AI Applications Lecture 17

Fundamentals of AI Tuning

SUZUKI, Atsushi
Jing WANG

Outline

Introduction

Machine Learning Framework: Learning and Inference

The machine learning framework used in many generative AIs is divided into two steps:

- the **learning** step, which appropriately determines the parameters of a parametric function, and
- the **inference** step, which actually executes the desired task using the function determined by the parameters set in the learning step.

This course has so far focused on the inference step, but starting from this lecture, we will focus on “learning”.

Throughout the broad field of generative AI, when we want to consider the problem of how to learn, that is, how to determine the parameters, one non-trivial difficulty is that, as we have seen, the graph structure of the neural network, i.e., the function system, is completely different depending on the type of data being

Different Objective Functions in Different Components

Furthermore, the objective function also differs depending on the field and component, even for the same type of data.

Examples of objective functions:

- **Text field:** Cross entropy when a neural network outputs a probability distribution over a set of tokens (see [1]).
- **Image generation:**
  - Contrastive learning loss by CLIP,
  - Squared error in the noise estimator,
  - Regularized reconstruction squared error in the VAE (in reality, it is more complex than this) [3, 5].

Even though the function systems and objective functions differ this much depending on the field and component, as long as feedforward neural networks are used, **all parameter determination can be done by backpropagation and stochastic gradient methods**. This is the wonderful flexibility of neural networks

1.1 Learning Outcomes

Upon completion of this lecture, students should be able to:

- Calculate the gradient vector of parameters for a neural network defined on a general directed acyclic multigraph using **backpropagation**.
- Determine appropriate parameters using the **stochastic gradient method** with the gradient vector of the neural network parameters.

Preliminaries: Mathematical Notations Revisited

Mathematical Notations (1): Definitions and Sets

We briefly revisit the basic mathematical notations.

- **Definition:**
  - $(\mathrm{LHS}) \coloneqq (\mathrm{RHS})$: Indicates that the left-hand side is defined by the right-hand side. For example, $a \coloneqq b$ indicates that a is defined by b.
- **Set:**
  - Sets are often denoted by uppercase calligraphic letters. Example: $\mathcal{A}$.
  - $x \in \mathcal{A}$: x belongs to $\mathcal{A}$.
  - $\{\}$: The empty set.
  - $\{a, b, c\}$: The set consisting of elements a, b, c (roster notation).
  - $\{x \in \mathcal{A} \mid P(x)\}$: The set of elements in $\mathcal{A}$ for which the proposition $P(x)$ is true (set-builder notation).
  - $\mathbb{R}$: The set of all real numbers.
  - $\mathbb{R}_{>0}$: The set of all positive real numbers.
  - $\mathbb{R}_{\geq 0}$: The set of all non-negative real numbers.
  - $\mathbb{Z}$: The set of all integers.
  - $\mathbb{Z}_{>0}$: The set of all positive integers.

Mathematical Notations (2): Functions, Vectors, Matrices, Tensors

- **Function:**
  - $f : \mathcal{X} \to \mathcal{Y}$: f maps elements of $\mathcal{X}$ to $\mathcal{Y}$.
  - $y = f(x)$: The output is $y \in \mathcal{Y}$ when $x \in \mathcal{X}$ is input to f.
- **Vector:**
  - A vector is a column of numbers arranged vertically.
  - Vectors are denoted by bold italic lowercase letters: $\boldsymbol{v}$.
  - $\boldsymbol{v} \in \mathbb{R}^n$: $\boldsymbol{v}$ is an n-dimensional real vector.
  - The i-th element is v_i:

$$\boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}.$$

- **Matrix:**
  - Matrices are bold italic uppercase letters: $\boldsymbol{A}$.
  - $\boldsymbol{A} \in \mathbb{R}^{m,n}$: an $m \times n$ real matrix.
  - (i, j)-th element: $a_{i,j}$.

Function Gradient Vectors and the Idea of Gradient Methods

Parametric Functions and Objective Functions

AI aims to obtain some appropriate input-output relationship using a **parametric function**. A parametric function $f_{(\cdot)}$ is a mathematical object such that giving a parameter $\boldsymbol{\theta}$ determines one function $f_{\boldsymbol{\theta}}$.

Ultimately, what we determine is the parameter vector $\boldsymbol{\theta}$.

To determine $\boldsymbol{\theta}$, we must provide information to the computer (directly or indirectly) about what kind of parameter is desirable for solving the task.

Objective function:

- Takes the parameter as an argument.
- Returns a real value representing its “goodness”.

Maximization, or minimization?

Remark

In many cases in AI, the objective function is designed so that the **smaller** its value, the better the parameter, so it can also be called a **cost function**.

Even if a larger value means better parameters, we can multiply by -1 and treat it as a cost function. Thus, we lose no generality by considering only the case where “smaller is better”. In this lecture, we will follow this convention.

Example of an Objective Function

Example

Suppose we want to create a prediction function from an input space $\mathcal{X} \subset \mathbb{R}^{d_{\text{in}}}$ to an output space $\mathcal{Y} \subset \mathbb{R}^{d_{\text{out}}}$ using a parametric function $f_{(\cdot)}$.

If we have past data $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})_{i=1}^{m}$, we can embed the desire for good predictions into a squared error function, and design the objective function as

$$\mathcal{L}(\boldsymbol{\theta}) \coloneqq \frac{1}{m} \sum_{i=1}^{m} \left\| \boldsymbol{y}^{(i)} - f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) \right\|_2^2, \tag{1}$$

where $\| \cdot \|_2$ is the Euclidean norm.

Objective function minimization is our objective.

Which objective function is optimal differs depending on the field, but once the objective function is determined, what needs to be done is aggregated into solving the objective function's minimization problem:

$$\underset{\boldsymbol{\theta} \in \mathbb{R}^{d_{\mathrm{param}}}}{\text{Minimize}} \ \mathcal{L}(\boldsymbol{\theta}). \tag{2}$$

In this lecture, we learn a practical way to solve this minimization problem using the concept of a gradient vector, and explain why using a gradient vector is promising.

Local Search, Directional Derivatives, and Gradients

Brute Force Search is Impossible in Practice

In modern generative AI models with a huge number of parameters, brute force search is computationally impossible.

Definition (Brute Force Search)

We decide on a discretization width $\delta > 0$ and assume that each component of $\boldsymbol{\theta}$ is restricted to a finite set of candidates $\{k\delta \mid k \in \mathbb{Z}\}$ (in practice, truncated to a finite range).

Brute force search is the method of performing a full search of $\mathcal{L}$ over the Cartesian product set of all candidate points and selecting the $\boldsymbol{\theta}$ that gives the minimum value.

Impossible at all

Example

Suppose there are $d_{\text{param}} = 10^9$ parameters, and each component takes $2^8 = 256$ possible values (like float8).

The total number of candidate points is

$$256^{10^9} = 2^{8 \cdot 10^9} \approx 10^{(8 \cdot 10^9) \log_{10} 2} \approx 10^{2.4 \times 10^9},$$

an astronomical scale.

