AI Application Lecture 20
Supervised Learning and Learning from Reward

SUZUKI, Atsushi
Jing WANG

2025-11-25

Contents

1 Introduction

The learning step of machine learning using parametric functions, including generative AI, is a problem of determining good parameters to solve a given task based on training data. In many cases, the basic flow has been to define an objective function that quantifies the poorness of the parameters based on the training data, and to optimize this objective function using gradient methods or similar approaches.

Since training data is often obtained from the real world, its availability—whether it is easy or difficult to obtain—is determined by the application and cannot be changed for the convenience of mathematics or computer science. Conversely, it is necessary to determine the objective function and methods in accordance with the format of the training data.

In this lecture, we will learn about several major formats of training data and how to define objective functions for each format. Specifically, in addition to the "input-output" pairs handled in the conventional framework of **supervised learning**, we will introduce frameworks that use **preference-based data** and **reward functions**, and organize at a mathematical level how these are used in the actual learning of LLMs (large language models).

1.1 Learning Outcomes

We explicitly state the Learning Outcomes of this lecture. By the end of this lecture, students should be able to:

- Explain major training data formats such as input-output pairs, input and reward functions, and preferences for input and output pairs.
- Explain the types of objective functions corresponding to the training data formats.

In the following, we first summarize the mathematical notation as in the previous lecture, then define typical formats of training data, and rigorously formulate what kind of objective functions are used for each training data format in the case of LLMs. Finally, we will overlook how to design and learn reward functions based on technical reports of actual large-scale models.

2 Preparation: Recap of 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 script letters. Example: $\mathcal{A}$.

- $x \in \mathcal{A}$: Indicates that element x belongs to set $\mathcal{A}$.
- $\{\}$: Empty set.
- $\{a, b, c\}$: Set consisting of elements a, b, c (roster notation).
- $\{x \in \mathcal{A} \mid P(x)\}$: Set consisting of elements of set $\mathcal{A}$ for which proposition $P(x)$ is true (set-builder notation).
- $\mathbb{R}$: Set of all real numbers.
- $\mathbb{R}_{>0}$: Set of all positive real numbers.
- $\mathbb{R}_{\geq 0}$: Set of all non-negative real numbers.
- $\mathbb{Z}$: Set of all integers.
- $\mathbb{Z}_{>0}$: Set of all positive integers.
- $\mathbb{Z}_{\geq 0}$: Set of all non-negative integers.
- $[1, k]_{\mathbb{Z}} \coloneqq \{1, 2, \ldots, k\}$: Set of integers from 1 to k for a positive integer k.

- **Function:**
  - $f : \mathcal{X} \to \mathcal{Y}$: Indicates that 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 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 vector $\boldsymbol{v}$ is an n-dimensional real vector.
  - The i-th element of vector $\boldsymbol{v}$ is denoted as 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 matrix $\boldsymbol{A}$ is a real matrix with m rows and n columns.

– The element in the i-th row and j-th column of matrix $\boldsymbol{A}$ is denoted as $a_{i,j}$.

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

– The transpose of matrix $\boldsymbol{A}$ is denoted as $\boldsymbol{A}^\top$. 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 term tensor simply refers to a multidimensional 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, such as $\underline{\boldsymbol{A}}$.

  – Students who have already learned abstract tensors in mathematics or physics may feel resistant to calling a simple multidimensional array a tensor. If we assume that the basis is always fixed to the standard basis and identify the mathematical tensor with its component representation (which becomes a multidimensional array), there is (tentatively) consistency in terminology.

3 Types of Training Data

In the learning phase of AI using machine learning, while there is room to freely change parametric functions, objective functions, and algorithms, training data is often obtained from the real world, so its format cannot be easily changed. Depending on the application, there are formats that are easy to obtain and those that are difficult. Therefore, it is necessary to appropriately design the objective function according to the format of the training data. To do this, it is necessary to understand the major formats of training data.

In this section, we introduce the major formats of training data. In the following, let the input space be $\mathcal{X}$ and the output space be $\mathcal{Y}$. Since both inputs and outputs are often vectors or sequences, we will denote them with bold italic lowercase letters like $\boldsymbol{x}$, $\boldsymbol{y}$.

