Cross Learning in Deep Q-Networks

Xing Wang. Alexander Vinel*

Department of Industrial Engineering
Auburn University
Auburn, AL, 36849, USA

Abstract

In this work, we propose a novel cross Q-learning algorithm, aim at alleviating the well-known overestimation problem in value-based reinforcement learning methods, particularly in the deep Q-networks where the overestimation is exaggerated by function approximation errors. Our algorithm builds on double Q-learning, by maintaining a set of parallel models and estimate the Q-value based on a randomly selected network, which leads to reduced overestimation bias as well as the variance. We provide empirical evidence on the advantages of our method by evaluating on some benchmark environment, the experimental results demonstrate significant improvement of performance in reducing the overestimation bias and stabilizing the training, further leading to better derived policies.

1 Introduction

Overestimation has been identified as one of the most severe problems in value-based reinforcement learning (RL) algorithms such as Q-learning [1], where the maximization of value estimates induces a consistent positive bias, and the error of the estimates is accumulated by the nature of temporal difference (TD) learning. In the function approximation setting such as deep Q-networks (DQN), the issue of value overestimation is more severe, given the noise induced by the inaccuracy of the approximation. As a result, learning DQN tends to have instability and variability for estimated Q-values, the derived policies according to the overestimated Q-values tend to be not optimal and often diverge.

To overcome this issue, double Q-learning [2] has become a standard approach for training DQNs. The main purpose of double Q-learning is to avoid the overestimation problem for the target Q-value, by introducing negative bias from the double estimates. The usual way to realize it in DQN is to maintain a target network which is a copy of the policy DQN which is either frozen for a period of time, or softly updated with exponential moving average. The target network then is used to estimate the TD target. This may alleviate the issue, however, double DQN still often suffer from overestimation in practice, partially because the policy and target estimates of Q-values are usually too similar, while the noise from high variance is propagated through the network and occasional large reward can produce great overestimation in the future. Another approach sometimes proposed is to impose a bias-correction term on the estimates for Q-learning [3], however, the error correction term is complicated to derive for deep networks, in which the finiteness of state space is no longer true. A more recent modification over double DQN favors underestimation and clips the Q-value estimates [4], that is, always chooses the minimum of the estimated targets over the two networks. The clipped double Q-learning is used on the critics in actor-critic methods for the deterministic policy gradient, which is referred to as TD3 (twin delayed deep deterministic policy gradient) and has shown state-of-the-art results on multiple tasks. However, the intentionally engineered underestimation lacks of rigorous theoretical guide, in addition, it may induce bias in the other direction, e.g., the

*Corresponding to: alexander.vinel@auburn.edu
underestimation can also accumulate through TD learning and derive suboptimal policies. Further, excessive underestimation can naturally lead to slower convergence.

Another direction to alleviate overestimation is through reducing the variance during training. For example, [5] uses the average of the learned estimated Q-values from multiple networks, which is designed to help reduce the target approximation error. There also exist various variance reduction techniques [6, 7, 8, 9] that focus on the general non-convex optimization procedure for accelerating the stochastic gradient descent, or their direct application on DQNs [10], in which the agent could obtain smaller approximated gradient errors. Reducing the variance can effectively stabilize the DQN training procedure, and overestimation alleviation can be seen as a by-product. However, these are indirect methods for overestimation control, and the positive bias due to the max operator in TD update are not taken care of.

To address these concerns, we propose a cross DQN algorithm, which can be seen as a direct extension of an earlier variant of double DQN, but can be more flexible. In cross DQN, we maintain more than two networks, and update them one at a time based on the estimation from another randomly selected one. As mentioned above, the averaged DQN [5] calculates the average of $K$ estimated Q-values, with the primary purpose of the overall variance reduction. For all $K$ networks, each step of TD updates as well as action selections are based on combining the $K$ estimates. Consequently, the networks are tangled together and cannot be implemented with a parallel simulation. In bootstrapped DQN [11], one of the $K$ networks (or heads) is bootstrapped for each action selection step during training, aiming at encouraging exploration early on. Thus the simulation is not independent among networks, while the TD updates are totally independent within each of the networks, by using its own estimation of Q-values as in standard (double) DQN. [12] investigates more general applications of traditional ensemble reinforcement learning on policies, i.e., majority voting, rank voting, Boltzmann addition, etc. to combine the different policies derived from multiple networks, by which they called the target ensembles, in addition to the averaged DQN which they called the temporal ensemble. All of the above-mentioned work that maintain multiple networks have achieved better performance by addressing different issues through some particular settings. Our method focuses on the variation of TD updates, in which the target Q-values are estimated with a bootstrapped network for calculating the gradients, with the direct goal of reducing overestimation. Each of the $K$ networks would perform its own TD updates, while maintaining flexibility in action selections: the networks can either interact with the environment independently, or through any other ensemble strategy. The detailed implementation options would be discussed in Section 3.

In supervised learning, ensemble strategies such as bagging, boosting, stacking, and hierarchical mixture of experts, etc. are commonly applied to achieve better performance, by simultaneously learning and combining multiple models. All of the abovementioned algorithms that maintain multiple models, including ours, can be seen as special cases of general ensemble DQNs. But our method has a deeper root in resampling and model selection. By bootstrapping another model to assess the values of current model, we introduce model bias for in-sample estimations, but reduce the variance of out-of-sample estimations (i.e., the squares of out-of-sample bias), in other words, the trained model can generalize better and alleviate overfitting. For squared errors, this can be expressed as the well-known bias-variance trade-off: $\text{MSE} = \text{Irreducible Error}^2 + \text{Bias}^2 + \text{Variance}$. In value-based reinforcement learning, the model easily overfits due to overestimation (which is caused by the max operator) during learning. Cross Q-learning introduces underestimation bias, and further reduces the variance, thus improves the generalization of the trained model.