Even if one evaluation could be processed at 10^{15} FLOP and executed in parallel at 10^{18} FLOP per second (exa-scale), this is impossibly out of scale.

Therefore, global exhaustive search is not realistic.

From Global Search to Local Search

Since global search is impossible, we consider repeating a **local search**.

Local search:

- Operation of searching for a better parameter (where $\mathcal{L}$ is smaller) in the vicinity of the current parameter vector.
- Uses only information from the current parameter vector or its vicinity.

We start with an initial parameter vector $\boldsymbol{\theta}^{[0]}$ (random or reasonably good) and repeat local search, updating

$$\boldsymbol{\theta}^{[0]} \to \boldsymbol{\theta}^{[1]} \to \boldsymbol{\theta}^{[2]} \to \cdots$$

to get closer to a better vector.

Local search

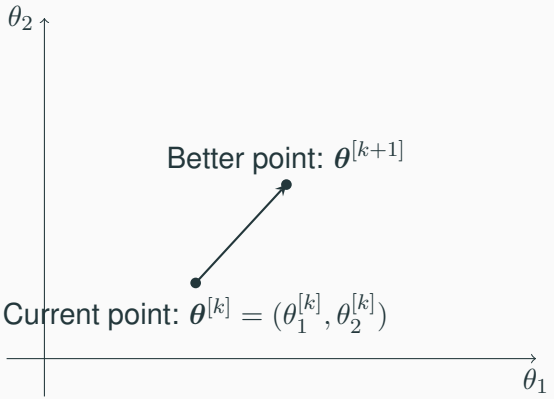

Figure 1: Conceptual diagram of local search from the current point $\boldsymbol{\theta}^{[k]}$ in the (θ_1, θ_2)-plane.

Directional Derivatives

As an efficient local search, we adopt the idea of slightly updating the parameter vector in the direction that reduces $\mathcal{L}$ as efficiently as possible.

The rate of change of $\mathcal{L}$ when moving $\boldsymbol{\theta}$ in the direction of a unit vector $\boldsymbol{u}$ is expressed by the **one-sided directional derivative**.

Definition (One-sided Directional Derivative)

Let $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$, $\boldsymbol{\theta} \in \mathbb{R}^d$, and $\boldsymbol{u} \in \mathbb{R}^d$ be a unit vector ($\|\boldsymbol{u}\|_2 = 1$).

The **one-sided directional derivative** $D^+_{\boldsymbol{u}}\mathcal{L}(\boldsymbol{\theta})$ is

$$D^+_{\boldsymbol{u}}\mathcal{L}(\boldsymbol{\theta}) \coloneqq \lim_{t \to 0+} \frac{\mathcal{L}(\boldsymbol{\theta} + t\boldsymbol{u}) - \mathcal{L}(\boldsymbol{\theta})}{t} \tag{3}$$

(if the limit exists).

Remarks: directional derivatives

Remark

The one-sided directional derivative can be defined even for non-unit $\boldsymbol{u}$, but then the "speed of advance" differs by direction.

If we want to compare directions fairly, we should restrict to unit vectors.

Remark

If $\mathcal{L}$ is differentiable at $\boldsymbol{\theta}$ (definition later), then for any unit vector $\boldsymbol{u}$, the limit as $t \to 0-$ also exists and coincides with the limit as $t \to 0+$.

In this case, we simply call it the **directional derivative** and may write $D_{\boldsymbol{u}}\mathcal{L}(\boldsymbol{\theta})$.

Directional Derivatives and Their Graphs

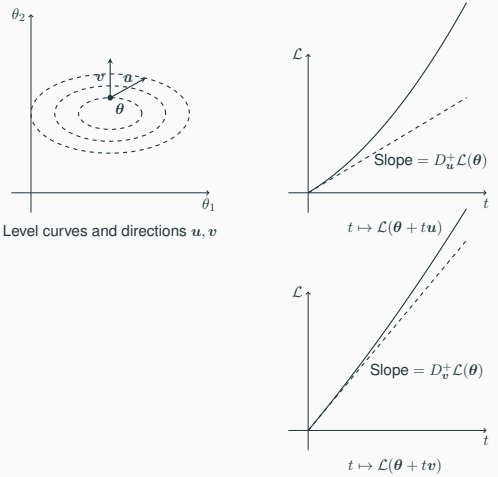

Figure 2: Graphs of $t \mapsto \mathcal{L}(\boldsymbol{\theta} + t\boldsymbol{u})$ and $t \mapsto \mathcal{L}(\boldsymbol{\theta} + t\boldsymbol{v})$ and their tangents at $t = 0$.

Optimizing the Directional Derivative

We want the direction that reduces $\mathcal{L}$ as efficiently as possible near the current parameter vector.

Since the one-sided directional derivative expresses exactly that efficiency, we want the direction that minimizes it.

Definition (Optimization of One-sided Directional Derivative)

We consider:

$$\text{Maximize:} \quad \underset{\|\boldsymbol{u}\|_2=1}{\text{Maximize}} \; D^+_{\boldsymbol{u}}\mathcal{L}(\boldsymbol{\theta}), \tag{4}$$

$$\text{Minimize:} \quad \underset{\|\boldsymbol{u}\|_2=1}{\text{Minimize}} \; D^+_{\boldsymbol{u}}\mathcal{L}(\boldsymbol{\theta}). \tag{5}$$

Under appropriate assumptions, the solution to this problem is given by the famous **gradient vector**.

Gradient vector

Definition (Gradient Vector)

When $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ has all partial derivatives at $\boldsymbol{\theta}$,

$$\frac{\partial \mathcal{L}}{\partial \theta_i}(\boldsymbol{\theta}) \coloneqq \lim_{t \to 0} \frac{\mathcal{L}(\boldsymbol{\theta} + t\boldsymbol{e}_i) - \mathcal{L}(\boldsymbol{\theta})}{t},$$

the **gradient** is

$$\nabla \mathcal{L}(\boldsymbol{\theta}) \coloneqq \begin{bmatrix} \partial \mathcal{L} / \partial \theta_1(\boldsymbol{\theta}) \\ \vdots \\ \partial \mathcal{L} / \partial \theta_d(\boldsymbol{\theta}) \end{bmatrix}.$$

Differentiability of Multivariate Functions

To relate gradients and directional derivatives, we use differentiability in the multivariate sense.

Definition (Differentiability of Multivariate Functions)

Let $\mathcal{L} : U \to \mathbb{R}$ on an open set $U \subset \mathbb{R}^d$ and $\boldsymbol{\theta} \in U$.

If there exists a linear map $A : \mathbb{R}^d \to \mathbb{R}$ such that

$$\lim_{\boldsymbol{h} \to \mathbf{0}} \frac{\mathcal{L}(\boldsymbol{\theta} + \boldsymbol{h}) - \mathcal{L}(\boldsymbol{\theta}) - A(\boldsymbol{h})}{\|\boldsymbol{h}\|_2} = 0, \tag{6}$$

we say that $\mathcal{L}$ is **differentiable** at $\boldsymbol{\theta}$, and call A the **derivative** $D\mathcal{L}(\boldsymbol{\theta})$.

