Reinforcement Learning in Enormous Action Spaces

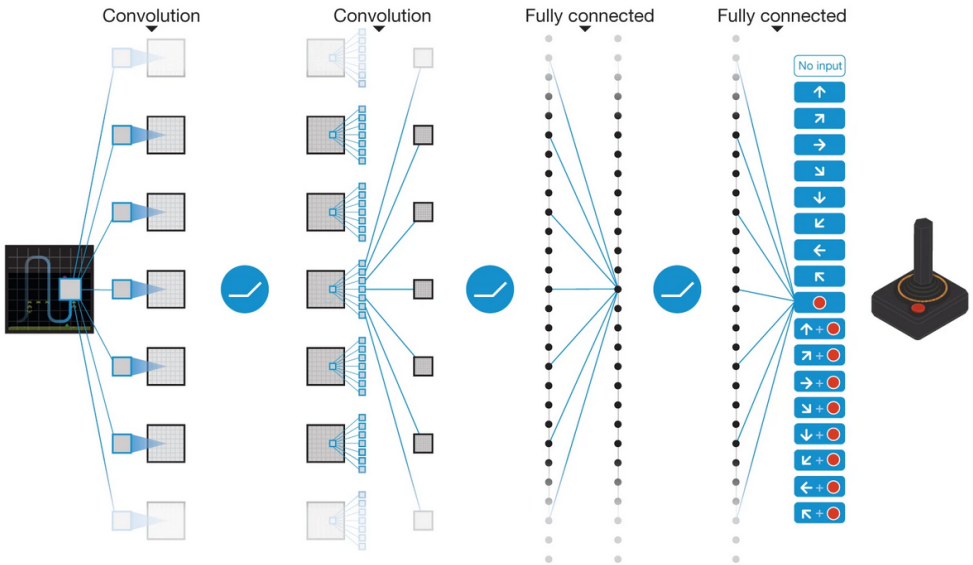
Varshant Dhar, Jack Weitze

March 30, 2021

Outline

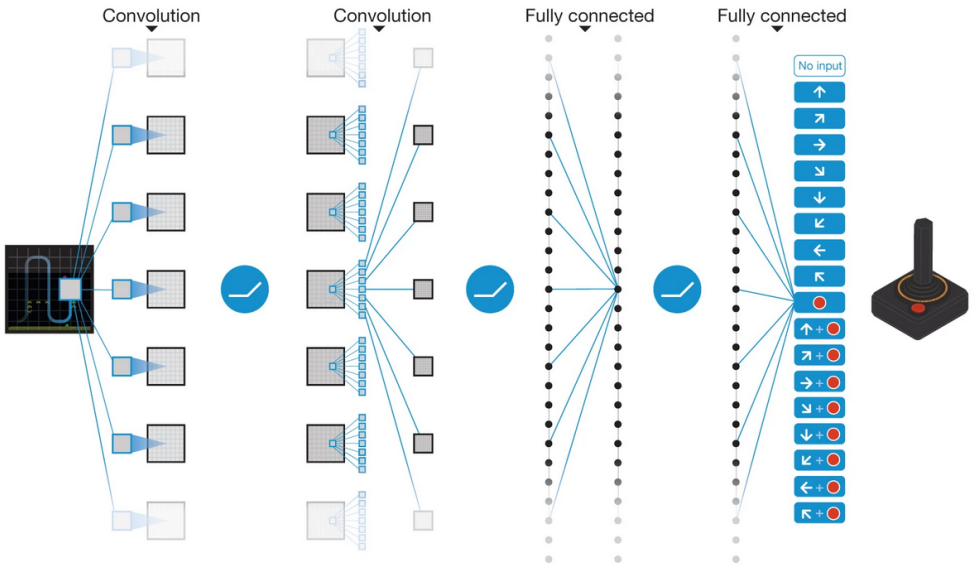
Background: Atari and DQNs

Deep Q-Learning has been used to achieve human-level control in Atari games.



Background: Atari and DQNs

Deep Q-Learning has been used to achieve human-level control in Atari games.



- Atari games have at most 18 actions.

The large action space problem

Many applications have large actions spaces:

- Continuous control for robotics
- Web-scale recommendation systems

The large action space problem

Many applications have large actions spaces:

- Continuous control for robotics
- Web-scale recommendation systems

Traditional Q-learning framework cannot scale to large action spaces.

The large action space problem

Many applications have large actions spaces:

- Continuous control for robotics
- Web-scale recommendation systems

Traditional Q-learning framework cannot scale to large action spaces.

- Training inefficiency in exploration of large action spaces.

The large action space problem

Many applications have large actions spaces:

- Continuous control for robotics
- Web-scale recommendation systems

Traditional Q-learning framework cannot scale to large action spaces.

- Training inefficiency in exploration of large action spaces.
- Maximization of action-value over all possible actions.

The large action space problem

Many applications have large actions spaces:

- Continuous control for robotics
- Web-scale recommendation systems

Traditional Q-learning framework cannot scale to large action spaces.

- Training inefficiency in exploration of large action spaces.
- Maximization of action-value over all possible actions.

Challenge: scale with action space size jAj

Why discrete action spaces?

- Reinforcement Learning in continuous domains often resort to parametric functions for a compact representation of distributions over actions. Often these are Gaussian.

Why discrete action spaces?

- Reinforcement Learning in continuous domains often resort to parametric functions for a compact representation of distributions over actions. Often these are Gaussian.

- A discrete policy, in practice, can represent much more flexible distributions than Gaussian when there are sufficient number of atomic actions. Intuitively, a discrete policy can represent a multi-modal action distribution while a Gaussian is by design uni-modal.

Why discrete action spaces?

- Reinforcement Learning in continuous domains often resort to parametric functions for a compact representation of distributions over actions. Often these are Gaussian.

- A discrete policy, in practice, can represent much more flexible distributions than Gaussian when there are sufficient number of atomic actions. Intuitively, a discrete policy can represent a multi-modal action distribution while a Gaussian is by design uni-modal.

