DPO

Umar Jamil

Topics

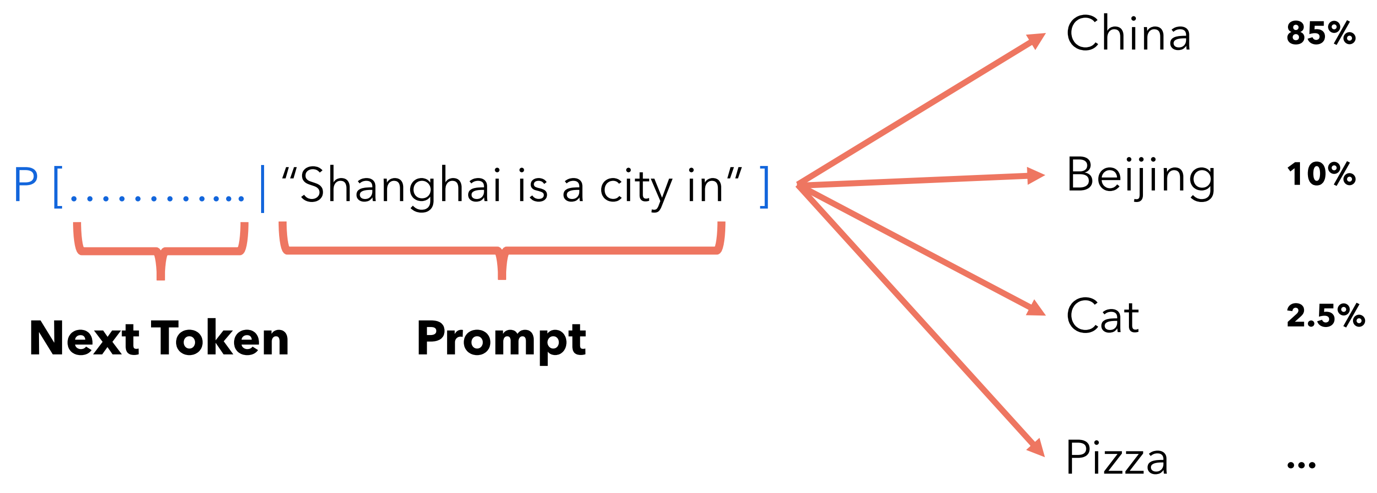
- Short intro to language models
- AI alignment
- Reinforcement Learning
    - The RL setup
    - Connection between RL and language models
    - Reward model
    - The Bradley-Terry model
    - The objective function in RLHF
- DPO
    - Loss function
    - Computing the log probabilities

Prerequisites

- Basics of probability and statistics.
- Basics of deep learning (gradient descent, loss functions, etc.)
- Basics of RLHF (best if you've watched my previous video on RLHF).
- Understanding of the Transformer model and language models

Short intro to language models

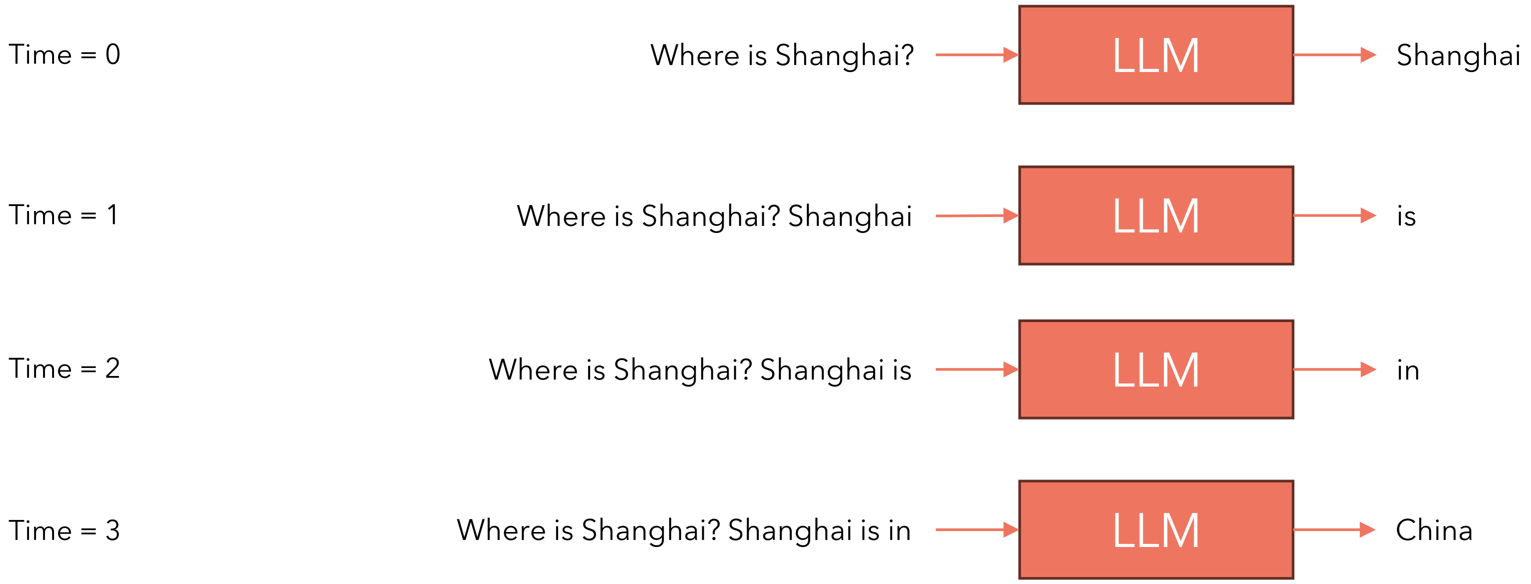
A language model is a probabilistic model that assign probabilities to sequence of words.
In practice, a language model allows us to compute the following:



A language model outputs a list of probabilities, one for every token in the vocabulary, indicating how likely a token is the next one.

Iteratively generating the next token

To generate a complete answer to a prompt, a language model is iteratively queried by adding the previously chosen token to the input.


Time = 0
Where is Shanghai?
LLM
Shanghai
Time = 1
Where is Shanghai? Shanghai
LLM
is
Time = 2
Where is Shanghai? Shanghai is
LLM
in
Time = 3
Where is Shanghai? Shanghai is in
LLM
China

AI alignment

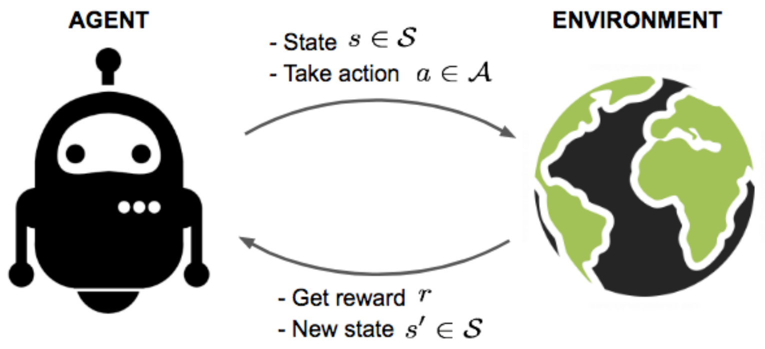
A large language model typically is pretrained on a massive amount of data, for example the entire Wikipedia and billions of web pages. This gives the language model a vast "knowledge" of information to complete any prompt in a reasonable way. However, to use an LLM as a chat assistant (for example ChatGPT) we want to force the language model to follow a particular style. For example, we may want the following:

- Do not use offensive language
- Do not use racist expressions
- Answer questions using a particular style

The goal of AI alignment is to align the model's behavior with a desired behavior.

Introduction to Reinforcement Learning

Reinforcement Learning is concerned with how an intelligent agent should take actions in an environment to maximize the cumulative reward.


AGENT
ENVIRONMENT
- State $s \in \mathcal{S}$
- Take action $a \in \mathcal{A}$
- Get reward r
- New state $s' \in \mathcal{S}$

Let me give you a concrete example with our channel's mascot.

The RL setup

Agent: the cat

State: the position of the cat (x, y) in the grid

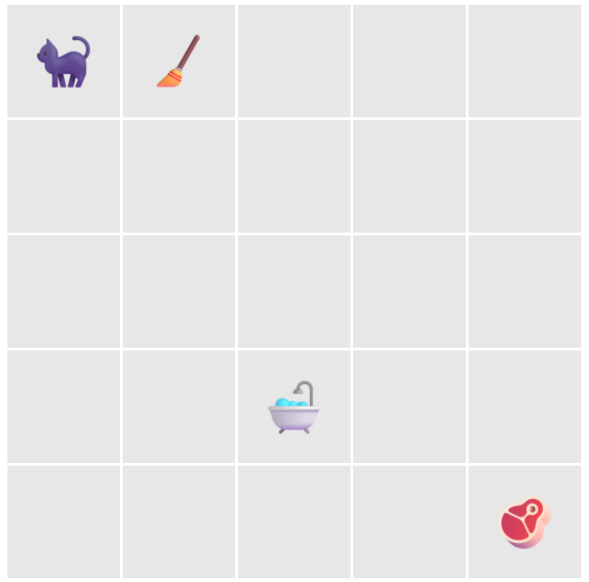
Action: at each position, the cat can move to one of the 4-directionally connected cells. If a move is invalid, the cell will not move and remain in the same position. Every time the cat makes a move, it results in a new state and a reward.

