# AI Applications Lecture 17
# Fundamentals of AI Tuning

SUZUKI, Atsushi
Jing WANG

2025-11-14

## Contents

# 1 Introduction

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 handled, such as text or images.

- Qwen3, a Chat AI, is based on scaled dot attention.
- In Stable Diffusion 1.5, the text encoder has a mixed structure of positional encoders, scaled dot attention for self-attention, and MLPs; the noise estimator is a U-Net containing scaled dot attention for cross-attention; and the natural image decoder was a VAE composed of a convolutional neural network [1, 2, 3, 4].
- In the first place, even if we speak of convolutional layers or multi-head attention layers in a single word, if the number of filters or filter width of the convolution, or the $Q, K, V$ and number of heads of the attention are different, the function systems are completely different. In other words, we need to handle as many different function systems as there are models in the world. From a classical statistical perspective, if the function system is slightly different, they have completely different names and fields, such as linear regression, ridge regression, LASSO, and logistic regression. This shows that the field of neural networks must handle a much more diverse range of function systems than those.

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

- Example: Cross entropy in the text field, when a neural network outputs a probability distribution over a set of tokens (see [5]).
- Example: In the image generation field, the contrastive learning loss by CLIP, the squared error in the noise estimator, and the regularized reconstruction squared error in the VAE (in reality, it is more complex than this) [4, 3].

The function systems and objective functions differ this much depending on the field and component. Nevertheless, 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.

# 2 Preliminaries: Mathematical Notations Revisited

- **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: $\mathcal{A}$.
  - $x \in \mathcal{A}$: Indicates that the element $x$ belongs to the set $\mathcal{A}$.
  - $\{\}$: The empty set.
  - $\{a, b, c\}$: The set consisting of elements $a, b, c$ (set extension notation).
  - $\{x \in \mathcal{A} \mid P(x)\}$: The set of elements in set $\mathcal{A}$ for which the proposition $P(x)$ is true (set comprehension 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.
  - $\mathbb{Z}_{\geq 0}$: The set of all non-negative integers.
  - $[1, k]_{\mathbb{Z}} \coloneqq \{1, 2, \ldots, k\}$: For a positive integer $k$, the set of integers from 1 to $k$.
- **Function:**
  - $f : \mathcal{X} \to \mathcal{Y}$: Indicates that the function $f$ is a map that takes an element of set $\mathcal{X}$ as input and outputs an element of set $\mathcal{Y}$.
  - $y = f(x)$: Indicates that the output is $y \in \mathcal{Y}$ when $x \in \mathcal{X}$ is input to the function $f$.
- **Vector:**

- In this course, a vector refers to a column of numbers arranged vertically.
- Vectors are denoted by bold italic lowercase letters. Example: $\boldsymbol{v}$.
- $\boldsymbol{v} \in \mathbb{R}^n$: Indicates that the vector $\boldsymbol{v}$ is an $n$-dimensional real vector.
- The $i$-th element of a vector $\boldsymbol{v}$ is denoted by $v_i$.

$$\boldsymbol{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}. \tag{1}$$

- **Matrix:**
  - Matrices are denoted by bold italic uppercase letters. Example: $\boldsymbol{A}$.
  - $\boldsymbol{A} \in \mathbb{R}^{m,n}$: Indicates that the matrix $\boldsymbol{A}$ is an $m \times n$ real matrix.
  - The element in the $i$-th row and $j$-th column of matrix $\boldsymbol{A}$ is denoted by $a_{i,j}$.

$$\boldsymbol{A} = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}. \tag{2}$$

  - The transpose of matrix $\boldsymbol{A}$ is denoted by $\boldsymbol{A}^\top$. If $\boldsymbol{A} \in \mathbb{R}^{m,n}$, then $\boldsymbol{A}^\top \in \mathbb{R}^{n,m}$, and

$$\boldsymbol{A}^\top = \begin{bmatrix} a_{1,1} & a_{2,1} & \cdots & a_{m,1} \\ a_{1,2} & a_{2,2} & \cdots & a_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1,n} & a_{2,n} & \cdots & a_{m,n} \end{bmatrix} \tag{3}$$

    .
  - A vector is also a matrix with 1 column, and its transpose can also be defined.

$$\boldsymbol{v}^\top = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix} \in \mathbb{R}^{1,n} \tag{4}$$

    .

- **Tensor:**
  - In this lecture, the word tensor simply refers to a multi-dimensional array. A vector can be regarded as a 1st-order tensor, and a matrix as a 2nd-order tensor. Tensors of 3rd-order or higher are denoted by underlined bold italic uppercase letters, like $\underline{\boldsymbol{A}}$.

– Students who have already learned about abstract tensors in mathematics or physics may be resistant to calling a mere multi-dimensional array a tensor. If we assume that the basis is always fixed to the standard basis and identify the tensor in the mathematical sense with its component representation (which becomes a multi-dimensional array), then the terminology is (arguably) consistent.

# 3 Function Gradient Vectors and the Idea of Gradient Methods

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 $f_{\boldsymbol{\theta}}$, so in the end, what we determine is the parameter $\boldsymbol{\theta}$.

When we say we determine the parameter $\boldsymbol{\theta}$, we must first provide information to the computer, whether indirectly or directly, about what kind of parameter is desirable for solving the task we are interested in. The objective function is what quantitatively provides this direct information about what kind of parameter is desirable. The objective function takes the parameter as an argument and returns a real value representing its goodness.

**Remark 3.1.** In many cases in the AI field, the objective function is designed so that the smaller its value, the better the parameter, so it can also be called a **cost function**. Furthermore, even if a larger value of the objective function means a better parameter, we can consider a new function with a negative sign and treat it as a cost function, so we do not lose generality by considering only the case where a smaller value of the objective function is better. In this lecture, we will follow this convention and hereafter only deal with the case where smaller is better for the objective function value.

By the way, designing an objective function from scratch is often difficult. Cases where this is possible are situations where humanity has already gained considerable knowledge about the task itself. For non-trivial tasks where this is not the case, the objective function is designed using past data. That is, even if we cannot directly determine how the parametric function should behave, we quantify the desire for the task to succeed in some sense, at least on past data. This is the basic framework of machine learning.

**Example 3.1.** 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 the prediction to work well on it into a squared error function, and design the objective function as

$$\mathcal{L}(\boldsymbol{\theta}) \coloneqq \frac{1}{m} \sum_{i=1}^{m} \left\| \boldsymbol{y}^{(i)} - f_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) \right\|_2^2 \tag{5}$$

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

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, namely

$$\underset{\boldsymbol{\theta}\in\mathbb{R}^{d_{\text{param}}}}{\text{Minimize}}\ \mathcal{L}(\boldsymbol{\theta}) \tag{6}$$

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

# 4 Local Search, Directional Derivatives, and Gradients

## 4.1 The Impossibility of Brute Force Search

In modern generative AI models with a huge number of parameters, brute force search is computationally impossible, no matter how much computing power has advanced. First, given an objective function, we define the brute force method as follows.

**Definition 4.1** (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, it is truncated within a finite range). At this time, **brute force search** is a 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.

**Example 4.1.** Suppose there are $d_{\text{param}} = 10^9$ (1B) parameters, and each component takes $2^8 = 256$ possible values, equivalent to **float8**. The total number of candidate points is $256^{10^9} = 2^{8\cdot 10^9}$. This is

$$256^{10^9} = 2^{8\cdot 10^9} \approx 10^{(8\cdot 10^9)\log_{10} 2} \approx 10^{2.4\times 10^9} \tag{7}$$

, an astronomical scale. Even if one step evaluation could be processed at $10^{15}$ FLOP (petaflop) and executed in parallel at $10^{18}$ FLOP per second (exa-scale), it is impossibly out of scale. Therefore, a global exhaustive search is not realistic.

## 4.2 The Strategy of Local Search

Since we found that a global search is impossible, we will consider repeating a local search. A local search is an operation of searching for a better parameter (where $\mathcal{L}$ is smaller) in the vicinity of the current parameter vector, using only information from the current parameter vector or its vicinity. Starting with a random vector or a vector that is already known to be reasonably good as the initial parameter vector $\boldsymbol{\theta}^{[0]}$, we repeat the local search, updating to $\boldsymbol{\theta}^{[1]}$, $\boldsymbol{\theta}^{[2]}$, ... to get closer to a better vector.

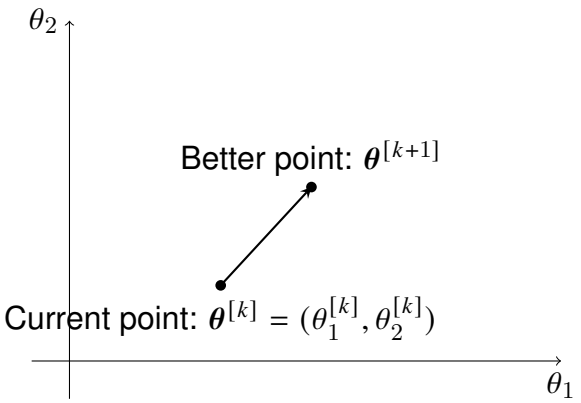



Figure 1: Conceptual diagram of local search from the current point $\boldsymbol{\theta}^{[k]}$ in the $(\theta_1, \theta_2)$-plane.

## 4.3 Directional Derivative: Rate of Change of the Function Value per Direction

As an efficient local search, we adopt the idea of slightly updating the parameter vector in the direction that reduces the value of the objective function as efficiently as possible. The rate of change of the objective function when the parameter vector is moved in the direction of a certain unit vector $\boldsymbol{u}$ is expressed by the **one-sided directional derivative**.

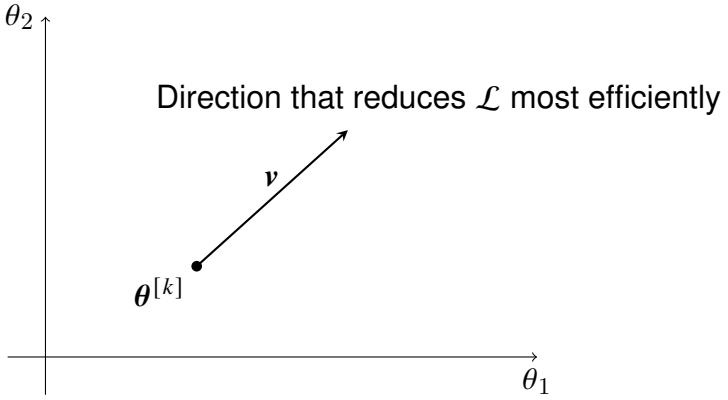



Figure 2: Conceptual diagram of the update direction.

