RL(HF)

Slide acknowledgment: Umar Jamil, https://github.com/hkproj/rlhf-ppo/

Outline

- Why use RL for Language Models
- Reinforcement Learning
  - RL setting
  - Connection between RL and Language Models
  - Reward Model
  - Trajectories
  - Policy Gradient Optimization
  - Reducing Variance
  - Advantage Estimation
  - Importance Sampling
  - Off-Policy Learning
- Proximal Policy Optimization (PPO)
  - Loss function
  - KL regularization

Remarks

- Attending Full RL course advised
- We will distill big field into short excursion
- Wonderful theory and intricate algorithms - we scratch the surface

RL for LLMs

- LLMs are pretrained on large corpora of text
  - Capabilities are determined by pre-training
  - But not useful for dialogue systems
- Supervised Fine-Tuning (SFT) on dialogue data
  - Aligns LLMs to output answers to questions and hold dialogue
  - Collecting suitable dialogue data is challenging
  - Further alignment might be desirable
- RL for alignment
  - Rank responses of LLM
  - Encourage better ones

Technical Differences

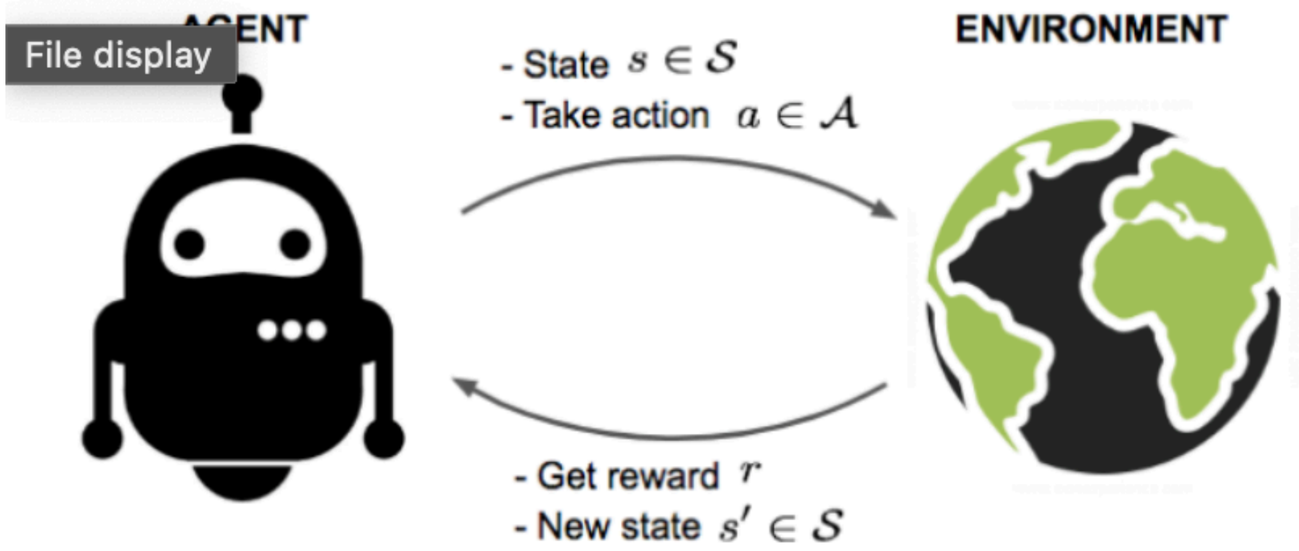
- Pretraining and SFT:
  - Cross-entropy loss on each token
  - Dense supervision
  - Easy training
- RL
  - Reward/Penalty on full response
  - Sparse supervision
  - Difficult training

RL for LLM Alignment

- Uses of RL
  - Safety: No racism/sexism/ offensiveness/right-wing
  - Style: Adopt particular idiom
  - Reasoning:
    - Encourage reasoning traces on difficult problems
    - Employ verificators on final solution as reward
    - R1 from DeepSeek