- A common argument against a discretized action space is that for an action space with M dimensions, discretizing K atomic actions per dimension leads to M^K combinations of joint atomic actions: *Curse of Dimensionality*.

Hierarchical Reinforcement Learning

- Hierarchical RL (HRL) works on decomposing the RL problem into sub-problems where solving each of these is more powerful than solving the entire problem.

Hierarchical Reinforcement Learning

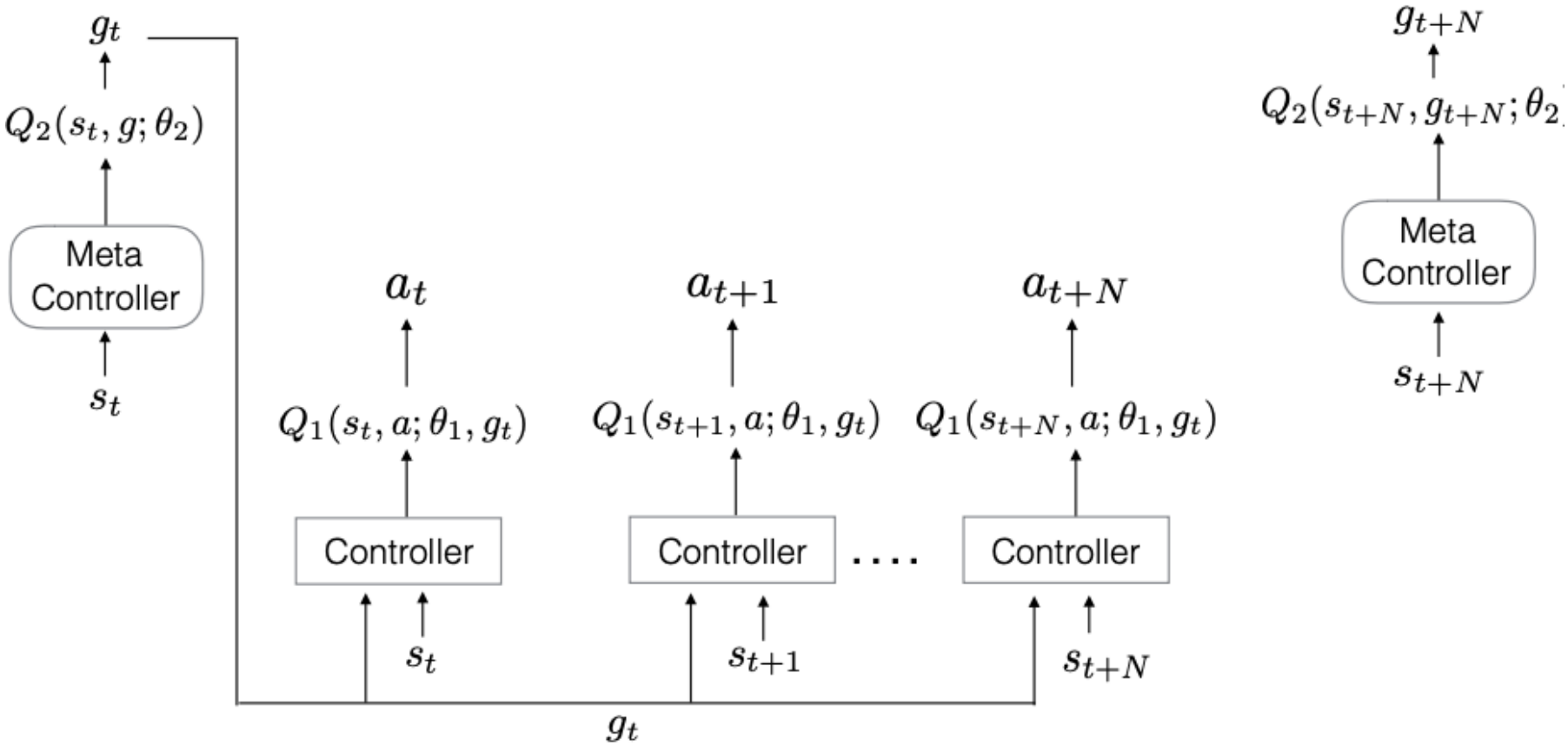
- Hierarchical RL (HRL) works on decomposing the RL problem into sub-problems where solving each of these is more powerful than solving the entire problem.
- HRL techniques use several forms of abstractions that have the ability to handle the exponentially increasing number of parameters.

Hierarchical Reinforcement Learning

- Hierarchical RL (HRL) works on decomposing the RL problem into sub-problems where solving each of these is more powerful than solving the entire problem.
- HRL techniques use several forms of abstractions that have the ability to handle the exponentially increasing number of parameters.
- A well-designed reward function in the HRL setting can decrease the number of impractical acts of exploration.

Hierarchical Reinforcement Learning

- Goal-conditioned Hierarchical RL comprises of a high-level policy that breaks the original task into a series of subgoals and a low-level policy that aims to reach those subgoals.



Generating Adjacency-Constrained Subgoals in Hierarchical Reinforcement Learning

Tianren Zhang, Shangqi Guo, Tian Tian, Xiaolin Hu, Feng Chen

June 2020

Generating Adjacency-Constrained Subgoals in Hierarchical Reinforcement Learning (2020)

- The subgoals are interpreted as high-level actions allowing direct training of the meta-controller to generate subgoals using external rewards as supervision.

Generating Adjacency-Constrained Subgoals in Hierarchical Reinforcement Learning (2020)

- The subgoals are interpreted as high-level actions allowing direct training of the meta-controller to generate subgoals using external rewards as supervision.

- **Problem**: Training inefficiency in large goal spaces for the meta-controller. Controller training also suffers as the agent tries to reach every possible subgoal produced by the meta-controller.

