Alignment of Language Models – Reward Maximization - II

Large Language Models: Introduction and Recent Advances

ELL881 · AIL821

Gaurav Pandey
Research Scientist, IBM Research

Regularized reward maximization

- Maximize the reward

$$\mathbb{E}_{y \sim \pi_\theta(y|x)} \, r(x,y) \quad \uparrow$$

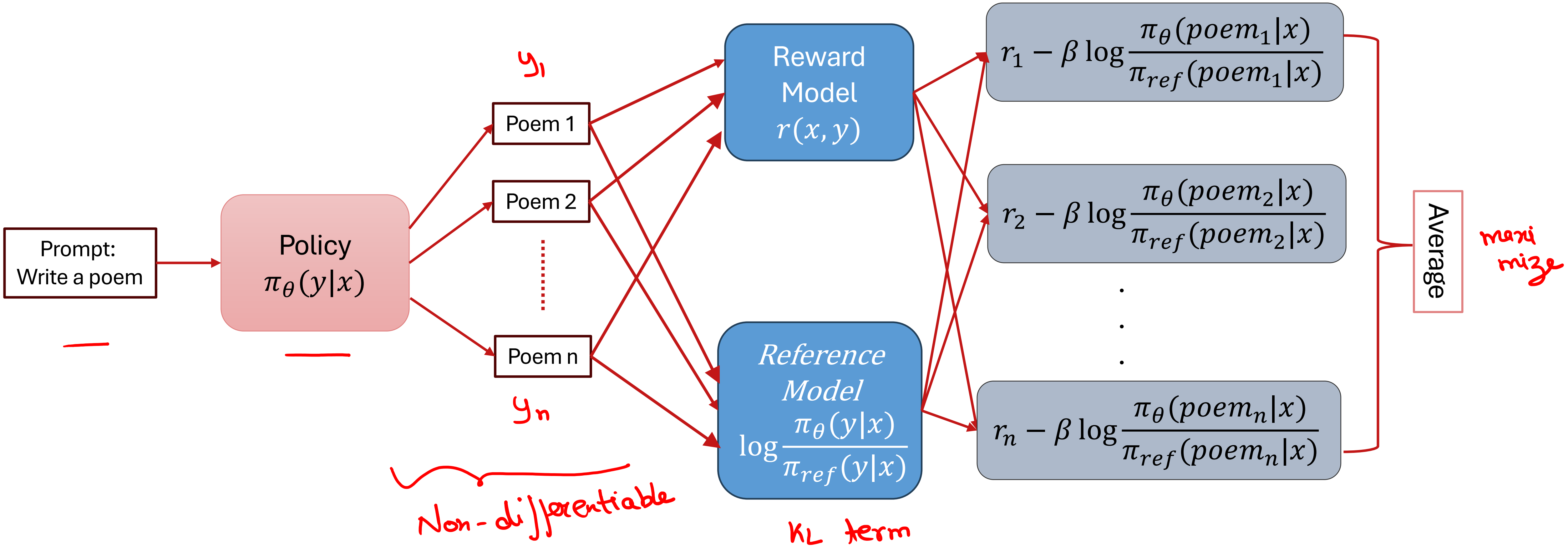
- Minimize the KL divergence

$$KL\left(\pi_\theta(y|x) \,||\, \pi_{ref}(y|x)\right) = \mathbb{E}_{\pi_\theta(y|x)}\left[\log \frac{\pi_\theta(y|x)}{\pi_{ref}(y|x)}\right] \quad \downarrow$$

- Add a scaling factor β & combine

$$\mathbb{E}_{y \sim \pi_\theta(y|x)}\left[r(x,y) - \beta \log \frac{\pi_\theta(y|x)}{\pi_{ref}(y|x)}\right]$$

The regularized reward maximization objective



Regularized reward

$$E_{\pi_\theta(y|x)}\left[r(x,y) - \beta \log \frac{\pi_\theta(y|x)}{\pi_{ref}(y|x)}\right] \equiv E_{\pi_\theta(y|x)} r_s(x,y)$$

$$where \;\; r_s(x,y) = r(x,y) - \beta \log \frac{\pi_\theta(y|x)}{\pi_{ref}(y|x)}$$

- $r_s(x,y)$ is the regularized reward
- Maximizing the regularized reward ensures
  - High reward outputs as decided by the reward model
  - Outputs that have reasonable probability under the reference model

How to maximize – The REINFORCE algorithm?