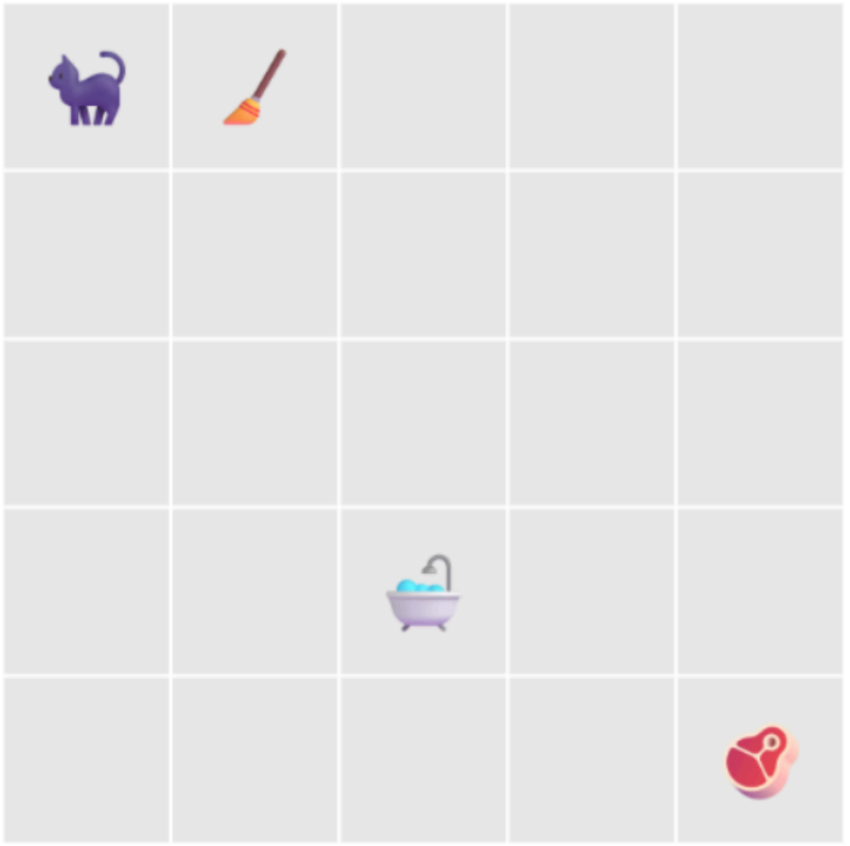
RL Intro

- Environment:
  - Determined by (partially) observable state $s \in \mathcal{S}$
  - Transits from one state to the other via actions $a \in \mathcal{A}$
  - Emits rewards after each step $r \in \mathbb{R}$
    - Transitions and rewards may be probabilistic
- Agent:
  - Observes state $s \in \mathcal{S}$
  - Takes an action $a \in \mathcal{A}$
  - Goes through variable length trajectory $\tau = (s_1, a_1, r_1, \ldots, s_n, a_n, r_n)$
- Objective:
  - Maximize return $R(\tau) = \sum_{i=1}^{n} \gamma^i r_i$ over trajectory
    - Discount factor $\gamma \in [0,1]$ determines how much we take into account future rewards



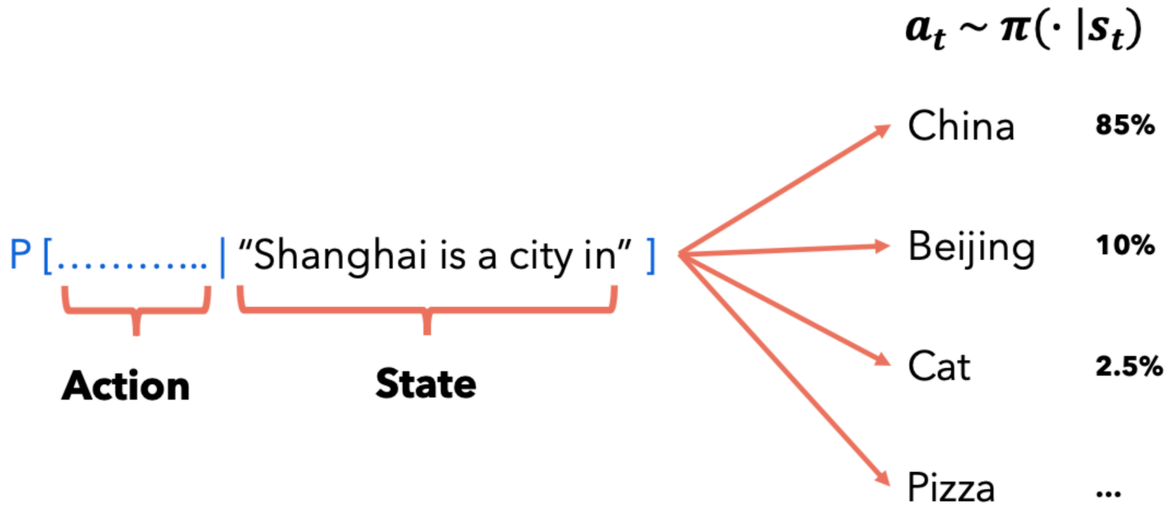
RL Toy Example

- Grid world environment:
  - State $(x, y) \in \{0, \ldots, k-1\}^2$
  - Actions: Move left/right/up/down
  - Reward:
    - 0 for move to empty cell
    - -1 for move to broom
    - -10 for move to bathtub
    - +100 for move to meat
- Agent:
  - Follows policy $a_t \sim \pi(\cdot \mid s_t)$
- Goal:
  - Find policy that collect maximum expected return



RL Setting for LLMs

- Agent: The LLM itself
- State: Previous input tokens - prompt and response until now
- Actions: Which token to select next
- Policy: $a_t \sim \pi(\cdot \mid s_t)$ logit distribution of last output head
- Reward: high reward for good responses



Reward for LLMs

- Difficulty: Assigning exact reward to text is hard
  - How good exactly is a response?

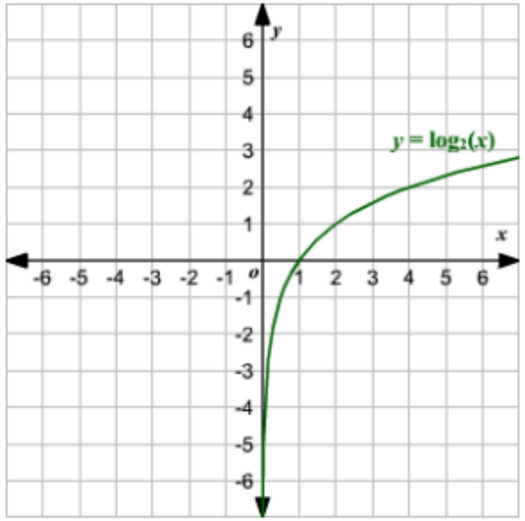
Question (Prompt)	Answer (Text generated by the language model)	Reward (0.0 ~ 1.0)
Where is Shanghai?	Shanghai is a city in China	???
Explain gravity like I'm 5	Gravity is what pulls things toward each other. It's why you stay on the ground and planets orbit the sun.	???
What is 2+2?	4	???

