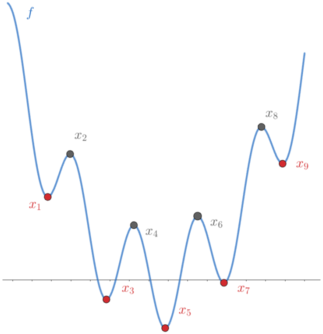
Questions

- 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

Calculus Review: Extreme Values

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



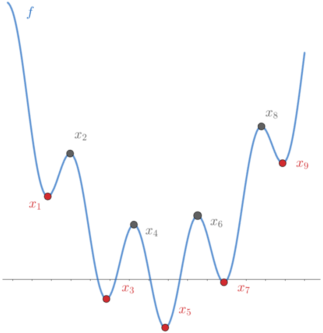
Local Min. x_1, x_3, x_5, x_7, x_9;
Local Max. x_2, x_4, x_6, x_8;

- 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*:

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

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.

James Stewart, "Calculus."

Hessian Matrix

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

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

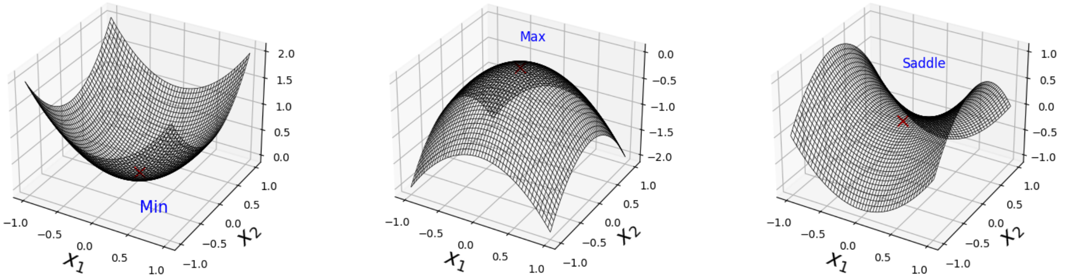
$$\nabla f(\boldsymbol{a}) = \begin{bmatrix} \frac{\partial f(\boldsymbol{a})}{\partial x_1} & \cdots & \frac{\partial f(\boldsymbol{a})}{\partial x_n} \end{bmatrix}^\top = \mathbf{0}$$

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

$$\nabla^2 f(\boldsymbol{x}) = \boldsymbol{H}(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$

- 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:

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

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

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

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:

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

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

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

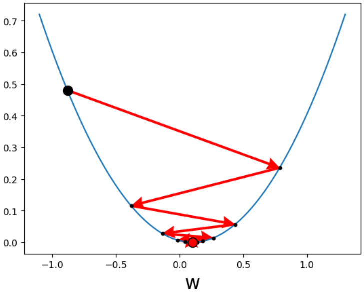
$$\nabla f(\boldsymbol{w}) = \boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}) = \begin{bmatrix} 3(3w_1 - 1) \\ w_2 \end{bmatrix} \quad \text{and} \quad \boldsymbol{H}(\boldsymbol{w}) = \begin{bmatrix} 9 & \\ & 1 \end{bmatrix},$$

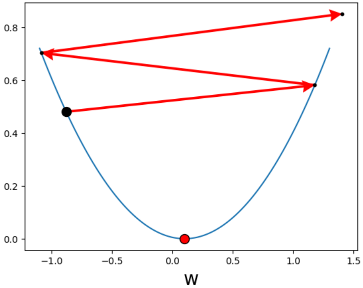
- Here 9 is the **largest eigenvalue** of $\boldsymbol{H}$, 1 is the **smallest eigenvalue** of $\boldsymbol{H}$, and their ratio is called **conditional number** $\kappa = 9$.

Outline

Learning Rate

Learning Rate





One-Dimensional Linear Regression

Consider a simple one-dimensional linear regression problem:

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

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**:

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

- 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).