Reward model:

- A move to another empty cell results in a reward of 0.
- A move towards the broom, will result in a reward of -1.
- A move towards the bathtub will result in a reward of -10 and the cat fainting (episode over). The cat will be respawned at the initial position again.
- A move towards the meat will result in a reward of +100

Policy: a policy rules how the agent selects the action to perform given the state it is in: $a_t \sim \pi(\cdot \mid s_t)$

The goal in RL is to select a policy that maximizes the expected return when the agent acts according to it.



The RL setup: connection to language models

Agent: the language model itself

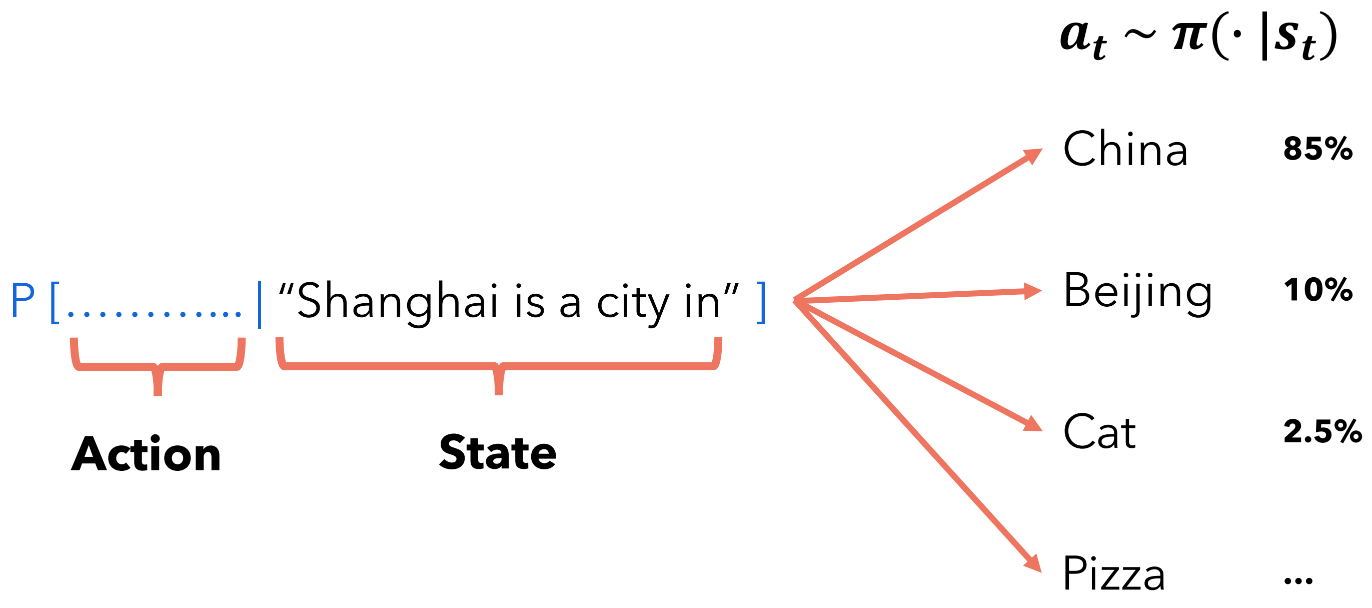
State: the prompt (input tokens)

Action: which token is selected as the next token

Reward model: the language model should be rewarded for generating "good responses" and should not receive any reward for generating "bad responses".

Policy: In the case of language models, the policy is the language model itself! Because it models the probability of the action space given the current state of the agent: $a_t \sim \pi(\cdot\,|s_t)$

Let's look at how we can define the reward model for our language model


$a_t \sim \pi(\cdot\,|s_t)$
P [............ | "Shanghai is a city in"]
Action
State
China 85%
Beijing 10%
Cat 2.5%
Pizza ...

Reward model for language models

It is not easy to create a reward model for language models, because this would require us to create a dataset of prompts and responses and assign a universally accepted "reward" for each answer.

Question (Prompt)	Answer (Text generated by the language model)	Reward (0.0 ~ 1.0)
Where is Shanghai?	Shanghai is a city in China	???
Explain gravity like I'm 5	Gravity is what pulls things toward each other. It's why you stay on the ground and planets orbit the sun.	???
What is 2+2?	4	???

People are so good good at finding a common ground for agreement, but unfortunately, they're good at comparing.