Like in [4], our work can be naturally extended to the state of the art actor-critic methods in continuous action space, such as the deep deterministic policy gradient [13], in which the critic network(s) are learned to give an estimate of the Q-value for the actor network to update its gradient and derive policies. Usually multiple critic networks are applied, however, rather than accumulating their learned gradients (either synchronously or asynchronously [14]) and optionally sharing network layers, no other information is shared among the critics. The extension of our method allows the critics to share their value estimates and utilize that of others, which leads to more accurate estimation of each critics, thus can improve the performance of these models. Similar to these actor-critic algorithms, our work can be implemented for parallel training easily, and the exchange of information among networks could take place either synchronously or asynchronously like the accumulation of gradients, as there is always tradeoff between synchronous and asynchronous update.
The rest of this paper is organized as follows. In Section 2, we resume the basics of value-based RL, and go through some recent related research. In Section 3, we formally define the estimators for the maximum expected values, along with their theoretical properties. The convergence of our cross estimator is shown in Section 4. Section 5 illustrates our cross DQN algorithm directly derived from the double DQN in details. We show some empirical results in Section 6. Finally, Section 7 draws conclusions and discusses future work.

2 Background

2.1 Value-based Reinforcement Learning

A natural abstraction for many sequential decision-making problems is to model the system as a Markov Decision Process (MDP) [15], in which the agent interacts with the environment over a sequence of discrete time steps. It is often represented as a 5-tuple: \( M = (S, A, T, R, \gamma) \), where \( S \) is a set of states; \( A \) is a set of actions that can be taken; \( T : S \times A \rightarrow \mathcal{P}_S \) is the transition function such that \( \sum_{s' \in S} T(s'|s,a) = 1 \), which denotes the (stationary) probability distribution over \( S \) of reaching a new state \( s' \), after taking action \( a \) in state \( s \); \( R \) is the reward function, which can take the form of either \( R : S \rightarrow \mathbb{R} \), \( R : S \times A \rightarrow \mathbb{R} \), or \( R : S \times A \times S \rightarrow \mathbb{R} \); and \( \gamma \in [0, 1) \) is the discount factor.

A policy \( \pi : S \rightarrow \mathcal{P}_A \) defines the conditional probability distribution of choosing each action while in state \( s \). For an MDP, once a stationary policy is fixed, the distribution of the reward sequence is then determined. Thus to evaluate a policy \( \pi \), it is natural to define the action value function under \( \pi \) as the expected cumulative discounted reward by taking action \( a \) starting from state \( s \) and following \( \pi \) thereafter:

\[
Q^\pi(s, a) \equiv \mathbb{E}_\pi \left[ \sum_{t=0}^{\infty} \gamma^t R_{t+\tau} | S_t = s, A_t = a \right] = R(s, a) + \gamma \int_{s'} T(s'|s,a)Q^\pi(s', \pi(s')). \tag{1}
\]

The goal of solving an MDP is to find an optimal policy \( \pi^* \) that maximizes the expected cumulative discounted reward in all states. The corresponding optimal action values satisfy \( Q^*(s, a) = \max_{\pi} Q^\pi(s, a) \), and Banach’s fixed-point theorem ensures the existence and uniqueness of the fixed-point solution of Bellman optimality equations [15]:

\[
Q^*(s, a) = R(s, a) + \gamma \int_{s'} T(s'|s,a) \max_{a'} Q^*(s', a') \tag{2}
\]

from which we can derive a deterministic optimal policy by being greedy with respect to \( Q^* \), i.e.,

\[
\pi^* = \arg\max_{a \in A} Q^*(s, a).
\]

In reinforcement learning problems, the agent must interact with the environment to learn the information about the transition and reward functions, meanwhile trying to produce an optimal policy. While interacting with the environment, at each time step \( t \), the agent senses some representation of current state \( s \), selects an action \( a \), then receives an immediate reward \( r \) from the environment and finds itself in a new state \( s' \). The experience tuple \( < s, a, r, s' > \) summarizes the observed transition for a single step. Based on the experiences through interacting with the environment, the agent can either learn the MDP model first by approximating the transition probabilities and reward functions, and then plan in the MDP to obtain an optimal policy (this is called the model-based approach in reinforcement learning); or without learning the model, directly learn the optimal value functions and upon which the optimal policy is derived (this is called the model-free approach).

As a model-free approach, Q-learning [16] updates one-step bootstrapped estimation of Q-values from the experience samples over time steps. The update rule upon observing \( < s, a, r, s' > \) is

\[
Q(s,a) \leftarrow Q(s,a) + \alpha \left( r + \gamma \max_{a'} Q(s', a') - Q(s,a) \right) \tag{3}
\]

in which \( \alpha \) is the learning rate, \( r + \max_{a'} Q(s', a') \) serves as the update target of the Q-value, which can be seen as a sample of the expected value of one-step look-ahead estimation for state-action pair \( (s,a) \), based on the the maximum estimated value over next state \( s' \), and the last term \( Q(s,a) \) is simply the current estimation. The difference \( \delta = r + \gamma \max_{a'} Q(s', a') - Q(s,a) \) is referred to as temporal difference (TD) error, or Bellman error. Note that one can bootstrap more than one
We consider Q-learning with function approximation in this paper. To update parameter vector $\theta$
Algorithm 1
A variety of extensions and generalizations have been proposed and shown successes in the literature. Experience replay
were made. Well known to be unstable and even to diverge when applied for reinforcement learning until deep Q-
action advantage function $A$ estimators, one for the state value function $V$ shared and followed by two streams of fully-connected layers, that are used to represent two separate
[20] proposed the dueling network architecture, in which lower layers of a deep neural network are
2.3 Dueling DQN
[20] proposed the dueling network architecture, in which lower layers of a deep neural network are
shared and followed by two streams of fully-connected layers, that are used to represent two separate estimators, one for the state value function $V(s)$ and the other for the associated state-dependent action advantage function $A(s, a)$. The two outputs are then combined to estimate the action value $Q(s, a)$:
\[ Q(s, a) = V(s) + A(s, a) - \frac{1}{|A|} \sum_{a'} A(s, a') \] 

(4)
Note here the average of advantage values across all possible actions are used to achieve better stability, instead of the max operator in the other form proposed in [20], i.e.,

\[ Q(s, a) = V(s) + A(s, a) - \max_{a'} A(s, a') \]  

(5)

The dueling factoring often leads to faster convergence and better policy evaluation, especially in the presence of similar-valued actions. The deployment of advantage values is more robust to noise, since it emphasizes the gaps between \( Q \)-values of different actions given the same state, which are usually tiny thus small amount of noise may result in reordering of actions. In addition, the subtraction of an action-irrelevant baseline in Equation (4) also effectively reduces variance, which helps stabilize learning and thus is more often used. The shared feature learning module also generalizes learning across actions, in which more frequent updating of the value stream \( V \) leads to more efficient learning of state values, contrasts with that in DQNs of a single stream output, only one of the action values is updated while other action values remain untouched.