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:

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

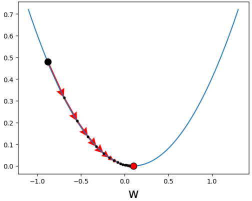
  where we use the geometric series $\sum_{i=0}^k a^i = \frac{1-a^{k+1}}{1-a}$ and $w^* = \frac{y}{x}$.

Impact of Learning Rate on Convergence

The recurrence relation:

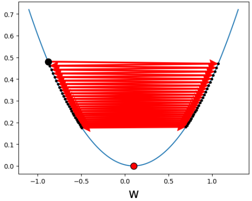
$$w^{k+1} = a^{k+1}(w^0 - w^*) + w^*$$

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

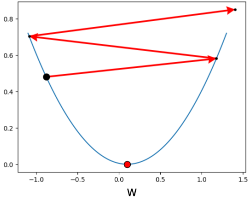


Slow

Just Right



Oscillation



Divergence

- **Convergence:** If $\eta < 2/x^2$, then $|a| < 1$, so $a^k \to 0$, 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 $a^k \to \infty$, causing w^k to diverge.

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}:

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

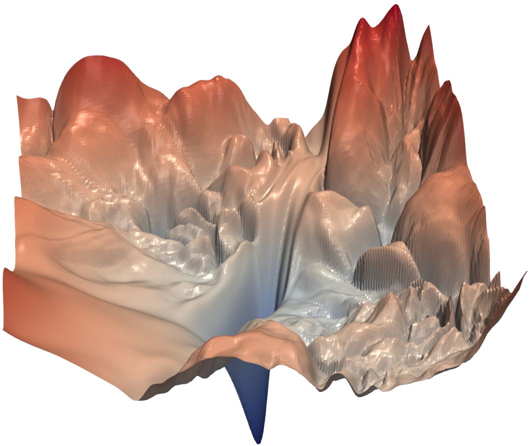
where $a := 1 - \eta x^2$ and $\varepsilon^k = w^k x - y$ is the error at step k.
Repeating this relation, we obtain:

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

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

Loss Landscape

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:

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

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

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

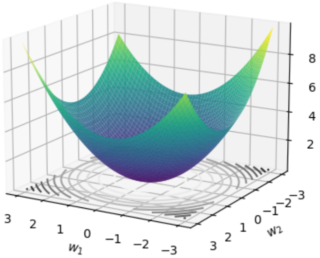
  where $\boldsymbol{A} := \boldsymbol{I} - \frac{\eta}{n}\boldsymbol{X}\boldsymbol{X}^\top$.
- Similarly, the dynamics of the residual $\boldsymbol{e}^k = \boldsymbol{X}\boldsymbol{w}^k - \boldsymbol{y}$ is given by:

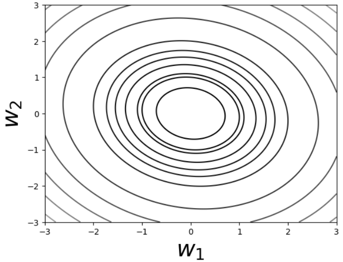
$$\boldsymbol{e}^{k+1} = \boldsymbol{A}\boldsymbol{e}^k = \boldsymbol{A}^{k+1}\boldsymbol{e}^0$$

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

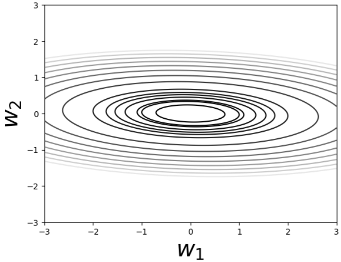
3D Loss Landscape Visualization

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





Well-Conditioned



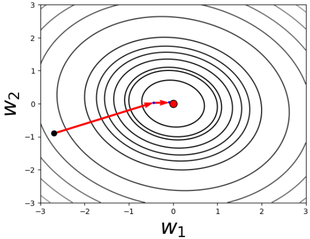
Ill-Conditioned

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

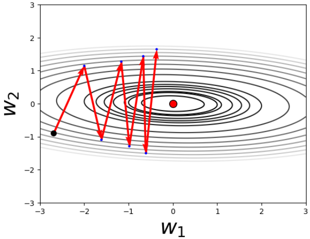
$$\boldsymbol{X}_1 = \begin{bmatrix} 1 & 0.1 \\ 0 & 1 \end{bmatrix}, \quad \text{v.s.} \quad \boldsymbol{X}_2 = \begin{bmatrix} 3 & 0.1 \\ 0 & 1 \end{bmatrix}$$