Generating Adjacency-Constrained Subgoals in Hierarchical Reinforcement Learning (2020)

- The subgoals are interpreted as high-level actions allowing direct training of the meta-controller to generate subgoals using external rewards as supervision.
- **Problem**: Training inefficiency in large goal spaces for the meta-controller. Controller training also suffers as the agent tries to reach every possible subgoal produced by the meta-controller.
- **Proposal**: The high-level action space can be restricted to a k-step adjacent region centered at the current state.

Preliminaries: Notation

- Consider a finite-horizon goal conditioned MDP defined as a tuple (S, G, A, P, R, γ), where S is a state set, G is a goal set and A is an action set.
- $P : S \quad A \quad S \quad ! \quad \mathrm{R}$ is the state transition function, $R : S \quad A \quad ! \quad \mathrm{R}$ is a reward function, $\gamma \; 2 \; [0, 1)$ is a discount factor and $\Psi : S \quad ! \quad G$ is a known mapping function.
- Meta-controller with policy $\pi^{h}_{\theta_h}(gjs)$, controller with policy $\pi^{l}_{\theta_l}(ajs, g)$ comprising a two-level hierarchy.

Thesis

- Distant subgoals can be substituted by closer subgoals, as long as they drive the controller to move towards the same "direction".

Thesis

- Distant subgoals can be substituted by closer subgoals, as long as they drive the controller to move towards the same "direction".

- The meta-controller policy only needs to explore in a subset of subgoals covering states that the controller can possibly reach within k steps.

Thesis

- Distant subgoals can be substituted by closer subgoals, as long as they drive the controller to move towards the same "direction".
- The meta-controller policy only needs to explore in a subset of subgoals covering states that the controller can possibly reach within k steps.
- For the controller, adjacent subgoals provide a stronger learning signal as the agent can be intrinsically rewarded with a higher frequency for reaching these subgoals.

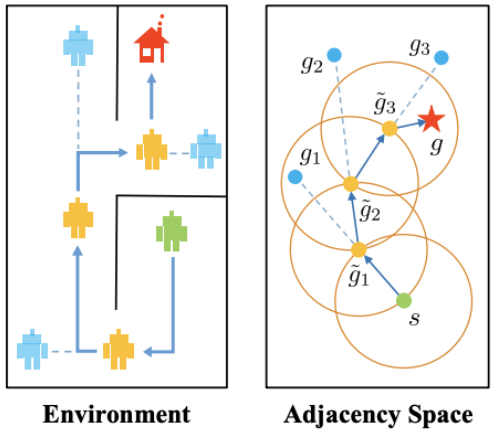


Figure 1: High-level illustration of our method: distant subgoals g_1, g_2, g_3 (blue) can be surrogated by closer subgoals $\tilde{g}_1$, $\tilde{g}_2$, $\tilde{g}_3$ (yellow) that fall into the k-step adjacent regions.

Shortest Transition Distance

Definition 1. Let s_1, s_2 *2 S*. Then, the *shortest transition distance* from s_1 to s_2 is defined as:

$$d_{st}(s_1, s_2) := \min_{\pi 2 \Pi} E[T_{s_1 s_2} j \pi] = \min_{\pi 2 \Pi} \sum_{t=0}^{1} tP(T_{s_1 s_2} = t j \pi)$$

where Π is the complete policy set and $T_{s_1 s_2}$ denotes the first hit time from s_1 to s_2.

Shortest Transition Distance

Definition 1. Let s_1, s_2 2 S. Then, the *shortest transition distance* from s_1 to s_2 is defined as:

$$d_{st}(s_1, s_2) := \min_{\pi 2 \Pi} E[T_{s_1 s_2} j \pi] = \min_{\pi 2 \Pi} \sum_{t=0}^{1} t P(T_{s_1 s_2} = t j \pi)$$

where Π is the complete policy set and $T_{s_1 s_2}$ denotes the first hit time from s_1 to s_2.

An optimal (deterministic) goal-conditioned policy $\pi \ : S \quad G \ ! \ A$ is:

$$\pi \ (s, g) \ 2 \ \text{argmin}_{a 2 A} \sum_{s^{\theta} 2 S} P(s^{\emptyset} j s, a) d_{st}(s^{\emptyset}, \Psi \ ^{1}(g))$$

$8s \ 2 \ S, 8g \ 2 \ G$

k-Step Adjacent Region

Definition 2. Let $s\ 2\ S$. Then, the k-step adjacent region of s is defined as:

$$G_A(s,k) = fg\ 2\ Gjd_{st}(s,\Psi\ \ ^1(g)) \quad kg$$

k-Step Adjacent Region

Definition 2. Let $s\ 2\ S$. Then, the k-step adjacent region of s is defined as:

$$G_A(s,k) = fg\ 2\ Gjd_{st}(s,\Psi\ \ ^1(g))\quad kg$$

- Consider a goal-conditioned hierarchical policy where the controller is required to reach the subgoals within k limited steps.

k-Step Adjacent Region

Definition 2. Let $s\ 2\ S$. Then, the k-step adjacent region of s is defined as:

$$G_A(s,k) = fg\ 2\ Gjd_{st}(s, \Psi\ ^{1}(g)) \quad kg$$

- Consider a goal-conditioned hierarchical policy where the controller is required to reach the subgoals within k limited steps.
- In deterministic MDPs, given an optimal controller policy $\pi^{l} = \pi$, subgoals that fall in the k-step adjacent region of the current state can represent all optimal subgoals in the whole goal space.

Using closer surrogate subgoals

Theorem 1: Let $s \in S$, $g \in G$ and let π be an optimal goal-conditioned policy. Under a deterministic MDP with strongly connected states, for all $k \in \mathbb{N}_+$ satisfying $k \geq d_{st}(s, \Psi^{-1}(g))$ there exists a surrogate goal $\tilde{g}$ such that:

$$\tilde{g} \in G_A(s, k)$$

$$\pi(s_i, \tilde{g}) = \pi(s_i, g)$$

$\forall s_i \in \tau (i \neq k)$ where $\tau := (s_0, s_1, \ldots, s_k)$ is the k-step trajectory starting from state $s_0 = s$ under π and g.