2.4 Bootstrapped DQN

The main purpose of Bootstrapped DQN [11] is to provide efficient “deep” exploration inspired by Thompson sampling or as probability matching in Bayesian reinforcement learning [21], but instead of maintaining a distribution over possible values and intractable exact posterior update, it takes a single sample from the posterior. Bootstrapped DQN maintains a \( Q \)-ensemble, represented by a multi-head deep neural network in order to parameterize a set of \( K \in \mathbb{N}_{+} \) different \( Q \)-value functions. The lower layers are shared by the \( K \) “heads”, and each head represents an independent estimate of the action value \( Q^k(s, a|\theta^k) \). For each episode at training, Bootstrapped DQN picks a single head uniformly at random, and follows the greedy policy with respect to the selected \( Q \)-value estimates, i.e., \( a_t = \arg\max_a Q^k(s_t, a) \), until the end of the episode.

Bootstrapped DQN diversifies the \( Q \)-estimates and improves exploration through independent initialization of the \( K \) heads as well as the fact that each head is trained with different experience samples. The \( K \) heads can be trained together with the help of so-called bootstrap mask \( m^k \), which decides whether the \( k \)-th head should be trained, i.e., the transition experience \( \tau \) updates \( Q^k \) only if \( m^k \) is nonzero. In addition, bootstrapped DQN adapts double DQN in order to avoid overestimation, i.e., the the estimates of TD targets are calculated using the target network \( Q^k_{\theta^k} \). The loss backpropagated to \( k \)-the head is then

\[ L(\theta^k) = \mathbb{E}_\tau [m^k \{ r + \gamma Q^k(s', a'|\theta^k) - Q^k(s, a|\theta^k) \}^2] \]  

(6)

Note the gradients should be further aggregated and normalized for updating the lower layers of the network.

3 Cross DQN

In this section, we elaborate our proposed cross Q-learning method and its variants. Cross DQN serves as an extension to the double DQN algorithm [2], which has been used as the default setting for most state-of-art DQN training.

Double DQN was proposed in the aim of reducing overestimation bias, in which the target network simply is a delayed-updated copy of the current network. Note that the original vanilla DQN also uses two networks, the purpose of periodic frozen and update of the target network is to stabilize learning. Specifically, in vanilla DQN, the target network is used to evaluate both the action and the value, i.e.,

\[ y \leftarrow r + \gamma Q_{\theta}(s', a'_\star) \quad \text{where } a'_\star = \arg\max_{a'} Q_{\theta}(s', a') \]  

(7)

On the other hand, in double DQN, the current network is used to evaluate the action and select \( a' \), while the target network is used for evaluate the value, so that action selection is decoupled from estimation of the target:

\[ y \leftarrow r + \gamma Q_{\theta}(s', a'_\star) \quad \text{where } a'_\star = \arg\max_{a'} Q_{\theta}(s', a') \]  

(8)

In practice however, it is common the case that little improvement can be gained by using double DQN, since the current and target networks are usually too similar due to slowly changed parameters.
in neural network models with SGD optimization. We can neither set the period of updating target too long, otherwise the derived policy would not exhibit learning and progress. As a result, double DQN does not entirely eliminate the overestimation bias. In Section 4 we will further experimentally show the elimination of overestimation is not effective nor sufficient in double DQN.

Instead of maintaining only two separate networks, we will use a set of $K$ models for estimating Q-values and selecting actions in our cross Q-learning. While update each network’s parameters, we will calculate its TD target Q-value using one of the other $K - 1$ models. More specifically, let the network with parameters we are about to adjust be our current network ($\theta_i$), and we randomly pick another network to be our target network ($\theta_j$, e.g., $j \in U[1, K]$). To compute the target Q-value, we will use the current network to evaluate the actions and select $a'$ in the next state $s'$, while the value is evaluated by using the target network, i.e.,

$$y \leftarrow r + \gamma Q_{\theta_j}(s', a'_i) \quad \text{where } a'_i = \arg\max_a Q_{\theta_i}(s', a)$$

(9)

**Algorithm 2 Cross-Learning DQN**

1: Initialize $K \in \mathbb{N}_+$ different Q-functions $Q(s, a|\theta^k)$ with random parameters $\theta^k$ for $k = 1, \cdots, K$.
2: Initialize replay buffer $B$.
3: for each episode until end of learning do
4: Initialize state $s$
5: for step $t = 1, \cdots$ until $s$ is terminal state of an episode do
6: Select action $a_t$ according to $Q$ with exploration, e.g., $a_t = \text{MajorityVote}\{\arg\max_a Q_{\theta}\}_{k=1}^K(s, a)$
7: Take action $a_t$, observe reward $r$ and next state $s'$
8: Store experience tuple $<s, a_t, r, s'>$ into $B$
9: Sample a mini-batch of experiences from $B$.
10: for all sampled experience in the mini-batch do
11: To train network $Q^i$, compute $a'_i = \arg\max_a Q^i(s', a|\theta^i)$
12: Randomly pick another network $Q^j$ to estimate TD target $y = r + Q^j(s', a'_i)$
13: Backpropagate TD error $\delta = y - Q^i(s, a|\theta^i)$ through $Q^i$, update $\theta^i$ with learning rate $\alpha_t$
14: end for
15: $s \leftarrow s'$
16: end for
17: end for