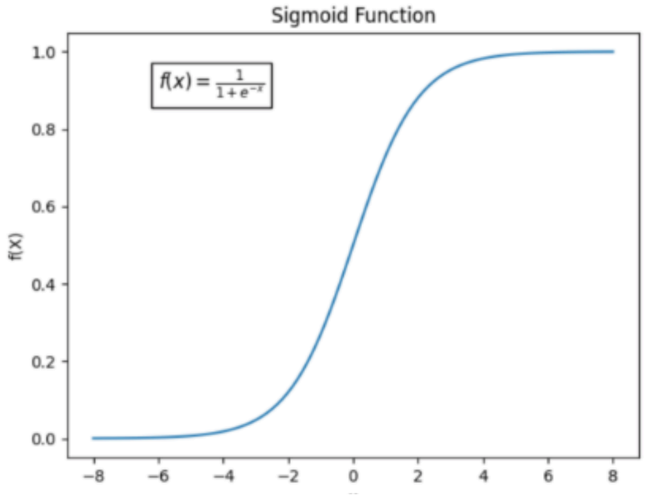
- But comparing two sentences and deciding which one is better is often easy.

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

Reward Loss for LLMs

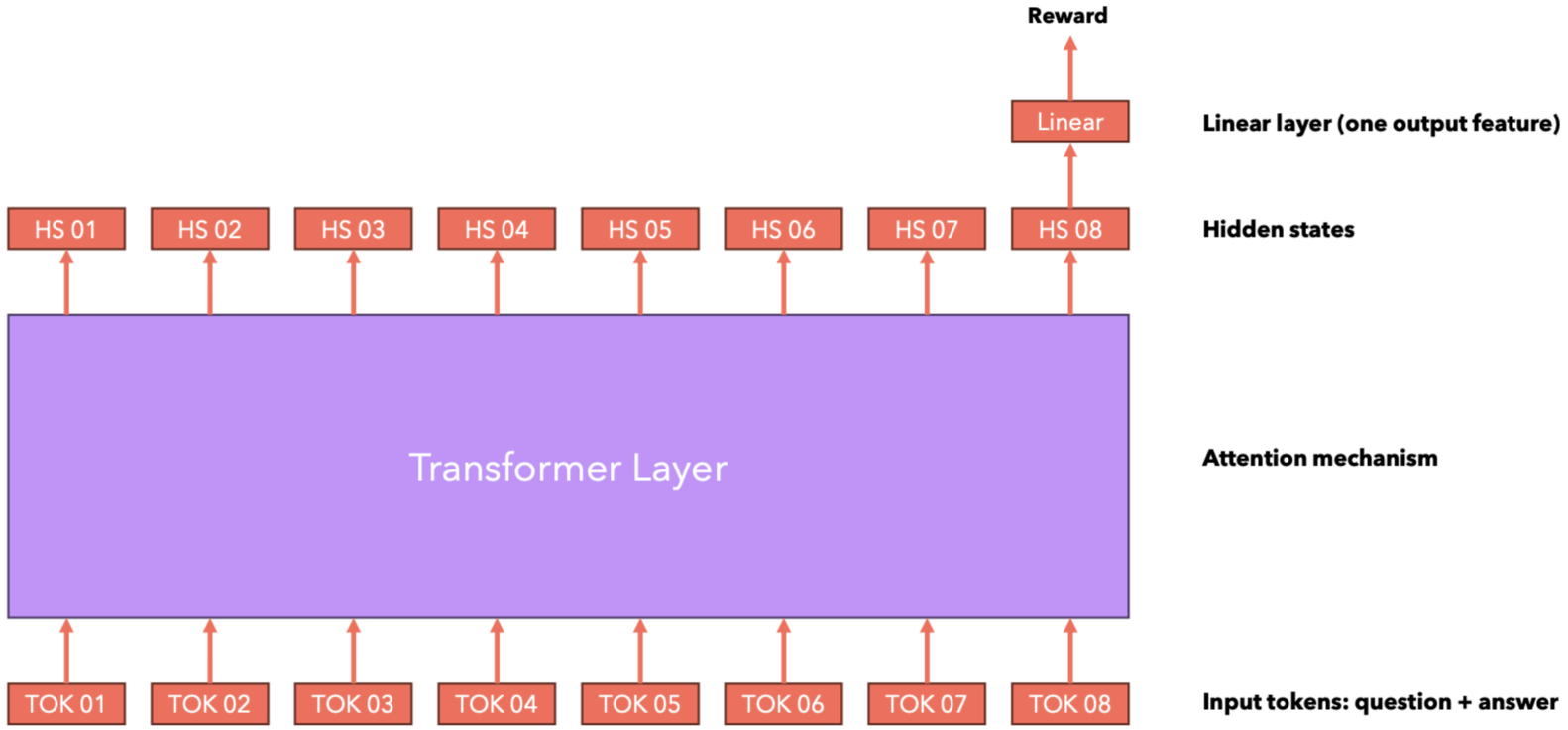
- Train reward model on pairs of responses
- Given bad/good pair of responses $(x, y_w)/(x, y_l)$ from human feedback:
  - $r(x, y_w) > r(x, y_l) \Leftrightarrow$ reward good response > reward bad response
- Loss: $-\log \sigma(r(x, y_w) - r(x, y_l))$
  - $r(x, y_w) > r(x, y_l) \rightarrow$ Sigmoid will return value $> 0.5 \rightarrow$ Loss small negative number
  - $r(x, y_w) < r(x, y_l) \rightarrow$ Sigmoid will return value $< 0.5 \rightarrow$ Loss large





