Q-SFT: Q-Learning as Supervised Fine-Tuning

12 August 2025

Motivation

- Value-based RL:

Motivation

- Value-based RL:
  - Compute the expected future reward of a state-action pair $Q(s, a)$, make decisions based on these values

Motivation

- Value-based RL:
  - Compute the expected future reward of a state-action pair $Q(s, a)$, make decisions based on these values
  - (Rather than learning a policy directly without a value function)

Motivation

- Value-based RL:
  - Compute the expected future reward of a state-action pair $Q(s, a)$, make decisions based on these values
  - (Rather than learning a policy directly without a value function)
  - Can be more sample-efficient than policy gradient methods

Motivation

- Value-based RL:
  - Compute the expected future reward of a state-action pair $Q(s, a)$, make decisions based on these values
  - (Rather than learning a policy directly without a value function)
  - Can be more sample-efficient than policy gradient methods
- But, applying value-based RL to language models is challenging:

Motivation

- Value-based RL:
  - Compute the expected future reward of a state-action pair $Q(s, a)$, make decisions based on these values
  - (Rather than learning a policy directly without a value function)
  - Can be more sample-efficient than policy gradient methods
- But, applying value-based RL to language models is challenging:
  - The action and state spaces are very large

Motivation

- Value-based RL:
  - Compute the expected future reward of a state-action pair $Q(s, a)$, make decisions based on these values
  - (Rather than learning a policy directly without a value function)
  - Can be more sample-efficient than policy gradient methods
- But, applying value-based RL to language models is challenging:
  - The action and state spaces are very large
  - The reward signal is sparse and delayed

Motivation

- Value-based RL:
  - Compute the expected future reward of a state-action pair $Q(s, a)$, make decisions based on these values
  - (Rather than learning a policy directly without a value function)
  - Can be more sample-efficient than policy gradient methods
- But, applying value-based RL to language models is challenging:
  - The action and state spaces are very large
  - The reward signal is sparse and delayed
  - Training is often unstable

Motivation

- Value-based RL:
  - Compute the expected future reward of a state-action pair $Q(s, a)$, make decisions based on these values
  - (Rather than learning a policy directly without a value function)
  - Can be more sample-efficient than policy gradient methods
- But, applying value-based RL to language models is challenging:
  - The action and state spaces are very large
  - The reward signal is sparse and delayed
  - Training is often unstable
- The idea: use the pretrained model's output probs/logits to compute Q-values ($Q^\pi(s, a) =$ expected future reward from action a in state s following the policy π).

Motivation

- Value-based RL:
  - Compute the expected future reward of a state-action pair $Q(s, a)$, make decisions based on these values
  - (Rather than learning a policy directly without a value function)
  - Can be more sample-efficient than policy gradient methods
- But, applying value-based RL to language models is challenging:
  - The action and state spaces are very large
  - The reward signal is sparse and delayed
  - Training is often unstable
- The idea: use the pretrained model's output probs/logits to compute Q-values ($Q^\pi(s, a) =$ expected future reward from action a in state s following the policy π).
  - Logits incorporate high-quality information from pretraining

Motivation

- Value-based RL:
  - Compute the expected future reward of a state-action pair $Q(s, a)$, make decisions based on these values
  - (Rather than learning a policy directly without a value function)
  - Can be more sample-efficient than policy gradient methods
- But, applying value-based RL to language models is challenging:
  - The action and state spaces are very large
  - The reward signal is sparse and delayed
  - Training is often unstable
- The idea: use the pretrained model's output probs/logits to compute Q-values ($Q^\pi(s, a) =$ expected future reward from action a in state s following the policy π).
  - Logits incorporate high-quality information from pretraining
  - Doesn't require a change in architecture or a separate value function network/head

Motivation

- Value-based RL:
  - Compute the expected future reward of a state-action pair $Q(s, a)$, make decisions based on these values
  - (Rather than learning a policy directly without a value function)
  - Can be more sample-efficient than policy gradient methods
- But, applying value-based RL to language models is challenging:
  - The action and state spaces are very large
  - The reward signal is sparse and delayed
  - Training is often unstable
- The idea: use the pretrained model's output probs/logits to compute Q-values ($Q^{\pi}(s, a) =$ expected future reward from action a in state s following the policy π).
  - Logits incorporate high-quality information from pretraining
  - Doesn't require a change in architecture or a separate value function network/head
  - Should provide a stable initialisation and training signal

Key RL Components

- **State (s)**: The current dialogue history/text sequence.