In implementation, we have flexibility and various options in how to utilize the $K$ different Q-networks. There always exist tradeoffs among different choices that we need to consider in order to pick the one that meets our goal most. For example, we can have different design of neural network architectures. A natural choice of having $K$ independent models is to maintain a list of separate neural networks with the same architecture, the difference between their outputs (i.e., $K$ streams of Q-values derived from the same $(s, a)$-pair as the input) comes from different random parameter initialization of each model, also is due to that different data that each model is trained upon, i.e., for each step of backpropagation, each model randomly samples a mini-batch of experiences and performs SGD optimization with the mini-batch. Maintaining $K$ copies of models implies that not only the storage for the models would be $K$ times large as a single network, also the forward propagation would take $K$ times amount of computations. Instead, we can utilize the shared network design for the $K$ models, in which the $K$ models shared their weights except for the last layer, which consists of $K$ value function heads from which the value functions $Q_k(s, a|\theta_k)$ are derived, and the weights on the last layer are generally different. Thus we have much less parameters in total to be trained, and the computational burden can be greatly alleviated. Moreover, as recent deep learning research reveals, the first few layers of neural network are mainly about representations learning, the shared layers provide the same features expressed for computing $Q$, this can be seen as online transfer of learned knowledge among models. Note that in shared learning settings, in order to avoid premature learning and suboptimal convergence, the gradients of the network except the last layer are usually normalized by $1/K$, but this also results in slower learning early on. On the other hand, the separate models are simpler yet provide more variability in Q-values, also are more stable during training. In addition, when we train the networks in distributed system, the separate networks do not depend on others’ weights thus can be learned independently, which requires much less information exchange and this could be a huge advantage for distributed learning. The comparison of the separate and shared network architectural design is shown in Figure 1.
With \( K \) different models (or heads), while each could derive a possibly different policy, there is no doubt that during test phase we should take advantage of ensembles, for instance by choosing the action with the majority votes across the outputs. However, we can make choices on how to combine action selections into a single policy during training. With ensemble action selection such as majority voting, the derived policy is often superior than any individual one, thus greatly reduces the variance during training, as we will experimentally show in Section 4. This in turn refines exploitation, results in great variance reduction of \( Q \)-values and speeds up learning. Note that to deal with exploration-exploitation dilemma, \( \varepsilon \)-greedy strategy is needed to encourage exploration.

On the other hand, we may also randomly pick a single network from the \( K \) models, and act as it suggests during training. This falls into the paradigm of Bootstrapped DQN [11], which encourages exploration, in the cost of slower early learning (see Section 4), but may learn better policy later with more exploration. Another advantage of bootstrapped action selection is that it can slightly reduce computational burden, since instead of forward passing and computing all \( K \) of the \( Q \)-values for action selection, we can calculate only one of them. The procedure of bootstrapped version of cross DQN is presented in Algorithm 3.

**Algorithm 3 Bootstrapped Cross DQN**

1: Initialize \( K \in \mathbb{N}_+ \) different Q-functions \( Q(s,a|\theta^k) \) with random parameters \( \theta^k \) for \( k = 1, \cdots, K \).
2: Initialize replay buffer \( B \).
3: for each episode until end of learning do
4: Initialize state \( s \)
5: Randomly pick a network \( Q^h \) to act, where \( k \in \{1, \cdots, K\} \).
6: for step \( t = 1, \cdots \) until \( s \) is terminal state of an episode do
7: Select action \( a_t = \arg\max_a Q^h(s,a') \) with exploration
8: Take action \( a_t \), observe reward \( r \) and next state \( s' \)
9: Store experience tuple \( <s, a_t, r, s'> \) into \( B \)
10: Sample a mini-batch of experiences from \( B \).
11: for all sampled experience in the mini-batch do
12: To train network \( Q' \), compute \( a' = \arg\max_a Q'(s',a'|\theta^k) \)
13: Randomly pick another network \( Q' \) to stimate TD target \( y = r + Q'(s',a'|\theta^k) \)
14: Backpropagate TD error \( \delta = y - Q'(s,a_t|\theta^k) \) through \( Q' \), update \( \theta^k \) with learning rate \( \alpha_t \)
15: end for
16: \( s \leftarrow s' \)
17: end for
18: end for

Another choice we can make is the training frequency. In our cross DQN settings, when backpropagation occurs, we can either choose to train on a single network (e.g., the single model that
provides the action selection), or each of the $K$ networks could independently sample a mini-batch of experiences and perform SGD optimization. The latter would increase the sample efficiency and speed up learning, while the former would reduce the computational burden, in which the number of backpropagation (which is the most computational expensive) remains the same as in a single DQN. In addition, with the former setting, our cross Q-learning does not require maintaining copies of the networks as the target. Experimentally, we found that freezing targets merely has any effect on stabilization of learning, but only costs doubled memory for model storage. This is due to two reasons. First, we bootstrap a model that is different than the current one, when $K \geq 2$, the variety of models ensures the difference in parameter initialization, as well as the difference of mini-batch data their learning based upon, which in turn ensures the independence of Q-value estimates. Secondly, with less frequent update of each network, the bootstrapped target $Q$-value changes less as well, also helps stabilize learning.

4 Experimental Results

In this paper, we conducted experiments on two classical control problems, CartPole and LunarLander, for extended tests. We selected these testbeds in the aim of covering different challenges, especially in terms of complexity. As both environments interfaced through OpenAI gym environment [22], unless specified otherwise. The neural networks have a number of hyperparameters. The combinatorial space of hyperparameters is too large for an exhaustive search, therefore we have performed limited tuning. For each component, we started with the same settings as in [23] in order to make comparisons with states of the art results.

4.1 CartPole

4.1.1 Experimental Setup

The CartPole, also known as an inverted pendulum, in which a pole (or pendulum) is attached by an un-actuated joint to a cart (i.e., the pivot point). The pendulum starts upright at the center of a 2D track but is unstable since the center of gravity is above the pivot point. The goal of this task is to keep the pole balanced and prevent it from falling over, by applying appropriate force to the pivot point, while the force could move the cart along the frictionless track with finite length of 4.8 units. An immediate reward of $+1$ is provided for every timestep that the pole remains not falling over, and the maximum cumulative rewards in an episode are clipped to 200. An episode also ends when the pole is slanted with degree $> 15^\circ$ from vertical, or the cart moves out of the track [24]. In each timestep, the agent is provided with current state $s \in \mathbb{R}^4$, which represents cart position, cart velocity, pole angle, and pole angular velocity, respectively. A unit force either from left or right can be applied, thus the actions are discrete with $a \in \{-1, 0, +1\}$.

As in [23], we approximate the $Q$-values using a neural network with two fully-connected hidden layers (which consist of 64 and 32 neurons, respectively). We train each of the neural networks for 1000 episodes (approximately a little less than 200000 steps), with a FIFO memory of size $5 \times 10^4$ transitions for experience replay. A target network is updated every 500 steps to further stabilize learning. The adaptive moment estimation (Adam) optimizer with learning rate $\alpha = 0.001$ is used to train the network, since it is in general less sensitive to the choice of the learning rate than other stochastic gradient descent algorithms [25]. The optimization is performed on mini-batches of size 32, sampled uniformly from the experience replay. The discount factor $\gamma$ is set to 0.99, and $\varepsilon$-greedy policy is used for choosing actions throughout interacting with the environment, which starts with exploration $\varepsilon = 1$, and annealed to 0.02 in the first 10000 steps.