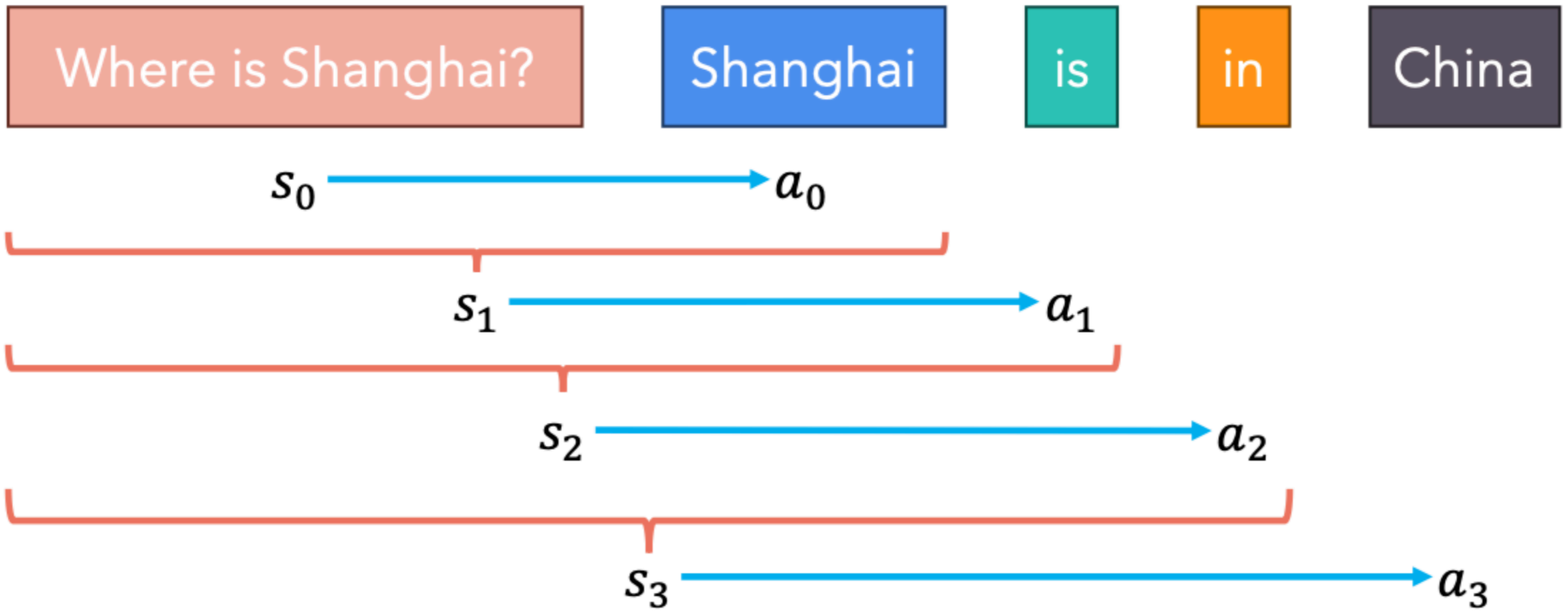
Reward Model for LLMs

- Train LLM to judge how good a response is
  - Human labelled comparison dataset
  - Reward computed on latent features of last layer of last response token



Trajectory for LLMs

- Trajectory is series of states, actions and rewards
  - State: All text until now
  - Action: Next token to produce
  - Reward: Given by reward LLM after all of text is produced



Loss on Trajectories

- Goal: Maximize expected return of trajectories generated with policy

$$\pi^* = \arg\max_\pi J(\pi), \quad J(\pi) = \sum_\tau P(\tau \mid \pi) R(\tau) = \mathbb{E}_{\tau \sim \pi}[R(\tau)]$$

- Probability of trajectory:

$$P(\tau \mid \pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1} \mid s_t, a_t) \pi(a_t \mid s_t)$$

  - Note: Probability is Markovian, i.e. next state depends only on previous one
  - Therefore state for LLM must contain all previously generated text

Policy Gradient Optimization

- Parametrize $\pi_\theta \leftarrow$ Weights of LLM
- We want to do gradient-based optimization $\theta_{t+1} \leftarrow \theta_t + \alpha \nabla_\theta J(\pi_\theta)\,|_{\theta_k}$
- How to get the policy gradient?
  - $J(\pi)$ is a sum over all trajectories $\rightarrow$ sample trajectories
  - But trajectories are generated based on current policy
    $J(\pi) = \mathbb{E}_{\tau\sim\pi}[R(\tau)] \rightarrow$ resample trajectories regularly after policy changes

Policy Gradient Optimization II

- Expected Policy Return is involved function of probability of full trajectory
  - Rewrite to get gradient on each distribution for each state/action pair of each trajectory

$$
\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \underset{\tau\sim\pi_\theta}{\mathrm{E}}[R(\tau)] \\
&= \nabla_\theta \int_\tau P(\tau|\theta)R(\tau) && \text{Expand expectation} \\
&= \int_\tau \nabla_\theta P(\tau|\theta)R(\tau) && \text{Bring gradient under integral} \\
&= \int_\tau P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)R(\tau) && \text{Log-derivative trick} \\
&= \underset{\tau\sim\pi_\theta}{\mathrm{E}}[\nabla_\theta \log P(\tau|\theta)R(\tau)] && \text{Return to expectation form} \\
\nabla_\theta J(\pi_\theta) &= \underset{\tau\sim\pi_\theta}{\mathrm{E}}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau)\right] && \text{Expression for grad-log-prob}
\end{aligned}
$$

$$
P(\tau|\pi) = \rho_0(s_0)\prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t).
$$

$$
\begin{aligned}
\nabla_\theta \log P(\tau|\theta) &= \cancel{\nabla_\theta \log \rho_0(s_0)} + \sum_{t=0}^{T}\left(\cancel{\nabla_\theta \log P(s_{t+1}|s_t, a_t)} + \nabla_\theta \log \pi_\theta(a_t|s_t)\right) \\
&= \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t).
\end{aligned}
$$