Reward model by comparison

Imagine if we could instead create a dataset of queries and answers and we could ask people to just select which one they prefer. This would be much easier!

Question (Prompt)	Answer 1	Answer 2	Chosen
Where is Shanghai?	Shanghai is a city in China	Shanghai does not exist	1
Explain gravity like I'm 5	Gravity is a famous restaurant	Gravity is what pulls things toward each other. It's why you stay on the ground and planets orbit the sun.	2
What is 2+2?	4	2+2 is a very complicated math problem…	1

Using a dataset like this, we can train a model to assign a score to a given answer. **Let's see how it's done**.

The Bradley-Terry model

Now that we have a dataset of preferences, we need to find a way to convert the preferences into a score (reward). One way to do it is to model our preferences using the Bradley-Terry model.

$$P(y_w > y_l) = \frac{e^{r^*(x,y_w)}}{e^{r^*(x,y_w)} + e^{r^*(x,y_l)}}$$

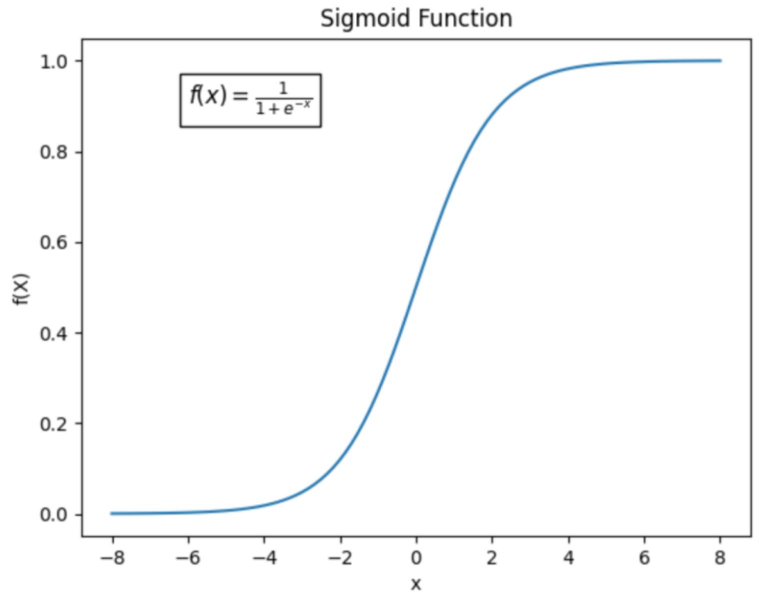
$$\overbrace{\hspace{5cm}}^{x}\overbrace{\hspace{5cm}}^{y_w}\overbrace{\hspace{5cm}}^{y_l}$$

Question/Prompt (x)	**Good/winning answer (y_w)**	**Bad/losing answer (y_l)**
Where is Shanghai?	Shanghai is a city in China	Shanghai does not exist
Explain gravity like I'm 5	Gravity is what pulls things toward each other. It's why you stay on the ground and planets orbit the sun.	Gravity is a famous restaurant
What is 2+2?	4	2+2 is a very complicated math problem…

Of course we want to maximize the probability that the preference model ranks our responses correctly. One way to do it is Maximum Likelihood Estimation. With MLE we can estimate the parameters of a reward model r^* such that the probability of ranking correctly the good and the bad answers is maximized. **Let's see how we can do it!**

Deriving the loss function of the reward model

Let's derive the loss function of the reward model. It's the equation (2) in the DPO paper.



$$P(y_w > y_l) = \frac{e^{r^{\varphi}(x,y_w)}}{e^{r^{\varphi}(x,y_w)} + e^{r^{\varphi}(x,y_l)}}$$

$$\frac{e^A}{e^A + e^B} \Rightarrow \sigma(A - B) \longleftarrow$$

Our goal is to write the Bradley-Terry model's expression as a sigmoid

$$\frac{e^A}{e^A + e^B} = \frac{\frac{e^A}{e^A}}{\frac{e^A + e^B}{e^A}} = \frac{1}{\frac{e^A + e^B}{e^A} + 1 - 1} = \frac{1}{1 + \left(\frac{e^A + e^B}{e^A} - 1\right)} = \frac{1}{1 + \left(\frac{e^A + e^B - e^A}{e^A}\right)} = \frac{1}{1 + \left(\frac{e^B}{e^A}\right)} = \frac{1}{1 + e^{B-A}} = \frac{1}{1 + e^{-(A-B)}} = \sigma(A - B)$$

This gives us the commonly known expression of the reward model. The minus sign is because we want to minimize the loss.