After every 20 training episodes, we conduct a performance test that plays 10 full episodes using the greedy policy deterministically derived from the current network. For the models with $K > 1$, majority voting is used for the action selection disregard whether or not bootstrapped $Q$-value head is used during training. The cumulative rewards of each test episode are used for comparison among different models. Moreover, in order to comparing the estimation of $Q$-values among models, every 20 training episodes, we randomly sample a batch of historical 1024 $(s, a)$-pairs from the replay buffer and compute their $Q$-values using current network. More than one thousand samples ensure that their mean is somewhat representative for $Q$-values under current model.
4.1.2 Analysis of Cross Q-learning Effects

We compared our cross Q-learning algorithms with vanilla DQN and double DQN. Note that vanilla DQN uses single estimators, while double DQN uses double estimators, and our cross DQN uses cross estimators. $K = 5$ and $K = 10$ are used in cross DQNs. Figure 2(a) illustrate the training history of episodic total rewards of the four models, from which we can see that although with a single network (vanilla and double DQNs), the agent starts to learn early on with less samples, in particular, double Q-learning helps the single network to learn even faster, however, the learned models are not stable. With cross Q-learning, although the networks learn slower at the beginning, in particular, cross DQN with $K = 10$ started to learn even later than cross DQN with $K = 5$, once cross DQNs start to learn, the performance improvement is substantial. Not only the total rewards are higher, the learning is also much more stable. After 300 episodes, the training total rewards converge to 200 for $K = 10$ cross DQN, with little variation (due to $\epsilon$ exploration). $K = 5$ cross DQN has more variation, but it also seems to converge after 900 episodes, while vanilla DQN and double DQN are easily deteriorated, and have much larger variations.

The performance improvement can be more clearly seen in Figure 2(b). After 300 episodes of training, the policies derived cross DQN with $K = 10$ become more and more stable, the variance of test total rewards become zero close to the end of training. Cross DQN with $K = 5$ deteriorates after 500 episodes of training, but later it also learns to derive stable policy that has total rewards of 200 with tiny variances. Whereas the policies derived from vanilla DQN and double DQN can only get score which is approximately half of cross DQNs, and with large variances. The policy derived from double DQN seems to be a little better than that from vanilla DQN, but the improvement is not as significant as that of using cross Q-learning.

Furthermore, part of the reason for slower start of cross DQN is due to our learning settings, in which we only perform SGD optimization on one of the networks (or heads). In other words, we reduce the learning frequency of each network (or head) down to $1/K$ to alleviate the computational effort, at the cost of slower start on learning. If we increase the learning frequency (i.e., backpropagate for each of the $K$ networks/heads every time), the learning should be faster.

We also plot the average $Q$-values from bootstrapped 1024 $(s, a)$-pairs as shown in Figure 2(c). We observe that the beginning of learning, vanilla DQN has highest estimates of $Q$-values, which is an evidence of overestimation. The estimates from double DQN is lower, but only for limited amount, therefore we say that double Q-learning may have not solve the overestimation problem completely. Cross DQNs have quite smaller estimations at the beginning, in particular, as $K$ gets larger, the estimates of $Q$-values become even lower. Overestimation is clearly an obstacle of effective learning, as a result, the estimated $Q$-values from cross DQNs are substantially higher than that from vanilla or double DQNs, since cross DQNs has derived better policies and obtained higher rewards. The $Q$-values estimates from cross DQNs start to converge after the derived policies stabilized. At the end of training, the estimated $Q$-values from the four different models are about at the same level, however, note that the estimates from vanilla and double DQNs continue increasing, and their derived policies are not stable, also have lower rewards. Our cross Q-learning algorithm has addressed the overestimation problem better.

4.1.3 Effects of dueling DQN & Bootstrapped DQN

As the cross learning architecture shares the same input-output interface with standard DQN, we can recycle many recent advances in DQN research. We have mentioned one variant in Section 3 that it can combined with Bootstrapped DQN for action selection during training, while in Section 4.1.2, our experiments for cross DQN are based on majority voting from $K$ different $Q$-functions. Furthermore, it is convenient to combine the dueling architecture into each of the $K$ networks. The goal of dueling DQN is to reduce variance for $Q$-value estimation, by subtracting a baseline and emphasizing the advantages among different actions, thus accelerates learning effectively. The variance reduction is performed on a single network’s estimation, while our cross Q-learning reduces variance from a different perspective. For each network, the target values were calculated with other models by bootstrapping from multiple $Q$-values, thus introduces some bias. Due to the bias-variance tradeoff, however, the variance of our estimates decreases, and thus the overall error becomes smaller. In addition, the maximum operator induces overestimation bias, while cross-estimator tends to introduce bias in the other direction, thus greatly alleviates overestimation problem.
Figure 2: Comparison of vanilla DQN, double DQN and cross DQNs of $K = 5$, $K = 10$ on CartPole.

Figure 3 and Figure 4 illustrate the training and testing performance of cross DQN with different architectures, for the cases of $K = 5$ and $K = 10$, respectively. We can see that dueling architecture speeds up early on learning effectively, without hurting the model performance later in general. On the other hand, Bootstrapped DQN slows learning at beginning, especially when $K$ is large, since the selected actions varies among networks at beginning quite a bit. For example, the $K = 10$ cross DQN with bootstrap converges around 400 episodes while the other cross learning agents converges before 200 episodes. But after learned something, the bootstrapped action selection won’t hurt the model. In fact, it might help learning for more complicated tasks because of more exploration early on. At least, using bootstrapped DQN can help our cross DQN agent make faster action selection during training and reduce computational burden slightly, since instead of calculate all $K$ Q-values, we can calculate only one of them. Moreover, by comparing the learning curves of bootstrapped cross DQNs with different $K$s, we can conclude that it is primarily our cross Q-learning rather than policy ensemble that greatly reduces the variance, as with $K = 10$ the variations are much smaller that that with $K = 5$, though policy ensemble further reduces the variance greatly, and during testing phase, our agent can definitely benefit from ensemble of multiple models. Naturally combined crossed Q-learning with dueling and bootstrapped DQN, our model aggregates the merits from all three perspectives.
4.2 Lunar Lander

