Direct Preference Optimization

How brainpower can save you GPU power

National Taiwan University, IA lab sharing, May the 20th 2024

Antoni Maciąg 馬安德

Introduction

- I will present a paper on training text generation models from human preferences.
- The paper: Direct Preference Optimization: Your Language Model is Secretly a Reward Model, Rafailov et al., 2023.
- But it will be a bit different than usual.
- I will not present a novel task, architecture, or way of training that empirically gives better results than previous methods.
- Instead, we will follow *theoretical* math to show that the previous methods of optimizing text generation from human feedback involve computation that nobody realized was unnecessary!
- We will end up with a training algorithm that is mostly equivalent to older ones, but simpler, faster, and more stable.
- The algorithm is called Direct Preference Optimization.

Introduction, continued

- There will be more math than usual.
  - Yay, math!
  - Oh no, math!
- I will also show some very practical consequences of this theoretical math, speaking from my own recent experiences.
- The paper makes use of the mathematical concept of *equivalence relations* (等價關係) that some people might not be familiar with. They make it easier to understand the reasoning, but they're not strictly necessary. We'll talk about them a bit at the end if we have time.
- And a note before we go: please stop me my English gets unclear, or anyone needs a translation of some term into Chinese (the Professor will probably know!).

High-level definition of the task

- Task: train a language generation model from human feedback.
- We are given a dataset *D* of prompts, and two responses for each prompt. We know which response is preferred by humans.
- We don't know "how much better" the better response is!
- We want to learn to generate responses that will be highly valued by humans.

Denotations and assumptions

- x - some prompt.
- y - some response.
- π - some *policy*, i.e. how the model decides what to generate.
- θ - the parameters of our model (i.e. weights of the neural network etc.).
- π_θ - a model's policy if its parameters are θ.
- $\pi(y \mid x)$ - the probability that the model will generate response y to prompt x under policy π.
- $r(x, y)$- some *reward* for generating y in response to x. A numerical measure of how good y is.
- p^* - the human preference distribution. We *assume* it exists, we never have access to it! We need to assume it exists, so that we have a strict mathematical model of what we're doing.
- $p^*(y_1 > y_2 \mid x)$ - the probability that a human will prefer y_1 to y_2 as response to prompt x. So we're assuming that humans act a bit randomly.

Denotations and assumptions - continued

- The Bradley-Terry model - again, we assume it so that we can model the world with math

  Eq. 6: $$p^*(y_1 \succ y_2 \mid x) = \frac{\exp\left(r^*(x, y_1)\right)}{\exp\left(r^*(x, y_1)\right) + \exp\left(r^*(x, y_2)\right)}$$

- (Equation numbers from the paper)
- Humans assign their *preferences* to responses, i.e. which one is better, but not numerical scores. This model assumes that there is some function r^* that reflects human preference numerically. But it's imaginary, so again, we never have access to it!
- $\mathbb{D}_{KL}(d_1 \,||\, d_2)$ - Kullback-Leibler divergence between probabilities distributions d_1 and d_2. Just a numerical score of how different these probabilities are. Precise formula is not important here.
- In particular, $\mathbb{D}_{KL}(\pi_1(y \mid x), || \, \pi_2(y \mid x))$ - the divergence between probability distributions of responses y to prompt x between policies π_1 and π_2. So this just measures how different the policies are for prompt x.

Formal definition of the task

Eq. 11: $$\max_{\pi} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi}\left[r(x,y)\right] - \beta\mathbb{D}_{\mathrm{KL}}\left[\pi(y|x)||\pi_{\mathrm{ref}}(y|x)\right]$$

- We want to find the policy π that maximizes the expected value of this expression, assuming that r is some reward function, prompts x follow the distribution of our dataset D, and we are given some reference policy π_{ref}
- Intuitively, we want to find a policy that gives us highly rewarded responses, without deviating too much from the reference policy.
- “Reference policy” is probably some pretrained model that we are tuning, that already generates grammatical text, so trying not to deviate is a way of making sure that we will not start generating gibberish.
- β is a constant float that determines how much we don’t want to deviate.
- Any questions about the definitions?

Traditional RLHF pipeline

- First, learn to simulate human preferences. Create a reward function r_ϕ (a reward model, neural net) to simulate the imaginary ground-truth human preference function r^*.
- Learn r_ϕ as a function that can be used for binary classification of responses into *better* (y_w) and *worse* (y_l).
-

$$p^*(y_1 \succ y_2 \mid x) = \frac{\exp\left(r^*(x, y_1)\right)}{\exp\left(r^*(x, y_1)\right) + \exp\left(r^*(x, y_2)\right)}. \tag{1}$$

Assuming access to a static dataset of comparisons $\mathcal{D} = \left\{x^{(i)}, y_w^{(i)}, y_l^{(i)}\right\}_{i=1}^N$ sampled from p^*, we can parametrize a reward model $r_\phi(x, y)$ and estimate the parameters via maximum likelihood. Framing the problem as a binary classification we have the negative log-likelihood loss:

$$\mathcal{L}_R(r_\phi, \mathcal{D}) = -\mathbb{E}_{(x, y_w, y_l) \sim \mathcal{D}}\left[\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))\right] \tag{2}$$