Confirming the differentiability is practically tractable.

Many functions used in neural networks are differentiable (linear maps, common activations, etc.), and compositions of differentiable functions are differentiable:

Proposition (Differentiability of Composite Functions)

Let $U \subset \mathbb{R}^d$, $V \subset \mathbb{R}^m$ open, and $\mathcal{L}_1 : U \to V$, $\mathcal{L}_2 : V \to \mathbb{R}$.

If $\mathcal{L}_1$ is differentiable at $\boldsymbol{\theta} \in U$ and $\mathcal{L}_2$ is differentiable at $\mathcal{L}_1(\boldsymbol{\theta}) \in V$, then $\mathcal{L}_2 \circ \mathcal{L}_1$ is differentiable at $\boldsymbol{\theta}$ and

$$D(\mathcal{L}_2 \circ \mathcal{L}_1)(\boldsymbol{\theta}) = D\mathcal{L}_2(\mathcal{L}_1(\boldsymbol{\theta})) \circ D\mathcal{L}_1(\boldsymbol{\theta}). \tag{7}$$

Steepest Ascent/Descent and the Gradient Vector

Theorem (Directions of Steepest Ascent/Descent)

Let $\mathcal{L} : U \to \mathbb{R}$ be differentiable on an open set $U \subset \mathbb{R}^d$, and $\boldsymbol{\theta} \in U$. Assume all partial derivatives $\partial \mathcal{L}/\partial \theta_i(\boldsymbol{\theta})$ exist.

1. *For any unit vector $\boldsymbol{u}$, the one-sided directional derivative exists and*

$$D^+_{\boldsymbol{u}} \mathcal{L}(\boldsymbol{\theta}) = \nabla \mathcal{L}(\boldsymbol{\theta})^\top \boldsymbol{u}.$$

2. *If $\nabla \mathcal{L}(\boldsymbol{\theta}) \neq \mathbf{0}$, the maximizer of (4) is $\boldsymbol{u}^\star = \frac{\nabla \mathcal{L}(\boldsymbol{\theta})}{\|\nabla \mathcal{L}(\boldsymbol{\theta})\|_2}$, and the maximum value is $\|\nabla \mathcal{L}(\boldsymbol{\theta})\|_2$. If $\nabla \mathcal{L}(\boldsymbol{\theta}) = \mathbf{0}$, then $D^+_{\boldsymbol{u}} \mathcal{L}(\boldsymbol{\theta}) = 0$ for all unit $\boldsymbol{u}$.*
3. *If $\nabla \mathcal{L}(\boldsymbol{\theta}) \neq \mathbf{0}$, the minimizer of (5) is $\boldsymbol{u}_\star = -\frac{\nabla \mathcal{L}(\boldsymbol{\theta})}{\|\nabla \mathcal{L}(\boldsymbol{\theta})\|_2}$, and the minimum value is $-\|\nabla \mathcal{L}(\boldsymbol{\theta})\|_2$. If $\nabla \mathcal{L}(\boldsymbol{\theta}) = \mathbf{0}$, again $D^+_{\boldsymbol{u}} \mathcal{L}(\boldsymbol{\theta}) = 0$ for all unit $\boldsymbol{u}$.*

Idea of the Proof (Steepest Ascent/Descent)

Sketch of proof idea:

- Differentiability implies

$$\mathcal{L}(\boldsymbol{\theta} + \boldsymbol{h}) = \mathcal{L}(\boldsymbol{\theta}) + A(\boldsymbol{h}) + r(\boldsymbol{h}),$$

  where A is linear and $r(\boldsymbol{h})$ is small compared to $\|\boldsymbol{h}\|_2$.

- For a unit vector $\boldsymbol{u}$ and small t,

$$\frac{\mathcal{L}(\boldsymbol{\theta} + t\boldsymbol{u}) - \mathcal{L}(\boldsymbol{\theta})}{t} = A(\boldsymbol{u}) + \frac{r(t\boldsymbol{u})}{t} \to A(\boldsymbol{u}).$$

- Thus, $D^+_{\boldsymbol{u}}\mathcal{L}(\boldsymbol{\theta}) = A(\boldsymbol{u})$.
- By relating A to partial derivatives, we show $A(\boldsymbol{u}) = \nabla\mathcal{L}(\boldsymbol{\theta})^\top \boldsymbol{u}$.
- Maximizing/minimizing $\nabla\mathcal{L}^\top \boldsymbol{u}$ subject to $\|\boldsymbol{u}\|_2 = 1$ is done via the Cauchy–Schwarz inequality, yielding the claimed optimal directions.

Gradients are Orthogonal to Level Sets

One helpful geometric property: under suitable assumptions, the gradient is orthogonal to level sets of $\mathcal{L}$.

Proposition (Gradient is Orthogonal to Level Sets)

Assume $\mathcal{L} : U \to \mathbb{R}$ is differentiable on an open set $U \subset \mathbb{R}^d$.

*Fix $c \in \mathbb{R}$ and define the **level set***

$$S := \{\boldsymbol{\theta} \in U \mid \mathcal{L}(\boldsymbol{\theta}) = c\}.$$

Take $\boldsymbol{\theta}_0 \in S$ and assume $\nabla \mathcal{L}(\boldsymbol{\theta}_0) \neq \mathbf{0}$.

*If $\boldsymbol{w}$ is a **tangent vector** to S at $\boldsymbol{\theta}_0$ (i.e., there exists a C^1 curve γ in S with $\gamma(0) = \boldsymbol{\theta}_0$, $\gamma'(0) = \boldsymbol{w}$), then*

$$\nabla \mathcal{L}(\boldsymbol{\theta}_0)^\top \boldsymbol{w} = 0.$$

That is, the gradient vector is orthogonal to any tangent vector of the level set.

The gradient vector is orthogonal to the level set

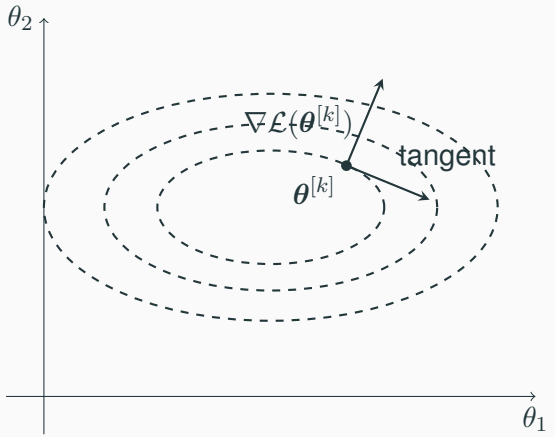

Figure 3: Gradient is perpendicular to the level curve of $\mathcal{L}$.

Steepest Descent Method

Since the negative gradient represents the direction of steepest local decrease, a natural idea is to repeatedly move in that direction.