- Compute the gradient of the objective.
- Train using Adam/Adagrad optimization algorithms

$$\nabla_\theta E_{\pi_\theta(y|x)} r_s(x, y) = \nabla_\theta \sum_{y \in Y} \pi_\theta(y|x) \underbrace{r_s(x,y)}_{\text{fixed}}$$

$$= \sum_{y \in Y} \nabla_\theta \pi_\theta(y|x) \, r_s(x,y)$$

Computing the derivative

$$\sum_{y \in Y} \underbrace{\nabla_\theta \pi_\theta(y|x)} \underbrace{r_s(x,y)} \Bigg\} \quad \text{using samples}$$

- Exact computation of the gradient is intractable
  - Output space is too large
- Can we approximate it using samples?
- To be able to do that, we need an expression of the form

$$\underline{E_{\pi_\theta(y|x)}[\dots]} = \underline{\sum_{y \in Y} \pi_\theta(y|x)\,[\dots]}$$

- How to transform the derivative to this desired form?

The log-derivative trick

$$\nabla_\theta \log \pi_\theta(y|x) = \frac{1}{\pi_\theta(y|x)} \underbrace{\nabla_\theta \pi_\theta(y|x)} \iff \underbrace{\nabla_\theta \pi_\theta(y|x)} = \pi_\theta(y|x)\, \nabla_\theta \log \pi_\theta(y|x) \checkmark$$

Replacing it in the derivative, we get

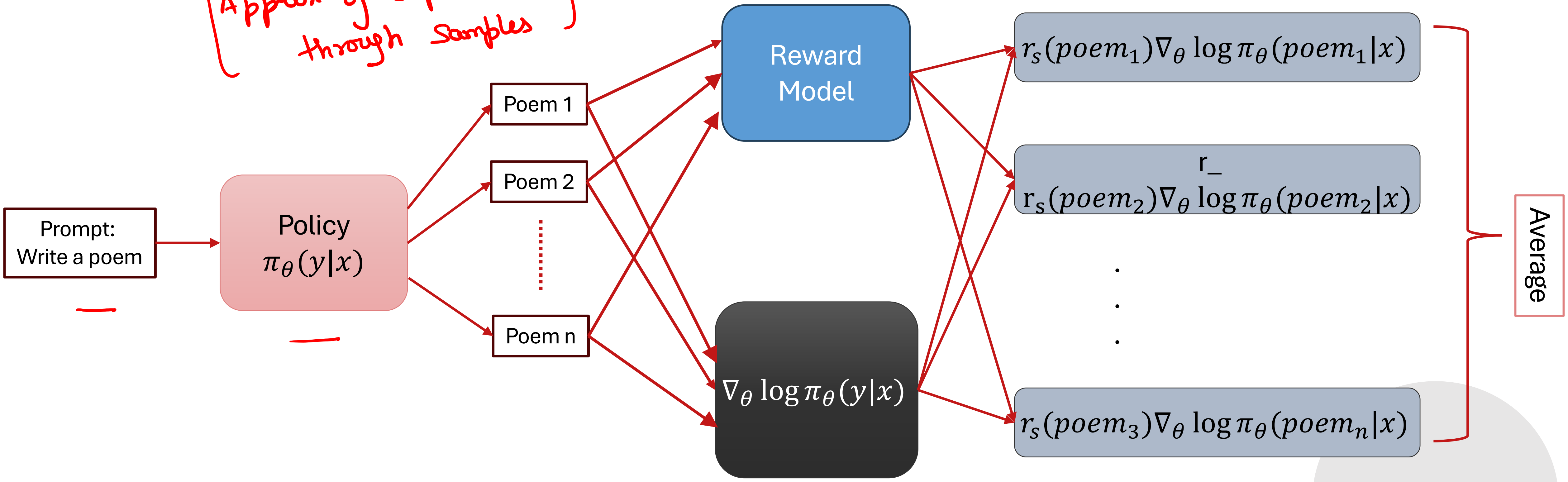
$$\sum_{y \in Y} \underbrace{\nabla_\theta \pi_\theta(y|x)}\, r_s(x,y)$$

$$= \sum_{y \in Y} \left[\pi_\theta(y|x)\, \nabla_\theta \log \pi_\theta(y|x)\right] r_s(x,y)$$

$$= \mathbb{E}_{y \sim \underline{\pi_\theta(y|x)}} \left[r_s(x,y)\, \nabla_\theta \log \pi_\theta(y|x) \right]$$