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

The RLHF objective

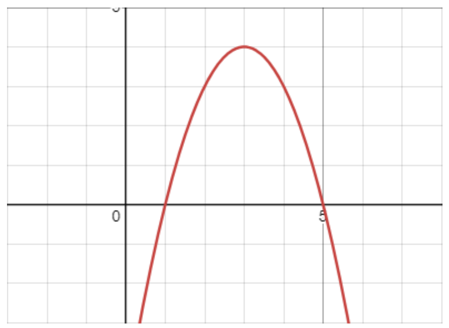
Our goal is to solve the following constrained optimization: we want to find the policy that maximizes the reward obtained when using that policy and at the same time we want the policy to not behave too differently from the original unoptimized policy. This constraint is added to avoid what is known as "reward hacking": the language model (the policy) may just choose sequences of tokens that achieve high reward but may be total gibberish.

$$J_{\text{RLHF}} = \max_{\pi_\theta} \mathbb{E}_{x\sim D, y\sim \pi_\theta(y|x)} \left[\underbrace{r_\varphi(x, y)}_{\text{Maximize the reward}} - \beta \mathbb{D}_{KL}\underbrace{\left[\pi_\theta(y|x) \| \pi_{ref}(y|x)\right]}_{\text{Constraint the model to be not so different from the original one}} \right]$$

What does it mean to maximize an objective function?

The optimization problem

Maximizing a function means to find all the values of one or more variables such that the function is maximized. The same applies for minimization problems. For example, if we have a very simple function like the following: $\mathrm{f}(\mathrm{x}) = -(x-3)^2 + 4$, to maximize it means to find the values of the variable x such that the function is maximized. One way to do it is to calculate the derivative of the function w.r.t to the variable x and set it to zero. In our case, the derivative is $f'(x) = -2x + 6$. If we set it to zero and solve for x, we find that $x^* = 3$ is the value that maximizes the function.



$$x^* = \underset{x}{\operatorname{argmax}} -(x-3)^2 + 4$$

The simple parabola problem is an unconstrained optimization problem. In the case of RLHF we have a **constrained optimization** problem, which means that we want to maximize the reward but at the same time we want the KL divergence to be small.

$$J_{\text{RLHF}} = \max_{\pi_\theta} \mathbb{E}_{x\sim D, y\sim\pi_\theta(y|x)} \left[r_\varphi(x,y) - \beta \mathbb{D}_{KL}[\pi_\theta(y|x) \| \pi_{ref}(y|x)] \right]$$

If you're thinking "why not just run gradient descent on this objective function"? Well, we can't, because the variable y is sampled from the language model itself using various strategies (greedy, beam search, top-k, and similar). **This sampling process is not differentiable. This is the reason we were forced to use RL algorithms like PPO.**

The optimal policy

The solution to the constrained optimization problem is the following. It is the equation (4) in the DPO paper.

$$\pi_r(y|x) = \frac{1}{Z(x)}\pi_{\text{ref}}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)$$

Where $Z(x) = \sum_y \pi_{\text{ref}}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)$. **Great! We have an exact solution to this constrained optimization problem, so now we're all set... right?**

Not really. Evaluating the $Z(x)$ term is not tractable computationally, because it means we would have to generate all possible answers y that can be generated by our language model for every given prompt x.

Suppose that somehow, magically, we have access to an optimal policy $\pi_r(y|x)$ … how would the reward function $r(x,y)$ be for such an optimal policy?

$$\log\pi^*(y|x) = \log\left[\frac{1}{Z(x)}\pi_{\text{ref}}(y|x)\exp\left(\frac{1}{\beta}r(x,y)\right)\right] = \log\pi_{\text{ref}}(y|x) - \log Z(x) + \log\exp\left(\frac{1}{\beta}r(x,y)\right) = \log\pi_{\text{ref}}(y|x) - \log Z(x) + \frac{1}{\beta}r(x,y)$$

$$r(x,y) = \beta\log\frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} + \beta\log Z(x)$$

So, if we had access to an optimal policy, we could use it to obtain a reward function. **OK, but what can we do with it now?**

… remember the Bradley-Terry model?

$$P(y_w > y_l) = \frac{e^{r(x,y_w)}}{e^{r(x,y_w)} + e^{r(x,y_l)}}$$

… and as shown in previous slides…

$$\frac{e^A}{e^A + e^B} = \cdots = \frac{1}{1 + e^{B-A}} = \frac{1}{1 + e^{-(A-B)}} = \sigma(A - B)$$

What if we plugged the expression of the reward function into the expression of the Bradley-Terry model?