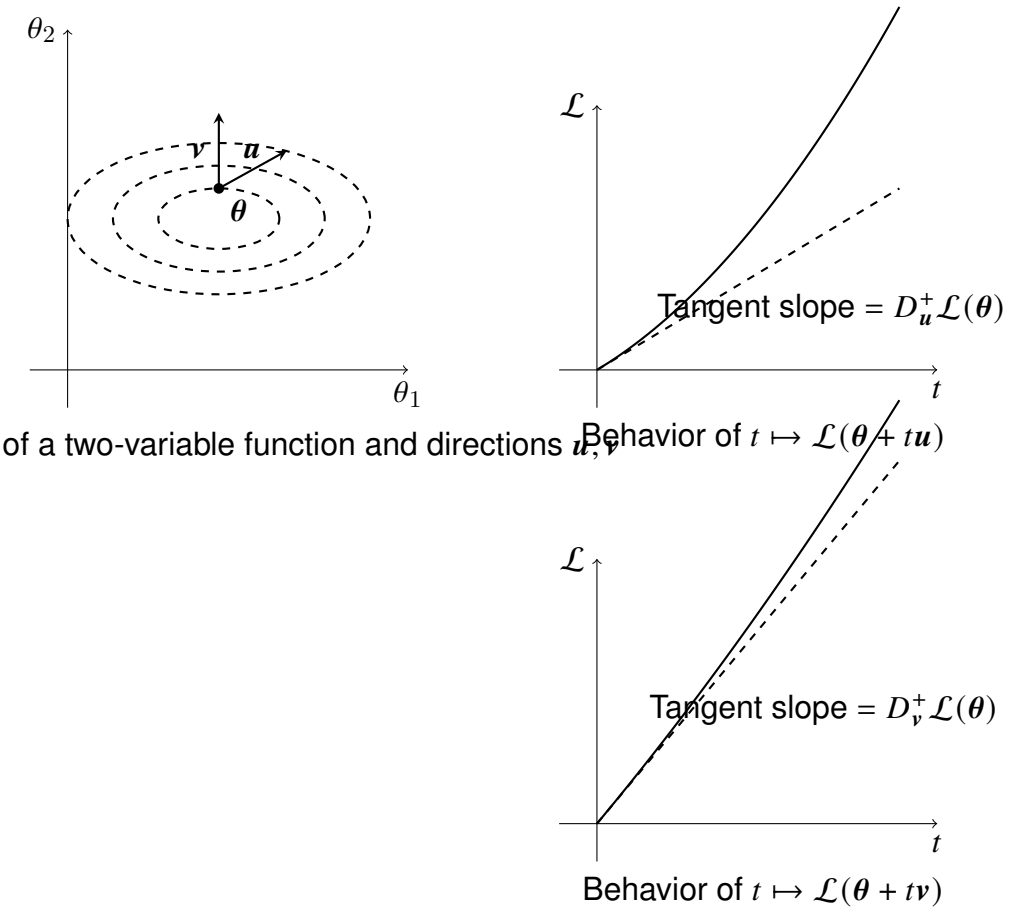
**Remark 4.1.** The one-sided directional derivative itself can be defined even for a direction vector $\boldsymbol{u}$ that is not a unit vector. However, the "speed of advance" when measuring the rate of change is no longer $1$, so if we want to look at the rate of change fairly for each direction, we should observe the one-sided directional derivative for unit vectors.

**Definition 4.2** (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 defined as

$$D_{\boldsymbol{u}}^{+}\mathcal{L}(\boldsymbol{\theta}) \coloneqq \lim_{t \to 0+} \frac{\mathcal{L}(\boldsymbol{\theta} + t\boldsymbol{u}) - \mathcal{L}(\boldsymbol{\theta})}{t} \tag{8}$$

(if the limit exists).

**Remark 4.2.** If $\mathcal{L}$ is differentiable at the point $\boldsymbol{\theta}$ in the sense defined 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 do not distinguish between the right and left sides, and simply call it the **directional derivative**, and we may write it as $D_{\boldsymbol{u}}\mathcal{L}(\boldsymbol{\theta})$.




Level curves of a two-variable function and directions $\boldsymbol{u}, \boldsymbol{v}$

Figure 3: Graphs of $t \mapsto \mathcal{L}(\boldsymbol{\theta} + t\boldsymbol{u})$ and $t \mapsto \mathcal{L}(\boldsymbol{\theta} + t\boldsymbol{v})$ with respect to the directions $\boldsymbol{u}$ and $\boldsymbol{v}$, and their tangents at $t = 0$. The slopes of the tangents are the one-sided directional derivatives $D_{\boldsymbol{u}}^{+}\mathcal{L}(\boldsymbol{\theta})$ and $D_{\boldsymbol{v}}^{+}\mathcal{L}(\boldsymbol{\theta})$.

## 4.4 Directional Optimization and the Gradient Vector

We want to know the direction that reduces the value of the objective function as efficiently as possible when performing a local search near the current parameter vector. Since the one-sided directional derivative expresses exactly that efficiency, we just need to know the direction that minimizes the one-sided directional derivative.

We want to maximize/minimize the one-sided directional derivative over the search space of the set of unit vectors $\{\boldsymbol{u} \in \mathbb{R}^d \mid \|\boldsymbol{u}\|_2 = 1\}$.

**Definition 4.3** (Maximization/Minimization Problems of One-sided Directional Derivative)**.** We will call the following optimization problems the maximization/minimization problems of

the one-sided directional derivative.

$$\text{Maximization:} \quad \underset{\|\boldsymbol{u}\|_2=1}{\text{Maximize}} \; D_{\boldsymbol{u}}^{+}\mathcal{L}(\boldsymbol{\theta}), \tag{9}$$

$$\text{Minimization:} \quad \underset{\|\boldsymbol{u}\|_2=1}{\text{Minimize}} \; D_{\boldsymbol{u}}^{+}\mathcal{L}(\boldsymbol{\theta}). \tag{10}$$

Under appropriate assumptions, the solution to this problem is the famous **gradient vector**. We will define the gradient vector first.

**Definition 4.4** (Gradient Vector)**.** When $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$ has all partial derivatives with respect to each component

$$\frac{\partial \mathcal{L}}{\partial \theta_i}(\boldsymbol{\theta}) \coloneqq \lim_{t \to 0} \frac{\mathcal{L}(\boldsymbol{\theta} + t\boldsymbol{e}_i) - \mathcal{L}(\boldsymbol{\theta})}{t} \tag{11}$$

(where $\boldsymbol{e}_i$ is the standard basis vector with $1$ only in the $i$-th component) at the point $\boldsymbol{\theta} \in \mathbb{R}^d$, the **gradient** is defined by

$$\nabla \mathcal{L}(\boldsymbol{\theta}) \coloneqq \begin{bmatrix} \dfrac{\partial \mathcal{L}}{\partial \theta_1}(\boldsymbol{\theta}) \\ \vdots \\ \dfrac{\partial \mathcal{L}}{\partial \theta_d}(\boldsymbol{\theta}) \end{bmatrix}. \tag{12}$$

Each component of the gradient vector is defined by a partial derivative with respect to one variable. This characteristic makes the gradient vector relatively easy to handle computationally.

Now, we want to see that the gradient vector becomes the solution to the maximization or minimization problems of the one-sided directional derivative, that is, it gives the locally most efficient local search method, but mathematically, conditions are required for that to hold. A useful sufficient condition is differentiability, so we define the differentiability of a multivariate function. In short, being differentiable means being smooth enough that a linear approximation exists.

**Definition 4.5** (Differentiability of Multivariate Functions)**.** Take a function $\mathcal{L} : U \to \mathbb{R}$ on an open set $U \subset \mathbb{R}^d$ and a point $\boldsymbol{\theta} \in U$. If a linear map $A : \mathbb{R}^d \to \mathbb{R}$ exists such that

$$\lim_{\boldsymbol{h} \to \boldsymbol{0}} \frac{\mathcal{L}(\boldsymbol{\theta} + \boldsymbol{h}) - \mathcal{L}(\boldsymbol{\theta}) - A(\boldsymbol{h})}{\|\boldsymbol{h}\|_2} = 0 \tag{13}$$

is satisfied, we say that $\mathcal{L}$ is **differentiable** at the point $\boldsymbol{\theta}$, and call such a linear map $A$ the **derivative** of $\mathcal{L}$ at the point $\boldsymbol{\theta}$, and write it as $D\mathcal{L}(\boldsymbol{\theta})$.

Of course, knowing a sufficient condition that cannot be checked is of little significance in application, but regarding neural networks, they are constructed by composing simple functions, so there are many situations where differentiability is easy to check. This is because, as described next, a function constructed by composing differentiable functions is also differentiable.

Linear functions are clearly differentiable, and the differentiability of single-variable functions is a topic often studied even at the high school level. In fact, many single-variable functions that are important in applications are differentiable. For this reason, whether the function represented by a neural network is differentiable can often be checked in practical situations.

Now, let's state mathematically that a function constructed by composing differentiable functions is also differentiable.

**Proposition 4.1** (Differentiability of Composite Functions)**.** Take open sets $U \subset \mathbb{R}^d$, $V \subset \mathbb{R}^m$, and functions $\mathcal{L}_1 : U \to V$, $\mathcal{L}_2 : V \to \mathbb{R}$. Assume that $\mathcal{L}_1$ is differentiable at $\boldsymbol{\theta} \in U$, and $\mathcal{L}_2$ is differentiable at the point $\mathcal{L}_1(\boldsymbol{\theta}) \in V$. Then, the composite function $\mathcal{L}_2 \circ \mathcal{L}_1 : U \to \mathbb{R}$ is differentiable at the point $\boldsymbol{\theta}$, and its derivative is given by

$$D(\mathcal{L}_2 \circ \mathcal{L}_1)(\boldsymbol{\theta}) = D\mathcal{L}_2(\mathcal{L}_1(\boldsymbol{\theta})) \circ D\mathcal{L}_1(\boldsymbol{\theta}). \tag{14}$$

For a composition of a finite number of functions, it is similarly inductively differentiable, and the derivative is given by the composition of the respective derivatives.

*Proof.* At point $\boldsymbol{\theta} \in U$, write $D\mathcal{L}_1(\boldsymbol{\theta}) =: A_1 : \mathbb{R}^d \to \mathbb{R}^m$, and at point $\mathcal{L}_1(\boldsymbol{\theta}) \in V$, write $D\mathcal{L}_2(\mathcal{L}_1(\boldsymbol{\theta})) =: A_2 : \mathbb{R}^m \to \mathbb{R}$. By definition,

$$\mathcal{L}_1(\boldsymbol{\theta} + \boldsymbol{h}) = \mathcal{L}_1(\boldsymbol{\theta}) + A_1(\boldsymbol{h}) + \boldsymbol{r}_1(\boldsymbol{h}), \quad \lim_{\boldsymbol{h} \to \boldsymbol{0}} \frac{\|\boldsymbol{r}_1(\boldsymbol{h})\|_2}{\|\boldsymbol{h}\|_2} = 0, \tag{15}$$

$$\mathcal{L}_2(\boldsymbol{z} + \boldsymbol{k}) = \mathcal{L}_2(\boldsymbol{z}) + A_2(\boldsymbol{k}) + r_2(\boldsymbol{z}, \boldsymbol{k}), \quad \lim_{\boldsymbol{k} \to \boldsymbol{0}} \frac{|r_2(\mathcal{L}_1(\boldsymbol{\theta}), \boldsymbol{k})|}{\|\boldsymbol{k}\|_2} = 0. \tag{16}$$

Here, let $\boldsymbol{z} = \mathcal{L}_1(\boldsymbol{\theta})$ and $\boldsymbol{k} = A_1(\boldsymbol{h}) + \boldsymbol{r}_1(\boldsymbol{h})$. Then

$$\begin{aligned}(\mathcal{L}_2 \circ \mathcal{L}_1)(\boldsymbol{\theta} + \boldsymbol{h}) &= \mathcal{L}_2(\mathcal{L}_1(\boldsymbol{\theta}) + A_1(\boldsymbol{h}) + \boldsymbol{r}_1(\boldsymbol{h})) \\ &= \mathcal{L}_2(\mathcal{L}_1(\boldsymbol{\theta})) + A_2(A_1(\boldsymbol{h}) + \boldsymbol{r}_1(\boldsymbol{h})) + r_2(\mathcal{L}_1(\boldsymbol{\theta}), A_1(\boldsymbol{h}) + \boldsymbol{r}_1(\boldsymbol{h})) \\ &= (\mathcal{L}_2 \circ \mathcal{L}_1)(\boldsymbol{\theta}) + (A_2 \circ A_1)(\boldsymbol{h}) + R(\boldsymbol{h}) \end{aligned} \tag{17}$$

can be written. Here

$$R(\boldsymbol{h}) \coloneqq A_2(\boldsymbol{r}_1(\boldsymbol{h})) + r_2(\mathcal{L}_1(\boldsymbol{\theta}), A_1(\boldsymbol{h}) + \boldsymbol{r}_1(\boldsymbol{h})) \tag{18}$$

was set. Since $A_2$ is a continuous linear map, there exists a constant $C > 0$ satisfying $\|A_2(\boldsymbol{r}_1(\boldsymbol{h}))\| \leq C\|\boldsymbol{r}_1(\boldsymbol{h})\|_2$, and

$$\lim_{\boldsymbol{h} \to \boldsymbol{0}} \frac{|A_2(\boldsymbol{r}_1(\boldsymbol{h}))|}{\|\boldsymbol{h}\|_2} \leq C \lim_{\boldsymbol{h} \to \boldsymbol{0}} \frac{\|\boldsymbol{r}_1(\boldsymbol{h})\|_2}{\|\boldsymbol{h}\|_2} = 0 \tag{19}$$

follows. Also, since $A_1$ is linear, there exists a constant $C' > 0$ satisfying $\|A_1(\boldsymbol{h})\|_2 \leq C'\|\boldsymbol{h}\|_2$,

and when $\boldsymbol{h} \to \mathbf{0}$, $A_1(\boldsymbol{h}) \to \mathbf{0}$ and $\boldsymbol{r}_1(\boldsymbol{h}) \to \mathbf{0}$, so $A_1(\boldsymbol{h}) + \boldsymbol{r}_1(\boldsymbol{h}) \to \mathbf{0}$. Therefore, by definition

$$\lim_{\boldsymbol{h}\to\mathbf{0}} \frac{|r_2(\mathcal{L}_1(\boldsymbol{\theta}), A_1(\boldsymbol{h}) + \boldsymbol{r}_1(\boldsymbol{h}))|}{\|\boldsymbol{h}\|_2} = \lim_{\boldsymbol{h}\to\mathbf{0}} \left( \frac{|r_2(\mathcal{L}_1(\boldsymbol{\theta}), A_1(\boldsymbol{h}) + \boldsymbol{r}_1(\boldsymbol{h}))|}{\|A_1(\boldsymbol{h}) + \boldsymbol{r}_1(\boldsymbol{h})\|_2} \cdot \frac{\|A_1(\boldsymbol{h}) + \boldsymbol{r}_1(\boldsymbol{h})\|_2}{\|\boldsymbol{h}\|_2} \right) = 0. \tag{20}$$