3.1 Input-Output Pair Sequence

Definition 3.1 (Input-Output Pair Sequence)**.** Let $m \in \mathbb{Z}_{>0}$ be the number of training data points. An **input-output pair sequence** is a sequence

$$(\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^{m} \in (\mathcal{X} \times \mathcal{Y})^m \tag{5}$$

where for each $i \in [1, m]_{\mathbb{Z}}$, $\boldsymbol{x}_i \in \mathcal{X}$ is an input, and $\boldsymbol{y}_i \in \mathcal{Y}$ is an element that is **expected to be an appropriate output** for that input.
In this case, the training dataset is written as

$$\mathcal{D}^{\mathrm{IO}} := \{(\boldsymbol{x}_i, \boldsymbol{y}_i) \mid i \in [1, m]_{\mathbb{Z}}\}. \tag{6}$$

Remark 3.1 (Burden in Supervised Learning and Generative AI)**.** The problem setting dealing with data in the format of Definition 3.1 is called (narrowly defined) **supervised learning**. That is, it is a situation where $\boldsymbol{y}_i$ corresponding to a "correct label" or "model answer" is given for each input $\boldsymbol{x}_i$.
A characteristic of generative AI is that the output space $\mathcal{Y}$ is extremely complex, such as natural language sequences, code sequences, or images. Therefore, the task of manually creating an appropriate $\boldsymbol{y}_i$ for an input $\boldsymbol{x}_i$ imposes a greater burden compared to other machine learning applications (e.g., class labeling in image classification). In this sense, while "input-output" pair sequences are theoretically easy to understand, they are not necessarily a data format that is easy to obtain in practice.

3.2 Input-Two-Outputs-Preference Tuple Sequence

Definition 3.2 (Input-Two-Outputs-Preference Tuple Sequence)**.** Let $m \in \mathbb{Z}_{>0}$ be the number of training data points. An **input-two-outputs-preference tuple sequence** is a sequence

$$(\boldsymbol{x}_i, \boldsymbol{y}_i^-, \boldsymbol{y}_i^+, b_i)_{i=1}^{m} \in (\mathcal{X} \times \mathcal{Y} \times \mathcal{Y} \times \{-1, +1\})^m \tag{7}$$

where for each $i \in [1, m]_{\mathbb{Z}}$:

- $\boldsymbol{x}_i \in \mathcal{X}$ is an input,
- $\boldsymbol{y}_i^-, \boldsymbol{y}_i^+ \in \mathcal{Y}$ are two candidate outputs for the same input $\boldsymbol{x}_i$,
- $b_i \in \{-1, +1\}$ is a label representing which output is preferred for the input $\boldsymbol{x}_i$ by a sign, and has the meaning

$$b_i = \begin{cases} +1 & \text{when } \boldsymbol{y}_i^+ \text{ is expected to be more suitable than } \boldsymbol{y}_i^-, \\ -1 & \text{when } \boldsymbol{y}_i^- \text{ is expected to be more suitable than } \boldsymbol{y}_i^+. \end{cases} \tag{8}$$

In this case, the training dataset is written as

$$\mathcal{D}^{\mathrm{pref}} := \left\{(\boldsymbol{x}_i, \boldsymbol{y}_i^-, \boldsymbol{y}_i^+, b_i) \mid i \in [1, m]_{\mathbb{Z}}\right\}. \tag{9}$$

Remark 3.2 (Reinforcement Learning from Human Feedback)**.** In dialogue-based generative models such as OpenAI ChatGPT, an interface where users select b_i in the format of Definition 3.2 is often used. The user simply needs to choose "which response is preferable" from two options and does not need to write an absolute correct answer from scratch.
Generally, the problem setting of collecting **comparative evaluations (preferences)** for multiple outputs generated by the model and performing learning using them is often called **reinforcement learning from human feedback (RLHF)**[1].
Even if it is difficult to generate an output by oneself, it is relatively easy to compare "which one is better," so the burden of data collection is often smaller for input-two-outputs-preference tuple sequences compared to "input-output" pair sequences.

3.3 Input-Output Reward Function Pair Sequence

Definition 3.3 (Input-Output Reward Function Pair Sequence)**.** Let $m \in \mathbb{Z}_{>0}$ be the number of training data points. An **input-output reward function** is a situation where a real-valued function for the output

$$R_i : \mathcal{Y} \to \mathbb{R} \tag{10}$$

is given for each input $\boldsymbol{x}_i \in \mathcal{X}$, and in this case, $R_i(\boldsymbol{y})$ is interpreted as

$$R_i(\boldsymbol{y}) \text{ is a value quantifying how good the output } \boldsymbol{y} \text{ is for the input } \boldsymbol{x}_i. \tag{11}$$

An **input-reward-function pair sequence** refers to the sequence

$$\left(\boldsymbol{x}_i, R_i\right)_{i=1}^m \in (\mathcal{X} \times \mathbb{R}^{\mathcal{Y}})^m. \tag{12}$$

In this case, the training dataset is written as

$$\mathcal{D}^{\mathrm{rew}} := \left\{(\boldsymbol{x}_i, R_i) \mid i \in [1, m]_{\mathbb{Z}}\right\}. \tag{13}$$