Monte Carlo approximation

[Approx of expectation through samples]



Expanding the gradient

- Let $y = (a_1, \ldots, a_{\cancel{m}\,T})$ be the tokens of y.
- $r_s(x, y)\nabla_\theta \log \pi_\theta(y|x) =$ $\quad r_s(x,y)\,\nabla_\theta \sum_{t=1}^{T} \log \pi_\theta(a_t | s_t)$ $\qquad s_t = (x, a_0, \ldots, a_{t-1})$

$$= r_s(x,y) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t | s_t)$$

$$= \sum_{t=1}^{T} r_s(x,y)\, \nabla_\theta \log \pi_\theta(a_t | s_t)$$

Implementing REINFORCE

$r = r_s(x, y)$
is the total reward of
the complete output

Sum & backpropagate

$\text{r} \log \pi_\theta(eot|s_5)$

$\text{r} \log \pi_\theta(Mahal|s_1)$

$\text{r} \log \pi_\theta(in|s_3)$

Replace with an avg

$\text{r} \log \pi_\theta(Taj|s_0)$

$\text{r} \log \pi_\theta(is|s_2)$

$\text{r} \log \pi_\theta(Agra|s_4)$

LLM Policy $\pi_\theta(a|s_t)$

$\underbrace{\text{User: Where is Taj Mahal? Assistant:}}_{x}$ $\underbrace{\text{Taj Mahal is in Agra}}_{y\text{ - generated}}$

Problems with REINFORCE

- The reward at token “Taj” depends on the tokens generated in the future
- If the model had generated “Taj Mahal is in Paris”
  - The reward would be negative
  - The probability of generating Taj would be decreased
- If the model had generated “Taj Mahal is in Agra”
  - The reward would be positive
  - The probability of generating Taj would be increased
- This variance in the reward leads to unstable training.
- To reduce variance – take the average reward over all likely sequences (under the policy) that generate “Taj” for the first token.
- This is called the $Q - function$

REINFORCE with Q functions

Sum & backpropagate

$r = r(x, y)$
is the reward of
the final output

average reward over all seq likely that start with Taj $=$ $\mathrm{Q}(\mathrm{s}_0, \mathrm{Taj}) \log \pi_\theta(Taj|s_0)$

$\mathrm{Q}(\mathrm{s}_1, \mathrm{Mahal}) \log \pi_\theta(Mahal|s_1)$

$\mathrm{Q}(\mathrm{s}_2, \mathrm{is}) \log \pi_\theta(is|s_2)$

$\mathrm{Q}(\mathrm{s}_3, \mathrm{in}) \log \pi_\theta(in|s_3)$

$\mathrm{Q}(\mathrm{s}_4, \mathrm{Agra}) \log \pi_\theta(Agra|s_4)$

$\mathrm{Q}(\mathrm{s}_5, \mathrm{eot}) \log \pi_\theta(eot|s_5)$

LLM Policy $\pi_\theta(a|s_t)$

User: Where is Taj Mahal? Assistant:	Taj	Mahal	is	in	Agra

Doesn't matter what gets generated in the future. The "reward" at token "Taj" is fixed.

Q-function & Value function

- The Q-function for a state-action pair is the average discounted cumulative reward received at the state after taking taking the specified action.

$s_{t+1} = (s_t, a_t)$

$$Q(s_t, a_t) = \mathbb{E}_{\pi_\theta(a_{t+1}, a_{t+2}, \ldots a_{t+T} \mid s_t)} \left[r(s_t, a_t) + \underbrace{\gamma}_{\text{discount factor.}} r(s_{t+1}, a_{t+1}) + \gamma^2 \ldots \right]$$

- The discount factor γ ensures that immediate rewards get higher weight.
- The Value function of a state is the average discounted cumulative reward received after reaching the state.

$$V(s_t) = \mathbb{E}_{\pi_\theta(a_t, a_{t+1} \ldots a_{t+T} \mid s_t)} \left[r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 \ldots \right]$$

From Q-function to Advantage function

- For text generation using language models

$$s_{t+1} = (s_t, a_t)$$

- That is, once you have generated the next token, the next state is determined completely.
- Hence, the Q-function for a state-action pair can be written as