From (19) and (20)

$$\lim_{\boldsymbol{h}\to\mathbf{0}} \frac{|R(\boldsymbol{h})|}{\|\boldsymbol{h}\|_2} = 0 \tag{21}$$

follows, so the form of (13) holds for $\mathcal{L}_2 \circ \mathcal{L}_1$ and the linear map $A_2 \circ A_1$. Therefore (14) is shown. For a finite number of compositions, it follows similarly by induction. □

Now that we are prepared, we will rigorously show that under the condition of differentiability, the directions that maximize/minimize the one-sided directional derivative are given by the gradient vector.

**Theorem 4.1** (Directions of Steepest Ascent/Descent are Given by the Gradient Vector)**.** Take a function $\mathcal{L} : U \to \mathbb{R}$ on an open set $U \subset \mathbb{R}^d$ and a point $\boldsymbol{\theta} \in U$. Assume that $\mathcal{L}$ is differentiable at the point $\boldsymbol{\theta}$, and that the partial derivatives $\frac{\partial \mathcal{L}}{\partial \theta_i}(\boldsymbol{\theta})$ $(i = 1, \ldots, d)$ exist. Then the following hold.

1. For any unit vector $\boldsymbol{u} \in \mathbb{R}^d$ $(\|\boldsymbol{u}\|_2 = 1)$, the one-sided directional derivative $D^+_{\boldsymbol{u}}\mathcal{L}(\boldsymbol{\theta})$ exists, and

   $$D^+_{\boldsymbol{u}}\mathcal{L}(\boldsymbol{\theta}) = \nabla\mathcal{L}(\boldsymbol{\theta})^\top \boldsymbol{u} \tag{22}$$

   holds. In particular, since the limits from both the left and right sides exist and coincide, we may write $D_{\boldsymbol{u}}\mathcal{L}(\boldsymbol{\theta})$ in this case.

2. When $\nabla\mathcal{L}(\boldsymbol{\theta}) \neq \mathbf{0}$, the solution to the maximization problem (9) is

   $$\boldsymbol{u}^\star = \frac{\nabla\mathcal{L}(\boldsymbol{\theta})}{\|\nabla\mathcal{L}(\boldsymbol{\theta})\|_2} \tag{23}$$

   , and the maximum value at that time is

   $$D^+_{\boldsymbol{u}^\star}\mathcal{L}(\boldsymbol{\theta}) = \|\nabla\mathcal{L}(\boldsymbol{\theta})\|_2 \tag{24}$$

   . On the other hand, when $\nabla\mathcal{L}(\boldsymbol{\theta}) = \mathbf{0}$, $D^+_{\boldsymbol{u}}\mathcal{L}(\boldsymbol{\theta}) = 0$ for all unit vectors $\boldsymbol{u}$, and any unit vector is a solution to the maximization problem.

3. When $\nabla\mathcal{L}(\boldsymbol{\theta}) \neq \mathbf{0}$, the solution to the minimization problem (10) is

   $$\boldsymbol{u}_\star = -\frac{\nabla\mathcal{L}(\boldsymbol{\theta})}{\|\nabla\mathcal{L}(\boldsymbol{\theta})\|_2} \tag{25}$$

   , and the minimum value at that time is

   $$D^+_{\boldsymbol{u}_\star}\mathcal{L}(\boldsymbol{\theta}) = -\|\nabla\mathcal{L}(\boldsymbol{\theta})\|_2 \tag{26}$$

. On the other hand, when $\nabla\mathcal{L}(\boldsymbol{\theta}) = \mathbf{0}$, $D^+_{\boldsymbol{u}}\mathcal{L}(\boldsymbol{\theta}) = 0$ for all unit vectors $\boldsymbol{u}$, and any unit vector is also a solution to the minimization problem.

*Proof.* First, we show (1). From the assumption of differentiability at point $\boldsymbol{\theta}$, there exists a linear map $A : \mathbb{R}^d \to \mathbb{R}$ such that

$$\mathcal{L}(\boldsymbol{\theta} + \boldsymbol{h}) = \mathcal{L}(\boldsymbol{\theta}) + A(\boldsymbol{h}) + r(\boldsymbol{h}) \tag{27}$$

and

$$\lim_{\boldsymbol{h}\to\mathbf{0}} \frac{r(\boldsymbol{h})}{\|\boldsymbol{h}\|_2} = 0 \tag{28}$$

hold. For any unit vector $\boldsymbol{u}$ and real number $t \neq 0$, let $\boldsymbol{h} = t\boldsymbol{u}$, then

$$\frac{\mathcal{L}(\boldsymbol{\theta} + t\boldsymbol{u}) - \mathcal{L}(\boldsymbol{\theta})}{t} = \frac{A(t\boldsymbol{u})}{t} + \frac{r(t\boldsymbol{u})}{t} = A(\boldsymbol{u}) + \frac{r(t\boldsymbol{u})}{t}. \tag{29}$$

Here, since $\|\boldsymbol{u}\|_2 = 1$, as $t \to 0$, $\|t\boldsymbol{u}\|_2 = |t| \to 0$, and

$$\left|\frac{r(t\boldsymbol{u})}{t}\right| = \frac{\|r(t\boldsymbol{u})\|}{\|t\boldsymbol{u}\|_2} \cdot \|\boldsymbol{u}\|_2 \xrightarrow[t\to 0]{} 0 \tag{30}$$

follows. Therefore, the limit as $t \to 0$ exists and

$$\lim_{t\to 0} \frac{\mathcal{L}(\boldsymbol{\theta} + t\boldsymbol{u}) - \mathcal{L}(\boldsymbol{\theta})}{t} = A(\boldsymbol{u}). \tag{31}$$

In particular, the limit as $t \to 0+$ also converges to the same value $A(\boldsymbol{u})$, so $D^+_{\boldsymbol{u}}\mathcal{L}(\boldsymbol{\theta})$ exists, and

$$D^+_{\boldsymbol{u}}\mathcal{L}(\boldsymbol{\theta}) = A(\boldsymbol{u}). \tag{32}$$

Next, we express $A$ using the gradient vector. Take the standard basis vectors $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_d$, and define $g_i \coloneqq A(\boldsymbol{e}_i)$. By linearity, for any $\boldsymbol{h} = \sum_{i=1}^d h_i \boldsymbol{e}_i$,

$$A(\boldsymbol{h}) = \sum_{i=1}^d h_i A(\boldsymbol{e}_i) = \sum_{i=1}^d g_i h_i \tag{33}$$

holds. On the other hand, considering the single-variable function

$$\varphi_i(t) \coloneqq \mathcal{L}(\boldsymbol{\theta} + t\boldsymbol{e}_i), \tag{34}$$

and applying (27) to $\boldsymbol{h} = t\boldsymbol{e}_i$, we get

$$\frac{\varphi_i(t) - \varphi_i(0)}{t} = \frac{\mathcal{L}(\boldsymbol{\theta} + t\boldsymbol{e}_i) - \mathcal{L}(\boldsymbol{\theta})}{t} = A(\boldsymbol{e}_i) + \frac{r(t\boldsymbol{e}_i)}{t}. \tag{35}$$

As $t \to 0$, the second term on the right-hand side converges to $0$ similarly to (30), so

$$\frac{\partial \mathcal{L}}{\partial \theta_i}(\boldsymbol{\theta}) = \varphi_i'(0) = A(\boldsymbol{e}_i) = g_i \tag{36}$$

. Therefore $g_i = \frac{\partial \mathcal{L}}{\partial \theta_i}(\boldsymbol{\theta})$, and

$$A(\boldsymbol{h}) = \sum_{i=1}^{d} h_i \frac{\partial \mathcal{L}}{\partial \theta_i}(\boldsymbol{\theta}) = \nabla \mathcal{L}(\boldsymbol{\theta})^\top \boldsymbol{h} \tag{37}$$

holds for any $\boldsymbol{h}$. In particular, setting $\boldsymbol{h} = \boldsymbol{u}$,

$$A(\boldsymbol{u}) = \nabla \mathcal{L}(\boldsymbol{\theta})^\top \boldsymbol{u}, \tag{38}$$

and substituting this into (32) yields (22). This proves (1).

Next, we show (2) and (3). For simplicity,

$$\boldsymbol{g} \coloneqq \nabla \mathcal{L}(\boldsymbol{\theta}) \tag{39}$$

is set. From (1), the one-sided directional derivative for a unit vector $\boldsymbol{u}$ is

$$D_{\boldsymbol{u}}^{+} \mathcal{L}(\boldsymbol{\theta}) = \boldsymbol{g}^\top \boldsymbol{u}, \tag{40}$$

so it is equivalent to the problem of maximizing/minimizing $\boldsymbol{g}^\top \boldsymbol{u}$ under the constraint $\|\boldsymbol{u}\|_2 = 1$.

First, assume $\boldsymbol{g} \neq \boldsymbol{0}$. For any unit vector $\boldsymbol{u}$, the squared Euclidean norm of the vector

$$\boldsymbol{u} - \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|_2} \tag{41}$$

is non-negative, and

$$\begin{aligned} 0 &\leq \left\| \boldsymbol{u} - \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|_2} \right\|_2^2 \\ &= \|\boldsymbol{u}\|_2^2 - 2\frac{\boldsymbol{g}^\top \boldsymbol{u}}{\|\boldsymbol{g}\|_2} + \left\| \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|_2} \right\|_2^2 \\ &= 1 - 2\frac{\boldsymbol{g}^\top \boldsymbol{u}}{\|\boldsymbol{g}\|_2} + 1 \\ &= 2\left(1 - \frac{\boldsymbol{g}^\top \boldsymbol{u}}{\|\boldsymbol{g}\|_2}\right). \end{aligned} \tag{42}$$

Therefore

$$1 - \frac{\boldsymbol{g}^\top \boldsymbol{u}}{\|\boldsymbol{g}\|_2} \geq 0 \quad \Longrightarrow \quad \boldsymbol{g}^\top \boldsymbol{u} \leq \|\boldsymbol{g}\|_2 \tag{43}$$

is obtained. Equality holds only when the norm of (41) is $0$, i.e.,

$$\boldsymbol{u} = \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|_2}. \tag{44}$$

Therefore, the solution that maximizes $\boldsymbol{g}^\top \boldsymbol{u}$ under the constraint $\|\boldsymbol{u}\|_2 = 1$ is given by (44), and the maximum value is $\|\boldsymbol{g}\|_2$. Substituting (40) and $\boldsymbol{g} = \nabla \mathcal{L}(\boldsymbol{\theta})$ yields (23) and (24).