Remark 3.3 (Implementation of Reward Functions by Programs)**.** The reward function R_i in Definition 3.3 does not necessarily need to be explicitly given in a closed form; it may be implemented as a "program that receives input $\boldsymbol{x}_i$ and candidate output $\boldsymbol{y}$ and returns a score." For example:

- In the case of a math problem, extract the final numerical solution, and set $R_i(\boldsymbol{y}) = 1$ if it matches the correct answer, and $R_i(\boldsymbol{y}) = 0$ otherwise.
- In the case of a programming problem, compile and execute the submitted code, and set $R_i(\boldsymbol{y})$ according to the number of test cases passed.

In this way, when R_i can be defined by a program, even if a new candidate output y is generated during the training process, it can be evaluated immediately, so it is often practically easier to handle compared to "input-output" pair sequences.

Example 3.1 (Design Example of Reward Functions in DeepSeek Series Models)**.** In the technical report of DeepSeek-R1[2], when performing reinforcement learning using GRPO (Group Relative Policy Optimization)[3] on the base model, math problems and programming problems are used as inputs, and the reward function R_i is designed based on rules. Specifically, the input is a "problem statement" or "programming task," and the output y is a natural language sequence including a long Chain-of-Thought (CoT) and a final answer. At this time, the reward function R_i is roughly decomposed additively as

$$R_i(y) = R_i^{\mathrm{acc}}(y) + R_i^{\mathrm{fmt}}(y). \tag{14}$$

Here:

- $R_i^{\mathrm{acc}}(y)$ is the **accuracy reward**, evaluating the correctness of the problem. For math problems, the final answer is required to be output in a specific format (e.g., a format like `\boxed{...}`), and a positive reward is given if it matches the correct answer, otherwise 0 or a negative reward is given. For programming problems, correctness is automatically judged using a compiler and a set of test cases, and a reward is given according to the number of passed tests.
- $R_i^{\mathrm{fmt}}(y)$ is the **format reward**, evaluating whether the output format constraints, such as enclosing the thought process with `<think>` and `</think>` tags, are observed.

The report states that for training DeepSeek-R1-Zero, they adopt a "rule-based reward system that mainly consists of two types of rewards," explaining that learning proceeds based on two types of rule-based rewards: accuracy and format[2]. Thus, in actual large-scale models, although R_i is implemented as a fairly complex program, it often maintains a simple decomposition structure like Equation (14).

Remark 3.4 (Extension by Separately Learned Reward Models)**.** The reward function R_i can also be defined using a **learned reward model** instead of a rule-based one. Specifically, a parametric function family

$$r_\phi : \mathcal{X} \times \mathcal{Y} \to \mathbb{R} \tag{15}$$

is introduced, and ϕ is learned using preference data or data labeled with "good/bad." Using the learned reward model, the reward function can be defined as

$$R_i(y) \coloneqq r_\phi(x_i, y). \tag{16}$$

In the case of generative AI, since the output y is often a long sequence of tokens, R_i is

sometimes defined for **subsequences** as well. For example, **process-based reward** has been proposed, where $R_i(\boldsymbol{y}_{1:t})$ is given for the reasoning process up to an intermediate point $\boldsymbol{y}_{1:t}$ to evaluate the quality during inference. Even in this case, it can essentially be handled within the framework of Definition 3.3.

3.4 Relationship with More General Frameworks in Reinforcement Learning

Other problem settings can also be considered, such as history sequences of states, actions, and rewards handled by **offline reinforcement learning**

$$(s_t, a_t, r_t)_{t=1}^{T} \tag{17}$$

or implicit state transitions and reward functions handled by **online reinforcement learning**

$$p(s_{t+1} \mid s_t, a_t), \quad r(s_t, a_t). \tag{18}$$

These are general frameworks where rewards may be given for parts of the data points.

However, in the case of generative AI, it is often natural to evaluate the ”completed output” (the full natural language response or code), and it can often be sufficiently expressed in the ”input-output reward function” format like Definition 3.3.

Furthermore, bringing in the general reinforcement learning framework including state transitions and long-term discounted rewards as is, while theoretically increasing expressiveness, also causes problems such as:

- Increased number of necessary hyperparameters and design freedom,
- Increased instability of learning and implementation complexity,
- Difficulty in practical debugging.

Unnecessary generalization may invite learning difficulties, and in many practical studies related to generative AI, simple settings focusing on **outputs completed per input and their rewards** are preferred.