Definition (Steepest Descent Method)

Given initial point $\boldsymbol{\theta}^{[0]} \in \mathbb{R}^d$, number of steps $K \in \mathbb{Z}_{>0}$, and learning rates $(\eta^{[k]})_{k=0}^{K-1} \subset \mathbb{R}_{>0}$, we update

$$\boldsymbol{\theta}^{[k+1]} := \boldsymbol{\theta}^{[k]} - \eta^{[k]} \nabla \mathcal{L}(\boldsymbol{\theta}^{[k]}), \quad k = 0, 1, \ldots, K-1. \tag{8}$$

This algorithm is called the **steepest descent method**.

In actual generative AI learning, we use large-scale data (big data) to define $\mathcal{L}$, which makes each exact gradient computation expensive.

Thus, the naive steepest descent method can be too costly, and we need more practical variants, which we discuss next.

Practical Gradient Methods: SGD and AdamW

Objective Functions with Big Data

Consider the case where the objective function depends on big data.

Suppose we use $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})_{i=1}^{m}$ and m is large, and the objective has the form

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \ell_{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})}(\boldsymbol{\theta}). \tag{9}$$

Then the gradient is

$$\nabla \mathcal{L}(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \nabla \ell_{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})}(\boldsymbol{\theta}), \tag{10}$$

and the summation over i becomes a bottleneck.

Therefore, in large-scale learning, we usually employ algorithms that **do not** sum over all data at each parameter update.

We now introduce the general framework of stochastic gradient methods and two important instances: SGD and AdamW.

Definition (General Stochastic Gradient Methods)

At each step k, suppose we can obtain a random vector $\boldsymbol{g}^{[k]}$ satisfying

$$\mathbb{E}[\boldsymbol{g}^{[k]} \mid \mathcal{F}^{[k]}] = \nabla \mathcal{L}(\boldsymbol{\theta}^{[k]}),$$

where $\mathcal{F}^{[k]}$ represents past information.

A general update rule using all past $\boldsymbol{g}^{[0]}, \ldots, \boldsymbol{g}^{[k]}$ is:

$$\boldsymbol{h}^{[k]} := \Phi_k(\boldsymbol{g}^{[0]}, \ldots, \boldsymbol{g}^{[k]}), \quad \boldsymbol{\theta}^{[k+1]} := \Psi_k\big(\boldsymbol{\theta}^{[k]}, \boldsymbol{h}^{[k]}\big), \tag{11}$$

where Φ_k, Ψ_k are deterministic maps.

The unbiased estimator appears naturally in big data situations.

Example (Unbiased Minibatch Gradient)

In the situation of (9), take a uniform random minibatch $S^{[k]} \subset \{1, \ldots, m\}$ of any size B, and define

$$\boldsymbol{g}^{[k]} := \frac{1}{|S^{[k]}|} \sum_{i \in S^{[k]}} \nabla \ell_{(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})}(\boldsymbol{\theta}^{[k]}). \tag{12}$$

Then $\mathbb{E}[\boldsymbol{g}^{[k]} \mid \boldsymbol{\theta}^{[k]}] = \nabla \mathcal{L}(\boldsymbol{\theta}^{[k]})$ holds (assuming uniform sampling).

Definition (SGD)

Given learning rates $(\eta^{[k]})_{k\geq 0}$ and an unbiased estimate $\boldsymbol{g}^{[k]}$ at each step, we update:

$$\boldsymbol{\theta}^{[k+1]} := \boldsymbol{\theta}^{[k]} - \eta^{[k]}\boldsymbol{g}^{[k]}. \tag{13}$$

It is related to the classic Robbins–Monro stochastic approximation [4].

AdamW: Adaptive Moment Estimation with Decoupled Weight Decay

Definition (AdamW [2])

Hyperparameters: $\beta_1, \beta_2 \in [0, 1)$, $\epsilon > 0$, learning rate $\eta^{[k]}$, decay coefficient $\lambda \geq 0$.

Let $\boldsymbol{g}^{[k]}$ be the minibatch gradient estimate. Initialize $\boldsymbol{m}^{[-1]} = \mathbf{0}, \quad \boldsymbol{v}^{[-1]} = \mathbf{0}$.

For each $k = 0, 1, \ldots$:

$$\boldsymbol{m}^{[k]} \coloneqq \beta_1 \boldsymbol{m}^{[k-1]} + (1 - \beta_1)\boldsymbol{g}^{[k]}, \tag{14}$$

$$\boldsymbol{v}^{[k]} \coloneqq \beta_2 \boldsymbol{v}^{[k-1]} + (1 - \beta_2)\big(\boldsymbol{g}^{[k]} \odot \boldsymbol{g}^{[k]}\big), \tag{15}$$

$$\widehat{\boldsymbol{m}}^{[k]} \coloneqq \boldsymbol{m}^{[k]} / (1 - \beta_1^{k+1}), \tag{16}$$

$$\widehat{\boldsymbol{v}}^{[k]} \coloneqq \boldsymbol{v}^{[k]} / (1 - \beta_2^{k+1}), \tag{17}$$

$$\boldsymbol{\theta}^{[k+1]} \coloneqq \boldsymbol{\theta}^{[k]} - \eta^{[k]} \frac{\widehat{\boldsymbol{m}}^{[k]}}{\sqrt{\widehat{\boldsymbol{v}}^{[k]}} + \epsilon} - \eta^{[k]} \lambda \, \boldsymbol{\theta}^{[k]}. \tag{18}$$

Here $\odot$ is element-wise product, and division between vectors is element-wise.

Intuition behind AdamW update

Remark

- (14): first moment (moving average of gradients).
- (15): second moment (moving average of squared gradients).
- (16)–(17): initial bias correction.
- (18): per-coordinate adaptive learning rate via $\widehat{v}^{[k]}$, plus **decoupled weight decay** $-\eta^{[k]}\lambda\boldsymbol{\theta}^{[k]}$.

Backpropagation for General Neural Networks

Goal: Gradients for General Neural Networks

Up to now:

- If we can compute the gradient of the objective function for a given data point or minibatch,
- we can apply SGD or AdamW to minimize the objective.

Key question: Can we compute the gradient vector for various structured neural networks and objective functions used in generative AI?

Answer: Yes. For feedforward neural networks and objective functions depending on their node outputs, gradients can be computed efficiently via **backpropagation** [7], a dynamic programming algorithm.

Remark

Backpropagation is often introduced for MLPs (multi-layer perceptrons), but MLPs are rarely used directly in generative AI.

MLP Structure

Definition (MLP Structure)

An MLP with input dimension d_0 and number of layers $L \in \mathbb{Z}_{>0}$ is defined as follows:

For each layer $\ell \in [1, L]_{\mathbb{Z}}$:

- Weight matrix $\boldsymbol{W}^{(\ell)} \in \mathbb{R}^{d_\ell, d_{\ell-1}}$,
- Bias vector $\boldsymbol{b}^{(\ell)} \in \mathbb{R}^{d_\ell}$,
- Differentiable activation function $\varphi^{(\ell)} : \mathbb{R}^{d_\ell} \to \mathbb{R}^{d_\ell}$.