Meta-Controller Optimizing Objective

- This suggests that the k-step action sequence generated by an optimal controller policy conditioned on a distant subgoal can be induced using a subgoal that is closer.

Meta-Controller Optimizing Objective

- This suggests that the k-step action sequence generated by an optimal controller policy conditioned on a distant subgoal can be induced using a subgoal that is closer.

- In the deterministic setting they constrain the meta-controller's action space to state-wise k-step adjacent regions without a loss of optimality.

Meta-Controller Optimizing Objective

- This suggests that the k-step action sequence generated by an optimal controller policy conditioned on a distant subgoal can be induced using a subgoal that is closer.

- In the deterministic setting they constrain the meta-controller's action space to state-wise k-step adjacent regions without a loss of optimality.

- Employing relaxation methods they derive the following optimizing objective:

$$\max_{\theta_h} E_{\pi_{\theta_h}} \sum_{t=0}^{T \quad 1} [\gamma^t r_{kt}^h \quad \eta H(d_{st}(s_{kt}, \Psi \quad ^1(g_{kt})), k)],$$

where r_{kt}^h is the reward for the meta-controller's policy,
$H(x,k) = max(x/k \quad 1, 0)$ is a hinge loss function, and η is a balancing coefficient.

Aggregated Adjacency Matrix and Adjacency Network

- Instead of a perfect shortest transition distance between states, a discriminator of k-step adjacency is needed.

Aggregated Adjacency Matrix and Adjacency Network

- Instead of a perfect shortest transition distance between states, a discriminator of k-step adjacency is needed.
- Use the agent's trajectories to construct and update an aggregate binary k-step adjacency matrix.

Aggregated Adjacency Matrix and Adjacency Network

- Instead of a perfect shortest transition distance between states, a discriminator of k-step adjacency is needed.
- Use the agent's trajectories to construct and update an aggregate binary k-step adjacency matrix.
- However, the adjacency matrix has a tough time generalizing to newly-visited sets of states and is non-differentiable.

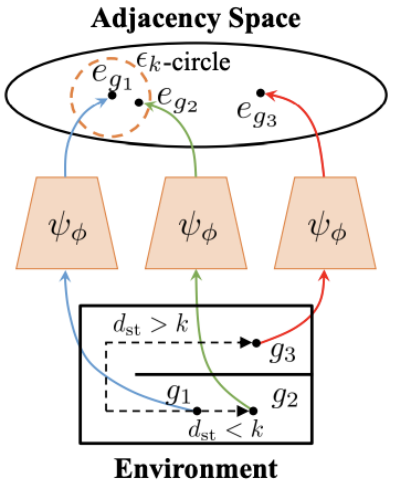


Figure 4: The functionality of the adjacency network. The k-step adjacent region is mapped to an ϵ_k-circle in the adjacency space, where $e_{g_i} = \psi_\theta(g_i)$, $i = 1, 2, 3$.

Aggregated Adjacency Matrix and Adjacency Network

- Employ an adjacency network, Φ_ϕ with parameter ϕ, that learns a mapping from the goal space to an adjacency space ensuring a binary relation for implementing the adjacency constraint.

Aggregated Adjacency Matrix and Adjacency Network

- Employ an adjacency network, Φ_ϕ with parameter ϕ, that learns a mapping from the goal space to an adjacency space ensuring a binary relation for implementing the adjacency constraint.

- Ex: $jj\Phi_\phi(g_1) \quad \Phi_\phi(g_2)jj_2 > \epsilon_k$ for $d_{st}(s_1, s_2) > k$ and $jj\Phi_\phi(g_1) \quad \Phi_\phi(g_2)jj_2 < \epsilon_k$ for $d_{st}(s_1, s_2) < k$.

Aggregated Adjacency Matrix and Adjacency Network

- Employ an adjacency network, Φ_ϕ with parameter ϕ, that learns a mapping from the goal space to an adjacency space ensuring a binary relation for implementing the adjacency constraint.

- Ex: $jj\Phi_\phi(g_1) \quad \Phi_\phi(g_2)jj_2 > \epsilon_k$ for $d_{st}(s_1, s_2) > k$ and $jj\Phi_\phi(g_1) \quad \Phi_\phi(g_2)jj_2 < \epsilon_k$ for $d_{st}(s_1, s_2) < k$.

- The adjacency network is trained by minimizing an objective that penalizes adjacent state embeddings with large Euclidean distances in the adjacency space and non-adjacent state embeddings with small Euclidean distances.

Updated Meta-Controller Objective

- With a learned adjacency network, Φ_ϕ, they incorporate the adjacency constraint into the goal-conditioned HRL framework.

Updated Meta-Controller Objective

- With a learned adjacency network, Φ_ϕ, they incorporate the adjacency constraint into the goal-conditioned HRL framework.

- They minimize the following objective for the meta-controller:

$$L(\theta_h) = \quad E_{\pi_{\theta_h}} \sum_{t=0}^{T \quad 1} [\gamma^t r_{kt}^h \quad \eta L_{adj}]$$

where $L_{adj}(\theta_h)$ / max$(jj\Phi_\phi(\Psi(s_{kt})) \quad \Phi_\phi(g_{kt})jj_2 \quad \epsilon_k, 0)$ is defined as an adjacency loss and $g_{kt} \quad \pi_{\theta_h}^h(g_{kt}js)$.

Limitations

- A "k" constraint is manually defined as a hyper-parameter. Thus, extension of this procedure to various settings, that may require a different constraint, is resolved through simulation and cross-validation.

Limitations

- A "k" constraint is manually defined as a hyper-parameter. Thus, extension of this procedure to various settings, that may require a different constraint, is resolved through simulation and cross-validation.

- The theorems are derived in the context of the deterministic MDP. They empirically show that their method is robust to **certain** types of stochasticity.