4 Design Methods of Objective Functions for Various Training Data in the Case of LLMs

When LLMs, more specifically **stochastic language models**, are viewed as parametric conditional probability mass functions, we organize what kind of objective functions are used practically for the training data formats explained in the previous section, corresponding to the default implementation of the practical library Hugging Face TRL.

4.1 Basic Formulation of LLMs

First, we formally define an LLM as a conditional probability mass function.

Definition 4.1 (LLM and Conditional Probability Mass Function)**.** Let the vocabulary set be a finite set $\mathcal{V}$, and denote a token sequence as

$$\boldsymbol{y} = (y_1, \ldots, y_T) \in \mathcal{V}^T. \tag{19}$$

For an output sequence $\boldsymbol{y}$ given an input $\boldsymbol{x} \in \mathcal{X}$, we write the **parametric conditional probability mass function** as

$$P_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}). \tag{20}$$

Here, $\boldsymbol{\theta} \in \Theta \subset \mathbb{R}^d$ represents the model parameters.
In many cases, $P_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x})$ is decomposed autoregressively as follows:

$$P_{\boldsymbol{\theta}}(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=1}^{T} P_{\boldsymbol{\theta}}(y_t \mid \boldsymbol{x}, y_{<t}), \tag{21}$$

where $y_{<t} := (y_1, \ldots, y_{t-1})$.

In the above equation, $P_{\boldsymbol{\theta}}(y_t \mid \boldsymbol{x}, y_{<t})$ is often implemented by a neural network as a classifier using softmax, but in this lecture, we will not go into those details and treat it abstractly as a conditional probability mass function.

In the following, we define the learning objective function for $P_{\boldsymbol{\theta}}$ corresponding to the TRL implementation for each training data format given in Definitions 3.1, 3.2, and 3.3.

4.2 Supervised Fine-Tuning (SFT) for Input-Output Pair Sequences

For the input-output pair sequence in Definition 3.1, TRL's `SFTTrainer` is basically implemented to minimize the cross-entropy objective function corresponding to "supervised fine-tuning (SFT)."

Definition 4.2 (SFT Objective Function (Token-level Cross-Entropy Minimization))**.** Let an input-output pair sequence $(\boldsymbol{x}_i, \boldsymbol{y}_i)_{i=1}^m$ be given according to Definition 3.1. Represent each $\boldsymbol{y}_i$ as a token sequence

$$\boldsymbol{y}_i = (y_{i,1}, \ldots, y_{i,T_i}) \in \mathcal{V}^{T_i}. \tag{22}$$

In this case, the **SFT objective function** for the conditional probability mass function $P_{\boldsymbol{\theta}}$ in Definition 4.1 is defined as

$$\mathcal{L}_{\mathrm{SFT}}(\boldsymbol{\theta}) := -\frac{1}{\sum_{i=1}^m T_i} \sum_{i=1}^{m} \sum_{t=1}^{T_i} \log P_{\boldsymbol{\theta}}\left(y_{i,t} \middle| \boldsymbol{x}_i, \boldsymbol{y}_{i,<t}\right). \tag{23}$$

That is, it is the average of the negative log conditional probabilities per token.

Remark 4.1 (Behavior of SFTTrainer in TRL)**.** The goal of **SFT (supervised fine-tuning)** is "to adjust P_θ so that high conditional probabilities are given to correct/preferred output sequences."

4.3 DPO for Input-Two-Outputs-Preference Tuple Sequences

Next, we define the objective function and algorithm for **Direct Preference Optimization (DPO)** [4] for preference data.

Definition 4.3 (Objective Function and Algorithm of DPO)**.** According to Definition 3.2, assume that an input-two-outputs-preference tuple sequence

$$\mathcal{D}^{\mathrm{pref}} = \left\{(\boldsymbol{x}_i, \boldsymbol{y}_i^-, \boldsymbol{y}_i^+, b_i) \mid i \in [1, m]_{\mathbb{Z}}\right\} \tag{24}$$

is given. Here $b_i \in \{-1, +1\}$ is a signed label indicating "which one is preferred," as in Definition 3.2. Furthermore, assume that a fixed conditional probability mass function $P_{\mathrm{ref}}(\cdot|\cdot)$ called the **reference model** is given.
For each data point i, let the log probability difference in the model and the reference model be

$$\Delta \log P_{\boldsymbol{\theta}}(i) \coloneqq \log P_{\boldsymbol{\theta}}(\boldsymbol{y}_i^+ \mid \boldsymbol{x}_i) - \log P_{\boldsymbol{\theta}}(\boldsymbol{y}_i^- \mid \boldsymbol{x}_i), \tag{25}$$

$$\Delta \log P_{\mathrm{ref}}(i) \coloneqq \log P_{\mathrm{ref}}(\boldsymbol{y}_i^+ \mid \boldsymbol{x}_i) - \log P_{\mathrm{ref}}(\boldsymbol{y}_i^- \mid \boldsymbol{x}_i). \tag{26}$$