MLP Forward propagation

Definition (MLP Forward Propagation)

Given an input $\boldsymbol{x} \in \mathbb{R}^{d_0}$, define

$$\boldsymbol{h}^{(0)} \coloneqq \boldsymbol{x}.$$

For $\ell \in [1, L]_{\mathbb{Z}}$:

$$\boldsymbol{z}^{(\ell)} \coloneqq \boldsymbol{W}^{(\ell)} \boldsymbol{h}^{(\ell-1)} + \boldsymbol{b}^{(\ell)} \in \mathbb{R}^{d_\ell}, \tag{19}$$

$$\boldsymbol{h}^{(\ell)} \coloneqq \varphi^{(\ell)}\left(\boldsymbol{z}^{(\ell)}\right) \in \mathbb{R}^{d_\ell}. \tag{20}$$

The final output

$$\boldsymbol{h}^{(L)} = f_{(\boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(L)}, \boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(L)})}(\boldsymbol{x})$$

is the MLP output.

MLP forward propagation intuition

Remark

Each layer performs an affine transformation (linear map plus bias) followed by a non-linear activation. Repeating this constructs a complex non-linear map $f_{(\cdot)}$ from $\boldsymbol{x}$ to $\boldsymbol{h}^{(L)}$.

Backpropagation for MLPs: Setup

Fix the input x and consider a scalar objective function $\mathcal{J} = \mathcal{J}\big(\boldsymbol{h}^{(L)}\big) \in \mathbb{R}$ depending on all parameters $\Theta \coloneqq \big(\boldsymbol{W}^{(1)}, \dots, \boldsymbol{W}^{(L)}, \boldsymbol{b}^{(1)}, \dots, \boldsymbol{b}^{(L)}\big)$.

Definition (MLP Backpropagation)

The backpropagation algorithm computes $\partial \mathcal{J} / \partial \boldsymbol{W}^{(\ell)}$ and $\partial \mathcal{J} / \partial \boldsymbol{b}^{(\ell)}$:

- **Forward pass**: compute and store $\boldsymbol{z}^{(\ell)}$, $\boldsymbol{h}^{(\ell)}$ using (19)–(20).
- **Initialization** (output layer): $\boldsymbol{\delta}^{(L)} \coloneqq \dfrac{\partial \mathcal{J}}{\partial \boldsymbol{h}^{(L)}} \odot \varphi^{(L)\prime}\big(\boldsymbol{z}^{(L)}\big)$.
- **Backward recursion**: for $\ell = L-1, \dots, 1$,

$$\boldsymbol{\delta}^{(\ell)} \coloneqq \big(\boldsymbol{W}^{(\ell+1)\top} \boldsymbol{\delta}^{(\ell+1)}\big) \odot \varphi^{(\ell)\prime}\big(\boldsymbol{z}^{(\ell)}\big), \tag{21}$$

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{W}^{(\ell)}} \coloneqq \boldsymbol{\delta}^{(\ell)}\, \boldsymbol{h}^{(\ell-1)\top}, \tag{22}$$

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{b}^{(\ell)}} \coloneqq \boldsymbol{\delta}^{(\ell)}. \tag{23}$$

MLP backpropagation intuition

Remark

$\boldsymbol{\delta}^{(\ell)}$ corresponds to $\partial \mathcal{J} / \partial \boldsymbol{z}^{(\ell)}$. Equation for $\boldsymbol{\delta}^{(\ell)}$ implements the chain rule, passing gradients from layer $\ell + 1$ back to ℓ, then multiplying by activation derivatives. The gradients w.r.t. $\boldsymbol{W}^{(\ell)}$ and $\boldsymbol{b}^{(\ell)}$ come from differentiating the affine relation $\boldsymbol{z}^{(\ell)} = \boldsymbol{W}^{(\ell)} \boldsymbol{h}^{(\ell-1)} + \boldsymbol{b}^{(\ell)}$.

Correctness of MLP Backpropagation

Theorem (Correctness of MLP Backprop)

Assume each activation $\varphi^{(\ell)}$ is differentiable at $\boldsymbol{z}^{(\ell)}$, and $\mathcal{J}$ is differentiable at $\boldsymbol{h}^{(L)}$.

Define $F(\Theta) \coloneqq \mathcal{J}(\boldsymbol{h}^{(L)}(\Theta))$.

Then the matrices/vectors from Definition 16 satisfy, for all ℓ and indices i, j:

$$\left(\frac{\partial \mathcal{J}}{\partial \boldsymbol{W}^{(\ell)}}\right)_{i,j} = \frac{\partial F}{\partial w_{i,j}^{(\ell)}}, \tag{24}$$

$$\left(\frac{\partial \mathcal{J}}{\partial \boldsymbol{b}^{(\ell)}}\right)_{i} = \frac{\partial F}{\partial b_{i}^{(\ell)}}. \tag{25}$$

That is, the algorithm yields the correct partial derivatives of F.

General Feedforward Neural Network as a DAG

We generalize from MLPs to arbitrary feedforward networks defined on directed acyclic multigraphs (DAGs).

Definition (General Feedforward Neural Network)

The **architecture** is a tuple:

- **Computation Graph**: DAG $G = (\mathcal{V}, \mathcal{E}, \mathsf{Tail}, \mathsf{Head})$ with nodes $\mathcal{V} \subset \mathbb{Z}_{>0}$ topologically ordered and $\mathsf{Tail}(e) < \mathsf{Head}(e)$ for all e.
- **Input/Output dimensions**: $d_{\text{input}}, d_{\text{output}}$.
- **Node specification**: Each $v \in \mathcal{V}$ has
  - activation $\mathrm{g}^{(v)} : \mathbb{R}^{d_{\text{arg}}^{(v)}} \to \mathbb{R}^{d_{\text{ret}}^{(v)}}$,
  - argument connection tuple $\mathsf{NodeSrc}^{(v)}$ specifying sources from input or incoming edges.

General Feedforward Neural Network as a DAG

Definition (General Feedforward Neural Network)

- **Edge specification**: Each $e \in \mathcal{E}$ has $\mathsf{EdgeSrc}(e)$ specifying which component of the tail node output it takes.
- **Parameter specification**: Parameter dimension d_{param} and surjection $\mathsf{Weight} : \mathcal{E} \to [1, d_{\text{param}}]_{\mathbb{Z}}$, allowing weight sharing.
- **Output specification**: OutSrc maps output indices to node-output components.