For the minimum value, similarly, by expanding the squared norm of the vector

$$\boldsymbol{u} + \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|_2} \tag{45}$$

for any unit vector $\boldsymbol{u}$,

$$\boldsymbol{g}^\top \boldsymbol{u} \geq -\|\boldsymbol{g}\|_2 \tag{46}$$

is obtained, and equality holds only if

$$\boldsymbol{u} = -\frac{\boldsymbol{g}}{\|\boldsymbol{g}\|_2}. \tag{47}$$

Therefore, the minimum value of $\boldsymbol{g}^\top \boldsymbol{u}$ is $-\|\boldsymbol{g}\|_2$, and the solution to the minimization problem is (47). Again, using (40) and $\boldsymbol{g} = \nabla \mathcal{L}(\boldsymbol{\theta})$ yields (25) and (26).

Finally, consider the case $\boldsymbol{g} = \boldsymbol{0}$. In this case, for any unit vector $\boldsymbol{u}$,

$$D^+_{\boldsymbol{u}} \mathcal{L}(\boldsymbol{\theta}) = \boldsymbol{g}^\top \boldsymbol{u} = 0, \tag{48}$$

so the maximum and minimum values are both $0$, and any $\boldsymbol{u}$ satisfying $\|\boldsymbol{u}\|_2 = 1$ is a solution to the maximization and minimization problems. With this, (2) and (3) are also shown. □

## 4.5 Orthogonality of Level Sets and Gradients

One feature that helps understand the gradient vector is that, under appropriate assumptions, the gradient vector is orthogonal to the level surface of the objective function. The level surface of the objective function is the set of points where the value of the objective function does not change from the current one. If the gradient vector were not orthogonal to the level surface, it would mean that the gradient vector has a component in the direction included in the level surface, but from the definition of the level surface, the direction included in the level surface does not locally change the value of the objective function, so it is a wasteful component when we want to efficiently change the value of the objective function. The fact that the gradient vector is orthogonal to the level surface means that the gradient vector does not contain such wasteful components. The orthogonality of the gradient vector to the level surface can be stated mathematically as follows.

**Proposition 4.2** (Gradient is Orthogonal to Level Sets)**.** Assume that a function $\mathcal{L} : U \to \mathbb{R}$

on an open set $U \subset \mathbb{R}^d$ is differentiable at all points in $U$. Fix $c \in \mathbb{R}$, and let the **level set** be

$$S \coloneqq \{\boldsymbol{\theta} \in U \mid \mathcal{L}(\boldsymbol{\theta}) = c\}. \tag{49}$$

Take one point $\boldsymbol{\theta}_0 \in S$, and assume $\nabla\mathcal{L}(\boldsymbol{\theta}_0) \neq \mathbf{0}$. If $\boldsymbol{w} \in \mathbb{R}^d$ is a **tangent vector** to $S$ at $\boldsymbol{\theta}_0$, i.e., there exists some $\epsilon > 0$ and a $C^1$ curve (a vector-valued function whose components are single-variable $C^1$ functions) $\gamma : (-\epsilon, \epsilon) \to S$ such that

$$\gamma(0) = \boldsymbol{\theta}_0, \qquad \gamma'(0) = \boldsymbol{w} \tag{50}$$

is satisfied, then the following holds:

$$\nabla\mathcal{L}(\boldsymbol{\theta}_0)^\top \boldsymbol{w} = 0. \tag{51}$$

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

*Proof.* Since $\boldsymbol{w}$ is a tangent vector, there exists a $C^1$ curve $\gamma$ satisfying (50). Since $\gamma(t) \in S$, for all $t \in (-\epsilon, \epsilon)$,

$$\mathcal{L}(\gamma(t)) = c \tag{52}$$

holds. Here, considering the single-variable function

$$\phi(t) \coloneqq \mathcal{L}(\gamma(t)), \tag{53}$$

from (52), $\phi(t) \equiv c$, so

$$\phi'(0) = 0 \tag{54}$$

follows. On the other hand, since $\gamma$ is $C^1$ and $\mathcal{L}$ is differentiable on $U$, applying the differentiability of composite functions (chain rule in continuous time) from Proposition 4.1 to $\mathcal{L}_1 = \gamma$, $\mathcal{L}_2 = \mathcal{L}$, $\phi = \mathcal{L}_2 \circ \mathcal{L}_1$ is differentiable at $t = 0$, and

$$\phi'(0) = D\mathcal{L}(\gamma(0))(\gamma'(0)) = D\mathcal{L}(\boldsymbol{\theta}_0)(\boldsymbol{w}) \tag{55}$$

holds. Furthermore, as shown in the proof of Theorem 4.1, the derivative $D\mathcal{L}(\boldsymbol{\theta}_0)$ is expressed by the gradient vector $\nabla\mathcal{L}(\boldsymbol{\theta}_0)$ as

$$D\mathcal{L}(\boldsymbol{\theta}_0)(\boldsymbol{h}) = \nabla\mathcal{L}(\boldsymbol{\theta}_0)^\top \boldsymbol{h}, \tag{56}$$

so

$$\phi'(0) = \nabla\mathcal{L}(\boldsymbol{\theta}_0)^\top \boldsymbol{w} \tag{57}$$

. Comparing (54) and (57) yields (51). □

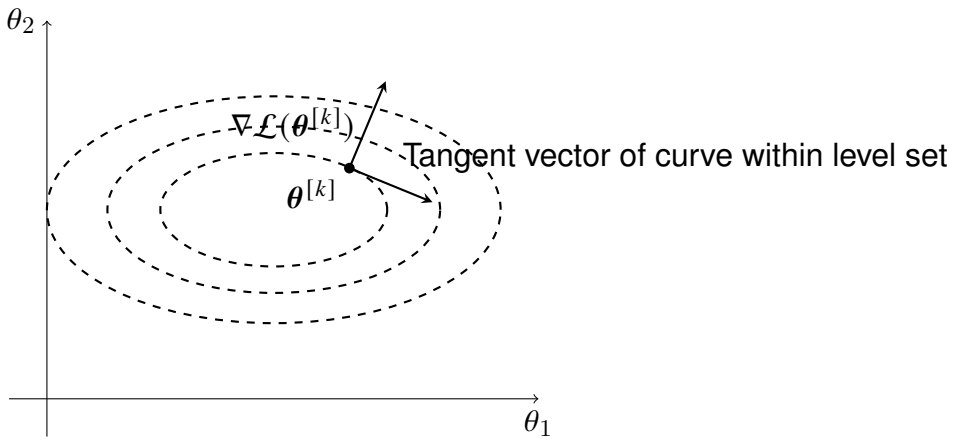



Figure 4: Conceptual diagram of the orthogonality of level curves and the gradient

### 4.6 Definition of the Steepest Descent Method

Since the negative gradient vector represents the direction that can locally reduce the value of the objective function most efficiently, it is a natural idea to repeatedly move the parameter vector in that direction. The **steepest descent method** is a naive implementation of that idea.

**Definition 4.6** (Steepest Descent Method)**.** Given an initial point $\boldsymbol{\theta}^{[0]} \in \mathbb{R}^d$, the number of steps $K \in \mathbb{Z}_{>0}$, and a sequence of learning rates $(\eta^{[k]})_{k=0}^{K-1} \subset \mathbb{R}_{>0}$. The algorithm that updates at each step

$$\boldsymbol{\theta}^{[k+1]} := \boldsymbol{\theta}^{[k]} - \eta^{[k]} \nabla\mathcal{L}(\boldsymbol{\theta}^{[k]}), \quad k = 0, 1, \ldots, K-1 \tag{58}$$

is called the **steepest descent method**.

We look at the steepest descent method from the framework of actual generative AI learning. In actual learning, we collect as much big data as we can get and minimize the objective function determined by it. In situations where we can assume that the quality of the data is uniform, the more data we have, the more information we can get about the task, and the quality of the objective function is considered to improve, so we want to take the policy of using big data for AI if possible. On the other hand, when using big data, the naive steepest descent method causes difficulties from the viewpoint of computational complexity. In the next Section 5, we will describe how to deal with this.

## 5 Practical Gradient Methods: SGD and AdamW

### 5.1 Objective Functions Dependent on Large-Scale Data

Let's consider the case where the objective function depends on big data. More specifically, suppose we use $(\boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)})_{i=1}^m$ and $m$ is large. And suppose the objective function takes the

form

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \ell_{(\boldsymbol{x}^{(i)},\boldsymbol{y}^{(i)})}(\boldsymbol{\theta}). \tag{59}$$

At this time, the gradient vector of this objective function is

$$\nabla\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{m}\sum_{i=1}^{m} \nabla\ell_{(\boldsymbol{x}^{(i)},\boldsymbol{y}^{(i)})}(\boldsymbol{\theta}), \tag{60}$$

but here, the summation calculation $\sum_{i=1}^{m}$ becomes a bottleneck. Therefore, in the actual learning of large-scale architectures using large-scale data, algorithms are actually used that do not take the sum over all $i$ at each step of updating the parameter vector. Various algorithms have been proposed, and for each, some theoretical guarantee is often obtained under some assumptions. However, in reality, there is no way to know what assumptions the objective function defined on mysterious big data satisfies, so in the end, there is an aspect where the algorithm must be chosen based on heuristics. As algorithms frequently used for minimizing the objective function of generative AI, this lecture introduces Stochastic Gradient Descent (SGD) and AdamW. First, as a large framework that includes them, we define the framework of (broadly defined) stochastic gradient methods.

## 5.2 General Framework of Stochastic Gradient Methods

**Definition 5.1** (General Stochastic Gradient Methods)**.** At each step $k$, we can obtain a random variable vector $\boldsymbol{g}^{[k]}$, and assume $\mathbb{E}[\boldsymbol{g}^{[k]} \mid \mathcal{F}^{[k]}] = \nabla\mathcal{L}(\boldsymbol{\theta}^{[k]})$ ($\mathcal{F}^{[k]}$ is past information) is satisfied. A general update rule using all past $\boldsymbol{g}^{[0]}, \dots, \boldsymbol{g}^{[k]}$ is defined as

$$\boldsymbol{h}^{[k]} \coloneqq \Phi_k(\boldsymbol{g}^{[0]}, \dots, \boldsymbol{g}^{[k]}), \quad \boldsymbol{\theta}^{[k+1]} \coloneqq \Psi_k\big(\boldsymbol{\theta}^{[k]}, \boldsymbol{h}^{[k]}\big) \tag{61}$$

(where $\Phi_k, \Psi_k$ are given deterministic maps).

**Example 5.1** (Unbiasedness of Minibatch Gradient)**.** In the situation of (59), take a uniform random minibatch $S^{[k]} \subset \{1, \dots, m\}$ of any size $B$, and let

$$\boldsymbol{g}^{[k]} \coloneqq \frac{1}{|S^{[k]}|}\sum_{i\in S^{[k]}} \nabla\ell_{(\boldsymbol{x}^{(i)},\boldsymbol{y}^{(i)})}(\boldsymbol{\theta}^{[k]}). \tag{62}$$

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

## 5.3 Stochastic Gradient Descent (SGD)

**Definition 5.2** (SGD)**.** Given a sequence of learning rates $(\eta^{[k]})_{k\geq 0}$ and an unbiased estimate $\boldsymbol{g}^{[k]}$ from the minibatch at each step.

$$\boldsymbol{\theta}^{[k+1]} := \boldsymbol{\theta}^{[k]} - \eta^{[k]}\boldsymbol{g}^{[k]}. \tag{63}$$

It is said to originate from the classic Robbins–Monro type stochastic approximation [6].