$$r(x, y) = \beta \log \frac{\pi^*(y|x)}{\pi_{\text{ref}}(y|x)} + \beta \log Z(x)$$

$$P(y_w > y_l) = \sigma(r(x, y_w) - r(x, y_l)) = \sigma\left(\beta \log \frac{\pi^*(y_w|x)}{\pi_{\text{ref}}(y_w|x)} + \beta \log Z(x) - \beta \log \frac{\pi^*(y_l|x)}{\pi_{\text{ref}}(y_l|x)} - \beta \log Z(x)\right)$$

To learn a Bradley-Terry model that maximizes the probability of choosing y_w over y_l, we just need to maximize the expression above or minimize the negative expression below:

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

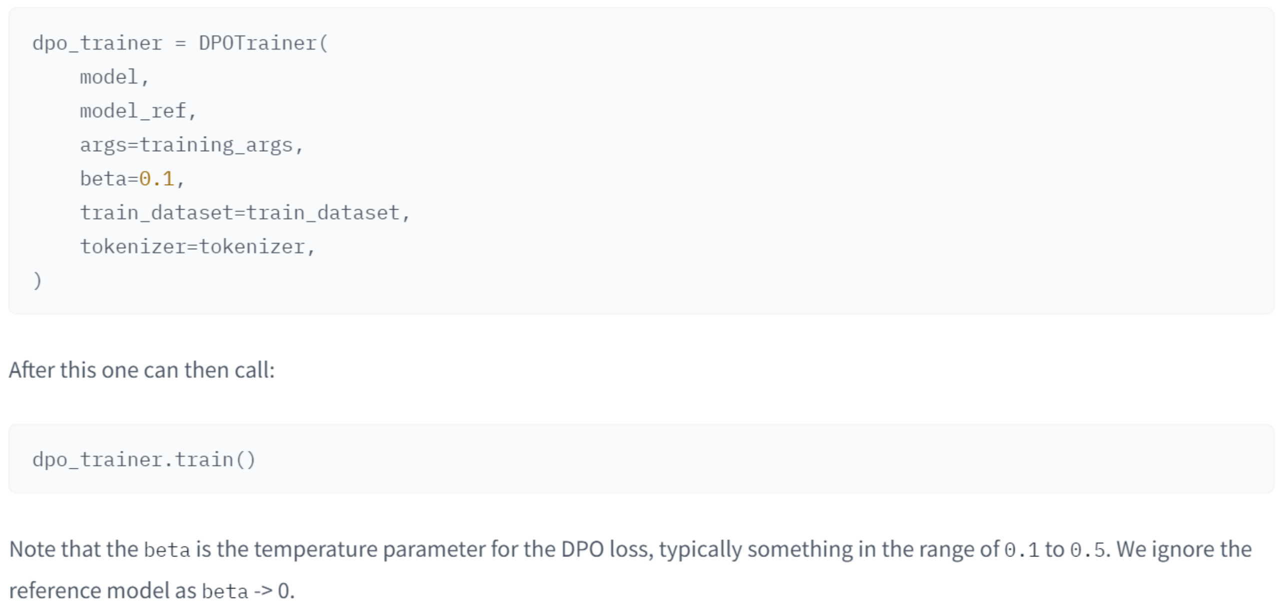
So instead of optimizing the reward function, we are optimizing the optimal policy (which depends on the optimal reward function through the expression in the previous slide). Brilliant solution!

How do we compute the log probabilities?

To evaluate the loss, we need to compute the log probabilities we have in the expression:

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

For example, $\pi_\theta(y_w|x)$ is the probability of generating the response y_w when the model is prompted with the input x. How do we compute this probability, **practically**?



```
dpo_trainer = DPOTrainer(
    model,
    model_ref,
    args=training_args,
    beta=0.1,
    train_dataset=train_dataset,
    tokenizer=tokenizer,
)
```

After this one can then call:

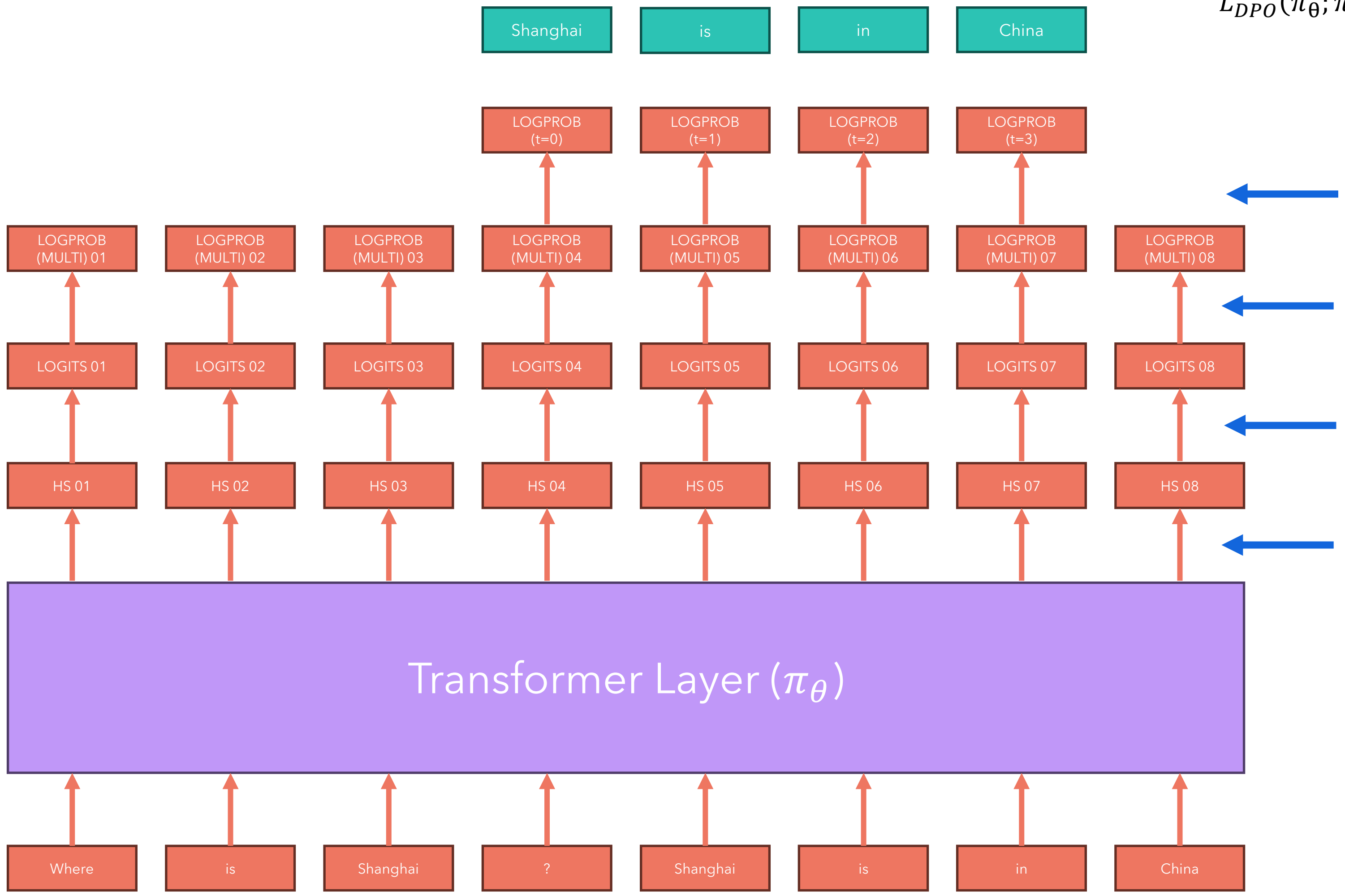
```
dpo_trainer.train()
```

Note that the `beta` is the temperature parameter for the DPO loss, typically something in the range of `0.1` to `0.5`. We ignore the reference model as `beta` -> 0.

Calculating log probabilities

Imagine our language model generated the following response for the given question.