Limitations

- A "k" constraint is manually defined as a hyper-parameter. Thus, extension of this procedure to various settings, that may require a different constraint, is resolved through simulation and cross-validation.
- The theorems are derived in the context of the deterministic MDP. They empirically show that their method is robust to **certain** types of stochasticity.
- For applications with vast state spaces, constructing a complete adjacency matrix will be problematic.

Q-Learning in Enormous Action Spaces Via Amortized Approximate Maximization

Tom Van de Wiele, David Warde-Farley, Adriy Mnih, Volodymyr Mnih

Deepmind. Jan 2020

Q-Learning Flexibility and Complexity

Why doesn't Q-learning scale? Recall Q-learning update:

- Maximization over action space A.
- $Q(s_t, a_t) \quad Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a2A} Q(s_{t+1}, a) \quad Q(s_t, a_t) \right)$

Q-Learning Flexibility and Complexity

Why doesn't Q-learning scale? Recall Q-learning update:

- Maximization over action space A.
- $Q(s_t, a_t) \quad Q(s_t, a_t) + \alpha \left(r_t + \gamma \max_{a2A} Q(s_{t+1}, a) \quad Q(s_t, a_t) \right)$
- Computing max on discrete A has complexity / jAj.
- Unable to maximize over continuous or hybrid A.

Extend DQN to Large Action Spaces

Challenge: traditional DQN cannot extend to large (continuous) action spaces.

Extend DQN to Large Action Spaces

Challenge: traditional DQN cannot extend to large (continuous) action spaces.

Goal:

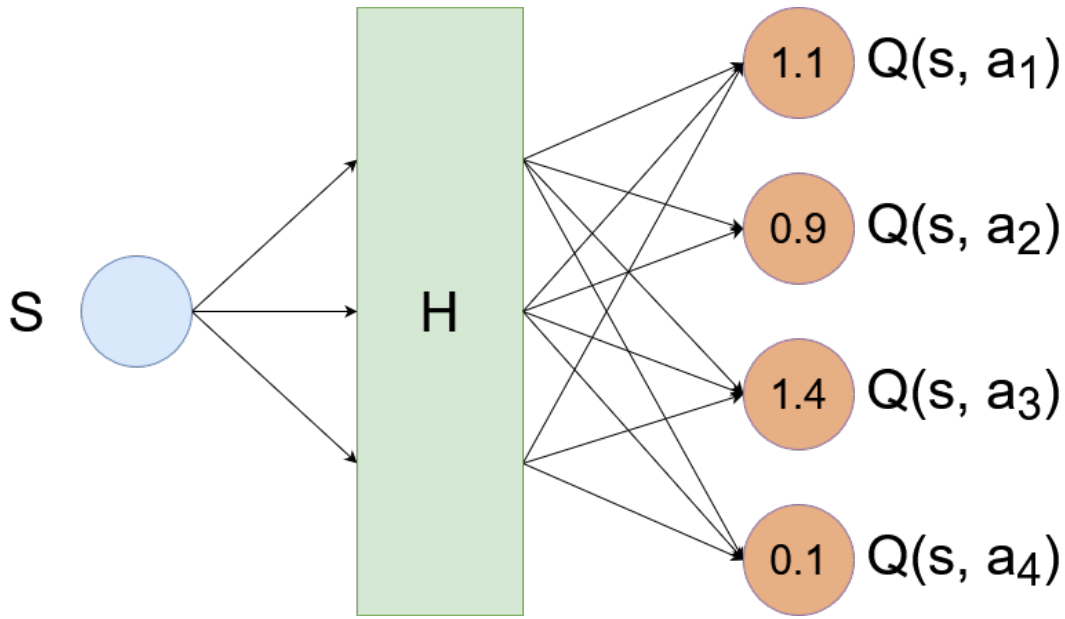
- Reduce computational cost dependence on jAj.
- Possibility of continuous/discrete action spaces.

DQN Review

Parameterize state-value function using a neural network Q_θ.

Usual case: network maps state input to (action, value) pairs.

- $Q_\theta : S \; ! \; A \quad V$
- # parameters $j\theta j$ roughly grows with jAj.



DQN Review: State, Action $!$ Value

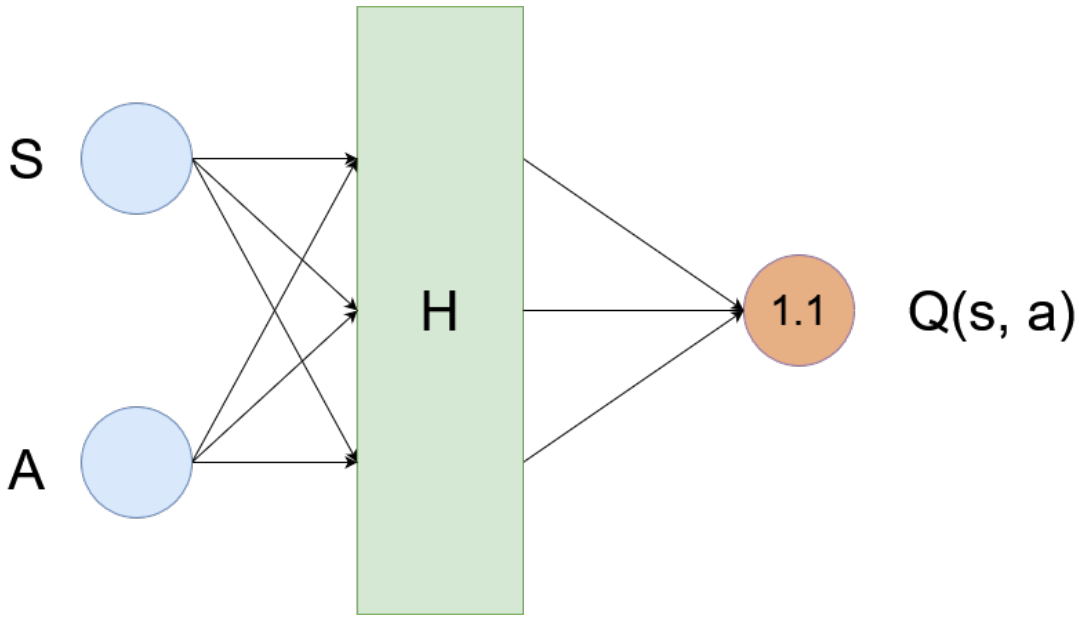
Can we reduce dependence of network size on action space?