## 5.4 AdamW

**Definition 5.3** (AdamW [7])**.** Give hyperparameters $\beta_1, \beta_2 \in [0, 1)$, $\epsilon > 0$, learning rate $\eta^{[k]}$, and decay coefficient $\lambda \geq 0$. $\boldsymbol{g}^{[k]}$ is the minibatch gradient estimate. Initialize $\boldsymbol{m}^{[-1]} = \boldsymbol{0}$, $\boldsymbol{v}^{[-1]} = \boldsymbol{0}$. For each $k = 0, 1, \ldots$

$$\boldsymbol{m}^{[k]} := \beta_1\boldsymbol{m}^{[k-1]} + (1-\beta_1)\boldsymbol{g}^{[k]}, \tag{64}$$

$$\boldsymbol{v}^{[k]} := \beta_2\boldsymbol{v}^{[k-1]} + (1-\beta_2)\big(\boldsymbol{g}^{[k]} \odot \boldsymbol{g}^{[k]}\big), \tag{65}$$

$$\widehat{\boldsymbol{m}}^{[k]} := \boldsymbol{m}^{[k]}/(1-\beta_1^{k+1}), \tag{66}$$

$$\widehat{\boldsymbol{v}}^{[k]} := \boldsymbol{v}^{[k]}/(1-\beta_2^{k+1}), \tag{67}$$

$$\boldsymbol{\theta}^{[k+1]} := \boldsymbol{\theta}^{[k]} - \eta^{[k]}\frac{\widehat{\boldsymbol{m}}^{[k]}}{\sqrt{\widehat{\boldsymbol{v}}^{[k]}} + \epsilon} - \eta^{[k]}\lambda\,\boldsymbol{\theta}^{[k]}. \tag{68}$$

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

**Remark 5.1.** Equation (64) is the first moment (moving average), (65) is the second moment (variance estimation), and (66)–(67) are initial bias corrections. (68) scales the learning rate for each coordinate and applies L2 regularization separately from the learning rate (applies weight decay independently of the ”gradient”). This part is called **decoupled weight decay**.

# 6 Backpropagation for General Neural Networks

Up to the previous chapter, we have seen that if we can calculate the gradient vector of the objective function, which is a function of the parameter vector defined on a certain data point or minibatch, we can apply SGD or AdamW to make the objective function as small as possible. Therefore, what we want to know is whether we can calculate the gradient vector for the various structured neural networks and objective functions used in generative AI. The answer to this problem is affirmative. Specifically, the gradient vector of the objective function, defined using a feedforward neural network and the return values of some of its nodes, can be obtained relatively efficiently by an algorithm based on dynamic programming called **backpropagation** [8].

**Remark 6.1.** Backpropagation is often defined for Multi-Layer Perceptrons (MLPs), which are neural networks composed of fully connected layers, but in reality, MLPs are rarely used in generative AI. In this lecture, while placing the starting point of the discussion on MLPs, the goal is to define backpropagation on a neural network defined on a general directed acyclic multigraph.

## 6.1 Backpropagation for MLPs

**Definition 6.1** (MLP Structure)**.** An MLP (multi-layer perceptron) with input dimension $d_0$ and number of layers $L \in \mathbb{Z}_{>0}$ is defined as follows. Each layer $\ell \in [1, L]_{\mathbb{Z}}$ has a weight matrix $\boldsymbol{W}^{(\ell)} \in \mathbb{R}^{d_\ell, d_{\ell-1}}$, a bias $\boldsymbol{b}^{(\ell)} \in \mathbb{R}^{d_\ell}$, and a fixed differentiable activation function $\varphi^{(\ell)} : \mathbb{R}^{d_\ell} \to \mathbb{R}^{d_\ell}$ that takes a vector input and outputs a vector. (A special case includes the component-wise parallel application of a scalar activation function.)

**Definition 6.2** (MLP Forward Propagation)**.** Given an input $\boldsymbol{x} \in \mathbb{R}^{d_0}$. First

$$\boldsymbol{h}^{(0)} := \boldsymbol{x} \tag{69}$$

is set, and for $\ell \in [1, L]_{\mathbb{Z}}$

$$\boldsymbol{z}^{(\ell)} := \boldsymbol{W}^{(\ell)} \boldsymbol{h}^{(\ell-1)} + \boldsymbol{b}^{(\ell)} \in \mathbb{R}^{d_\ell}, \tag{70}$$

$$\boldsymbol{h}^{(\ell)} := \varphi^{(\ell)}\left(\boldsymbol{z}^{(\ell)}\right) \in \mathbb{R}^{d_\ell} \tag{71}$$

are recursively defined. The final layer output

$$\boldsymbol{h}^{(L)} = f_{(\boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(L)}, \boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(L)})}(\boldsymbol{x}) \tag{72}$$

is taken as the output of the entire MLP.

**Remark 6.2.** Equation (70) expresses that each layer is an affine transformation consisting of a linear transformation and bias addition, and Equation (71) expresses that a non-linear activation function is applied to its output. By repeating this layer by layer, a complex non-linear map $f_{(\cdot)}$ for the input $\boldsymbol{x}$ is obtained.

For the MLP defined as above, we define the **backpropagation algorithm** that uses the intermediate quantities obtained in forward propagation to efficiently calculate the gradient vector of the objective function. Here, we consider the case where the objective function depends only on the final layer output.

**Definition 6.3** (MLP Backpropagation (Objective Function Dependent Only on Final Layer Output))**.** Fix the input $\boldsymbol{x}$, and consider a scalar-valued objective function

$$\mathcal{J} = \mathcal{J}\left(\boldsymbol{h}^{(L)}\right) \in \mathbb{R} \tag{73}$$

that depends on all parameters

$$\Theta \coloneqq \left(\boldsymbol{W}^{(1)}, \ldots, \boldsymbol{W}^{(L)}, \boldsymbol{b}^{(1)}, \ldots, \boldsymbol{b}^{(L)}\right). \tag{74}$$

Here $\boldsymbol{h}^{(L)}$ is determined by the forward propagation (69)–(72).
The MLP backpropagation algorithm is a map that calculates the partial derivatives $\frac{\partial \mathcal{J}}{\partial \boldsymbol{W}^{(\ell)}}$, $\frac{\partial \mathcal{J}}{\partial \boldsymbol{b}^{(\ell)}}$ for each parameter by the following procedure.

- Forward Propagation Step: For all layers $\ell \in [1, L]_{\mathbb{Z}}$, calculate $\boldsymbol{z}^{(\ell)}$ and $\boldsymbol{h}^{(\ell)}$ according to Equation (70) and Equation (71), and save them.
- Backpropagation Step (Initialization): For the final layer, define the vector

$$\boldsymbol{\delta}^{(L)} \coloneqq \frac{\partial \mathcal{J}}{\partial \boldsymbol{h}^{(L)}} \odot \varphi^{(L)\prime}\left(\boldsymbol{z}^{(L)}\right) \in \mathbb{R}^{d_L}. \tag{75}$$

  Here $\odot$ is the element-wise product (Hadamard product), and $\varphi^{(L)\prime}$ is the component-wise single-variable derivative of $\varphi^{(L)}$.

- Backpropagation Step (Recursion): In the order $\ell = L-1, L-2, \ldots, 1$, calculate the following.

$$\boldsymbol{\delta}^{(\ell)} \coloneqq \left(\boldsymbol{W}^{(\ell+1)\top} \boldsymbol{\delta}^{(\ell+1)}\right) \odot \varphi^{(\ell)\prime}\left(\boldsymbol{z}^{(\ell)}\right) \in \mathbb{R}^{d_\ell}, \tag{76}$$

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{W}^{(\ell)}} \coloneqq \boldsymbol{\delta}^{(\ell)} \, \boldsymbol{h}^{(\ell-1)\top} \in \mathbb{R}^{d_\ell, d_{\ell-1}}, \tag{77}$$

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{b}^{(\ell)}} \coloneqq \boldsymbol{\delta}^{(\ell)} \in \mathbb{R}^{d_\ell}. \tag{78}$$

- Output: Output the $\frac{\partial \mathcal{J}}{\partial \boldsymbol{W}^{(\ell)}}$ and $\frac{\partial \mathcal{J}}{\partial \boldsymbol{b}^{(\ell)}}$ obtained for all $\ell \in [1, L]_{\mathbb{Z}}$ as components of the gradient vector regarding the parameter $\Theta$.

**Remark 6.3.** The vector $\boldsymbol{\delta}^{(\ell)}$ is a quantity corresponding to the gradient of the objective function with respect to the affine input $\boldsymbol{z}^{(\ell)}$ of layer $\ell$, $\frac{\partial \mathcal{J}}{\partial \boldsymbol{z}^{(\ell)}}$. Equation (76) implements the chain rule by "passing back" the gradient from layer $\ell+1$ to the previous layer via the linear transformation $\boldsymbol{W}^{(\ell+1)\top}$, and further multiplying by the derivative of the activation function $\varphi^{(\ell)\prime}$. Equation (77) and Equation (78) are the formulas for the partial derivatives for the linear part $\boldsymbol{z}^{(\ell)} = \boldsymbol{W}^{(\ell)}\boldsymbol{h}^{(\ell-1)} + \boldsymbol{b}^{(\ell)}$, written together in matrix/vector form.

The following theorem asserts that, under the assumption of differentiability, the algorithm of Definition 6.3 gives the correct gradient vector with respect to $\Theta$.

**Theorem 6.1** (Correctness of MLP Backpropagation)**.** Fix the input $\boldsymbol{x}$. Assume that the activation function $\varphi^{(\ell)} : \mathbb{R}^{d_\ell} \to \mathbb{R}^{d_\ell}$ of each layer is differentiable at the point $\boldsymbol{z}^{(\ell)}$, and the objective function $\mathcal{J} : \mathbb{R}^{d_L} \to \mathbb{R}$ is differentiable at the point $\boldsymbol{h}^{(L)}$. At this time, when $\mathcal{J}$ is

regarded as a function of the parameter $\Theta$

$$F(\Theta) \coloneqq \mathcal{J}\big(\boldsymbol{h}^{(L)}(\Theta)\big), \tag{79}$$

the $\frac{\partial \mathcal{J}}{\partial \boldsymbol{W}^{(\ell)}}$ and $\frac{\partial \mathcal{J}}{\partial \boldsymbol{b}^{(\ell)}}$ obtained by Definition 6.3 coincide with the partial derivative matrix/vector regarding $F$. That is, for any $\ell \in [1, L]_{\mathbb{Z}}$ and components $i \in [1, d_\ell]_{\mathbb{Z}},\ j \in [1, d_{\ell-1}]_{\mathbb{Z}}$,

$$\left(\frac{\partial \mathcal{J}}{\partial \boldsymbol{W}^{(\ell)}}\right)_{i,j} = \frac{\partial F}{\partial w_{i,j}^{(\ell)}}, \tag{80}$$

$$\left(\frac{\partial \mathcal{J}}{\partial \boldsymbol{b}^{(\ell)}}\right)_{i} = \frac{\partial F}{\partial b_{i}^{(\ell)}} \tag{81}$$

hold.

*Proof.* First, we show by backward mathematical induction on $\ell$ that $\boldsymbol{\delta}^{(\ell)}$ defined in Definition 6.3 satisfies

$$\boldsymbol{\delta}^{(\ell)} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{z}^{(\ell)}} \quad (\ell \in [1, L]_{\mathbb{Z}}). \tag{82}$$

**Step 1: Base case ($\ell = L$).** The objective function $\mathcal{J}$ is a function of $\boldsymbol{h}^{(L)}$, and $\boldsymbol{h}^{(L)}$ is a function of $\boldsymbol{z}^{(L)}$, so the composite function

$$\mathcal{J} \circ \varphi^{(L)} : \mathbb{R}^{d_L} \to \mathbb{R} \tag{83}$$

is defined. From Proposition 4.1 (differentiability of composite functions) and Definition 4.5, $\mathcal{J} \circ \varphi^{(L)}$ is differentiable at the point $\boldsymbol{z}^{(L)}$, and its gradient vector is given, corresponding to the single-variable chain rule, by

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{z}^{(L)}} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{h}^{(L)}} \odot \varphi^{(L)\prime}\big(\boldsymbol{z}^{(L)}\big). \tag{84}$$

Comparing with Equation (75) of Definition 6.3, $\boldsymbol{\delta}^{(L)} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{z}^{(L)}}$ follows.

**Step 2: Induction hypothesis.** Assume that for some $\ell \in [1, L-1]_{\mathbb{Z}}$, the proposition

$$\boldsymbol{\delta}^{(\ell+1)} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{z}^{(\ell+1)}} \tag{85}$$

holds.