For a temperature parameter $\beta > 0$, define

$$\alpha_i(\boldsymbol{\theta}) \coloneqq \beta b_i \Big[\Delta \log P_{\boldsymbol{\theta}}(i) - \Delta \log P_{\mathrm{ref}}(i)\Big]. \tag{27}$$

Using the logistic function

$$\sigma(u) \coloneqq \frac{1}{1 + \exp(-u)}, \tag{28}$$

define the DPO loss per sample as

$$\ell_i^{\mathrm{DPO}}(\boldsymbol{\theta}) \coloneqq -\log \sigma\big(\alpha_i(\boldsymbol{\theta})\big). \tag{29}$$

The DPO objective function is the average of this, which is

$$\mathcal{L}_{\mathrm{DPO}}(\boldsymbol{\theta}) \coloneqq \frac{1}{m} \sum_{i=1}^{m} \ell_i^{\mathrm{DPO}}(\boldsymbol{\theta}). \tag{30}$$

a

[a] Hugging Face TRL's `DPOTrainer` implements an objective function isomorphic to equations (27)–(30) under default settings (`loss_type="sigmoid"`, `label_smoothing=0`, `f_divergence_type="reverse_kl"`, etc.).

Remark 4.2 (Intuition of DPO Objective Function and TRL Implementation)**.** $\alpha_i(\boldsymbol{\theta})$ in Equation (27) is the amount of change in the log probability difference between "preferred output" and "less preferred output," taking the difference between

- the difference $\Delta \log P_{\boldsymbol{\theta}}(i)$ under $P_{\boldsymbol{\theta}}$ and
- the difference $\Delta \log P_{\mathrm{ref}}(i)$ under the reference model P_{ref}.

That is, it is a metric measuring how much the preferred output can be favored when compared to the reference model.
The reference model P_{ref} is not updated and functions as a reference point for updates, providing implicit regularization so as not to deviate too significantly.

4.4 GRPO for Input-Output Reward Function Pair Sequences

Finally, as a design example of an objective function for input-output reward function pair sequences, we define the objective function and algorithm of **Group Relative Policy Optimization (GRPO)**[3, 2], which is used in DeepSeek-R1 and others.

Definition 4.4 (Objective Function and Algorithm of GRPO)**.** Assume that an input-output reward function pair sequence in Definition 3.3

$$\mathcal{D}^{\mathrm{rew}} = \left\{(\boldsymbol{x}_i, R_i) \mid i \in [1, m]_{\mathbb{Z}}\right\} \tag{31}$$

and a fixed conditional probability mass function P_{ref} called the reference policy are given. In one step of GRPO, letting the current parameter vector be $\boldsymbol{\theta}_{\mathrm{old}}$, we proceed as follows.

1. **Group Sampling:** Using a fixed group size $G \in \mathbb{Z}_{>0}$, independently sample G outputs from the old policy for each input $\boldsymbol{x}_i$ as

$$\boldsymbol{y}_{i,j} \sim P_{\boldsymbol{\theta}_{\mathrm{old}}}(\cdot \mid \boldsymbol{x}_i), \quad j \in [1, G]_{\mathbb{Z}}. \tag{32}$$

2. **Group-wise Reward and Advantage Normalization:** For each (i, j), calculate the reward of the sampled output

$$r_{i,j} \coloneqq R_i(\boldsymbol{y}_{i,j}) \tag{33}$$

and define the group relative advantage normalized by it as

$$A_{i,j} \coloneqq \frac{r_{i,j} - \mu_i}{\sigma_i}. \tag{34}$$

Here, the group mean reward and standard deviation corresponding to input i are

defined as

$$\mu_i \coloneqq \frac{1}{G}\sum_{j=1}^{G} r_{i,j}, \qquad \sigma_i \coloneqq \sqrt{\frac{1}{G}\sum_{j=1}^{G}(r_{i,j}-\mu_i)^2}. \tag{35}$$

3. **GRPO Objective Function:** For each (i,j) and token position t, define the probability ratio of the current policy and the old policy as

$$\rho_{i,j,t}(\boldsymbol{\theta}) \coloneqq \frac{P_{\boldsymbol{\theta}}(y_{i,j,t} \mid \boldsymbol{x}_i, y_{i,j,<t})}{P_{\boldsymbol{\theta}_{\text{old}}}(y_{i,j,t} \mid \boldsymbol{x}_i, y_{i,j,<t})}. \tag{36}$$