DQN Review: State, Action ! Value

Can we reduce dependence of network size on action space?

Yes! Redefine Q_θ to map (state, actions) to values.

- $Q_\theta : S \quad A \;!\; V$.
- No explicit size dependence on jAj.
- Forward pass for each action to compute max value.



Amortized Q-Learning (AQL)

- Q-learning with cost *less* dependent on jAj.

Amortized Q-Learning (AQL)

- Q-learning with cost *less* dependent on jAj.
- Main idea: learn to search for good candidate actions.
- Only estimate value for candidate actions.
- Maximize over (much smaller) proposed set.

AQL: Approach

Learn a proposal distribution μ over possible actions $a\ 2\ A$.

- $\mu(ajs; \theta) =$ probability of being a high-value action.
- Hopefully: learn μ with high probability for optimal action.

AQL: Approach

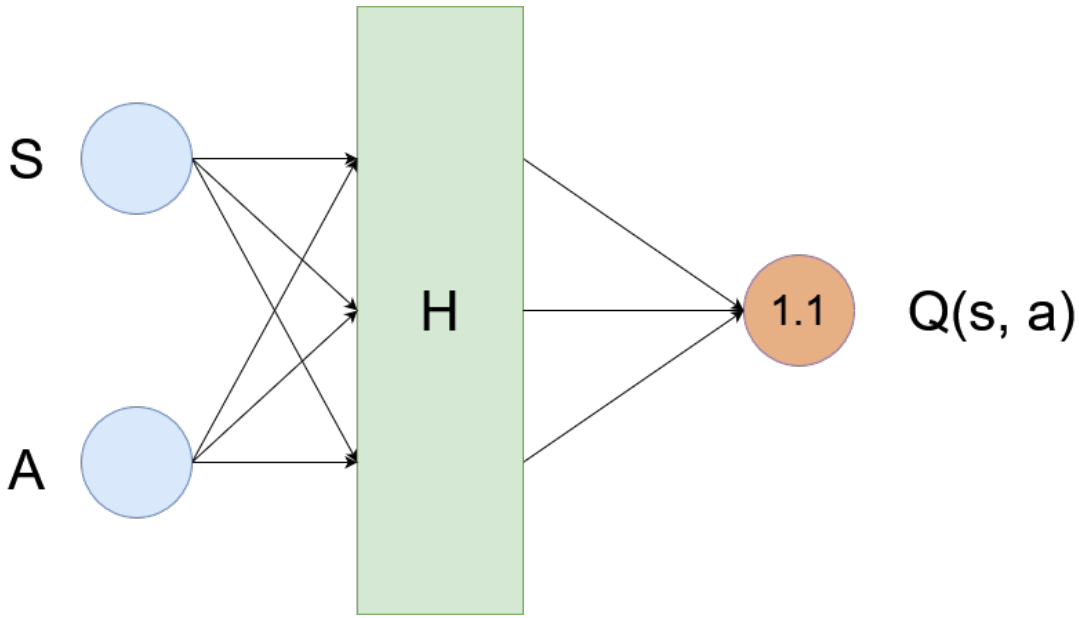
Learn a proposal distribution μ over possible actions $a \; 2 \; A$.

- $\mu(ajs; \theta) =$ probability of being a high-value action.
- Hopefully: learn μ with high probability for optimal action.

Parameterize μ with a neural network.

AQL: Q network

- $Q : S \quad A \;\; ! \;\; V$
- Usual DQN training procedure.



AQL: Algorithm

Two NNs: proposal network μ, Q-network $Q(s, a)$.

In a state s:

1. Proposal network forward pass computes $\mu(a\, j\, s)$.

AQL: Algorithm

Two NNs: proposal network μ, Q-network $Q(s, a)$.

In a state s:

1. Proposal network forward pass computes $\mu(a\, j\, s)$.
2. Sample actions: $A_{samp} = A_\mu \; [\; A_U$, where:
   $A_\mu := \{a_i\}_{j=1}^{N}$, $a_i \quad \mu(a\, j\, s)$
   $A_U := \{a_j\}_{j=1}^{M}$, $a_j \quad$ Uniform(A)

AQL: Algorithm

Two NNs: proposal network μ, Q-network $Q(s, a)$.

In a state s:

1. Proposal network forward pass computes $\mu(a\ j\ s)$.
2. Sample actions: $A_{samp} = A_\mu\ [\ A_U$, where:
   $A_\mu := \{a_i\}_{j=1}^{N},\ a_i \quad \mu(a\ j\ s)$
   $A_U := \{a_j\}_{j=1}^{M},\ a_j \quad \text{Uniform}(A)$
3. Q-network forward pass computes $Q(s, a)$ for each $a\ 2\ A_{samp}$.

AQL: Algorithm

Two NNs: proposal network μ, Q-network $Q(s, a)$.

In a state s:

1. Proposal network forward pass computes $\mu(a\, j\, s)$.
2. Sample actions: $A_{samp} = A_\mu\ [\ A_U$, where:
   $A_\mu := f a_i g_{j=1}^{N},\ a_i \quad \mu(a\, j\, s)$
   $A_U := f a_j g_{j=1}^{M},\ a_j \quad \text{Uniform}(A)$
3. Q-network forward pass computes $Q(s, a)$ for each $a\ 2\ A_{samp}$.
4. Choose optimal action as:

$$a\ (s) = \arg\max_{a2A_{samp}} Q(s, a)$$

AQL: Algorithm

Two NNs: proposal network μ, Q-network $Q(s, a)$.