Given $\boldsymbol{\theta} \in \mathbb{R}^{d_{\text{param}}}$, $f_{\boldsymbol{\theta}} : \mathbb{R}^{d_{\text{input}}} \to \mathbb{R}^{d_{\text{output}}}$ is defined by evaluating each node v:

$$a_m^{(v)} = \begin{cases} x_j & \text{if } \mathsf{NodeSrc}_m^{(v)} = j, \\ \theta_{\mathsf{Weight}(e)}\, r_{\mathsf{EdgeSrc}(e)}^{(\mathsf{Tail}(e))} & \text{if } \mathsf{NodeSrc}_m^{(v)} = e, \end{cases} \tag{26}$$

$$\boldsymbol{r}^{(v)} \coloneqq \mathrm{g}^{(v)}\big(\boldsymbol{a}^{(v)}\big), \quad \text{(output)}. \tag{27}$$

Forward Propagation on a General DAG

Definition (Forward Propagation in Topological Order)

List nodes as $\mathcal{V} = \{v_1, \ldots, v_N\}$ in a topological order.

Compute for $i = 1, \ldots, N$:

- inputs $\boldsymbol{a}^{(v_i)}$ by collecting from $\boldsymbol{x}$ or parent nodes,
- outputs $\boldsymbol{r}^{(v_i)} = \mathrm{g}^{(v_i)}(\boldsymbol{a}^{(v_i)})$.

Then construct $\boldsymbol{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$ from the specified node-output components.

This procedure is called forward propagation on the general neural network.

Remark

Topological ordering ensures that all inputs to a node are available when it is evaluated. This generalizes layer-wise evaluation in MLPs to arbitrary graph structures.

Backpropagation on a General DAG: Setup

We extend backpropagation to this general DAG.

Consider a scalar objective function

$$\mathcal{J} = \mathcal{J}(\boldsymbol{y}) \in \mathbb{R}, \quad \boldsymbol{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}).$$

Introduce an **objective node** $v_{\mathcal{J}}$ as the last node in the topological order, with activation $\mathrm{g}^{(v_{\mathcal{J}})}$ applying $\mathcal{J}$ to $\boldsymbol{y}$.

For each node v, we will maintain

$$\boldsymbol{\delta}^{(v)} \coloneqq \frac{\partial \mathcal{J}}{\partial \boldsymbol{r}^{(v)}}.$$

We also maintain gradient accumulators G_p for each parameter index p.

Backpropagation on a General DAG: Algorithm

Definition (Backpropagation on General DAG)

- **Forward pass**: Compute and store $\boldsymbol{a}^{(v)}$, $\boldsymbol{r}^{(v)}$ for all v in topological order and $\mathcal{J}$ at $v_{\mathcal{J}}$.
- **Initialization**: For all v, set $\boldsymbol{\delta}^{(v)} = 0$. For the objective node, set

$$\boldsymbol{\delta}^{(v_{\mathcal{J}})} \coloneqq 1.$$

  For each parameter index p, set $G_p \coloneqq 0$.

Backpropagation on a General DAG: Algorithm ii

Definition (Backpropagation on General DAG)

- **Backprop loop**: Process nodes in reverse topological order $v_N, \ldots, v_1$; denote current node by w.
  - Let $\boldsymbol{J}^{(w)}(\boldsymbol{a}^{(w)}) \coloneqq \frac{\partial \mathrm{g}^{(w)}}{\partial \boldsymbol{a}^{(w)}}$ be the Jacobian. Define $\boldsymbol{\gamma}^{(w)} \coloneqq \frac{\partial \mathcal{J}}{\partial \boldsymbol{a}^{(w)}} = \boldsymbol{J}^{(w)}(\boldsymbol{a}^{(w)})^\top \boldsymbol{\delta}^{(w)}$.
  - For each argument index m of node w:
    1. If $\mathsf{NodeSrc}_m^{(w)}$ is an input component, do nothing.
    2. If $\mathsf{NodeSrc}_m^{(w)} = e$ is an incoming edge from $u = \mathsf{Tail}(e)$, with parameter index $p = \mathsf{Weight}(e)$ and source component $\mathsf{EdgeSrc}(e)$, then:

$$G_p \leftarrow G_p + \gamma_m^{(w)}\, r^{(u)}_{\mathsf{EdgeSrc}(e)}, \tag{28}$$

$$\delta^{(u)}_{\mathsf{EdgeSrc}(e)} \leftarrow \delta^{(u)}_{\mathsf{EdgeSrc}(e)} + \gamma_m^{(w)}\, \theta_p. \tag{29}$$

- **Output**: After the loop, set $\partial \mathcal{J} / \partial \theta_p \coloneqq G_p$.

Backpropagation intuition

Remark

$\boldsymbol{\delta}^{(v)}$ is the gradient w.r.t. node output $\boldsymbol{r}^{(v)}$, and $\boldsymbol{\gamma}^{(w)}$ is the gradient w.r.t. node input $\boldsymbol{a}^{(w)}$.

By propagating $\boldsymbol{\delta}$ backward through Jacobians and accumulating contributions along edges, we implement the chain rule on the DAG. Weight sharing is handled by summing contributions into the same G_p.

Correctness of Backpropagation on General DAG

Theorem (Correctness on General DAG)

Assume each node activation $\mathrm{g}^{(v)}$ *is differentiable at* $\boldsymbol{a}^{(v)}$*, and the objective node activation at* $\boldsymbol{a}^{(v_{\mathcal{J}})}$ *is differentiable.*

For fixed input $\boldsymbol{x}$*, define*

$$F(\boldsymbol{\theta}) \coloneqq \mathcal{J}\big(f_{\boldsymbol{\theta}}(\boldsymbol{x})\big).$$

Then F *is differentiable w.r.t.* $\boldsymbol{\theta}$*, and for each* p*,*

$$G_p = \frac{\partial F}{\partial \theta_p}.$$

That is, the algorithm produces the correct gradient vector $\nabla_{\boldsymbol{\theta}} F$*.*

Worked Example: Network Structure

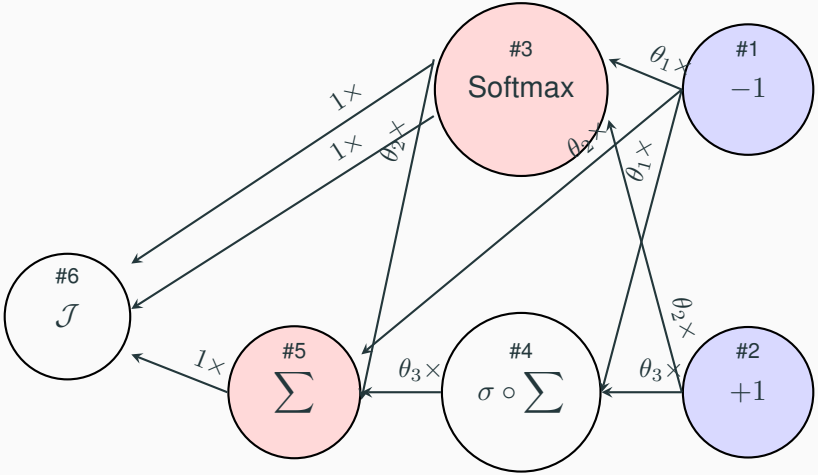

Figure 4: Example network structure (with objective node 6). Information flows from right to left.