Challenges in Gradient Descent: Zig-Zag Patterns

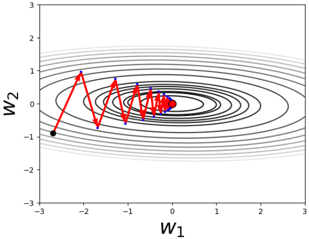
- In ill-conditioned systems, gradient descent can only progress with a **small** learning rate. The following examples illustrate different behaviors:



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.

Ill-Conditioned Systems

Consider the recurrence relation for ill-conditioned systems:

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

where we use $n = 2$ and

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

- 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 $\eta < \frac{4}{9}$.
- One direction converges may be slower than the other, leading to the zig-zag behavior.
- This occurs because the **condition number** κ of the Hessian $\boldsymbol{H}(\boldsymbol{w})$ 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 $\boldsymbol{x}^0 = \boldsymbol{x}$,

$$\boldsymbol{z}^\ell = \boldsymbol{W}^\ell \boldsymbol{x}^{\ell-1}, \quad \forall \ell \in \{0, 1, 2, \ldots, L\},$$
$$\boldsymbol{x}^\ell = \phi(\boldsymbol{z}^\ell),$$

  where $\phi(\boldsymbol{z})$ is the activation function.
- Assuming a linear activation function $\phi(\boldsymbol{z}) = \boldsymbol{z}$ for simplicity:

$$\boldsymbol{x}^\ell = \boldsymbol{W}^\ell \boldsymbol{x}^{\ell-1} = \begin{bmatrix} a & \\ & a \end{bmatrix}^\ell \boldsymbol{x}^0 = a^\ell \boldsymbol{x}^0.$$

  As ℓ increases:
  - If $a > 1$, then $\boldsymbol{x}^\ell$ grows exponentially (explodes).
  - If $a < 1$, then $\boldsymbol{x}^\ell$ diminishes exponentially (vanishes).

Backward Propagation and Gradient Behavior

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

$$
\begin{aligned}
d\boldsymbol{z}^\ell &= \left[(\boldsymbol{W}^{\ell+1})^\top d\boldsymbol{z}^{\ell+1}\right] \odot \phi'(\boldsymbol{z}^\ell), \quad \forall \ell \in \{1, 2, \ldots, L-1\}, \\
d\boldsymbol{W}^\ell &= d\boldsymbol{z}^\ell \boldsymbol{x}^{(\ell-1)\top}.
\end{aligned}
$$

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

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

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

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.