Recap: REINFORCE

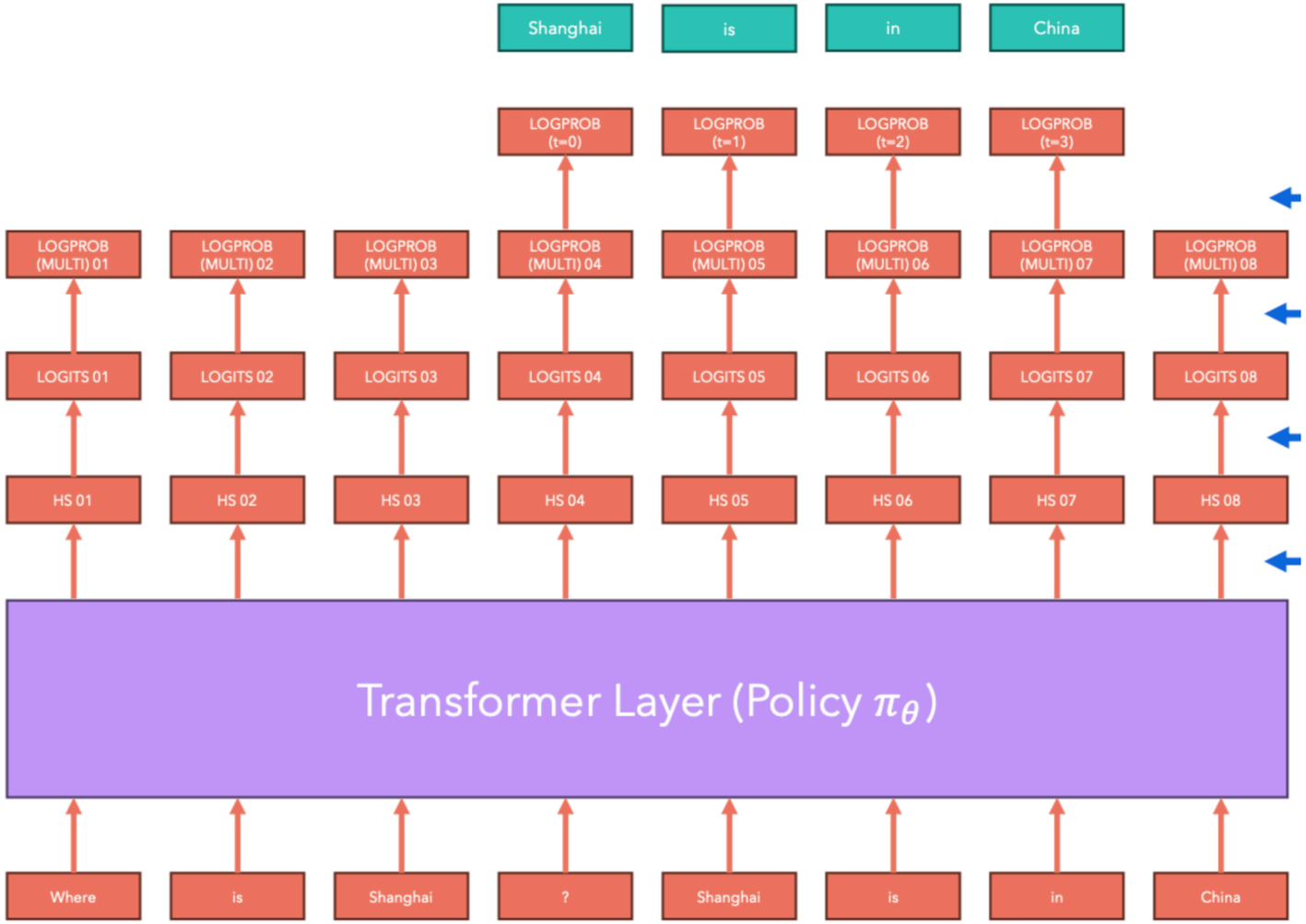
- We have derived the REINFORCE algorithm
    - The simplest policy gradient based algorithm

REINFORCE:

1. Initialize neural network that parametrizes the agent
2. Sample a number of trajectories/rewards with current network
3. Use samples to calculate approximation of policy gradient
4. Run stochastic gradient ascent to update the parameters of the policy/network
5. Go back to 2.

- Trajectories: Generate text with LLM.
    - Prompt is a set of questions/user requests
- Rewards: Given by trained reward model
    - Reward model trained initially / but data can be collected in intervals and annotated by humans and reward model can be retrained