Furthermore, introduce reverse KL regularization with the reference policy P_{ref}, using the sample-based approximation

$$\text{KL}^{\text{rev}}_{i,j,t}(\boldsymbol{\theta}) \coloneqq \log P_{\boldsymbol{\theta}}(y_{i,j,t} \mid \boldsymbol{x}_i, y_{i,j,<t}) - \log P_{\text{ref}}(y_{i,j,t} \mid \boldsymbol{x}_i, y_{i,j,<t}). \tag{37}$$

Using the KL regularization coefficient $\beta > 0$, the empirical objective function of GRPO is defined as

$$\mathcal{L}^{(k)}_{\text{GRPO}}(\boldsymbol{\theta}) \coloneqq -\frac{1}{\sum_{i=1}^{m}\sum_{j=1}^{G} T_{i,j}} \sum_{i=1}^{m}\sum_{j=1}^{G}\sum_{t=1}^{T_{i,j}} \Big[\rho_{i,j,t}(\boldsymbol{\theta})A_{i,j} - \beta\,\text{KL}^{\text{rev}}_{i,j,t}(\boldsymbol{\theta})\Big]. \tag{38}$$

GRPO minimizes this using stochastic gradient methods or similar approaches.

Remark 4.3 (Intuition of GRPO Objective Function)**.** When applying gradient methods to the GRPO objective function, for samples with $A_{i,j} > 0$, the gradient works in the direction of increasing $\rho_{i,j,t}(\boldsymbol{\theta})$, that is, increasing the probability $P_{\boldsymbol{\theta}}(y_{i,j,t} \mid \boldsymbol{x}_i, y_{i,j,<t})$ under the current policy, and for samples with $A_{i,j} < 0$, the gradient works in the direction of decreasing the probability. Also, since $A_{i,j}$ is a group-normalized advantage, it is robust to differences in reward scales.
The reverse Kullback–Leibler regularization term $\text{KL}^{\text{rev}}_{i,j,t}(\boldsymbol{\theta})$ suppresses $P_{\boldsymbol{\theta}}$ from moving excessively to new modes compared to P_{ref}.

Remark 4.4 (Clipping of GRPO in DeepSeek Technical Papers [3, 2])**.** GRPO was proposed in DeepSeek's technical papers[3, 2], where clipping is also proposed. We describe it here. Define the clipped ratio

$$\tilde{\rho}_{i,j,t}(\boldsymbol{\theta}) \coloneqq \text{clip}\big(\rho_{i,j,t}(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon\big). \tag{39}$$

The empirical objective function of GRPO with clipping is defined as

$$\mathcal{L}^{(k)}_{\text{GRPOClip}}(\boldsymbol{\theta}) \coloneqq -\frac{1}{\sum_{i=1}^{m}\sum_{j=1}^{G} T_{i,j}} \sum_{i=1}^{m}\sum_{j=1}^{G}\sum_{t=1}^{T_{i,j}} \Big[\min\big(\rho_{i,j,t}(\boldsymbol{\theta})A_{i,j},\ \tilde{\rho}_{i,j,t}(\boldsymbol{\theta})A_{i,j}\big) - \beta\,\text{KL}^{\text{rev}}_{i,j,t}(\boldsymbol{\theta})\Big]. \tag{40}$$

Let us look in detail at when clipping and the minimum value operation change the objective function. By the clipping function $\mathrm{clip}(\cdot, 1-\epsilon, 1+\epsilon)$ in Equation (39), $\tilde{\rho}_{i,j,t}(\boldsymbol{\theta})$ is given by

$$\tilde{\rho}_{i,j,t}(\boldsymbol{\theta}) = \begin{cases} 1-\epsilon & \text{if } \rho_{i,j,t}(\boldsymbol{\theta}) < 1-\epsilon, \\ \rho_{i,j,t}(\boldsymbol{\theta}) & \text{if } 1-\epsilon \le \rho_{i,j,t}(\boldsymbol{\theta}) \le 1+\epsilon, \\ 1+\epsilon & \text{if } \rho_{i,j,t}(\boldsymbol{\theta}) > 1+\epsilon. \end{cases} \tag{41}$$

Using this, $\min\bigl(\rho_{i,j,t}(\boldsymbol{\theta})A_{i,j}, \tilde{\rho}_{i,j,t}(\boldsymbol{\theta})A_{i,j}\bigr)$ can be divided into cases according to the sign of the advantage $A_{i,j}$.