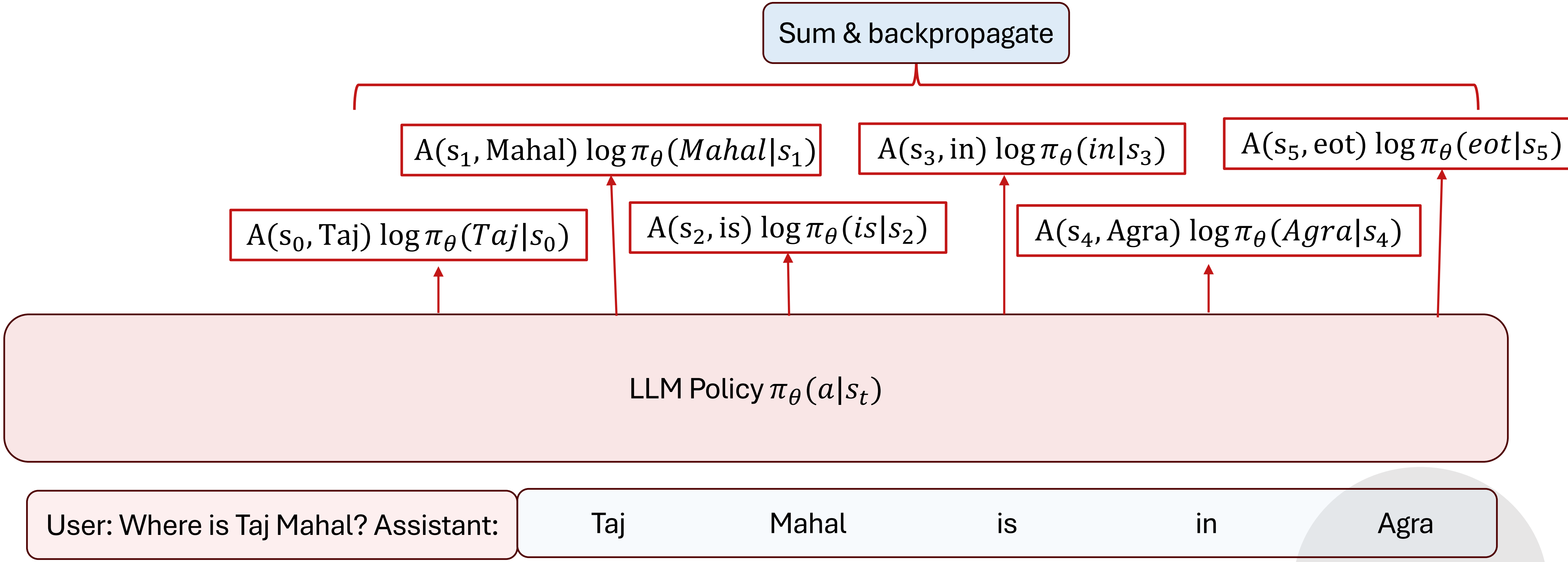
$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \underbrace{V(s_{t+1})}_{} \text{ next state}$$

- To further reduce variance, the advantage function $A(s_t, a_t)$ is used instead of Q-function

$$\underbrace{A(s_t, a_t)}_{} = Q(s_t, a_t) - V(s_t) \} \rightarrow \text{ average over all actions at } s_t$$
$$= r(s_t, a_t) + \gamma V(s_{t+1}) - V(s_t)$$