Key RL Components

- **State (s)**: The current dialogue history/text sequence.
- **Action (a)**: The next token to be generated.

Key RL Components

- **State (s)**: The current dialogue history/text sequence.
- **Action (a)**: The next token to be generated.
- **Reward (r)**: Usually sparse and delayed.

Key RL Components

- **State (s)**: The current dialogue history/text sequence.
- **Action (a)**: The next token to be generated.
- **Reward (r)**: Usually sparse and delayed.
- **Discount factor ($\gamma \in [0, 1]$)**: The discount factor for future rewards.

Key RL Components

- **State (s)**: The current dialogue history/text sequence.
- **Action (a)**: The next token to be generated.
- **Reward (r)**: Usually sparse and delayed.
- **Discount factor ($\gamma \in [0, 1]$)**: The discount factor for future rewards.
    - (Prioritizes immediate rewards over delayed ones.)

Key RL Components

- **State (s)**: The current dialogue history/text sequence.
- **Action (a)**: The next token to be generated.
- **Reward (r)**: Usually sparse and delayed.
- **Discount factor ($\gamma \in [0, 1]$)**: The discount factor for future rewards.
  - (Prioritizes immediate rewards over delayed ones.)
  - (We can set it to 1 and ignore it for single-turn interactions with rewards only on the final token.)

Key RL Components

- **State (s)**: The current dialogue history/text sequence.
- **Action (a)**: The next token to be generated.
- **Reward (r)**: Usually sparse and delayed.
- **Discount factor ($\gamma \in [0, 1]$)**: The discount factor for future rewards.
  - (Prioritizes immediate rewards over delayed ones.)
  - (We can set it to 1 and ignore it for single-turn interactions with rewards only on the final token.)
- **Policy (π)**: The policy we are trying to improve (distribution over next-tokens).

The Bellman Equation and Q-Learning

- **Q-function (Q^π)**: The discounted expected future reward from a state-action pair following policy π.

The Bellman Equation and Q-Learning

- **Q-function (Q^π)**: The discounted expected future reward from a state-action pair following policy π.
- Q^π satisfies the Bellman recurrence:

$$Q^\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a),\ a' \sim \pi(\cdot|s')} \left[Q^\pi(s',a') \right]$$

The Bellman Equation and Q-Learning

- **Q-function (Q^π)**: The discounted expected future reward from a state-action pair following policy π.
- Q^π satisfies the Bellman recurrence:

$$Q^\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a),\ a' \sim \pi(\cdot|s')} \left[Q^\pi(s',a') \right]$$

- The Q-function of the optimal policy, Q^*, maximizes the expected future reward:

$$Q^*(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a'} Q^*(s',a') \right]$$

The Bellman Equation and Q-Learning

- **Q-function (Q^π)**: The discounted expected future reward from a state-action pair following policy π.
- Q^π satisfies the Bellman recurrence:

$$Q^\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a),\ a' \sim \pi(\cdot|s')} \left[Q^\pi(s',a') \right]$$

- The Q-function of the optimal policy, Q^*, maximizes the expected future reward:

$$Q^*(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a'} Q^*(s',a') \right]$$

- The goal of RL is to find the optimal policy π^*:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{s \sim P(\cdot)} \left[\max_{a} Q^\pi(s,a) \right]$$

The Bellman Equation and Q-Learning

- **Q-function (Q^π)**: The discounted expected future reward from a state-action pair following policy π.
- Q^π satisfies the Bellman recurrence:

$$Q^\pi(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a),\ a' \sim \pi(\cdot|s')} \left[Q^\pi(s',a') \right]$$