- Case $A_{i,j} > 0$. Since $A_{i,j}$ is positive,

$$\min\bigl(\rho_{i,j,t}(\boldsymbol{\theta})A_{i,j}, \tilde{\rho}_{i,j,t}(\boldsymbol{\theta})A_{i,j}\bigr) = A_{i,j}\min\bigl(\rho_{i,j,t}(\boldsymbol{\theta}), \tilde{\rho}_{i,j,t}(\boldsymbol{\theta})\bigr). \tag{42}$$

  From Equation (41):

  - When $\rho_{i,j,t}(\boldsymbol{\theta}) \le 1+\epsilon$, since $\tilde{\rho}_{i,j,t}(\boldsymbol{\theta}) \ge \rho_{i,j,t}(\boldsymbol{\theta})$, we have $\min(\rho_{i,j,t}(\boldsymbol{\theta}), \tilde{\rho}_{i,j,t}(\boldsymbol{\theta})) = \rho_{i,j,t}(\boldsymbol{\theta})$, so clipping does not change the objective function.
  - When $\rho_{i,j,t}(\boldsymbol{\theta}) > 1+\epsilon$, since $\tilde{\rho}_{i,j,t}(\boldsymbol{\theta}) = 1+\epsilon < \rho_{i,j,t}(\boldsymbol{\theta})$, we have $\min(\rho_{i,j,t}(\boldsymbol{\theta}), \tilde{\rho}_{i,j,t}(\boldsymbol{\theta})) = 1+\epsilon$, so $(1+\epsilon)A_{i,j}$ is used instead of $\rho_{i,j,t}(\boldsymbol{\theta})A_{i,j}$. Therefore, for samples with $A_{i,j} > 0$, the objective function is modified only in the region where the probability ratio $\rho_{i,j,t}(\boldsymbol{\theta})$ exceeds $1+\epsilon$, suppressing the update amount of the policy. This is a mechanism to prevent unstable updates caused by excessively increasing the probability for high-reward samples.

- Case $A_{i,j} < 0$. Since $A_{i,j}$ is negative,

$$\min\bigl(\rho_{i,j,t}(\boldsymbol{\theta})A_{i,j}, \tilde{\rho}_{i,j,t}(\boldsymbol{\theta})A_{i,j}\bigr) = A_{i,j}\max\bigl(\rho_{i,j,t}(\boldsymbol{\theta}), \tilde{\rho}_{i,j,t}(\boldsymbol{\theta})\bigr). \tag{43}$$

  From Equation (41):

  - When $\rho_{i,j,t}(\boldsymbol{\theta}) \ge 1-\epsilon$, since $\tilde{\rho}_{i,j,t}(\boldsymbol{\theta}) \le \rho_{i,j,t}(\boldsymbol{\theta})$ or they are equal, we have $\max(\rho_{i,j,t}(\boldsymbol{\theta}), \tilde{\rho}_{i,j,t}(\boldsymbol{\theta})) = \rho_{i,j,t}(\boldsymbol{\theta})$, so clipping does not change the objective function.
  - When $\rho_{i,j,t}(\boldsymbol{\theta}) < 1-\epsilon$, since $\tilde{\rho}_{i,j,t}(\boldsymbol{\theta}) = 1-\epsilon > \rho_{i,j,t}(\boldsymbol{\theta})$, we have $\max(\rho_{i,j,t}(\boldsymbol{\theta}), \tilde{\rho}_{i,j,t}(\boldsymbol{\theta})) = 1-\epsilon$, so $(1-\epsilon)A_{i,j}$ is used instead of $\rho_{i,j,t}(\boldsymbol{\theta})A_{i,j}$. Therefore, for samples with $A_{i,j} < 0$, the objective function is modified only in the region where the probability ratio $\rho_{i,j,t}(\boldsymbol{\theta})$ becomes smaller than $1-\epsilon$, suppressing the extreme reduction of probability to near zero by the policy update. This is a mechanism to prevent loss of exploration and instability of learning caused by rapidly reducing probability for low-reward samples.

5 Practice of Reward Functions

In the case of frameworks using reward functions, the problem is how to obtain the reward function. Broadly speaking, there are:

- Cases where it is given by rules, and
- Methods to obtain the reward function itself by learning it as a parametric function (specifically, a neural network).

We introduce prominent examples for each.

5.1 Design Example of Reward Functions Used in GRPO: DeepSeek-R1-Zero

In the technical report of DeepSeek-R1[2], there is a detailed description of reward design when training DeepSeek-R1-Zero. R1-Zero is a model that acquired reasoning capabilities only through large-scale RL without using SFT, and its core is the GRPO algorithm and rule-based rewards.

The outline of the reward design is as described in Example 3.1: two types of rule-based rewards based on ”accuracy” and ”format.” In the report, for example, the total reward is defined as a weighted linear combination like

$$R_i(\boldsymbol{y}) = R_i^{\mathrm{acc}}(\boldsymbol{y}) + \lambda_{\mathrm{fmt}} R_i^{\mathrm{fmt}}(\boldsymbol{y}) \tag{44}$$

(λ_{fmt} is a hyperparameter).