In a state s:

1. Proposal network forward pass computes $\mu(a\, j\, s)$.
2. Sample actions: $A_{samp} = A_\mu\ [\ A_U$, where:
   $A_\mu := f a_i g_{j=1}^{N},\ a_i \quad \mu(a\, j\, s)$
   $A_U := f a_j g_{j=1}^{M},\ a_j \quad \text{Uniform}(A)$
3. Q-network forward pass computes $Q(s, a)$ for each $a\ 2\ A_{samp}$.
4. Choose optimal action as:

$$a\ (s) = \arg\max_{a2A_{samp}} Q(s, a)$$

Note: complexity proportional to $jA_{samp}j$, not jAj.
(Amortized cost!)

AQL: Proposal network

- Output softmax over actions.
- Train with (regularized) proposal loss:
- $L(\theta^{\mu}; s) = \quad log\mu(a\ (s)js; \theta^{\mu}) \quad \lambda H(\mu(ajs; \theta^{\mu}))$

AQL: Proposal network

- Output softmax over actions.
- Train with (regularized) proposal loss:
- $L(\theta^{\mu}; s) = \quad log\mu(a\ (s)js; \theta^{\mu}) \quad \lambda H(\mu(ajs; \theta^{\mu}))$

First term: increases likelihood of sampling optimal action from proposal.

- Recall: a was not necessarily sampled from μ (exploration).

AQL: Proposal network

- Output softmax over actions.
- Train with (regularized) proposal loss:
- $L(\theta^{\mu}; s) = \quad log\mu(a\ (s)js; \theta^{\mu}) \quad \lambda H(\mu(ajs; \theta^{\mu}))$

First term: increases likelihood of sampling optimal action from proposal.

- Recall: a was not necessarily sampled from μ (exploration).

Second term: encourages uncertainty in proposal distribution (exploration).

AQL: Sampling from proposal network

Discrete case:

- Apply softmax to μ output layer.
- Sample from resulting distribution.

AQL: Sampling from proposal network

Discrete case:

- Apply softmax to μ output layer.
- Sample from resulting distribution.

Continuous case:

- Option 1: discretize action space.

AQL: Sampling from proposal network

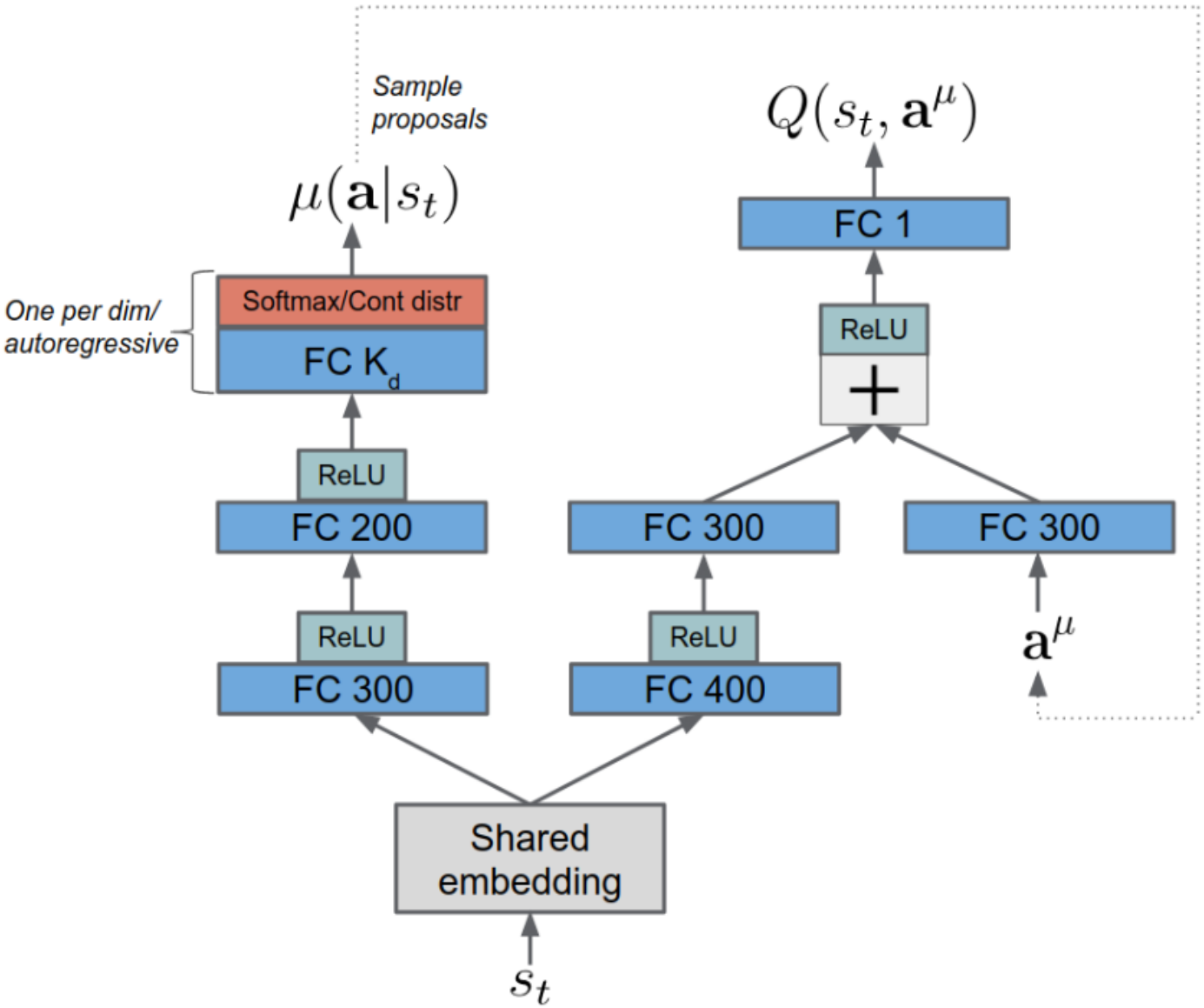
Discrete case:

- Apply softmax to μ output layer.
- Sample from resulting distribution.