Calculating Log-Probabilities in LLMs



Select the log-prob of the token taken by the policy

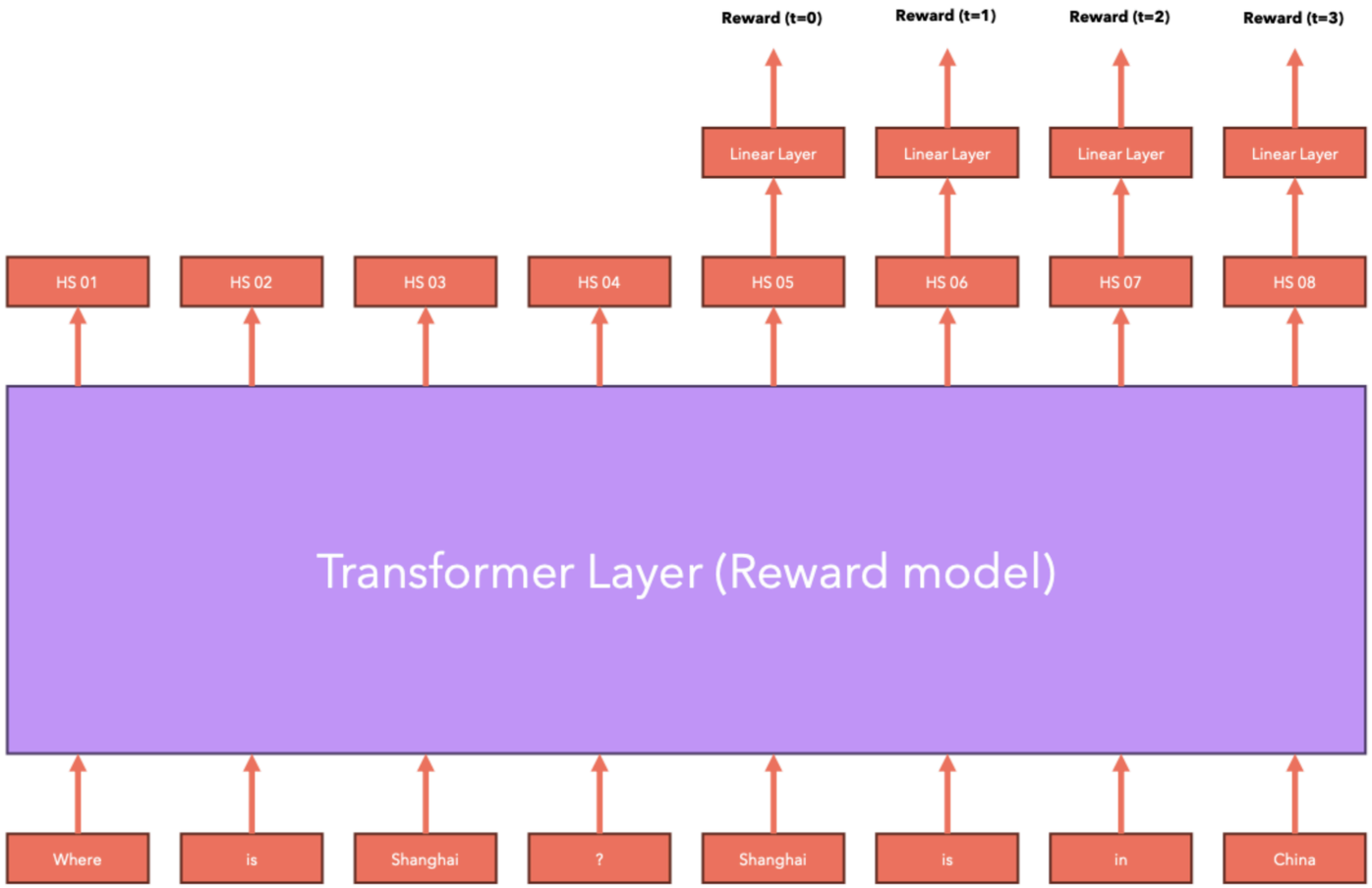
Apply $\log(\text{softmax}(x))$

Generate logits with the output head

Can take hidden states of all tokens due to causal masking

Calculating Rewards

- Reward LLM is Transformer again



Reward is calculated from last sequence token's hidden state with linear layer

REINFORCE Gradient: Reducing Variance

- Gradient has high variance

$$\hat{\nabla}_\theta J(\theta) = \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) R(\tau)$$

- Replace return with reward to go:

$$\hat{\nabla}_\theta J(\theta) = \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=1}^{T} \left(\nabla_\theta \log \pi_\theta(a_t \mid s_t) (\sum_{t'=t}^{T} r(s_{t'}, a_{t'})) \right)$$

  - Past terms vanish in expectation → still an unbiased estimator

- Baseline:

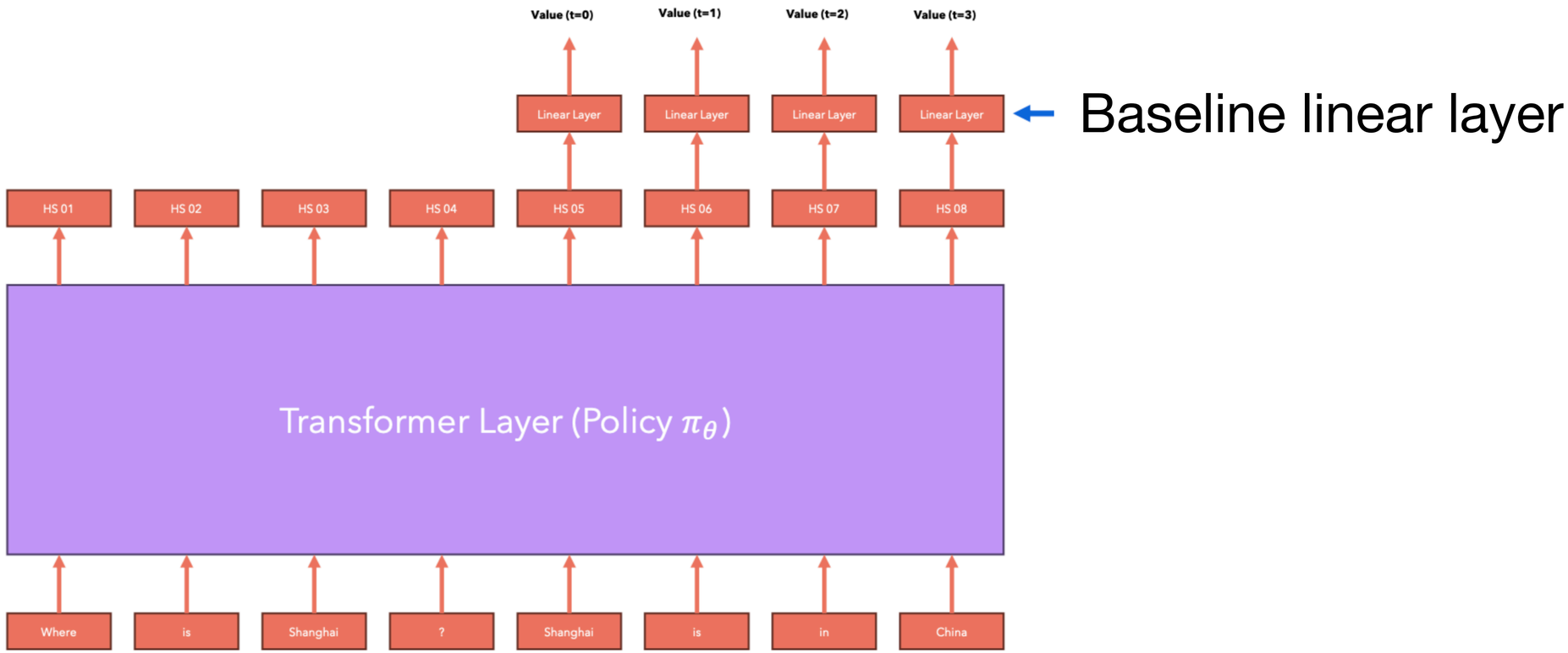
$$\hat{\nabla}_\theta J(\theta) = \frac{1}{|D|} \sum_{\tau \in D} \sum_{t=1}^{T} \left(\nabla_\theta \log \pi_\theta(a_t \mid s_t) (\sum_{t'=t}^{T} r(s_{t'}, a_{t'}) - b(s_t)) \right)$$