Exercise on Example i

Consider a network with a node sequence $(1, 2, 3, 4, 5)$, and additionally an objective function node 6 (Figure **??**). The edge and parameter correspondence is as follows.

$$(1, 3_1) : \theta_1, \quad (2, 3_2) : \theta_2, \quad (1, 4) : \theta_1, \quad (2, 4) : \theta_3, \quad (1, 5) : \theta_2, \quad (3_1, 5) : \theta_2, \quad (4, 5) : \theta_3. \tag{30}$$

The activation functions are defined as follows.

- Node 1: Input node (input value -1), Node 2: Input node (input value $+1$).
- Node 3: Softmax (2D input, 2D output).
- Node 4: Apply standard sigmoid $\sigma(z) = 1/(1 + e^{-z})$ after summing the inputs.

Exercise on Example ii

- Node 5: Linear node that sums the inputs (identity function).
- Node 6: Calculates the sum of the cross entropy $-\log s_2$ for data $(0, 1)$ with respect to the 2D output $\boldsymbol{s} = (s_1, s_2)$ of node 3, and the square y_5^2 of the node 5 output y_5.

Exercise on Example iii

Forward Propagation (Topological order $1 \to 2 \to 3 \to 4 \to 5 \to 6$) is.

$$r^{(1)} = -1, \quad r^{(2)} = +1, \tag{31}$$

$$\text{Node 3: } z_1 = \theta_1 r^{(1)},\ z_2 = \theta_2 r^{(2)},\ s_j = \frac{e^{z_j}}{e^{z_1} + e^{z_2}},\ j = 1, 2, \tag{32}$$

$$\text{Node 4: } u = \theta_1 r^{(1)} + \theta_3 r^{(2)},\ y_4 = \sigma(u), \tag{33}$$

$$\text{Node 5: } y_5 = \theta_2 r^{(1)} + \theta_2 s_1 + \theta_3 y_4, \tag{34}$$

$$\text{Node 6: } \mathcal{J} = -\log s_2 + y_5^2. \tag{35}$$

Below, we apply the general backpropagation of Definition **??** to this specific example. We follow the calculations at each node according to the reverse topological sort order $6 \to 5 \to 4 \to 3 \to 2 \to 1$.

Exercise on Example iv

Processing Node 6 (Objective Function Node).

Since the output of node 6 is $\mathcal{J}$ itself

$$\delta^{(6)} = \frac{\partial \mathcal{J}}{\partial \mathcal{J}} = 1 \tag{36}$$

. The input to node 6 is (s_1, s_2, y_5), and

$$\frac{\partial \mathcal{J}}{\partial s_1} = 0, \quad \frac{\partial \mathcal{J}}{\partial s_2} = -\frac{1}{s_2}, \quad \frac{\partial \mathcal{J}}{\partial y_5} = 2y_5 \tag{37}$$

. This corresponds to $\gamma^{(6)}$ in Definition **??** and gives the initial gradient for the output (s_1, s_2) of node 3 and the output y_5 of node 5.

Processing Node 5 ($\sum$).

The gradient with respect to the output of node 5 is, from Equation (37),

$$\delta^{(5)} = \frac{\partial \mathcal{J}}{\partial y_5} = 2y_5. \tag{38}$$

Node 5 is a linear node that takes the sum of the inputs $(\theta_2 r^{(1)}, \theta_2 s_1, \theta_3 y_4)$, so the gradient with respect to the inputs is

$$\frac{\partial \mathcal{J}}{\partial \theta_2 r^{(1)}} = 2y_5, \quad \frac{\partial \mathcal{J}}{\partial \theta_2 s_1} = 2y_5, \quad \frac{\partial \mathcal{J}}{\partial \theta_3 y_4} = 2y_5. \tag{39}$$

Exercise on Example vi

From here, according to Equation (**??**) and Equation (**??**) of Definition **??**, we propagate the gradient to the parameters and parent nodes through edges $(1, 5)$, $(3_1, 5)$, and $(4, 5)$. Specifically

$$\text{Edge } (1,5): \quad G_2 \leftarrow G_2 + (2y_5) \cdot r^{(1)}, \quad \delta^{(1)} \leftarrow \delta^{(1)} + (2y_5)\theta_2, \tag{40}$$

$$\text{Edge } (3_1,5): \quad G_2 \leftarrow G_2 + (2y_5) \cdot s_1, \quad \delta_1^{(3)} \leftarrow \delta_1^{(3)} + (2y_5)\theta_2, \tag{41}$$

$$\text{Edge } (4,5): \quad G_3 \leftarrow G_3 + (2y_5) \cdot y_4, \quad \delta^{(4)} \leftarrow \delta^{(4)} + (2y_5)\theta_3. \tag{42}$$

Processing Node 4 ($\sigma \circ \sum$).

Exercise on Example vii

The gradient with respect to the output of node 4 is given by $\delta^{(4)}$ in Equation (42). The activation of node 4 is $y_4 = \sigma(u)$, $u = \theta_1 r^{(1)} + \theta_3 r^{(2)}$, so from the single-variable chain rule

$$\frac{\partial \mathcal{J}}{\partial u} = \frac{\partial \mathcal{J}}{\partial y_4}\sigma'(u) = \delta^{(4)} \cdot \sigma(u)(1 - \sigma(u)). \tag{43}$$

Furthermore, from the parameter dependency of u

$$\frac{\partial \mathcal{J}}{\partial \theta_1} \leftarrow \frac{\partial \mathcal{J}}{\partial \theta_1} + \frac{\partial \mathcal{J}}{\partial u} \cdot r^{(1)} = G_1 + \frac{\partial \mathcal{J}}{\partial u}\, r^{(1)}, \tag{44}$$

$$\frac{\partial \mathcal{J}}{\partial \theta_3} \leftarrow \frac{\partial \mathcal{J}}{\partial \theta_3} + \frac{\partial \mathcal{J}}{\partial u} \cdot r^{(2)} = G_3 + \frac{\partial \mathcal{J}}{\partial u}\, r^{(2)}, \tag{45}$$

Exercise on Example viii

. Also, the gradients to input nodes 1, 2 are updated as

$$\delta^{(1)} \leftarrow \delta^{(1)} + \frac{\partial \mathcal{J}}{\partial u}\theta_1, \tag{46}$$

$$\delta^{(2)} \leftarrow \delta^{(2)} + \frac{\partial \mathcal{J}}{\partial u}\theta_3. \tag{47}$$

Processing Node 3 (Softmax).

At node 3, softmax is applied to the input (z_1, z_2) to obtain the output (s_1, s_2). From the standard derivation for the combination of cross entropy $-\log s_2$ and softmax (chain rule using the Jacobian as a matrix)

$$\frac{\partial \mathcal{J}}{\partial z_j} = s_j - t_j, \quad t = (0, 1) \tag{48}$$

Exercise on Example ix

holds. Here, since there is an additional gradient of $2y_5$ in the s_1 direction as a contribution from node 5 (Equation (41)), applying Equation (**??**) of Definition **??** gives

$$\frac{\partial \mathcal{J}}{\partial z_1} = s_1 + 2y_5 \cdot s_1(1 - s_1), \tag{49}$$

$$\frac{\partial \mathcal{J}}{\partial z_2} = (s_2 - 1) - 2y_5 \cdot s_1 s_2. \tag{50}$$

This corresponds to $\delta_1^{(3)}, \delta_2^{(3)}$.