Continuous case:

- Option 1: discretize action space.
- Option 2: μ parameterizes a Gaussian:
- I.e. Samples $a_i \quad N(\mu(a \mid s), \sigma^2)$

AQL: Architecture



AQL: Dependence on jAj

Does AQL address the problem of learning in large action spaces?

AQL: Dependence on jAj

Does AQL address the problem of learning in large action spaces?

- Proposal network has softmax over all actions.
- Why is this better than a DQN predicting (action, value) pairs?
- Both AQL and DQN have an output layer with jAj nodes.

AQL: Dependence on jAj

In general, it's "easier" to classify than regress,
(easier: requires fewer parameters, less training, etc...)

AQL: Dependence on jAj

In general, it's "easier" to classify than regress,
(easier: requires fewer parameters, less training, etc...)

- AQL: learning softmax over actions is multi-class classification.
- DQN: learning (action, value) pairs is many regressions.

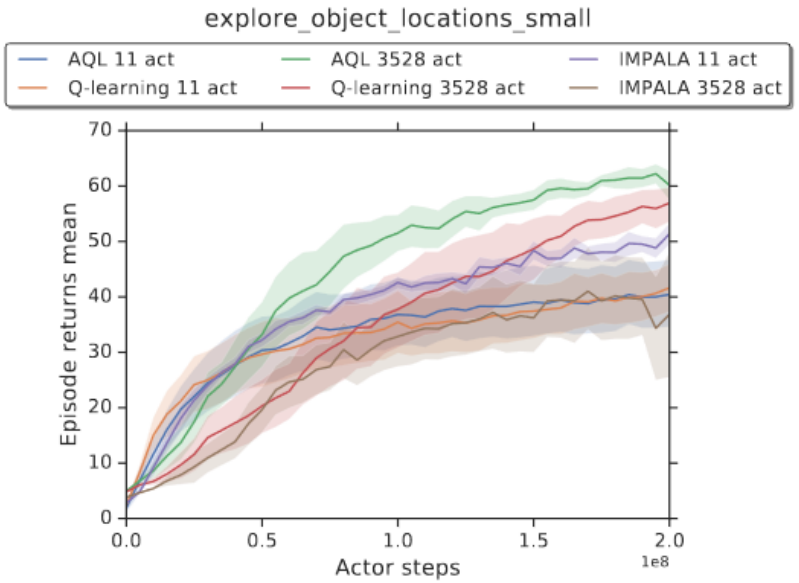
AQL: Dependence on jAj

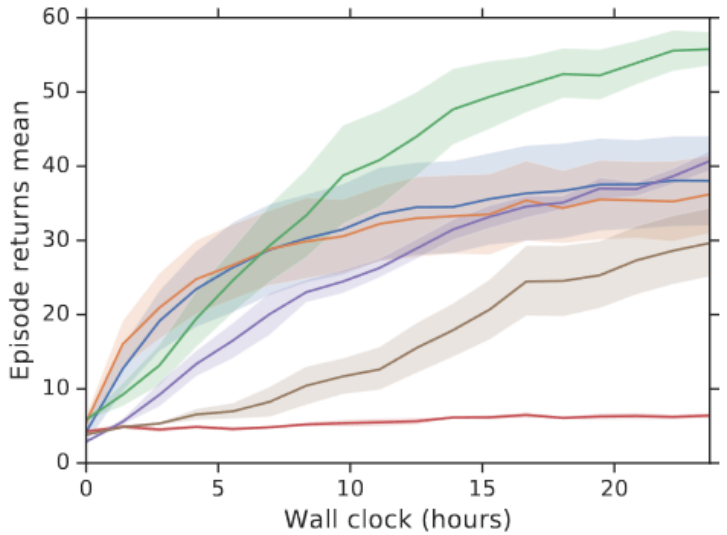
In general, it's "easier" to classify than regress,
(easier: requires fewer parameters, less training, etc...)

- AQL: learning softmax over actions is multi-class classification.
- DQN: learning (action, value) pairs is many regressions.

Intuition: classifying an action as good is "easier" than regressing *how* good it is.

Results: AQL per-step similar, per-second better than QL





AQL: Limitations

Hierarchical RL can exploit structure in problem. E.g.

- Inductive bias that the problem can be decomposed into sub-goals.
- Or that *local* subgoals are optimal (k-step adjacency HRL).

AQL: Limitations

Hierarchical RL can exploit structure in problem. E.g.

- Inductive bias that the problem can be decomposed into sub-goals.
- Or that *local* subgoals are optimal (k-step adjacency HRL).

AQL is "fixed" in its adaptability,

- but does amortize cost wrt action space size.

AQL: Limitations

Hierarchical RL can exploit structure in problem. E.g.

- Inductive bias that the problem can be decomposed into sub-goals.
- Or that *local* subgoals are optimal (k-step adjacency HRL).

AQL is "fixed" in its adaptability,

- but does amortize cost wrt action space size.

Our project: can we exploit structure in action space and reduce complexity dependence on jAj?

Questions?

References

- Tejas D. Kulkarni et al. *Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation*. 2016. arXiv: 1604.06057 [cs.LG].
- Volodymyr Mnih et al. "Playing atari with deep reinforcement learning". In: *arXiv preprint arXiv:1312.5602* (2013).
- Yunhao Tang and Shipra Agrawal. *Discretizing Continuous Action Space for On-Policy Optimization*. 2020. arXiv: 1901.10500 [cs.LG].
- Tom Van de Wiele et al. *Q-Learning in enormous action spaces via amortized approximate maximization*. 2020. arXiv: 2001.08116 [cs.LG].
- Tianren Zhang et al. *Generating Adjacency-Constrained Subgoals in Hierarchical Reinforcement Learning*. 2020. arXiv: 2006.11485 [cs.LG].