**Step 3: Inductive step.** The affine transformation of layer $\ell + 1$ is

$$\boldsymbol{z}^{(\ell+1)} = \boldsymbol{W}^{(\ell+1)}\boldsymbol{h}^{(\ell)} + \boldsymbol{b}^{(\ell+1)}, \tag{86}$$

and $\boldsymbol{z}^{(\ell+1)}$ is a linear function of $\boldsymbol{h}^{(\ell)}$. From Definition 4.5 and the differentiability of linear

maps, the gradient vector of $\mathcal{J}$ viewed as a function of $\boldsymbol{h}^{(\ell)}$ is given by

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{h}^{(\ell)}} = \boldsymbol{W}^{(\ell+1)\top} \frac{\partial \mathcal{J}}{\partial \boldsymbol{z}^{(\ell+1)}}. \tag{87}$$

Substituting the induction hypothesis (85),

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{h}^{(\ell)}} = \boldsymbol{W}^{(\ell+1)\top} \boldsymbol{\delta}^{(\ell+1)} \tag{88}$$

is obtained.

On the other hand, the activation part of layer $\ell$ is

$$\boldsymbol{h}^{(\ell)} = \varphi^{(\ell)}\left(\boldsymbol{z}^{(\ell)}\right), \tag{89}$$

so the gradient vector with respect to $\boldsymbol{z}^{(\ell)}$ is, from the component-wise chain rule,

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{z}^{(\ell)}} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{h}^{(\ell)}} \odot \varphi^{(\ell)\prime}\left(\boldsymbol{z}^{(\ell)}\right). \tag{90}$$

Substituting Equation (88),

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{z}^{(\ell)}} = \left(\boldsymbol{W}^{(\ell+1)\top} \boldsymbol{\delta}^{(\ell+1)}\right) \odot \varphi^{(\ell)\prime}\left(\boldsymbol{z}^{(\ell)}\right), \tag{91}$$

but this is none other than $\boldsymbol{\delta}^{(\ell)}$ defined by Equation (76) of Definition 6.3. Therefore

$$\boldsymbol{\delta}^{(\ell)} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{z}^{(\ell)}} \tag{92}$$

is shown.

By the above, by backward mathematical induction, (82) holds for all $\ell \in [1, L]_{\mathbb{Z}}$.

**Step 4: Partial derivatives with respect to parameters.** Next, we derive the formulas for the partial derivatives with respect to $\boldsymbol{W}^{(\ell)}$ and $\boldsymbol{b}^{(\ell)}$. Writing Equation (70) in components,

$$z_i^{(\ell)} = \sum_{j=1}^{d_{\ell-1}} w_{i,j}^{(\ell)} h_j^{(\ell-1)} + b_i^{(\ell)}. \tag{93}$$

Here, regarding $F(\Theta) = \mathcal{J}$ as a function depending on $\boldsymbol{W}^{(\ell)}$ and $\boldsymbol{b}^{(\ell)}$ through $\boldsymbol{z}^{(\ell)}$, from the chain rule

$$\frac{\partial F}{\partial w_{i,j}^{(\ell)}} = \sum_{k=1}^{d_\ell} \frac{\partial \mathcal{J}}{\partial z_k^{(\ell)}} \frac{\partial z_k^{(\ell)}}{\partial w_{i,j}^{(\ell)}} \tag{94}$$

$$= \frac{\partial \mathcal{J}}{\partial z_i^{(\ell)}} \cdot h_j^{(\ell-1)} \tag{95}$$

is obtained. Here $\frac{\partial z_k^{(\ell)}}{\partial w_{i,j}^{(\ell)}} = 0$ $(k \neq i)$, $\frac{\partial z_i^{(\ell)}}{\partial w_{i,j}^{(\ell)}} = h_j^{(\ell-1)}$ was used. Similarly

$$\frac{\partial F}{\partial b_i^{(\ell)}} = \sum_{k=1}^{d_\ell} \frac{\partial \mathcal{J}}{\partial z_k^{(\ell)}} \frac{\partial z_k^{(\ell)}}{\partial b_i^{(\ell)}} \tag{96}$$

$$= \frac{\partial \mathcal{J}}{\partial z_i^{(\ell)}}. \tag{97}$$

On the other hand, from Equation (82), $\frac{\partial \mathcal{J}}{\partial z_i^{(\ell)}} = \delta_i^{(\ell)}$, so

$$\frac{\partial F}{\partial w_{i,j}^{(\ell)}} = \delta_i^{(\ell)} h_j^{(\ell-1)}, \tag{98}$$

$$\frac{\partial F}{\partial b_i^{(\ell)}} = \delta_i^{(\ell)} \tag{99}$$

are obtained. This is exactly the component representation of the right-hand sides of Equation (77) and Equation (78). Therefore

$$\frac{\partial \mathcal{J}}{\partial \boldsymbol{W}^{(\ell)}} = \boldsymbol{\delta}^{(\ell)} \boldsymbol{h}^{(\ell-1)\top}, \quad \frac{\partial \mathcal{J}}{\partial \boldsymbol{b}^{(\ell)}} = \boldsymbol{\delta}^{(\ell)} \tag{100}$$

coincide with the partial derivatives regarding $F$. Thus (80) and (81) are shown. □

## 6.2 Gradient Calculation by Backpropagation for General Feedforward NNs

So far we have considered MLPs, but we will now explain that the same can be done for general feedforward neural networks. Considering the case of MLPs abstractly, forward propagation could recursively define the output in the order of layer dependencies, and backpropagation could recursively define the gradient vector in the reverse order of forward propagation, while also using the calculation results from forward propagation. For a general feedforward neural network, since it is defined by a directed acyclic graph (DAG), a one-way dependency relationship always holds between any nodes, and the same can be said. Specifically, forward propagation can be defined in the topological sort order of the directed graph, and backpropagation can be calculated in its reverse order. We will look at this rigorously with mathematical formulas.

**Definition 6.4** (General Feedforward Neural Network)**.** The **architecture** of a neural network is rigorously defined as a tuple of the following elements.

1. Computation Graph: A directed acyclic multigraph $G = (\mathcal{V}, \mathcal{E}, \mathrm{Tail}, \mathrm{Head})$. The node set $\mathcal{V} \subset \mathbb{Z}_{>0}$ is topologically ordered, and $\forall e \in \mathcal{E}$, $\mathrm{Tail}(e) < \mathrm{Head}(e)$.

2. Input/Output Dimensions: $d_{\mathrm{input}} \in \mathbb{Z}_{>0}$, $d_{\mathrm{output}} \in \mathbb{Z}_{>0}$.

3. Node Specification: Each $v \in \mathcal{V}$ has an activation function $\mathrm{g}^{(v)} : \mathbb{R}^{d_{\mathrm{arg}}^{(v)}} \to \mathbb{R}^{d_{\mathrm{ret}}^{(v)}}$ and an argument connection source tuple $\mathrm{NodeSrc}^{(v)} \in ([1, d_{\mathrm{input}}]_{\mathbb{Z}} \cup \mathcal{E}_{\mathrm{in}}^{(v)})^{[1, d_{\mathrm{arg}}^{(v)}]_{\mathbb{Z}}}$. Here $\mathcal{E}_{\mathrm{in}}^{(v)} \subset \mathcal{E}$ is the set of incoming edges satisfying $\mathrm{Head}(e) = v$.

4. Edge Specification: Each $e \in \mathcal{E}$ has $\mathrm{EdgeSrc}(e) \in [1, d_{\mathrm{ret}}^{(u)}]_{\mathbb{Z}}$ for $\mathrm{Tail}(e) = u$.

5. Parameter Specification: Has $d_{\mathrm{param}} \in \mathbb{Z}_{>0}$, a surjection $\mathrm{Weight} : \mathcal{E} \to [1, d_{\mathrm{param}}]_{\mathbb{Z}}$, a variable index set $\mathcal{I}_{var} \subseteq [1, d_{\mathrm{param}}]_{\mathbb{Z}}$, a fixed index set $\mathcal{I}_{fix}$, and variable/fixed edge sets $\mathcal{E}_{var}, \mathcal{E}_{fix}$. Here $\theta_p$ ($p \in [1, d_{\mathrm{param}}]_{\mathbb{Z}}$) is a parameter.

6. Output Specification: $\mathrm{OutSrc} : [1, d_{\mathrm{output}}]_{\mathbb{Z}} \to \bigcup_{v \in \mathcal{V}} (\{v\} \times [1, d_{\mathrm{ret}}^{(v)}]_{\mathbb{Z}})$ specifies which node's return value component corresponds to the output component.

When a checkpoint $\boldsymbol{\theta} \in \mathbb{R}^{d_{\mathrm{param}}}$ is given, $f_{\boldsymbol{\theta}} : \mathbb{R}^{d_{\mathrm{input}}} \to \mathbb{R}^{d_{\mathrm{output}}}$ is defined as follows. The return value $\boldsymbol{r}^{(v)} \in \mathbb{R}^{d_{\mathrm{ret}}^{(v)}}$ of node $v$ is determined in topological order by

$$a_m^{(v)} = \begin{cases} x_j & \text{if } \mathrm{NodeSrc}_m^{(v)} = j \in [1, d_{\mathrm{input}}]_{\mathbb{Z}}, \\ \theta_{\mathsf{Weight}(e)}\, r_{\mathsf{EdgeSrc}(e)}^{(\mathsf{Tail}(e))} & \text{if } \mathrm{NodeSrc}_m^{(v)} = e \in \mathcal{E}_{\mathrm{in}}^{(v)}, \end{cases} \tag{101}$$

$$\boldsymbol{r}^{(v)} := \mathrm{g}^{(v)}\left(\boldsymbol{a}^{(v)}\right) \tag{102}$$

, and finally $\boldsymbol{y} = (y_1, \ldots, y_{d_{\mathrm{output}}})^\top$ is determined as $y_j = r_k^{(v)}$ where $(v, k) = \mathrm{OutSrc}(j)$.

The above is a definition as a composite function, but by actually calculating the return value of each node according to this definition in topological sort order, the output of the above neural network for a given input can be calculated.

**Definition 6.5** (Forward Propagation in Topological Order)**.** For $\mathcal{V} = \{v_1, \ldots, v_N\}$ arranged topologically, the operation of evaluating Equation (101) and Equation (102) in the order $i = 1, \ldots, N$, and finally constructing $\boldsymbol{y}$ to obtain $f_{\boldsymbol{\theta}}(\boldsymbol{x})$ is called the forward propagation of the general neural network.

**Remark 6.4.** By arranging the nodes in topological order, the input of each node is always constructed from the output of the preceding nodes, so forward propagation can be implemented with a simple loop. This is an extension of the layer-order calculation of MLPs to a general graph structure.

Finally, we define the backpropagation algorithm for a general feedforward neural network. As in the case of MLPs, we explicitly add the objective function as a scalar-valued function to the graph, and let the value returned by that node be $\mathcal{J}$.

**Definition 6.6** (Backpropagation on General DAG)**.** In addition to the situation of Definition 6.4, consider a scalar-valued objective function

$$\mathcal{J} = \mathcal{J}(\boldsymbol{y}) \in \mathbb{R}. \tag{103}$$

Here $\boldsymbol{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$. We newly add an **objective node** $v_{\mathcal{J}}$ representing this objective function, and let it be the last node in the topological order. The activation function $\mathrm{g}^{(v_{\mathcal{J}})}$ of $v_{\mathcal{J}}$ is assumed to be a function that takes $\boldsymbol{y}$ as input and outputs $\mathcal{J}$.
The backpropagation algorithm on a general DAG is for calculating the partial derivative

$$\frac{\partial \mathcal{J}}{\partial \theta_p} \tag{104}$$

for any parameter index $p \in [1, d_{\mathrm{param}}]_{\mathbb{Z}}$ by the following procedure.