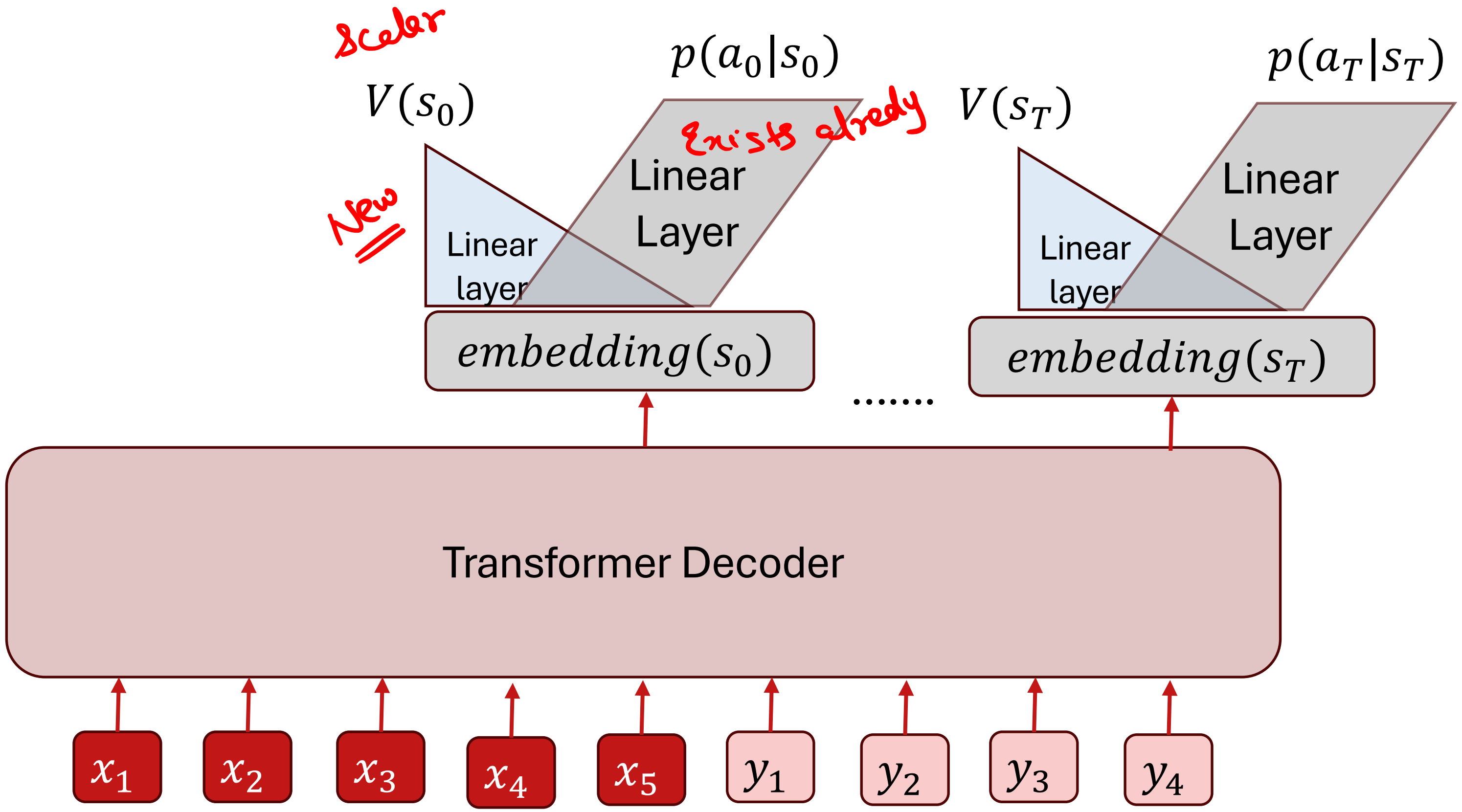
- Intuitively, advantage function captures contribution of the action a_t over an average action at the same state.

REINFORCE with advantage functions



Doesn't matter what gets generated in the future. The "reward" at token "Taj" is fixed.

Implementing the Value function



Learning the Value function

- Given an input x, sample $y = (a_0, \dots, a_T)$ from the policy $\pi_\theta(y|x)$
- Compute the cumulative discounted reward for each time-step

$$R_t = \underbrace{r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 r(s_{t+2}, a_{t+2})}_{\text{Reward-to-go}} + \cdots$$

- Minimize the mean-squared error

$$\min_{\phi} \sum_{t=0}^{T} \left(V_\phi(s_t) - R_t\right)^2$$

Vanilla Policy Gradient

- Repeat until convergence
  - Sample a batch of prompts B
  - For each prompt, sample one-or more outputs
  - For each output $y = (a_1, \ldots, a_T)$
    - Compute the reward r_t at each token a_t
    - Compute cumulative discounted reward R_t for each token
    - Compute the value & advantage function A_t for each token
  - Apply few gradient updates using REINFORCE with the advantage values computed above
  - Apply few gradient updates to train the value function by minimizing the MSE.