- The Q-function of the optimal policy, Q^*, maximizes the expected future reward:

$$Q^*(s,a) = r(s,a) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a)} \left[\max_{a'} Q^*(s',a') \right]$$

- The goal of RL is to find the optimal policy π^*:

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{s \sim P(\cdot)} \left[\max_{a} Q^\pi(s,a) \right]$$

- **Summary: if we have Q^*, we can find π^* by taking the argmax over the Q-values.**

Q-SFT

Q-SFT: Q-Learning for Language Models via Supervised Fine-Tuning

Joey Hong **Anca Dragan** **Sergey Levine**
University of California, Berkeley
{jxihong,anca,svlevine}@berkeley.edu

The Core Idea: Q-Learning as SFT

- Reframe Q-learning as a supervised fine-tuning problem using an offline dataset $\mathcal{D} = \{(s_i, a_i, r_i)\}_{i=1}^{N}$.

The Core Idea: Q-Learning as SFT

- Reframe Q-learning as a supervised fine-tuning problem using an offline dataset $\mathcal{D} = \{(s_i, a_i, r_i)\}_{i=1}^{N}$.
- Finetune a base model to output probabilities $\hat{p}_\theta(a|s)$ that are estimates of the optimal Q-values, $Q^*(s, a)$.

The Core Idea: Q-Learning as SFT

- Reframe Q-learning as a supervised fine-tuning problem using an offline dataset $\mathcal{D} = \{(s_i, a_i, r_i)\}_{i=1}^{N}$.
- Finetune a base model to output probabilities $\hat{p}_\theta(a|s)$ that are estimates of the optimal Q-values, $Q^*(s, a)$.
- (At inference time, we'll have to modify $\hat{p}_\theta$ slightly to get a good (non-greedy) policy π_θ.)

A Note on the Dataset

- In text, the dataset $\mathcal{D} = \{(s_i, a_i, r_i)\}_{i=1}^{N}$ has actions that are tokens and states that are token sequences.

A Note on the Dataset

- In text, the dataset $\mathcal{D} = \{(s_i, a_i, r_i)\}_{i=1}^{N}$ has actions that are tokens and states that are token sequences.
- E.g. s_i might be the question and partial response, `"Q: If a right-angled triangle has side lengths 3 and 4, what is the length of the hypotenuse?  A: We can use"`.

A Note on the Dataset

- In text, the dataset $\mathcal{D} = \{(s_i, a_i, r_i)\}_{i=1}^{N}$ has actions that are tokens and states that are token sequences.
- E.g. s_i might be the question and partial response, `"Q: If a right-angled triangle has side lengths 3 and 4, what is the length of the hypotenuse?  A: We can use"`.
- And a_i might be the next token, `"Pythagoras"`.

A Note on the Dataset

- In text, the dataset $\mathcal{D} = \{(s_i, a_i, r_i)\}_{i=1}^N$ has actions that are tokens and states that are token sequences.
- E.g. s_i might be the question and partial response, `"Q: If a right-angled triangle has side lengths 3 and 4, what is the length of the hypotenuse? A: We can use"`.
- And a_i might be the next token, `"Pythagoras"`.
- Only the final token can have a non-zero reward, but any token can have positive values of the Q-function (`"Pythagoras"` is a good action, but `"Fermat"` is not).

A Note on the Dataset

- In text, the dataset $\mathcal{D} = \{(s_i, a_i, r_i)\}_{i=1}^{N}$ has actions that are tokens and states that are token sequences.
- E.g. s_i might be the question and partial response, `"Q: If a right-angled triangle has side lengths 3 and 4, what is the length of the hypotenuse?  A: We can use"`.
- And a_i might be the next token, `"Pythagoras"`.
- Only the final token can have a non-zero reward, but any token can have positive values of the Q-function (`"Pythagoras"` is a good action, but `"Fermat"` is not).
- We can extend this to multi-turn interactions by concatenating the previous responses to the question (in which case we may get intermediate rewards).

Behavioural Cloning

- We can assume our offline dataset $\mathcal{D} = \{(s_i, a_i, r_i)\}_{i=1}^{N}$ is generated by some behaviour policy π_b over tokens.