  - Baseline term dependent upon state only still leads to an unbiased estimator

$$\nabla_\theta J(\theta) = \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a_t \mid s_t) (\sum_{t'=t}^{T} r_t - b(s_t))] = \mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a_t \mid s_t) (\sum_{t'=t}^{T} r_t)] - \underbrace{\mathbb{E}_\pi[\nabla_\theta \log \pi_\theta(a_t \mid s_t) b(s_t)])}_{=0}$$

Baseline Estimator $V^{\pi}(s)$

- Value function: Another additional linear layer on top of LLM
  - Use it for baseline: $b(s) = V^{\pi}(s)$
  - Train to estimate $V^{\pi}(s) = \mathbb{E}_{\tau:s_0=s}[R(\tau)]$



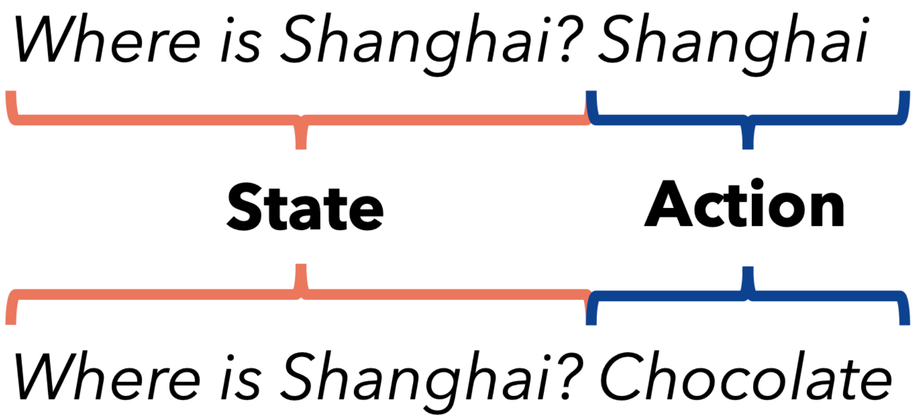
Advantage Function $A(s_t, a_t)$

- Advantage Function: $A^{\pi}(s, t) = \mathbb{E}_{\tau:s_0=s,a_0=a}[\tau] - V^{\pi}(s)$
- Interpretation:
  - How much better is it to take action a in state s over the average action of policy π
  - When used in policy gradient: Increase/decrease logits of actions with positive/negative advantage
- Recursively rewriting Advantage Function:
  - $\hat{A}^{\pi}(s_t, a_t) = [r(s_t, a_t) + \gamma V^{\pi}(s_{t+1})] - V^{\pi}(s_t)$
  - $\hat{A}^{\pi}(s_t, a_t) = [r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 V^{\pi}(s_{t+2})] - V^{\pi}(s_t)$
  - $\hat{A}^{\pi}(s_t, a_t) = [r(s_t, a_t) + \gamma r(s_{t+1}, a_{t+1}) + \gamma^2 r(s_{t+2}, a_{t+2}) + \gamma^3 V^{\pi}(s_{t+3})] - V^{\pi}(s_t)$
  - ...
- Generalized Advantage Estimation:
  - $\delta_t = r_t + \gamma V^{\pi}(s_{t+1}) - V^{\pi}(s_t)$
  - $\hat{A} = \delta_t + \gamma\lambda\hat{A}_{t+1}$ ← recursively update, hyperparameter λ

Advantage Function $A(s_t, a_t)$

- Policy Gradient Update with Advantage Function:

$$\nabla_\theta J(\theta) \approx \frac{1}{D} \sum_{i=1}^{D} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) A^\pi(s_t, a_t)$$



- Action „Shanghai”: Positive advantage, policy gradient will increase logins
- Action „Chocolate”: Negative advantage, policy gradient will decrease logins

(Off-Policy) Sampling Trajectories

- We need to sample trajectories $\tau \sim \pi$ to estimate $\nabla_\theta J(\theta)$ and make a gradient step
  - On-Policy: Sampling from current model before every update step
  - Inefficient: We do not reuse sampled trajectories
- Off-Policy: Use Importance Sampling to use stale trajectories and correct for error