Credit: https://spinningup.openai.com/en/latest/algorithms/ppo.html

Problems

- Sampling from the policy after every update can be challenging.
- Solution: Sample from an older fixed policy instead

$$\mathbb{E}_{\pi_\theta(y|x)} r_s(x,y) = \sum_{y \in Y} \pi_\theta(y|x)\, r_s(x,y) \times \left(\frac{\pi_{\theta_R}(y|x)}{\pi_{\theta_R}(y|x)} \right)$$

$$= \sum \pi_{\theta_R}(y|x) \left[\frac{\pi_\theta(y|x)}{\pi_{\theta_R}(y|x)} \right] r_s(x,y)$$

$$= \mathbb{E}_{\pi_{\theta_R}(y|x)} \underbrace{\left[\frac{\pi_\theta(y|x)}{\pi_{\theta_R}(y|x)} \right]}_{} r_s(x,y) \longrightarrow \text{importance weight}$$

REINFORCE with importance weights

$$\nabla_\theta \mathbb{E}_{y \sim \pi_{\theta_k}(y|x)} \left[\frac{\pi_\theta(y|x)}{\pi_{\theta_k}(y|x)} \right] r_s(x, y)$$

(Apply log-derivative trick)

$$= \underbrace{\mathbb{E}_{y \sim \pi_{\theta_k}(y|x)} \left[\frac{\pi_\theta(y|x)}{\pi_{\theta_k}(y|x)} \right] r_s(x, y) \nabla_\theta \log \pi_\theta(y|x)}$$

REINFORCE with importance weights

The term in the square brackets is kept constant during gradient update.
In Pytorch, this means using .detach() function

Sum & backpropagate

$$\left[\frac{\pi_\theta(a|s_0)}{\pi_{\theta_k}(a|s_0)} A(s_0, a_0)\right] \log \pi_\theta(a|s_0) \quad \ldots\ldots\ldots \quad \left[\frac{\pi_\theta(a|s_T)}{\pi_{\theta_k}(a|s_T)} A(s_T, a_T)\right] \log \pi_\theta(a|s_T)$$

LLM Policy $\pi_{\theta_k}(a|s_t)$ (Fixed) ✓

LLM Policy $\pi_\theta(a|s_t)$ (Trainable) ✓

User: Where is Taj Mahal? Assistant:	Taj	Mahal	is	in	Agra

User: Where is Taj Mahal? Assistant:	Taj	Mahal	is	in	Agra

Proximal Policy Optimization

- Keeping the batch of prompts & outputs fixed, how much can we update the policy?
- If we update too much, the importance weights can change drastically.
- PPO-CLIP

$$(1-\epsilon) \leq \left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}\right) \leq (1+\epsilon)$$

- This ensures that the no matter how many updates are done to π_{θ}, it stays close to π_{θ_k}

PPO-CLIP

$$(1-\epsilon) \leq \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} \leq (1+\epsilon)$$

To achieve above, maximize the following

- When advantage is positive

$$\max_\theta \left[\min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}, (1+\epsilon)\right) A(s_t, a_t) \right]$$

- When advantage is negative

$$\max_\theta \left[\max\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}, (1-\epsilon)\right) A(s_t, a_t) \right]$$

Credit: https://spinningup.openai.com/en/latest/algorithms/ppo.html

PPO-CLIP with +ve advantage

$$\max_{\theta} \left[\min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}, (1+\epsilon)\right) A(s_t, a_t) \right]$$

Initially $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)} = 1 < 1+\epsilon$

$\max_\theta \underbrace{\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_k}(a_t|s_t)}}_{> 1+\epsilon} A(s_t, a_t)$

$\min = (1+\epsilon) A(s_t, a_t)$

PPO-CLIP with -ve advantage

$$\max_{\theta} \max\left(\frac{\pi_{\theta}(a_t|s_t)}{\pi_{\theta_{\cancel{t}\,k}}(a_t|s_t)}, (1-\epsilon)\right) A\,(s_t, a_t)$$

The PPO-CLIP algorithm

- Repeat until convergence
  - Sample a batch of prompts B
  - For each prompt, sample one-or more outputs
  - For each output $y = (a_1, \dots, a_T)$
    - Compute the reward r_t at each token a_t
    - Compute cumulative discounted reward R_t for each token
    - Compute the value & advantage function A_t for each token
  - Apply few gradient updates using ~~REINFORCE~~ PPO-CLIP with the advantage values computed above
  - Apply few gradient updates to train the value function by minimizing the MSE.

Credit: https://spinningup.openai.com/en/latest/algorithms/ppo.html

Things to remember

- The log-derivative trick should be used to compute gradient in REINFORCE
- The log-probability of the tokens should be weighed by the advantage function to reduce variance
- Importance weights should be used to allow sampling from a fixed policy
- The importance weights should be clipped to prevent large gradient updates.