- **Forward Propagation Step**: According to Definition 6.5, calculate and save $\boldsymbol{a}^{(v)}$ and $\boldsymbol{r}^{(v)}$ for all nodes $v \in \mathcal{V}$. Furthermore, calculate $\mathcal{J}$ at the objective function node $v_{\mathcal{J}}$.
- **Initialization**: For each node $v$, prepare a variable representing the gradient vector of $\mathcal{J}$ with respect to $\boldsymbol{r}^{(v)}$

  $$\boldsymbol{\delta}^{(v)} := \frac{\partial \mathcal{J}}{\partial \boldsymbol{r}^{(v)}} \in \mathbb{R}^{d_{\mathrm{ret}}^{(v)}} \tag{105}$$

  , and initialize all to $0$. The output of the objective function node $v_{\mathcal{J}}$ is a scalar and is $\mathcal{J}$ itself, so

  $$\boldsymbol{\delta}^{(v_{\mathcal{J}})} := 1 \tag{106}$$

  is set. Also, for each parameter index $p$,

  $$G_p := 0 \tag{107}$$

  is set, and this is used as an accumulation variable for $\frac{\partial \mathcal{J}}{\partial \theta_p}$.
- **Backpropagation Loop**: For the topological order $\mathcal{V} \cup \{v_{\mathcal{J}}\} = \{v_1, \ldots, v_N, v_{\mathcal{J}}\}$, process the node $w := v_i$ in the order $i = N, N-1, \ldots, 1$.
  - Write the Jacobian of the activation function $\mathrm{g}^{(w)} : \mathbb{R}^{d_{\mathrm{arg}}^{(w)}} \to \mathbb{R}^{d_{\mathrm{ret}}^{(w)}}$ of node $w$ as

    $$\boldsymbol{J}^{(w)}\left(\boldsymbol{a}^{(w)}\right) := \frac{\partial \mathrm{g}^{(w)}}{\partial \boldsymbol{a}^{(w)}} \in \mathbb{R}^{d_{\mathrm{ret}}^{(w)}, d_{\mathrm{arg}}^{(w)}}. \tag{108}$$

    By the chain rule, the gradient vector with respect to $\boldsymbol{a}^{(w)}$

    $$\boldsymbol{\gamma}^{(w)} := \frac{\partial \mathcal{J}}{\partial \boldsymbol{a}^{(w)}} \in \mathbb{R}^{d_{\mathrm{arg}}^{(w)}} \tag{109}$$

    can be calculated by

    $$\boldsymbol{\gamma}^{(w)} = \boldsymbol{J}^{(w)}\left(\boldsymbol{a}^{(w)}\right)^{\top} \boldsymbol{\delta}^{(w)}. \tag{110}$$

  - Next, for each argument component $m \in [1, d_{\mathrm{arg}}^{(w)}]_{\mathbb{Z}}$ of node $w$, perform the following update according to the form of $\mathrm{NodeSrc}_m^{(w)}$.
    1. When $\mathrm{NodeSrc}_m^{(w)} = j \in [1, d_{\mathrm{input}}]_{\mathbb{Z}}$ (component of input vector $\boldsymbol{x}$), it is a pure copy containing no parameters, so nothing is updated.

2. When $\mathrm{NodeSrc}_m^{(w)} = e \in \mathcal{E}_{\mathrm{in}}^{(w)}$, the edge $e$ is an edge from $\mathrm{Tail}(e) = u$ to $\mathrm{Head}(e) = w$, and the input is

$$a_m^{(w)} = \theta_{\mathsf{Weight}(e)}\, r^{(u)}_{\mathsf{EdgeSrc}(e)}. \tag{111}$$

At this time, by the chain rule

$$\frac{\partial \mathcal{J}}{\partial \theta_{\mathsf{Weight}(e)}} \leftarrow \frac{\partial \mathcal{J}}{\partial \theta_{\mathsf{Weight}(e)}} + \gamma_m^{(w)}\, r^{(u)}_{\mathsf{EdgeSrc}(e)}, \tag{112}$$

$$\frac{\partial \mathcal{J}}{\partial r^{(u)}_{\mathsf{EdgeSrc}(e)}} \leftarrow \frac{\partial \mathcal{J}}{\partial r^{(u)}_{\mathsf{EdgeSrc}(e)}} + \gamma_m^{(w)}\, \theta_{\mathsf{Weight}(e)}. \tag{113}$$

We rewrite this according to the variable notation as

$$G_{\mathsf{Weight}(e)} \leftarrow G_{\mathsf{Weight}(e)} + \gamma_m^{(w)}\, r^{(u)}_{\mathsf{EdgeSrc}(e)}, \tag{114}$$

$$\delta^{(u)}_{\mathsf{EdgeSrc}(e)} \leftarrow \delta^{(u)}_{\mathsf{EdgeSrc}(e)} + \gamma_m^{(w)}\, \theta_{\mathsf{Weight}(e)}. \tag{115}$$

- **Output**: After the backpropagation loop finishes, output $G_p$ as the calculation result of $\frac{\partial \mathcal{J}}{\partial \theta_p}$ for each $p \in [1, d_{\mathrm{param}}]_{\mathbb{Z}}$. That is

$$\frac{\partial \mathcal{J}}{\partial \theta_p} \coloneqq G_p. \tag{116}$$

**Remark 6.5.** The vector $\boldsymbol{\delta}^{(v)}$ is the gradient of the objective function with respect to the output $\boldsymbol{r}^{(v)}$ of node $v$, and $\boldsymbol{\gamma}^{(w)}$ is the gradient with respect to the input $\boldsymbol{a}^{(w)}$ of node $w$. $\boldsymbol{\delta}^{(w)}$ is converted to $\boldsymbol{\gamma}^{(w)}$ by Equation (110), and by Equation (114) and Equation (115), the gradients to the parameter and the parent node are simultaneously updated along the edge. By performing this operation in the reverse of the topological order, the contributions from all paths reaching each node are appropriately accumulated. In the case of MLPs, Definition 6.3 is this algorithm organized into matrix operations for each layer.

**Remark 6.6.** The above algorithm is a general version that includes the case where Weight is not injective, that is, weight sharing is used. Therefore, backpropagation for neural networks including convolutional layers and multi-head attention layers is also all within the scope of the above algorithm. Note that the same result can be obtained by considering that backpropagation is first performed assuming that all edges have independent parameters, and then all the gradients of the multiple edges corresponding to one parameter element are added up.

The following theorem asserts that the backpropagation algorithm according to Definition 6.6 gives the correct gradient vector even for a general feedforward neural network.

**Theorem 6.2** (Correctness of Backpropagation on General DAG)**.** Consider the situation of Definition 6.4 and Definition 6.6. Assume that for all nodes $v \in \mathcal{V}$, the activation function $\mathrm{g}^{(v)}$ is differentiable at the point $\boldsymbol{a}^{(v)}$, and the activation function $\mathrm{g}^{(v_{\mathcal{J}})}$ of the objective function node $v_{\mathcal{J}}$ is differentiable at the point $\boldsymbol{a}^{(v_{\mathcal{J}})}$. At this time, the composite function when the input $\boldsymbol{x}$ is fixed

$$F(\boldsymbol{\theta}) \coloneqq \mathcal{J}\left(f_{\boldsymbol{\theta}}(\boldsymbol{x})\right) \tag{117}$$

is differentiable with respect to $\boldsymbol{\theta}$, and $G_p$ obtained by Definition 6.6 satisfies

$$G_p = \frac{\partial F}{\partial \theta_p}. \tag{118}$$

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

*Proof.* The policy of the proof is to combine backward mathematical induction using the topological sort order $v_1, \ldots, v_N, v_{\mathcal{J}}$ and the chain rule.

**Step 1: Inductive representation of the gradient regarding node output.** We show that for any node $v \in \mathcal{V} \cup \{v_{\mathcal{J}}\}$, $\boldsymbol{\delta}^{(v)}$ is the gradient with respect to $\boldsymbol{r}^{(v)}$

$$\boldsymbol{\delta}^{(v)} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{r}^{(v)}}. \tag{119}$$

For the topological order $\{v_1, \ldots, v_N, v_{\mathcal{J}}\}$, let $i$ move in reverse order $N, N-1, \ldots, 1$. At the start of each step $i$, assume that the node set

$$\mathcal{W}_i \coloneqq \{v_{i+1}, \ldots, v_N, v_{\mathcal{J}}\} \tag{120}$$

has already been processed, and that $\boldsymbol{\delta}^{(w)}$ $(w \in \mathcal{W}_i)$ is the complete gradient of $\mathcal{J}$ with respect to $\boldsymbol{r}^{(w)}$.

**Base case ($i = N$).** At this time $\mathcal{W}_N = \{v_{\mathcal{J}}\}$, and by Equation (106) of Definition 6.6, $\boldsymbol{\delta}^{(v_{\mathcal{J}})} = 1$. This is the partial derivative of $\mathcal{J}$ with respect to itself, and since $\boldsymbol{r}^{(v_{\mathcal{J}})} = \mathcal{J}$,

$$\boldsymbol{\delta}^{(v_{\mathcal{J}})} = \frac{\partial \mathcal{J}}{\partial \boldsymbol{r}^{(v_{\mathcal{J}})}} \tag{121}$$

holds. Therefore the base case of the induction holds.

**Inductive step.** Assume that for some $i \in [1, N]_{\mathbb{Z}}$, (119) holds for each node $w$ in $\mathcal{W}_i$. In step $i$, $w \coloneqq v_i$ is processed. The output $\boldsymbol{r}^{(w)}$ of node $w$ appears as an input to the activation functions $\mathrm{g}^{(u)}$ of the set of child nodes

$$C(w) \coloneqq \{u \in \mathcal{V} \cup \{v_{\mathcal{J}}\} \mid \exists e \in \mathcal{E},\ \mathrm{Tail}(e) = w,\ \mathrm{Head}(e) = u\} \tag{122}$$

, as some arguments of the objective function $\mathcal{J}$. The partial derivative of the objective

function with respect to the component $r_k^{(w)}$ of $\boldsymbol{r}^{(w)}$ is given by the chain rule as

$$\frac{\partial \mathcal{J}}{\partial r_k^{(w)}} = \sum_{u \in C(w)} \sum_{m:\ \mathsf{NodeSrc}_m^{(u)}=e,\ \mathsf{Tail}(e)=w,\ \mathsf{EdgeSrc}(e)=k} \frac{\partial \mathcal{J}}{\partial a_m^{(u)}} \frac{\partial a_m^{(u)}}{\partial r_k^{(w)}}. \tag{123}$$

Here $\frac{\partial \mathcal{J}}{\partial a_m^{(u)}}$ is calculated as $\boldsymbol{\gamma}_m^{(u)}$ by Equations (109)–(110) of Definition 6.6. Also, from Equation (111)

$$\frac{\partial a_m^{(u)}}{\partial r_k^{(w)}} = \begin{cases} \theta_{\mathsf{Weight}(e)} & \text{if } \mathsf{Tail}(e) = w,\ \mathsf{EdgeSrc}(e) = k, \\ 0 & \text{otherwise} \end{cases} \tag{124}$$

, so Equation (123) can be written as

$$\frac{\partial \mathcal{J}}{\partial r_k^{(w)}} = \sum_{u \in C(w)} \sum_{m:\ \mathsf{NodeSrc}_m^{(u)}=e,\ \mathsf{Tail}(e)=w,\ \mathsf{EdgeSrc}(e)=k} \gamma_m^{(u)}\, \theta_{\mathsf{Weight}(e)}. \tag{125}$$

Since Equation (115) of Definition 6.6 is exactly the operation of accumulating this sum into $\delta_k^{(w)}$, at the end of step $i$

$$\delta_k^{(w)} = \frac{\partial \mathcal{J}}{\partial r_k^{(w)}} \tag{126}$$

holds. Since they are equal component-wise, (119) also holds for $v = w$. Therefore, by backward mathematical induction, (119) holds for all nodes $v$.