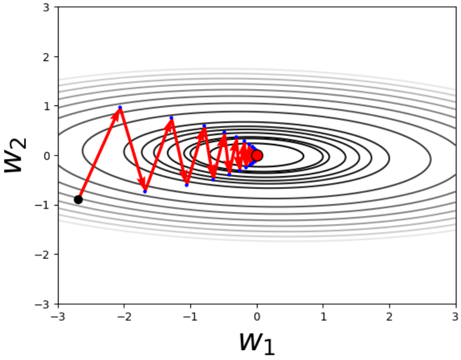
Outline

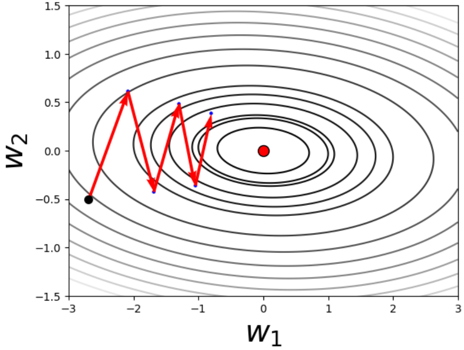
Outline

Gradient Descent with Momentum

The Trajectory of Gradient Descent

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





Average Search Direction

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

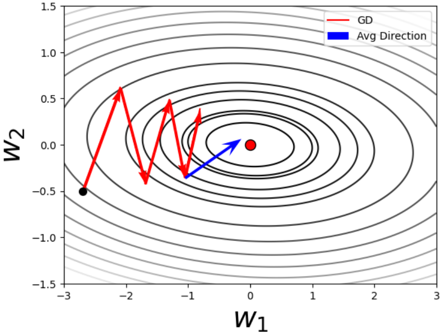
$$\boldsymbol{w}^+ = \boldsymbol{w} - \eta \cdot \boldsymbol{v},$$

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

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

$$\{\boldsymbol{g}^0, \boldsymbol{g}^1, \cdots, \boldsymbol{g}^{k-1}\}, \quad \text{where} \quad \boldsymbol{g}^i = \nabla \mathcal{L}(\boldsymbol{w}^i) \quad \Longrightarrow \quad \boldsymbol{v}^k = \frac{1}{k}\sum_{i=0}^{k-1} \boldsymbol{g}^i.$$

- 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:

$$\boldsymbol{v}^{k+1} = \frac{1}{k+1} \sum_{i=0}^{k} \boldsymbol{g}^i,$$
$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta \cdot \boldsymbol{v}^{k+1}.$$

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

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

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

$$\boldsymbol{v}^{k+1} = \beta_{k+1} \boldsymbol{v}^k + (1 - \beta_{k+1})\, \boldsymbol{g}^k,$$
$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta \cdot \boldsymbol{v}^{k+1}.$$

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

Gradient Descent with Momentum

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

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

- Here, β balances the influence of past gradients $\boldsymbol{v}^k$ and the current $\boldsymbol{g}^k$ 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**:

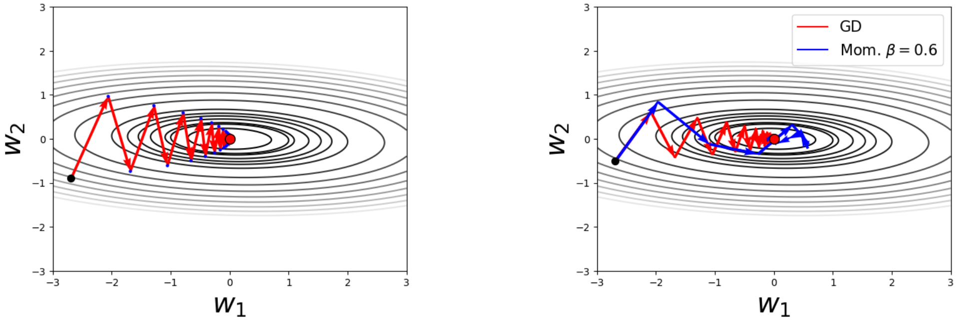
$$\begin{aligned} \boldsymbol{w}^{k+1} &= \boldsymbol{w}^k - \eta \cdot \boldsymbol{v}^{k+1} \\ &= \boldsymbol{w}^k - \eta \cdot \left[\beta \boldsymbol{v}^k + (1-\beta)\boldsymbol{g}^k\right] \\ &= \boldsymbol{w}^k - \underbrace{\eta(1-\beta)}_{:=\alpha} \cdot \boldsymbol{g}^k + \beta \cdot \underbrace{(\boldsymbol{w}^k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

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:

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



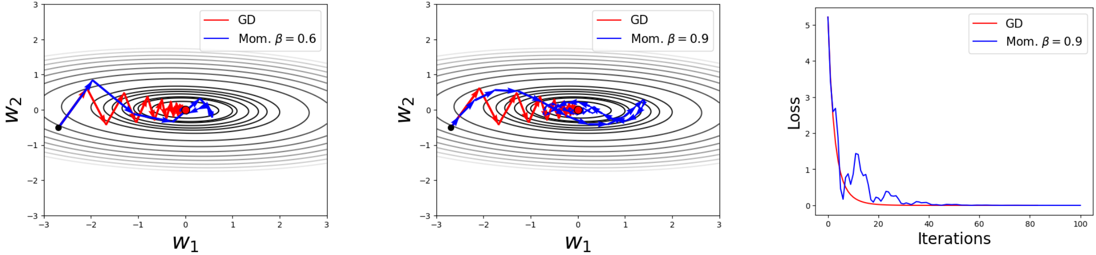
- Gradient Descent (GD) converges in 84 steps with $\eta = 0.22$, while GD with momentum converges in 36 steps with $\eta = 0.63$ and $\beta = 0.6$.
- For further reading, see this illustration on the impact of momentum.

Accelerated GD requires $\mathcal{O}(\sqrt{k})$ iteration to achieve the error level that standard GD achieves in $\mathcal{O}(k)$ iterations.

Damping in Gradient Descent with Momentum

Gradient Descent with momentum:

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



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:

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

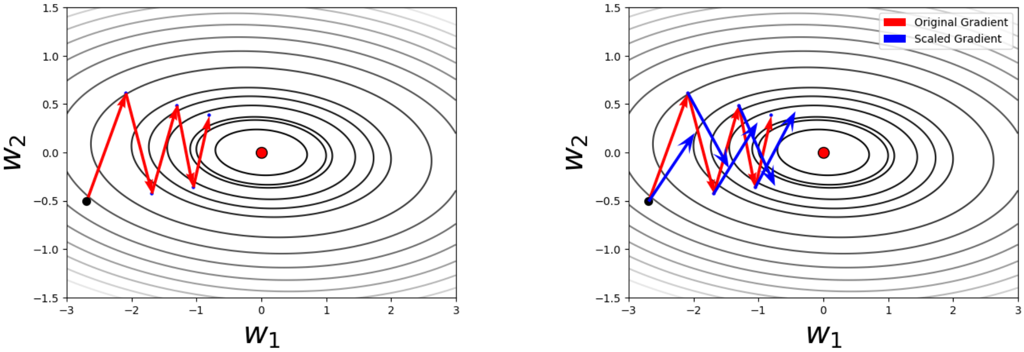
- The current update is influenced by **both** the most recent gradient and the past movement.
- The search direction in GD with momentum is a **running average** of past gradients.
- Momentum allows for **larger learning rates** and **faster** convergence.
- Too large a momentum factor β can cause **damping**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

Outline

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:

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

  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:

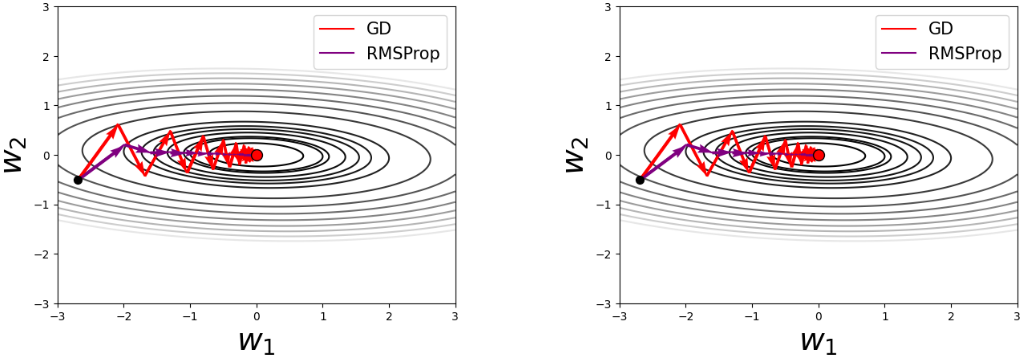
$$\boldsymbol{w}_i^+ = \boldsymbol{w}_i - {\color{red}\eta_i} \cdot \boldsymbol{g}_i,$$