Traditional RLHF pipeline - continued

- Substitute r_ϕ for r in Eq. 11 and we get

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(y|x)}\left[r_\phi(x,y)\right] - \beta\mathbb{D}_{\mathrm{KL}}\left[\pi_\theta(y \mid x) \,||\, \pi_{\mathrm{ref}}(y \mid x)\right] \quad (3)$$

- Which we can then optimize using RL algorithms, such as, in recent years, PPO (Proximal Policy Optimization).
- The divergence penalty gets more important, since it keeps the model's generations sensible (coherent text). If they aren't, r_ϕ may stop being accurate (e.g. assign very high rewards to nonsensical text, which it didn't see in the training).
- Is this methodology roughly clear to everyone?

Shortcomings of the pipeline (motivation for DPO)

- We need to train a reward function, which might not be accurate, might not generalize to new model generations, and certainly slows us down and imposes GPU memory constraints before and during the training.
- Good RL algorithms like PPO are simply hard to implement and debug.
- They are also unstable, susceptible to high gradients of the loss function, which can make the training collapse into generating nonsense.
- They are also often computationally heavy, for example PPO requires training an additional value model, in addition to the generator and reward model.

DPO - what it's all about

- We will do some purely mathematical magic and realize that we never needed the trained reward r_ϕ in the first place!
- We will show how to define a loss function that achieves the same objective as RLHF, but it is computationally cheaper, *and* more stable to optimize, *and* perhaps most importantly, *simpler*.
- Based on this loss function, we will *directly* train an optimal policy based on the human preference dataset.
- Hence the name - Direct Preference Optimization.

Sketch of the reasoning

- We will define an equivalence relation (等價關係) R between reward functions.
  - Again, precise definition later, if we have time.
  - Intuitively, this just means that some reward functions, even though they're not the same, will be the same (equivalent) to us. We won't care if we have one or the other.
  - R defines precisely which reward functions will be equivalent to us.
- We will show that if two reward functions are equivalent under R, they same policy is optimal for both of them.
- We will show how to find a Very Interesting Reward Function *r*. It has two Very Interesting properties:
  - It is equivalent to the imaginary ground-truth reward r^* we defined earlier, which perfectly reflects human preference.
  - It will allow us to define a loss function over policies, which we can then directly optimize without the need for RL.
- So, after this simple optimization, we get a policy that reflects human preference.

Equivalence

Definition 1. *We say that two reward functions* $r(x, y)$ *and* $r'(x, y)$ *are equivalent iff* $r(x, y) - r'(x, y) = f(x)$ *for some function* f.

- In other words, the functions are equivalent iff the difference between rewards assigned by them for every prompt *x*, response *y*, depends only on the prompt, and doesn't depend on the responses.

Lemma

- *If two reward functions are equivalent under R, the same policy is optimal for both of them.*
- So, looking back at equation 11. from earlier, which defined the optimal policy for a reward function r:
- $$\max_{\pi} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi}\left[r(x,y)\right] - \beta\mathbb{D}_{\mathrm{KL}}\left[\pi(y|x)||\pi_{\mathrm{ref}}(y|x)\right]$$
- this will mean that if r_1, r_2 are equivalent, then substituting r_1 or r_2 for r in this equation gives us the same policy π (the same π is the "max" π).
- Proof, as stated in the paper:

some algebra

- No, seriously, the proof is just some transformations of algebraic equations. Let's not follow it here because that's not very educational. Folks who are curious can check the paper for details.

More algebra

- We'll use the lemma in a moment. Now, starting again from Eq. 11:

$$\max_{\pi} \mathbb{E}_{x\sim\mathcal{D},y\sim\pi}\left[r(x,y)\right] - \beta\mathbb{D}_{\mathrm{KL}}\left[\pi(y|x)||\pi_{\mathrm{ref}}(y|x)\right]$$

- And doing some algebra again, we can show that the optimal policy π_r for reward r is

$$\pi_r(y \mid x) = \frac{1}{Z(x)}\pi_{\mathrm{ref}}(y \mid x)\exp\left(\frac{1}{\beta}r(x,y)\right) \qquad \text{where } Z(x) = \sum_y \pi_{\mathrm{ref}}(y \mid x)\exp\left(\tfrac{1}{\beta}r(x,y)\right)$$

- Which means (after more algebra, which we will skip again):

$$r(x,y) = \beta\log\frac{\pi_r(y \mid x)}{\pi_{\mathrm{ref}}(y \mid x)} + \beta\log Z(x)$$

- Of course this equation is boring, especially Z. Don't look at it too much!
- Z is some garbage that we will get rid of in a moment. The only thing that matters is that it's a function of x (the prompt), and doesn't depend on y (the response).
- **Why does this matter?**

More algebra, continued

- Because for our equivalence relation from before:

  Definition 1. *We say that two reward functions* $r(x, y)$ *and* $r'(x, y)$ *are equivalent iff* $r(x, y) - r'(x, y) = f(x)$ *for some function* f.