Behavioural Cloning

- We can assume our offline dataset $\mathcal{D} = \{(s_i, a_i, r_i)\}_{i=1}^{N}$ is generated by some behaviour policy π_b over tokens.
- We can train a policy π_ϕ to match the behaviour of the dataset by minimizing the cross-entropy loss:

$$L_{\mathrm{CE}}(\phi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\log \pi_\phi(a|s)\right]$$

Behavioural Cloning

- We can assume our offline dataset $\mathcal{D} = \{(s_i, a_i, r_i)\}_{i=1}^{N}$ is generated by some behaviour policy π_b over tokens.
- We can train a policy π_ϕ to match the behaviour of the dataset by minimizing the cross-entropy loss:
$$L_{\mathrm{CE}}(\phi) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[\log \pi_\phi(a|s)\right]$$
- This is just standard SFT, and let's us approximate $\pi_\phi \approx \pi_b$ as a reference policy later.

Weighted Cross-Entropy Loss

- We can modify the loss function to weight different tokens/actions with weights $w(s, a)$:

$$L_{\text{WCE}}(\theta) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[w(s, a) \log p_\theta(a|s) + (1 - w(a, s))\mathbb{E}_{a' \neq a}[\log p_\theta(a'|s)]\right]$$

Weighted Cross-Entropy Loss

- We can modify the loss function to weight different tokens/actions with weights $w(s, a)$:

$$L_{\text{WCE}}(\theta) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[w(s, a)\log p_\theta(a|s) + (1 - w(a, s))\mathbb{E}_{a'\neq a}[\log p_\theta(a'|s)]\right]$$

- Optimising this will give us $\hat{p}_\theta(s, a) \approx w(s, a)\pi_b(a|s)$.

Weighted Cross-Entropy Loss

- We can modify the loss function to weight different tokens/actions with weights $w(s,a)$:

$$L_{\mathrm{WCE}}(\theta) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[w(s,a)\log p_\theta(a|s) + (1 - w(a,s))\mathbb{E}_{a'\neq a}[\log p_\theta(a'|s)]\right]$$

- Optimising this will give us $\hat{p}_\theta(s,a) \approx w(s,a)\pi_b(a|s)$.
- The goal is to find weights $w(s,a)$ that make $\hat{p}_\theta(s,a) \approx Q^*(s,a)$.

Weighted Cross-Entropy Loss

- We can modify the loss function to weight different tokens/actions with weights $w(s, a)$:

$$L_{\mathrm{WCE}}(\theta) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[w(s, a)\log p_\theta(a|s) + (1 - w(a, s))\mathbb{E}_{a'\neq a}[\log p_\theta(a'|s)]\right]$$

- Optimising this will give us $\hat{p}_\theta(s, a) \approx w(s, a)\pi_b(a|s)$.
- The goal is to find weights $w(s, a)$ that make $\hat{p}_\theta(s, a) \approx Q^*(s, a)$.
- The authors use the *empirical Bellman probability operator*:

$$w(s, a) = \hat{\mathcal{B}}^* \bar{p}_\theta(a|s) = r + \gamma \max_{a'} \frac{\bar{p}_\theta(a'|s')}{\pi_b(a'|s')},$$

where r is the reward of moving from s to s' via action a, and $\bar{p}_\theta$ is a moving average of p_θ over training.

Weighted Cross-Entropy Loss

- We can modify the loss function to weight different tokens/actions with weights $w(s, a)$:

$$L_{\mathsf{WCE}}(\theta) = \mathbb{E}_{(s,a)\sim\mathcal{D}}\left[w(s,a)\log p_\theta(a|s) + (1 - w(a,s))\mathbb{E}_{a'\neq a}[\log p_\theta(a'|s)]\right]$$

- Optimising this will give us $\hat{p}_\theta(s, a) \approx w(s, a)\pi_b(a|s)$.
- The goal is to find weights $w(s, a)$ that make $\hat{p}_\theta(s, a) \approx Q^*(s, a)$.
- The authors use the *empirical Bellman probability operator*:

$$w(s,a) = \hat{\mathcal{B}}^*\bar{p}_\theta(a|s) = r + \gamma \max_{a'} \frac{\bar{p}_\theta(a'|s')}{\pi_b(a'|s')},$$

  where r is the reward of moving from s to s' via action a, and $\bar{p}_\theta$ is a moving average of p_θ over training.
- They prove that this leads to $\hat{p}_\theta$ being a good ("conservative") approximation of Q^*.