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



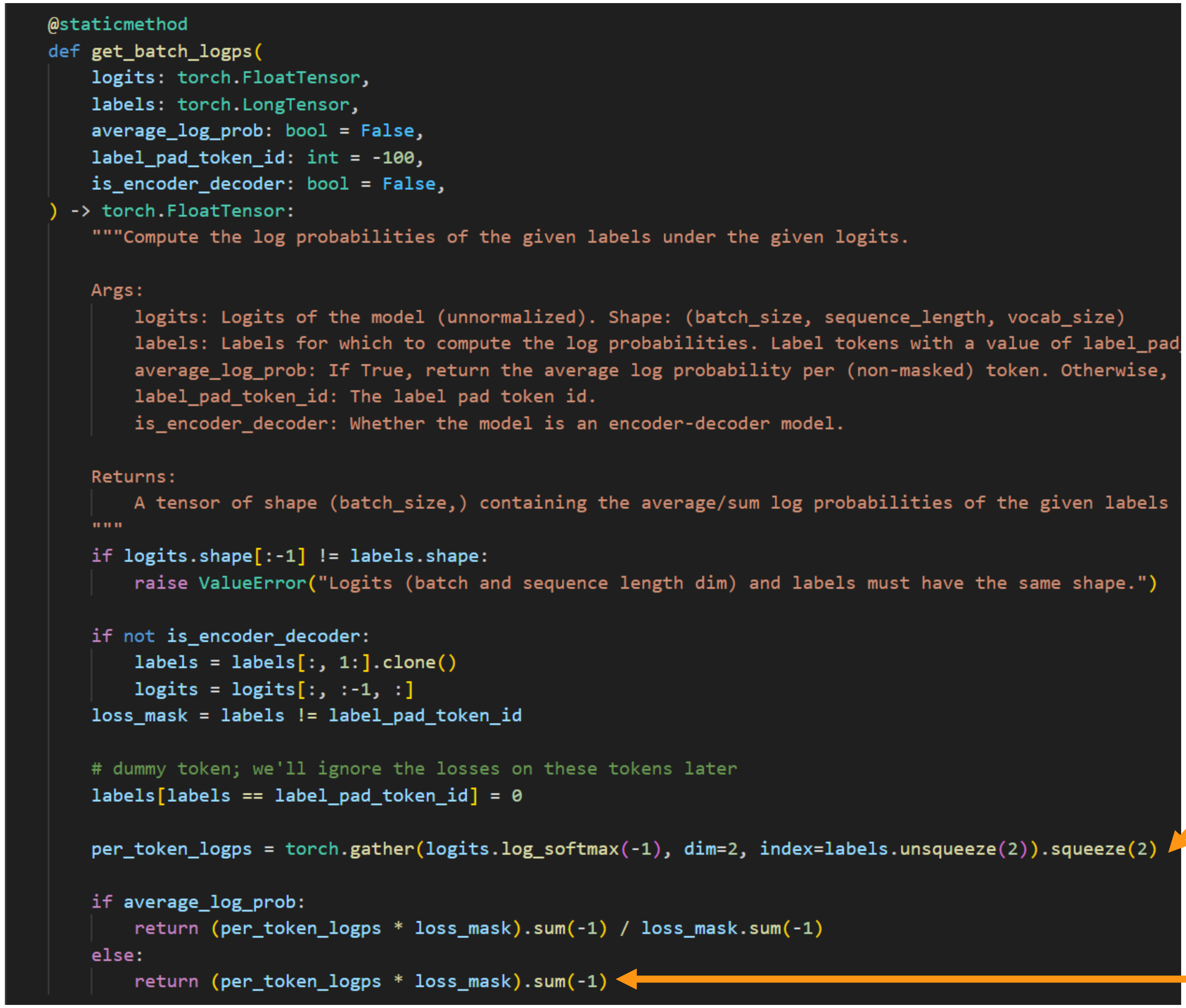
Select the log probably corresponding to the next token

Apply the $\log\_softmax$ function to generate log probabilities. $\log\_softmax(x) = \log(softmax(x))$

Apply the Linear projection to **all** the hidden states to generate logits for all the positions

Each hidden state encoder information only about previous tokens because we apply a **causal mask during self-attention**!

Calculating log probabilities



```
@staticmethod
def get_batch_logps(
    logits: torch.FloatTensor,
    labels: torch.LongTensor,
    average_log_prob: bool = False,
    label_pad_token_id: int = -100,
    is_encoder_decoder: bool = False,
) -> torch.FloatTensor:
    """Compute the log probabilities of the given labels under the given logits.

    Args:
        logits: Logits of the model (unnormalized). Shape: (batch_size, sequence_length, vocab_size)
        labels: Labels for which to compute the log probabilities. Label tokens with a value of label_pad
        average_log_prob: If True, return the average log probability per (non-masked) token. Otherwise,
        label_pad_token_id: The label pad token id.
        is_encoder_decoder: Whether the model is an encoder-decoder model.

    Returns:
        A tensor of shape (batch_size,) containing the average/sum log probabilities of the given labels
    """
    if logits.shape[:-1] != labels.shape:
        raise ValueError("Logits (batch and sequence length dim) and labels must have the same shape.")

    if not is_encoder_decoder:
        labels = labels[:, 1:].clone()
        logits = logits[:, :-1, :]
    loss_mask = labels != label_pad_token_id

    # dummy token; we'll ignore the losses on these tokens later
    labels[labels == label_pad_token_id] = 0

    per_token_logps = torch.gather(logits.log_softmax(-1), dim=2, index=labels.unsqueeze(2)).squeeze(2)

    if average_log_prob:
        return (per_token_logps * loss_mask).sum(-1) / loss_mask.sum(-1)
    else:
        return (per_token_logps * loss_mask).sum(-1)
```

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

Select the log probability corresponding to the next token

Sum the log probabilities

Thanks for watching!
Don't forget to subscribe for more amazing content on AI and Machine Learning!