  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:

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



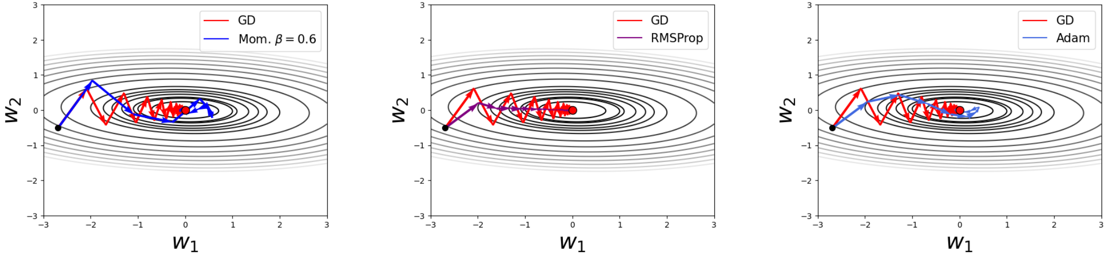
- GD converges in 84 steps with $\eta = 0.22$.
- RMSProp converges in 43 steps with $\eta = 0.07$ and in 10 steps with $\eta = 0.22$.
- **Note**: RMSProp may **not** perform well with large learning rates.

Adam

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

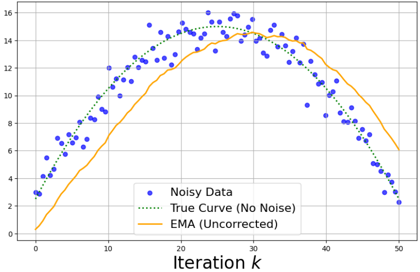
$$\boldsymbol{v}^+ = \beta_1 \boldsymbol{v} + (1-\beta_1)\boldsymbol{g},$$
$$\boldsymbol{s}^+ = \beta_2 \boldsymbol{s} + (1-\beta_2)\boldsymbol{g}^2,$$
$$\boldsymbol{w}^+ = \boldsymbol{w} - \eta \cdot \frac{\boldsymbol{v}^+}{\sqrt{\boldsymbol{s}^+ + \varepsilon}},$$

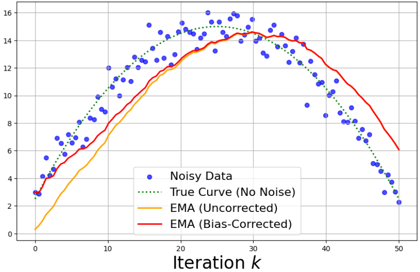
where typical values in training DNNs are $\beta_1 = 0.9$ and $\beta_2 = 0.99$.



- 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 $\eta = 0.74$.
- $\boldsymbol{v}^0 = \boldsymbol{0}$.

Bias-Corrected Adam





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

$$\boldsymbol{v}^{k+1} = \beta_1 \boldsymbol{v}^k + (1-\beta_1)\boldsymbol{g}^k, \quad \hat{\boldsymbol{v}}^{k+1} = \frac{\boldsymbol{v}^{k+1}}{1-\beta_1^k},$$

$$\boldsymbol{s}^{k+1} = \beta_2 \boldsymbol{s}^k + (1-\beta_2)(\boldsymbol{g}^k)^2, \quad \hat{\boldsymbol{s}}^{k+1} = \frac{\boldsymbol{s}^{k+1}}{1-\beta_2^k},$$

$$\boldsymbol{w}^{k+1} = \boldsymbol{w}^k - \eta \cdot \frac{\hat{\boldsymbol{v}}^{k+1}}{\sqrt{\hat{\boldsymbol{s}}^{k+1}+\varepsilon}},$$

- 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.

Outline

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) Overview

Recap: Training deep neural networks as an optimization problem over parameters $\boldsymbol{\theta}$:

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

- The gradient descent (GD) update rule is:

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

  where $\ell_i(\boldsymbol{\theta}) := \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), y_i)$ 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**):

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

- 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.