Q-SFT: Policy Extraction

We want to extract a policy $\hat{\pi}$ from the Q-SFT model.

- Greedy policy: $\hat{\pi}(a|s) = \mathbf{1}[a = \arg\max_a Q^*(s, a)]$.

Q-SFT: Policy Extraction

We want to extract a policy $\hat{\pi}$ from the Q-SFT model.

- Greedy policy: $\hat{\pi}(a|s) = \mathbf{1}[a = \arg\max_a Q^*(s, a)]$.
- Entropy-regularised policy: $\hat{\pi}(a|s) \propto \exp(Q^*(s, a))$.

Q-SFT: Policy Extraction

We want to extract a policy $\hat{\pi}$ from the Q-SFT model.

- Greedy policy: $\hat{\pi}(a|s) = \mathbf{1}[a = \arg\max_a Q^*(s, a)]$.
- Entropy-regularised policy: $\hat{\pi}(a|s) \propto \exp(Q^*(s, a))$.
- KL-regularised policy (suggested by the authors) with hyperparameter $\beta > 0$:

$$\begin{aligned}\hat{\pi}(a|s) &\propto \pi_b(a|s)\exp(\beta Q^*(s, a)) \\ &\approx \pi_\phi(a|s)\exp(\beta p_\theta(a|s))\end{aligned}$$

This is a well-known solution to the constrained optimisation problem:

$$\arg\max_\pi \mathbb{E}_{s\sim P(\cdot),\, a\sim\pi(\cdot|s)}[Q^*(s, a)] \quad s.t. \quad \mathbb{E}_{s\sim P(\cdot)}[D_{\mathrm{KL}}(\pi(\cdot|s) \,\|\, \pi_b(\cdot|s))] \le \epsilon$$

The Q-SFT Algorithm

Algorithm 1 Q-learning via Supervised Fine-tuning (Q-SFT)

Require: Dataset $\mathcal{D} = \{(s_i, a_i, r_i, s_i')\}_{i \in [N]}$, hyperparameter $\beta > 0$

Initialize $\phi, \theta, \bar{\theta}$ from pretrained model.
Optimize behavior policy:
for each gradient step **do**
    Update $\phi \leftarrow \phi - \lambda_\phi \nabla_\phi \mathcal{L}_{\mathrm{CE}}(\phi)$
end for
Optimize likelihood model:
for each gradient step **do**
    Update $\theta \leftarrow \theta - \lambda_\theta \nabla_\theta \mathcal{L}_{\mathrm{WCE}}(\theta)$
    Update target parameters: $\bar{\theta} \leftarrow (1 - \alpha)\bar{\theta} + \alpha\theta$
end for
At inference time, policy probabilites become: $\widehat{\pi}(a \mid s) \propto \pi_\phi(a \mid s) \exp\left(\beta\, p_\theta(a \mid s)\right)$

Experiment Settings

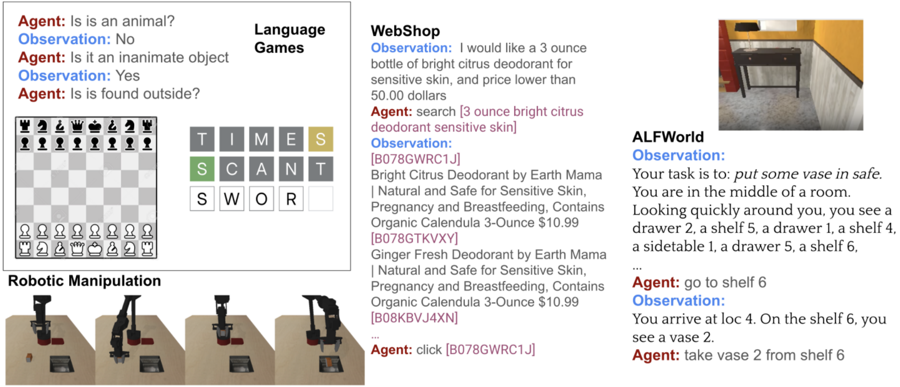

Figure 2: Overview of all the evaluated tasks, spanning both text and image inputs. Solving all the tasks effectively requires our algorithm to be able to be used to fine-tune LLMs, VLMs, and even robotics transformer models.