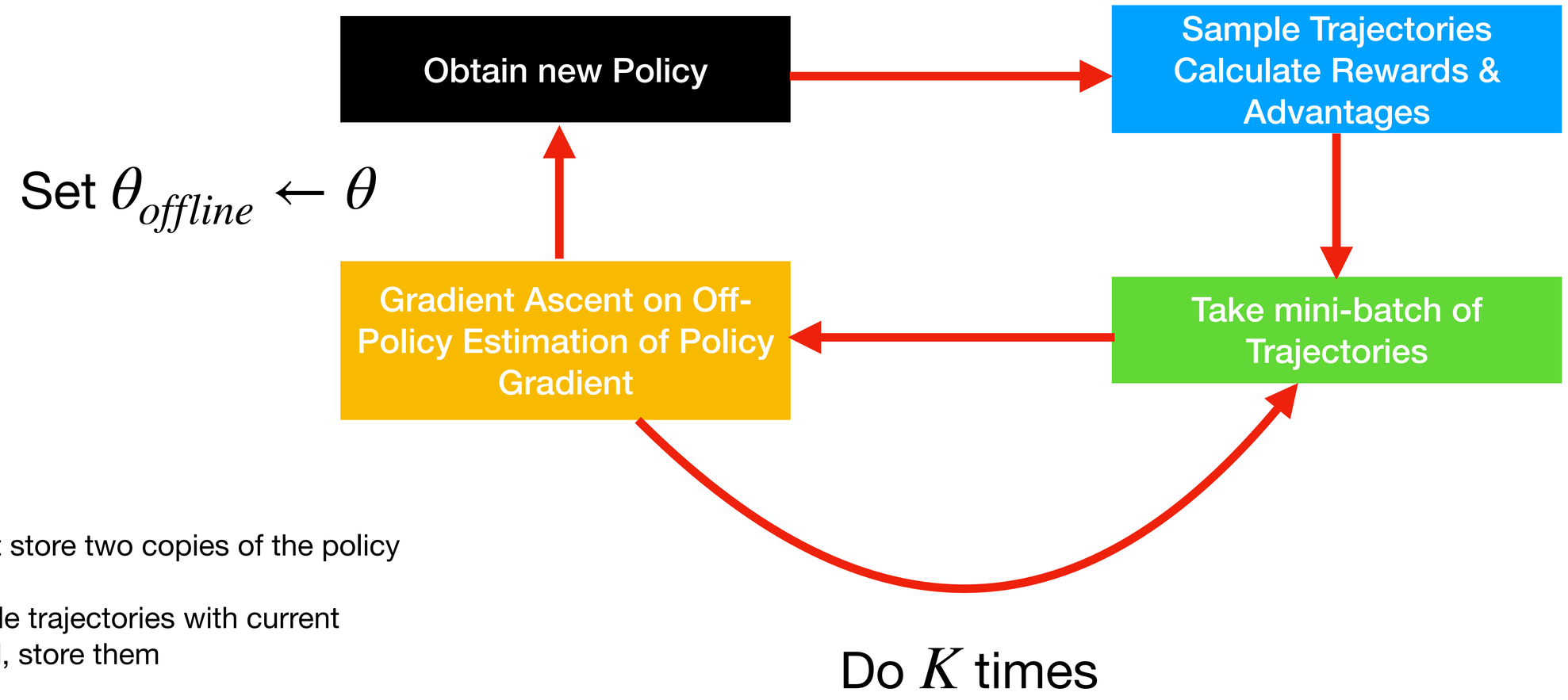
$$\mathbb{E}_{x\sim p(x)} \int p(x)f(x)dx = \int \frac{q(x)}{p(x)} p(x)f(x)dx = \int q(x)\frac{p(x)}{q(x)} f(x)dx = \mathbb{E}_{x\sim q(x)} \left[\frac{p(x)}{q(x)} f(x)\right]$$

- Use in estimating $\nabla_\theta J(\theta)$:

$$\underbrace{\frac{1}{D}\sum_{i=1}^{D}\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t \mid s_t) A^\pi(s_t, a_t)}_{\text{Sampling from } \pi_\theta} \rightarrow \underbrace{\frac{1}{D}\sum_{i=1}^{D}\sum_{t=0}^{T} \nabla_\theta \frac{\log \pi_\theta(a_t \mid s_t)}{\log \pi_{\theta_{offline}}(a_t \mid s_t)} A^\pi(s_t, a_t)}_{\text{Sampling from } \pi_{\theta_{offline}}}$$

  - In expectation correct, but has higher variance than on-policy sampling

Off-Policy Learning



- We do not store two copies of the policy
  - Sample trajectories with current model, store them
  - Then update the model

Proximal Policy Optimization Loss

- PPO = REINFORCE + Offline Sampling + Clipped Policy Loss + Regularization
  - Clipped Loss:

$$L_{policy} = \min\left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{offline}}(a_t \mid s_t)}\hat{A}_t, \text{clip}\left(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{offline}}(a_t \mid s_t)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right]$$

  - Value Function Loss: $L_{VF} = \frac{1}{2}\left|\left|V_\theta(s) - \left(\sum_{t=0}^{T}\gamma^t r_t \mid s_0 = s\right)\right|\right|_2^2$
  - Entropy Loss: $L_{entropy} = -\sum_x p(x)\log p(x)$
  - Overall: $L_{PPO} = L_{policy} + c_1 L_{VF} + c_2 L_{entropy}$

PPO Clipping Loss: Intuition

- Clipped Loss: $L_{policy} = \min \left[\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{offline}}(a_t \mid s_t)} \hat{A}_t, \text{clip} \left(\frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{offline}}(a_t \mid s_t)}, 1-\epsilon, 1+\epsilon \right) \hat{A}_t \right]$
  - Assume sample (s, a) with positive advantage $\hat{A}(s, t) > 0$ (analogous for negative advantage)
  - Objective will increase if action become $\pi_\theta(s, a)$ more likely
  - If $\pi_\theta(a \mid s) > (1+\epsilon)\pi_\theta(a \mid s)$ clipping becomes active
  - Objective reduces to $\min \left[\frac{\pi_\theta(a \mid s)}{\pi_{\theta_{offline}}(a \mid s)}, 1+\epsilon \right] \hat{A}_\pi(a, s)$
  - Due to $\min$ operator, the entire objective is limited to $(1+\epsilon)\hat{A}_\pi(a, s)$
  - Objective does not increase if policy goes further away from old policy
    - Implicitly encourage current policy to not deviate from previous one
    - More stable

Regularization

- If we apply vanilla PPO the LLM will degrade and only output what the rewards wants to see
  - We want the LLM to retain its pretrained capabilities
  - Penalize deviation from logit distribution generated from frozen pretrained LLM