Exercise on Example x

The gradients to the parameters and input nodes are, from a form similar to Equation (**??**), updated as

$$\frac{\partial \mathcal{J}}{\partial \theta_1} \leftarrow \frac{\partial \mathcal{J}}{\partial \theta_1} + \frac{\partial \mathcal{J}}{\partial z_1} \cdot r^{(1)}, \tag{51}$$

$$\frac{\partial \mathcal{J}}{\partial \theta_2} \leftarrow \frac{\partial \mathcal{J}}{\partial \theta_2} + \frac{\partial \mathcal{J}}{\partial z_2} \cdot r^{(2)}, \tag{52}$$

$$\delta^{(1)} \leftarrow \delta^{(1)} + \frac{\partial \mathcal{J}}{\partial z_1}\theta_1, \tag{53}$$

$$\delta^{(2)} \leftarrow \delta^{(2)} + \frac{\partial \mathcal{J}}{\partial z_2}\theta_2. \tag{54}$$

Processing Node 2 and Node 1 (Input Nodes).

Exercise on Example xi

Node 2 and Node 1 are nodes that only receive external inputs and have no further parent nodes. Therefore, when they are reached, $\delta^{(1)}, \delta^{(2)}$ have become their final values, and these can be used to read the sensitivity to the inputs.

Finally, by aggregating all of Equation (44), (45), (51), (52), and the contributions from node 5 (40)–(42), the gradients with respect to the parameters $\theta_1, \theta_2, \theta_3$

$$\frac{\partial \mathcal{J}}{\partial \theta_1} = \frac{\partial \mathcal{J}}{\partial z_1} \cdot r^{(1)} + \frac{\partial \mathcal{J}}{\partial u} \cdot r^{(1)}, \tag{55}$$

$$\frac{\partial \mathcal{J}}{\partial \theta_2} = \frac{\partial \mathcal{J}}{\partial z_2} \cdot r^{(2)} + (2y_5) r^{(1)} + (2y_5) s_1, \tag{56}$$

$$\frac{\partial \mathcal{J}}{\partial \theta_3} = \frac{\partial \mathcal{J}}{\partial u} \cdot r^{(2)} + (2y_5) y_4 \tag{57}$$

are obtained. By substituting $r^{(1)} = -1,\ r^{(2)} = +1$ here, a completely specific gradient is calculated. This series of procedures is a concrete calculation showing that the general backpropagation algorithm of Definition **??** gives the correct gradient even in a specific example.

Note: What Happens with Non-differentiable Functions? i

Practical neural networks use many differentiable functions, but also non-differentiable functions like ReLU. In fact, even if not differentiable, the functions practically used as activation functions in neural networks are mostly locally Lipschitz, and for locally Lipschitz functions, the **Clarke subdifferential**, which is a generalization of the gradient vector, can be defined. This is a set of vectors. For example, the subdifferential $\partial^\circ \mathrm{ReLU}$ of ReLU is

$$\partial^\circ \mathrm{ReLU}(x) = \begin{cases} \{0\} & \text{if } x < 0, \\ [0,1] & \text{if } x = 0, \\ \{1\} & \text{if } x > 0, \end{cases} \tag{58}$$

, which matches intuition.

Note: What Happens with Non-differentiable Functions? ii

The problem lies in the chain rule. Although we will not make a rigorous assertion here, the subdifferential of a composite function is a subset of the set consisting of elements obtained by applying the chain rule by arbitrarily taking elements of the subdifferential of each function. It is important that it is a subset, but not equal as a set. In other words, backpropagation based on the chain rule has validity for non-differentiable functions in the sense that if we appropriately choose an element of the subdifferential at each step, we can obtain one element of the overall subdifferential.

On the other hand, if we cannot appropriately choose an element of the subgradient at each step, there is no guarantee that the obtained vector will be in the overall subdifferential. It is difficult to appropriately choose an element of the

Note: What Happens with Non-differentiable Functions? iii

subgradient in the general case. That is where it is theoretically difficult. However, in reality, in PyTorch, the gradient of ReLU at the origin is defined by convention as 0, and moreover, the backpropagation method determined by it works well empirically.

Summary and Next Time

Summary

- The efficiency of local search can be defined by the directional derivative, and the solution to its maximization/minimization problem is given by the gradient vector. The gradient method is a technique of locally and sequentially updating parameters in the direction determined by the gradient vector. For objective functions determined by large-scale data, random vectors with expectation equal to the gradient vector are used as substitutes, and methods such as SGD and AdamW are widely used in the field of generative AI.

Summary

- The efficiency of local search can be defined by the directional derivative, and the solution to its maximization/minimization problem is given by the gradient vector. The gradient method is a technique of locally and sequentially updating parameters in the direction determined by the gradient vector. For objective functions determined by large-scale data, random vectors with expectation equal to the gradient vector are used as substitutes, and methods such as SGD and AdamW are widely used in the field of generative AI.
- The gradient vector of the objective function can be calculated by a combination of forward propagation and backpropagation, not only for MLPs composed of fully connected layers, but also for objective functions defined by feedforward neural networks defined on a general DAG. Forward propagation is dynamic programming in topological sort order, and backpropagation is dynamic programming in its reverse order.

Next topics

Next time, we will explain **parameter-efficient parameter tuning** and its most popular example, **Low-Rank Adaptation (LoRA)**.

References i

[1] Thomas M. Cover and Joy A. Thomas.
Elements of Information Theory.
Wiley, 2 edition, 2006.

[2] Ilya Loshchilov and Frank Hutter.
Decoupled weight decay regularization.
In International Conference on Learning Representations (ICLR) 2019, 2019.

References ii

[3] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever.
Learning transferable visual models from natural language supervision.
In Proceedings of the 38th International Conference on Machine Learning (ICML) 2021, 2021.

[4] Herbert Robbins and Sutton Monro.
A stochastic approximation method.
The Annals of Mathematical Statistics, 22(3):400–407, 1951.

References iii

[5] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer.
High-resolution image synthesis with latent diffusion models.
In IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) 2022, 2022.

[6] Olaf Ronneberger, Philipp Fischer, and Thomas Brox.
U-net: Convolutional networks for biomedical image segmentation.
In Medical Image Computing and Computer-Assisted Intervention (MICCAI) 2015, 2015.

References iv

[7] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams.
Learning representations by back-propagating errors.
Nature, 323(6088):533–536, 1986.

[8] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin.
Attention is all you need.
In Advances in Neural Information Processing Systems 30 (NeurIPS 2017), 2017.