Experimental Results

	language games			alfworld					
Method	Chess	Wordle	20Q	Pick	Examine	Clean	Heat	Cool	Pick2
ReAct	0	−4.96	−13.2	**45**	19	17	7	12	**24**
SFT	0.11	−3.81	−17.3	38	15	0	11	0	18
ILQL	0.09	−2.08	−14.2	28	7	0	5	2	15
Q-SFT (ours)	**0.15**	**−2.11**	**−13.1**	39	**21**	**19**	**14**	**18**	21

Table 1: Average scores (for language games), and success rates (for ALFWorld tasks) across 100 independent evaluations. Our method performs best or near-best across the table, and competitively with prompting a much more complex model.

- ReAct: CoT/prompt-based reasoning
- SFT: just using π_ϕ
- ILQL (Implicit Language Q-Learning): train an additional transformer to predict the Q-values directly.

Experimental Results

Method	Score
ReAct	0.60
SFT	0.55
Offline ArCHer	0.57
Q-SFT	**0.63**

Table 2: Average score across 100 held-out instructions in WebShop. Our method performs best, even against prompting a much larger model.

Method	Pick Object	Place Object Near Target
BC	44	32
CQL	78	57
QT	92	**68**
Q-SFT	**94**	64

Table 3: Success rate for 100 runs across robotic manipulation tasks. Our general method performs competitively with Q-transformer, a value-based RL method specifically designed for continuous control.

- ReAct: CoT/prompt-based reasoning
- SFT/BC (behavioural cloning): just using π_ϕ
- Offline ArCHer: hierarchical value modelling at multi-turn-level & token-level (seems complicated).
- CQL (Conservative Q-Learning) & QT (Q-Transformer): train Q-value networks.

Experimental Results

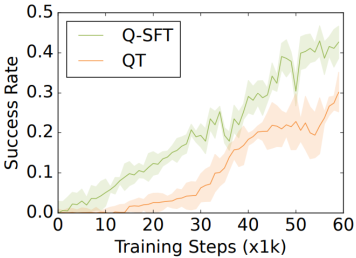

Figure 3: Success rate during initial training on the pick object task of the robotic manipulation benchmark. Though our method achieves similar final performance as Q-transformer, we perform much better on fewer samples.

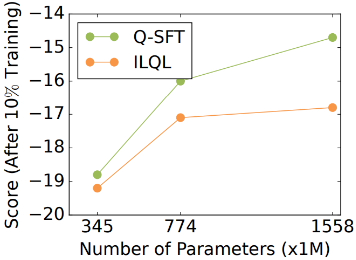

Figure 4: Scores after training on 10% of the offline dataset on the 20Q task, varying the size of the pretrained model. Our method benefits more from using more sophisticated pretrained models, suggesting our approach scales better.

- (Left) Q-SFT scales better than QT with fewer samples.
- (Right) Q-SFT scales better than ILQL with more parameters. ("We only train on 10% of the dataset, so that retaining prior knowledge from pretraining becomes crucial"...)

Summary of Contributions

- Reframes Q-learning as a supervised fine-tuning problem using a weighted cross-entropy loss (good for stability and simplicity).
- An effective way to leverage pretrained models without adding new layers or heads (this seems worthwhile).
- But requires training two models, π_ϕ and $\hat{p}_\theta$, plus the moving average $\bar{p}_\theta$.
- Some of the choices seem a bit arbitrary (e.g. the moving average to help with stability) and/or not well-motivated/explained.
- No comparison to online methods?