- **Accuracy rewards:** Automatically judge whether the final answer is correct for math problems and programming problems. In the case of math problems, after having the final answer output in a predetermined format, it is judged whether it matches the true answer. In the case of programming problems, correctness is judged by compiling the submitted code and running test cases. The design is such that $R_i^{\mathrm{acc}}(\boldsymbol{y})$ becomes a positive value if it is correct, and 0 or a negative value if it is incorrect.
- **Format rewards:** Require the model to output the thought process between `<think>` and `</think>` and the final answer between `<answer>` and `</answer>`, and give $R_i^{\mathrm{fmt}}(\boldsymbol{y})$ according to whether this format is observed. Penalties are given if tags are missing or the order is incorrect.

In the report[2], following the statement ”To train DeepSeek-R1-Zero, we adopt a rule-based reward system that mainly consists of two types of rewards,” details of these two types of rewards are explained. Such rule-based rewards are suitable for large-scale RL in that they do not require human annotation and can automatically evaluate a large number of problems.

5.2 Learning of Reward Models Used in PPO: InstructGPT

On the other hand, when one wants to obtain rewards reflecting **qualities difficult to describe by rules** (e.g., ”politeness” or ”helpfulness”), it is effective to learn the reward function itself as a neural network. A representative example is the RLHF pipeline in OpenAI's InstructGPT[1].

In InstructGPT, the reward model is roughly learned in the following 3 stages.

1. **Learning of SFT Model:** First, using SFT in Definition 4.2, learn an initial policy model $P_{\boldsymbol{\theta}_{\mathrm{SFT}}}$ that follows instructions.

2. **Collection of Preference Data:** Sample multiple responses from the SFT model and have human annotators label ”which response is preferable” in a form like

$$(\boldsymbol{x}_i, \boldsymbol{y}_i^{(1)}, \ldots, \boldsymbol{y}_i^{(K)}, \text{preference}). \tag{45}$$

3. **Learning of Reward Model:** Introduce a parametric reward model

$$r_{\boldsymbol{\phi}} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R} \tag{46}$$

   and for the preference data, for example, for a pair consisting of 2 responses $(\boldsymbol{y}_i^{(w)}, \boldsymbol{y}_i^{(l)})$ (the better one and the worse one), minimize a logistic loss like

$$\mathcal{L}_{\mathrm{RM}}(\boldsymbol{\phi}) \coloneqq -\frac{1}{N}\sum_{i=1}^{N} \log \sigma\Big(r_{\boldsymbol{\phi}}(\boldsymbol{x}_i, \boldsymbol{y}_i^{(w)}) - r_{\boldsymbol{\phi}}(\boldsymbol{x}_i, \boldsymbol{y}_i^{(l)})\Big). \tag{47}$$

   Here σ is the logistic function in Equation (28). In the InstructGPT paper, human preferences are approximated as a scalar value $r_{\boldsymbol{\phi}}$ through such reward model learning[1].

In practice, in [1], Proximal policy optimization (PPO) is applied to update the parameters of the stochastic language model using the learned reward model $r_{\boldsymbol{\phi}}$ obtained above. Note that PPO is a more complex algorithm compared to GRPO, as it learns a value function model together with the stochastic language model within the algorithm, while using the learned reward model.

6 Summary

Finally, we briefly summarize the contents up to this chapter for each Learning Outcome of this lecture.

6.1 Answers to Learning Outcomes

- Since the format of training data cannot necessarily be changed for the convenience of AI engineers, it is necessary to know various types of training data formats. Training data formats include sequences of input-output pairs, sequences of input-two-outputs-preference tuples, and sequences of input-reward function pairs.
- If the format of the training data changes, the most natural objective function also changes accordingly. SFT or similar is applicable to sequences of input-output pairs, DPO or similar to sequences of input-two-outputs-preference tuples, and GRPO or similar to sequences of input-reward function pairs.

References

[1] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, et al., "Training language models to follow instructions with human feedback," Advances in Neural Information Processing Systems, vol. 35, 2022. arXiv:2203.02155.

[2] DeepSeek-AI, "Deepseek-r1: Incentivizing reasoning capability in LLMs via reinforcement learning," arXiv preprint, vol. arXiv:2501.12948, 2025.

[3] Z. Shao, Z. Lin, W. Liu, et al., "Self-rewarding language models," arXiv preprint, vol. arXiv:2401.10020, 2024.

[4] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," Advances in neural information processing systems, vol. 36, pp. 53728–53741, 2023.