The task of Lunar Lander in Box2D \cite{zaremba2014learning} is to land the spaceship between the flags smoothly. In each step, the agent is provided with the current state $s$ of the lander in $\mathbb{R}^8$, in which 6 of the dimensions are in continuous space whereas the other 2 are dummy variables in discrete space, and the agent is allowed to make one of the 4 possible actions (i.e., the action space is discrete): fire the left, right, or down throttle so that the lander could obtain a force toward the opposite direction, or do nothing. At the end of each step, the agent receives a reward and moves to a new state $s'$. An episode finishes if the lander rest on the ground at zero speed (receives additional reward of $+100$), or hits the ground.
We built each network with two fully-connected hidden layers, which consist of 128 and 64 neurons, respectively. We train each of the neural networks for 10000 episodes for the LunarLander task, with a much larger replay buffer of size $10^6$. The target network update is set to every 1000 steps for vanilla and double DQN, and learning rate $\alpha = 0.001$ and batch size of 64 are used for Adam optimizer to train all the models. The discount factor $\gamma$ is again 0.99, and exploration rate $\varepsilon$ is set to annealed to 0.02 in the first 100000 steps. And again, $Q$-values for bootstrapped 1024 $(s, a)$-pairs are evaluated and 10 episodes of performance tests with current policy are conducted every 20 training episodes.

In Figure 5, we compared our cross Q-learning algorithms with vanilla DQN and double DQN. With slower learning in the first a few hundreds of episodes due to our experimental design of the learning frequencies, cross DQNs learned much better and more stable policies, while vanilla and double DQN have large variances in both learning curves and performance testing. Figure 5(c) clearly shows that from the beginning, vanilla DQN optimistically gathers the occasional large rewards which are due to the high variance, and produces great overestimations. Double DQN slightly alleviates the problem, but cannot avoid the overestimation effectively. The derived policies from these two networks are then not optimal nor stable. As learning going on, the estimated $Q$-values from both vanilla and double DQN explode, resulting in that the derived policies are no better than random actions. On the other hand, cross DQNs have much lower $Q$-value estimations at the beginning, and the estimates from model with $K = 10$ are even lower than that from model with $K = 5$.

After 1000 episodes, the estimates continue growing until convergence, and their values converge to a same level at about 105. The derived policies are very stable, with total rewards close to 300 and also have little variance. Note that double DQN has lower estimates of $Q$-values than cross DQNs after 8000 episodes of training. The reason is that the corresponding policies from double DQN are much worse, and it does not indicate that double DQN addresses overestimation better.

Comparing Figure 6 and Figure 7, $K = 5$ seems works even better than $K = 10$ for most of the time. Especially for $K = 10$ bootstrapped cross DQN, both the learning curve and the test scores are lower than other cross DQN models. This indicates that it is not always the larger $K$ the better, since cross estimator would induce underestimate bias, and too much underestimation may also hide the real better actions and thus hurt the model performance. In fact, $K = 10$ cross DQN might have too much underestimation at the beginning, which slows down the learning process significantly. But overall, the $K = 10$ bootstrapped cross learning with dueling architecture performs best among all models, including all $K = 5$ cross DQNs. We say that the DQN architectures are too complicated, and the aggregated effect may significantly change the performance of a particular model. Generally speaking, our cross DQNs favor underestimation, which should be much better than overestimation if no unbiased estimation can be achieved, since underestimations do not tend to propagate too much during training, as lower valued actions are avoided by the greedy action selection mechanism. And the bias-variance tradeoff tells us that the overall error can be reduced when the variance of our estimates is greatly decreased, by introducing slight negative bias, this in turn leads to better model performance.

Note that the derived policies from cross DQNs are much more stable in general, and hard to deteriorate. There are at least two reasons for this phenomena. First, cross Q-learning effectively addressed overestimation problem, thus premature policy would be more difficult to derived from cross DQN. In addition, we always ensemble policies using methods such as majority voting during test time, which in general is superior and has a stabilizing effect for action selections. The improved stability comes from larger barrier for altering the decision boundaries, and we could care much less about the early termination as an additional hyperparameter during training. This is yet another advantage of using multiple networks as in cross DQN.

5 Conclusions and Future Work

In this paper, we have presented the cross Q-learning algorithm, an extension to DQN that effectively reduces overestimation, stabilizes training, and improves performance. Cross DQN is a simple
extension that can be easily integrated with other algorithmic improvement such as dueling network and bootstrapped DQN, leads to dramatic performance enhancement. We have both shown in theory and demonstrated in several experiments of classical control problems that the proposed scheme is superior in reducing overestimation and leads to better policies derivation, compared to widely used approaches such as double DQN. Cross learning favors underestimation, the introduced negative bias can greatly help variance reduction. We analyze this effect from the famous bias-variance tradeoff point of view. However, this also indicates that it is not the case the larger $K$ the better model performance in cross DQN. Nevertheless, DQN models tolerate underestimation much more than overestimation, as lower valued actions can be avoided by the greedy action selection mechanism.

It is noted that the computation complexity of cross DQN is generally higher, comparing with that of single network DQNs. We can, however, greatly reduce the complexity given the flexibility provided by our model. In addition, ensemble policies from multiple networks help stabilize the decision space, which can be utilized optionally in stabilizing learning and definitely during testing.

As future work, we would apply cross learning to the state-of-the-art actor-critic methods in continuous control, further reduce the overestimation and stabilize those algorithms. Also, analysis from statistical learning theory could be helpful for us to derive more advanced cross learning strategies, for instance, better bootstrap estimations may be obtained by mimicking the $K$-fold cross validation [27], or from Bayesian perspective [28].

Figure 5: Comparison of vanillar DQN, double DQN and cross DQNs of $K = 5$, $K = 10$ on *LunarLander*.
Moreover, it worth noting that in each step of Q-learning (and more general value-based RL), we utilize $Q$-values in several different places. Now that a set of $K$ different $Q$-functions are applied, we can make different choices for picking particular one to use. We call them generalized cross learning in DQNs, and some existing work can be fell into a particular subclass of our generalized method. The first place that $Q$-values are utilized is when the agent makes decision for choosing an action $a_t$ at time step $t$ while observing $s_t$. We can pick a random $Q$-function for action selection, and this is exactly what bootstrapped DQN [11] does. We say the bootstrapped DQN is a special case of our generalized cross DQN. The next place is at TD update when the target $Q$-values need to be evaluated.
for choosing the next action $a'$, which might not be executed, but is used to evaluate the current target Q-value and derive the $\max$ operator. Recall in Q-learning we use the maximum estimator. Finally, after picking the next action $a'$, its value can be evaluated, again we have choices here for picking a $Q$-function to use. In the version of our cross DQN we presented in this work, which is directly derived from double DQN, we decoupled the selection and evaluation of the next action $a'$, where the current network is used for evaluating $a'$ while another target network is used for selecting $a'$. We could try to do the opposite in certain circumstances, i.e., select $a'$ with the current network and bootstrap another network to evaluate $a'$, which should have the effect of decrease bias but increase variance due to bias-variance tradeoff in general statistical learning scheme. One can further analyze and experiment with other generalized cross Q-learning variants.