- we can take $f(x) = \beta \log Z(x)$ and then from the lemma it follows that the policy r

$$r(x, y) = \beta \log \frac{\pi_r(y \mid x)}{\pi_{\text{ref}}(y \mid x)} + \beta \log Z(x)$$

- is equivalent to policy r':

$$r'(x, y) = \beta \log \frac{\pi_r(y \mid x)}{\pi_{\text{ref}}(y \mid x)}$$

- Which is the Very Interesting Reward Function that we wanted!
  - Equivalent to the reward that perfectly rewards human preference - we have already proved that.
  - Allowing us to define loss directly over policies - we'll show that next (just 1 more slide!).

The goal - loss over policies

- Recall the equation for trained reward loss used in RL (optimizing r_ϕ to human preferences)

$$\mathcal{L}_R(r_\phi, \mathcal{D}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma(r_\phi(x, y_w) - r_\phi(x, y_l))\right]$$

- But now instead of r_ϕ we can substitute the VIRF and define this loss function, directly optimizing π_θ to human preferences :

$$\mathcal{L}_{\text{DPO}}(\pi_\theta; \pi_{\text{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}}\left[\log \sigma\left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)\right]$$

- So that's what the goal was! A loss function defined directly over policies.
  - No need to train a reward function, or anything else, beside our tuned model.
  - Everything is differentiable. No need for RL.
  - Simple! I mean, we only have to look at the definition of VIRF. We can forget all the algebra that we needed to do to obtain it.
- Mission accomplished! Questions?

Reproduction - why DPO came up

- In fact I needed to implement it before deciding to 報告 about it, because I needed a training method for text generation in an experiment.
- This is still the same experiment that I started after my previous presentation: generating adversarial examples for text classification models in a seq2seq framework.
- It is still ongoing. Lots, lots of details to work out, by trial and error.
- The success rate is only around 10% now. But the adversarial examples are generated fast and they're natural (big improvement over most existing methods).
- I hope to give a more detailed research update in a few months, with better results.
- It will be the topic of my Master's thesis, hopefully in January 2025.

What DPO meant for the experiment

- Allowing me to run trainings with a decent speed and max length of the generation, easing the computational power and GPU memory constraints on pepper.
- Both of these things are important. Can’t graduate if you’re waiting for the model to finish running all the time, or if you retrain for longer sequences and the results don’t generalize.
- I first implemented PPO, not knowing about DPO, which took me about 10 working days.
- Then the Professor suggested DPO and it took like 2 working days from first opening the paper to having a working implementation (though to be fair, I did reuse some code from PPO).

Takeaways

- Math still matters in ML! It's not all about who has more GPU power. It can help us think in a structured way and sometimes realize that there's a simpler, faster way to do something.
- It's worth spending time to look for simple solutions. Easier to look through some papers than understand, implement and debug a complicated, error-prone algo.
    - I learned that the hard way…
- Open question: how much computation is going on in the world right now that we will one day realize was never necessary, like in this case?

Appendix - relations (關係)

- Mathematical relations in general: sets of tuples.
- For example, if the relation is equality on integers (=), then the set is {(0, 0), (1, 1), (2, 2) ...}
- If the relation is < on integers, then the set is {(0, 1), (0, 2), ... (1, 2), (1, 3), ...}.
- Relations have certain properties.
- A relation R is **reflexive (自反)**, if (x, x) belongs to the relation for every x (or we can write, xRx)
  - Examples: = on integers is reflexive, <= and >= are also reflexive, but < and > are not reflexive.
- A relation is **symmetric (對稱)** if xRy ⇔ yRx.
  - = is symmetric. Set inclusion is not symmetric. Being married is symmetric (because if A is married to B, then B is married to A). Being someone's parent is not symmetric (because if A is the parent of B, B is not a parent of A).
- A relation is **transitive (遞移)** xRy and yRz implies xRz.
  - < is transitive, because if x < y and y < z, then x < z. Line parallelism too, because if a || b and b || c, then a || c. Line perpendicularity isn't transitive, because if a ⊥ b and b ⊥ c, then it doesn't have to be true that a ⊥ c.

Appendix - equivalence relations (等價關係)

- If a relation is **reflexive**, **symmetric** and **transitive**, then we say it is an **equivalence** relation.
- We can use equivalence relations to think of some mathematical objects that are “the same” to us in some respect.
- For example, if two neural networks have different weights but always give the same results, we can think of them as equivalent. Indeed, we can verify that the relation “*nRm* ⇔ the net *n* gives the same results as net *m*” is reflexive, symmetrical, and transitive.
- Or maybe we’re doing operations on the uint8 type in C, which means that 255 + 1 = 0. Then 0 is equivalent to 256 and to 512, while 1 is equivalent to 257 and to 513, and so on.
- ERs make it easier to think about lots of stuff in maths. And as we’ve just seen, they can be useful in ML, too!
- Questions about equivalence relations?

謝謝大家