**Step 2: Partial derivatives with respect to parameters.** Next, we derive the formula for the partial derivative $\frac{\partial \mathcal{J}}{\partial \theta_p}$ with respect to the parameter index $p$. The parameter $\theta_p$ affects the objective function through all edges belonging to $\mathsf{Weight}^{-1}(p) \subset \mathcal{E}$. By the chain rule

$$\frac{\partial \mathcal{J}}{\partial \theta_p} = \sum_{e \in \mathsf{Weight}^{-1}(p)} \sum_{u:\ \mathsf{Head}(e)=u} \sum_{m:\ \mathsf{NodeSrc}_m^{(u)}=e} \frac{\partial \mathcal{J}}{\partial a_m^{(u)}} \frac{\partial a_m^{(u)}}{\partial \theta_p}. \tag{127}$$

From Equation (111), $a_m^{(u)} = \theta_p r_{\mathsf{EdgeSrc}(e)}^{(\mathsf{Tail}(e))}$, so

$$\frac{\partial a_m^{(u)}}{\partial \theta_p} = r_{\mathsf{EdgeSrc}(e)}^{(\mathsf{Tail}(e))} \tag{128}$$

, and furthermore $\frac{\partial \mathcal{J}}{\partial a_m^{(u)}} = \gamma_m^{(u)}$, so

$$\frac{\partial \mathcal{J}}{\partial \theta_p} = \sum_{e \in \mathsf{Weight}^{-1}(p)} \sum_{u:\ \mathsf{Head}(e)=u} \sum_{m:\ \mathsf{NodeSrc}_m^{(u)}=e} \gamma_m^{(u)}\, r_{\mathsf{EdgeSrc}(e)}^{(\mathsf{Tail}(e))}. \tag{129}$$

On the other hand, Equation (114) of Definition 6.6 performs updates in each step in the form

$$G_p \leftarrow G_p + \sum_{e \in \mathsf{Weight}^{-1}(p)} \sum_{u:\ \mathsf{Head}(e)=u} \sum_{m:\ \mathsf{NodeSrc}_m^{(u)}=e} \gamma_m^{(u)}\, r_{\mathsf{EdgeSrc}(e)}^{(\mathsf{Tail}(e))}, \tag{130}$$

and $G_p$ after processing all nodes from the initial value $G_p = 0$ becomes exactly equal to the right-hand side (129). Therefore

$$G_p = \frac{\partial \mathcal{J}}{\partial \theta_p} \tag{131}$$

holds. Reinterpreting $\mathcal{J}$ as $F(\boldsymbol{\theta})$, this is equivalent to (118). □

## 6.3 Worked Example: Forward and Backward Propagation on a Specified Graph

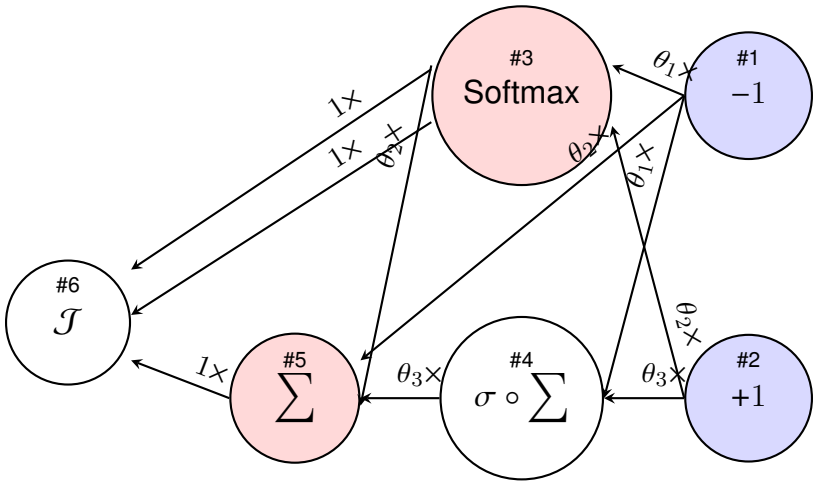



Figure 5: Example network structure (with objective function node). Information flows from right to left, and node $6$ represents the objective function $\mathcal{J}$.

**Example 6.1.** Consider a network with a node sequence $(1, 2, 3, 4, 5)$, and additionally an objective function node $6$ (Figure 5). The edge and parameter correspondence is as follows.

$$(1, 3_1) : \theta_1, \quad (2, 3_2) : \theta_2, \quad (1, 4) : \theta_1, \quad (2, 4) : \theta_3, \quad (1, 5) : \theta_2, \quad (3_1, 5) : \theta_2, \quad (4, 5) : \theta_3. \tag{132}$$

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.
- 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 $y_5$.

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

$$r^{(1)} = -1, \quad r^{(2)} = +1, \tag{133}$$

$$\text{Node 3: } z_1 = \theta_1 r^{(1)},\ z_2 = \theta_2 r^{(2)},\ s_j = \frac{e^{z_j}}{e^{z_1} + e^{z_2}},\ j = 1, 2, \tag{134}$$

$$\text{Node 4: } u = \theta_1 r^{(1)} + \theta_3 r^{(2)},\ y_4 = \sigma(u), \tag{135}$$

$$\text{Node 5: } y_5 = \theta_2 r^{(1)} + \theta_2 s_1 + \theta_3 y_4, \tag{136}$$

$$\text{Node 6: } \mathcal{J} = -\log s_2 + y_5^2. \tag{137}$$

Below, we apply the general backpropagation of Definition 6.6 to this specific example. We follow the calculations at each node according to the reverse topological sort order $6 \to 5 \to 4 \to 3 \to 2 \to 1$.

**Processing Node 6 (Objective Function Node)**.

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

$$\delta^{(6)} = \frac{\partial \mathcal{J}}{\partial \mathcal{J}} = 1 \tag{138}$$

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

$$\frac{\partial \mathcal{J}}{\partial s_1} = 0, \quad \frac{\partial \mathcal{J}}{\partial s_2} = -\frac{1}{s_2}, \quad \frac{\partial \mathcal{J}}{\partial y_5} = 2y_5 \tag{139}$$

. This corresponds to $\boldsymbol{\gamma}^{(6)}$ in Definition 6.6 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 (139),

$$\delta^{(5)} = \frac{\partial \mathcal{J}}{\partial y_5} = 2y_5. \tag{140}$$

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

$$\frac{\partial \mathcal{J}}{\partial \theta_2 r^{(1)}} = 2y_5, \quad \frac{\partial \mathcal{J}}{\partial \theta_2 s_1} = 2y_5, \quad \frac{\partial \mathcal{J}}{\partial \theta_3 y_4} = 2y_5. \tag{141}$$

From here, according to Equation (114) and Equation (115) of Definition 6.6, we propagate the gradient to the parameters and parent nodes through edges $(1, 5)$, $(3_1, 5)$, and $(4, 5)$.

Specifically

$$\text{Edge } (1,5): \quad G_2 \leftarrow G_2 + (2y_5)\cdot r^{(1)}, \quad \delta^{(1)} \leftarrow \delta^{(1)} + (2y_5)\theta_2, \tag{142}$$

$$\text{Edge } (3_1,5): \quad G_2 \leftarrow G_2 + (2y_5)\cdot s_1, \quad \delta_1^{(3)} \leftarrow \delta_1^{(3)} + (2y_5)\theta_2, \tag{143}$$

$$\text{Edge } (4,5): \quad G_3 \leftarrow G_3 + (2y_5)\cdot y_4, \quad \delta^{(4)} \leftarrow \delta^{(4)} + (2y_5)\theta_3. \tag{144}$$

**Processing Node 4 ($\sigma \circ \sum$).**
The gradient with respect to the output of node 4 is given by $\delta^{(4)}$ in Equation (144). 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

$$\frac{\partial \mathcal{J}}{\partial u} = \frac{\partial \mathcal{J}}{\partial y_4}\sigma'(u) = \delta^{(4)} \cdot \sigma(u)(1-\sigma(u)). \tag{145}$$

Furthermore, from the parameter dependency of $u$

$$\frac{\partial \mathcal{J}}{\partial \theta_1} \leftarrow \frac{\partial \mathcal{J}}{\partial \theta_1} + \frac{\partial \mathcal{J}}{\partial u}\cdot r^{(1)} = G_1 + \frac{\partial \mathcal{J}}{\partial u} r^{(1)}, \tag{146}$$

$$\frac{\partial \mathcal{J}}{\partial \theta_3} \leftarrow \frac{\partial \mathcal{J}}{\partial \theta_3} + \frac{\partial \mathcal{J}}{\partial u}\cdot r^{(2)} = G_3 + \frac{\partial \mathcal{J}}{\partial u} r^{(2)}, \tag{147}$$

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

$$\delta^{(1)} \leftarrow \delta^{(1)} + \frac{\partial \mathcal{J}}{\partial u}\theta_1, \tag{148}$$

$$\delta^{(2)} \leftarrow \delta^{(2)} + \frac{\partial \mathcal{J}}{\partial u}\theta_3. \tag{149}$$

**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)

$$\frac{\partial \mathcal{J}}{\partial z_j} = s_j - t_j, \quad t = (0,1) \tag{150}$$

holds. Here, since there is an additional gradient of $2y_5$ in the $s_1$ direction as a contribution from node 5 (Equation (143)), applying Equation (110) of Definition 6.6 gives

$$\frac{\partial \mathcal{J}}{\partial z_1} = s_1 + 2y_5 \cdot s_1(1-s_1), \tag{151}$$

$$\frac{\partial \mathcal{J}}{\partial z_2} = (s_2 - 1) - 2y_5 \cdot s_1 s_2. \tag{152}$$

This corresponds to $\delta_1^{(3)}, \delta_2^{(3)}$.
The gradients to the parameters and input nodes are, from a form similar to Equation (93),

updated as

$$\frac{\partial \mathcal{J}}{\partial \theta_1} \leftarrow \frac{\partial \mathcal{J}}{\partial \theta_1} + \frac{\partial \mathcal{J}}{\partial z_1} \cdot r^{(1)}, \tag{153}$$

$$\frac{\partial \mathcal{J}}{\partial \theta_2} \leftarrow \frac{\partial \mathcal{J}}{\partial \theta_2} + \frac{\partial \mathcal{J}}{\partial z_2} \cdot r^{(2)}, \tag{154}$$

$$\delta^{(1)} \leftarrow \delta^{(1)} + \frac{\partial \mathcal{J}}{\partial z_1}\theta_1, \tag{155}$$

$$\delta^{(2)} \leftarrow \delta^{(2)} + \frac{\partial \mathcal{J}}{\partial z_2}\theta_2. \tag{156}$$

**Processing Node 2 and Node 1 (Input Nodes)**.
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 (146), (147), (153), (154), and the contributions from node 5 (142)–(144), the gradients with respect to the parameters $\theta_1, \theta_2, \theta_3$

$$\frac{\partial \mathcal{J}}{\partial \theta_1} = \frac{\partial \mathcal{J}}{\partial z_1} \cdot r^{(1)} + \frac{\partial \mathcal{J}}{\partial u} \cdot r^{(1)}, \tag{157}$$

$$\frac{\partial \mathcal{J}}{\partial \theta_2} = \frac{\partial \mathcal{J}}{\partial z_2} \cdot r^{(2)} + (2y_5)r^{(1)} + (2y_5)s_1, \tag{158}$$

$$\frac{\partial \mathcal{J}}{\partial \theta_3} = \frac{\partial \mathcal{J}}{\partial u} \cdot r^{(2)} + (2y_5)y_4 \tag{159}$$

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 6.6 gives the correct gradient even in a specific example.

## 6.4 Note: What Happens with Non-differentiable Functions?

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

$$\partial^\circ \mathrm{ReLU}(x) = \begin{cases} \{0\} & \text{if } x < 0, \\ [0,1] & \text{if } x = 0, \\ \{1\} & \text{if } x > 0, \end{cases} \tag{160}$$

, which matches intuition.

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 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.

# 7 Summary and Next Time

## 7.1 Answers to Learning Outcomes

- 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.

## 7.2 Next Time

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

# References


[1] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in Medical Image Computing and Computer-Assisted Intervention (MICCAI) 2015, 2015.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Łukasz Kaiser, and I. Polosukhin, "Attention is all you need," in Advances in Neural Information Processing Systems 30 (NeurIPS 2017), 2017.

[3] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) 2022, 2022.

[4] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever, "Learning transferable visual models from natural language supervision," in Proceedings of the 38th International Conference on Machine Learning (ICML) 2021, 2021.

[5] T. M. Cover and J. A. Thomas, Elements of Information Theory. Wiley, 2 ed., 2006.

[6] H. Robbins and S. Monro, "A stochastic approximation method," The Annals of Mathematical Statistics, vol. 22, no. 3, pp. 400–407, 1951.

[7] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in International Conference on Learning Representations (ICLR) 2019, 2019.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," Nature, vol. 323, no. 6088, pp. 533–536, 1986.