References

[1] Sebastian Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning.
[2] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In AAAI, pages 2094–2100, 2016.
[3] Donghun Lee, Boris Defourny, and Warren B Powell. Bias-corrected q-learning to control max-operator bias in q-learning. In 2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL), pages 93–99. IEEE, 2013.
[4] Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. arXiv preprint arXiv:1802.09477, 2018.
[5] Oron Anschel, Nir Baram, and Nahum Shimkin. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 176–185. JMLR. org, 2017.
[6] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In Advances in neural information processing systems, pages 315–323, 2013.
[7] Aaron Defazio, Francis Bach, and Simon Lacoste-Julien. Saga: A fast incremental gradient method with support for non-strongly convex composite objectives. In Advances in neural information processing systems, pages 1646–1654, 2014.
[8] Mark Schmidt, Nicolas Le Roux, and Francis Bach. Minimizing finite sums with the stochastic average gradient. Mathematical Programming, 162(1-2):83–112, 2017.
[9] Zeyuan Allen-Zhu. Katyusha: The first direct acceleration of stochastic gradient methods. The Journal of Machine Learning Research, 18(1):8194–8244, 2017.
[10] Zengqiang Chen, Beibei Qin, Mingwei Sun, and Qinglin Sun. Q-learning-based parameters adaptive algorithm for active disturbance rejection control and its application to ship course control. Neurocomputing, 2019.
[11] Ian Osband, Charles Blundell, Alexander Pritzel, and Benjamin Van Roy. Deep exploration via bootstrapped dqn. In Advances in neural information processing systems, pages 4026–4034, 2016.
[12] Xi-liang Chen, Lei Cao, Chen-qi Li, Zhi-xiong Xu, and Jun Lai. Ensemble network architecture for deep reinforcement learning. Mathematical Problems in Engineering, 2018.
[13] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.
[14] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In International conference on machine learning, pages 1928–1937, 2016.
[15] ML Puterman. Markov decision processes. 1994. Jhon Wiley & Sons, New Jersey, 1994.
[16] Christopher JCH Watkins and Peter Dayan. Q-learning. Machine learning, 8(3-4):279–292, 1992.
[17] Richard S Sutton. Learning to predict by the methods of temporal differences. Machine learning, 3(1):9–44, 1988.
[18] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[19] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[20] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*, 2015.

[21] Malcolm Strens. A bayesian framework for reinforcement learning.

[22] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[23] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. Openai baselines. [https://github.com/openai/baselines](https://github.com/openai/baselines), 2017.

[24] Andrew G Barto, Richard S Sutton, and Charles W Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*, 5:834–846, 1983.

[25] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[26] Erin Catto. Box2d: A 2d physics engine for games, 2011.

[27] Hado Van Hasselt. Estimating the maximum expected value: an analysis of (nested) cross validation and the maximum sample average. *arXiv preprint arXiv:1302.7175*, 2013.

[28] Carlo D’Eramo, Marcello Restelli, and Alessandro Nuara. Estimating maximum expected value through gaussian approximation. In *International Conference on Machine Learning*, pages 1032–1040, 2016.

[29] Satinder Singh, Tommi Jaakkola, Michael L Littman, and Csaba Szepesvári. Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine learning*, 38(3):287–308, 2000.

[30] Hado V Hasselt. Double q-learning. In *Advances in Neural Information Processing Systems*, pages 2613–2621, 2010.
A Estimating the Maximum Expected Values

For Q-learning, the action is selected according to the estimated target Q-values. This is an instance of a more general maximum expected value estimation problem, which is formed as follows. Consider a set of $|A|$ random variables $Q = \{Q_{a_1}, \cdots , Q_{a_\Omega}\}$, we are interested in finding the maximum expected value among the set of variables, which is defined as

$$\max_a \mu_a = \max_a \mathbb{E}[Q_a]$$

while each $\mathbb{E}[Q_a]$ is usually estimated from samples. Let $\Omega_a$ denote the sample space for estimating $Q_a$, for $a \in A$, and we further assume that the samples in $\Omega_a$ are i.i.d. The sample mean $\hat{\mu}_a = \frac{1}{|\Omega_a|} \sum_{x \in \Omega_a} x$ is then an unbiased estimator for $\mathbb{E}[Q_a]$.

Let $f_a : \mathbb{R} \rightarrow \mathbb{R}$ be the probability density function (PDF) for the variable $Q_a$, and $F_a(x) = \int_{-\infty}^{x} f_a(x) dx$ be the cumulative density function (CDF). The maximum expected value is then

$$\max_a \mathbb{E}[Q_a] = \max_a \int_{-\infty}^{\infty} x f_a(x) dx. \quad (10)$$

### A.1 (Single) Maximum Estimator

The most straightforward way to approximate $\max_a \mathbb{E}[Q_a]$ is to take the maximum over the sample mean for each $a$, i.e., $\max_a \mathbb{E}[Q_a] \approx \max_a \hat{\mu}_a$. Note that the sample means $\hat{\mu}_a$ are unbiased estimates of the true means, thus $\max_a \hat{\mu}_a$ is an unbiased estimate for $\mathbb{E}[\max_a \mu_a] = \int_{-\infty}^{\infty} x f_{\max}(x) dx$, however, it is a biased estimate for $\max_a \mathbb{E}[Q_a]$.

Consider its CDF $F_{\max}^\mu = P\{\max_a \hat{\mu}_a \leq x\} = \Pi_a P\{\mu_a \leq x\} = \Pi_a F_a^\mu(x)$, we can write

$$\mathbb{E}[\max_a \hat{\mu}_a] = \int_{-\infty}^{\infty} x \frac{d}{dx} \Pi_a F_a^\mu(x) dx = \sum_a \int_{-\infty}^{\infty} x f_a^\mu(x) \Pi_a \neq a F_a^\mu(x) dx. \quad (11)$$

Comparing equations $(10)$ and $(11)$, clearly $\max_a \mathbb{E}[Q_a]$ and $\mathbb{E}[\max_a \hat{\mu}_a]$ are not equivalent. Moreover, the product term $\Pi_a \neq a F_a^\mu(x)$ in the integral introduces positive bias (since CDFs are monotonically increasing, the sum of their derivatives will be positive, the integral value would be monotonically increasing while more product terms are added). Therefore, we say that the expected value of the single estimator for the maximum is an overestimation of the maximum expected value.

### A.2 Double Estimator

Consider the case that we use two sets of estimators $\hat{\mu}^A = \{\hat{\mu}_{a_1}^A, \cdots , \hat{\mu}_{a_\Omega}^A\}$ and $\hat{\mu}^B = \{\hat{\mu}_{a_1}^B, \cdots , \hat{\mu}_{a_\Omega}^B\}$, in which each $\hat{\mu}_{a_i}^A$ is estimated from a set of samples independent of the one to estimate $\hat{\mu}_{a_i}^B$, i.e., $\hat{\mu}_{a_i}^A = \frac{1}{|\Omega_{a_i}^A|} \sum_{x \in \Omega_{a_i}^A} x$, $\hat{\mu}_{a_i}^B = \frac{1}{|\Omega_{a_i}^B|} \sum_{x \in \Omega_{a_i}^B} x$, and $\Omega_{a_i}^A \cap \Omega_{a_i}^B = \emptyset$. For all $a_i$, both $\hat{\mu}_{a_i}^A$ and $\hat{\mu}_{a_i}^B$ are unbiased estimator for $\mathbb{E}[Q_{a_i}]$, assuming all the samples in both sets are independently drawn from the population. That means $\mathbb{E}[\hat{\mu}_{a_i}^A] = \mathbb{E}[\hat{\mu}_{a_i}^A]$ for all $a_i$, including $a^*_B = \arg \max \hat{\mu}_{a_i}^B$, the action that maximizes the sample mean $\hat{\mu}_{a_i}^B$. Therefore, $\hat{\mu}_{a_i}^A$ can be used to estimate $\max_a \mathbb{E}[\hat{\mu}_{a_i}^A]$ as well as $\max_a \mathbb{E}[Q_{a_i}]$, i.e.,

$$\max_a \mathbb{E}[Q_{a_i}] = \max_a \mathbb{E}[\hat{\mu}_{a_i}^A] \approx \hat{\mu}_{a_i}^A.$$

The same argument holds for the opposite way considering the best action over $\Omega^A$ and the sample mean $\hat{\mu}^B_{a^*_A}$. The selection of $a^*$ means that all other $a$ gives lower estimation, i.e., $P(a = a^*) = \Pi_a \neq a^* P(\mu_a < \mu_{a^*})$. Let $f_{a_i}^A$ and $F_{a_i}^A$ be the PDF and CDF of $\hat{\mu}_{a_i}^A$, respectively. Then

$$P(a = a^*) = \int_{-\infty}^{\infty} P(\mu_a = x) \Pi_a \neq a^* P(\mu_a < x) dx = \int_{-\infty}^{\infty} x f_{a_i}^A(x) \Pi_a \neq a F_{a_i}^A(x) dx.$$  

The expected value of double estimator is a weighted sum of the sample means’ expected values in one sample space, weighted by the probability of each sample mean to be the maximum in the other
We can easily extend the double estimator to a more general case, in which instead of using two estimators, suppose now we have $K$ independent sets of estimators $\mu_1, \cdots, \mu_K$. We call it the cross estimator. The double estimator can be seen as a special case of the more general cross estimator. Similar argument as analyzing the double estimator can be applied here, for any two estimators $\mu$ and $\mu'$, as

$$\max_a \mathbb{E}[Q_a] = \max_a \mathbb{E}[\hat{\mu}_a^A] \approx \hat{\mu}_a^A.$$  

The cross estimator finally uses a convex combination of the $K$ sample means,

$$\sum_a P(a = a^*) \mathbb{E}[\hat{\mu}_a^A] = \sum_a \mathbb{E}[\hat{\mu}_a^A] \int_{-\infty}^{\infty} x f_a(x) \Pi_{a' \neq a} F_a^A(x) dx,$$

thus also underestimates the maximum expected value.

**Theorem 1.** [27] There does not exist an unbiased estimator for maximum expected values.

### B Convergence in the Limit

In this section, we first present a lemma which claims the convergence of SARSA from [29], and then use it to prove convergence of cross Q-learning. Note that this part heavily borrows the proof of the convergence of double Q-learning [30], but serves as a more general case.

**Lemma 2.** [29]. Consider a stochastic process $(\alpha_t, \Delta_t, F_t), t \geq 0$, where $\alpha_t, \Delta_t$ and $F_t : X \to \mathbb{R}$ satisfy the equation

$$\Delta_{t+1}(x) = (1 - \alpha_t(x)) \Delta_t(x) + \alpha_t(x) F_t(x), \quad \text{where } x \in X, t = 0, 1, 2, \cdots$$

Let $P_t$ be a sequence of increasing $\sigma$-fields such that $\alpha_0$ and $\Delta_0$ are $P_0$-measurable and $\alpha_t, \Delta_t$ and $F_t$ are $P_t$-measurable, for $t = 1, 2, \cdots$. $\Delta_t$ converges to zero with probability one (w.p.1) if the following hold:

1. the set $X$ is finite.
2. $0 \leq \alpha_t(x) \leq 1$, $\sum_t \alpha_t(x) = \infty$, and $\sum_t \alpha_t^2(x) < \infty$ w.p. 1.
3. $||\mathbb{E}[F_t | P_t]|| \leq \kappa ||\Delta_t|| + c_t$, where $\kappa \in [0, 1]$ and $c_t$ converges to zero w.p. 1.
4. $\text{Var}(F_t | P_t) \leq K (1 + ||\Delta_t||)^2$, where $K$ is a constant.

in which $|| \cdot ||$ denotes the maximum norm.

**Theorem 3.** In a given ergodic MDP, suppose that we have a set of $K$ Q-value functions, $Q^1, Q^2, \cdots, Q^K$, as updated by cross Q-learning, will converge to the optimal value function $Q^*$ with probability 1, if the following conditions hold:

1. The MDP is finite, i.e., $|S \times A| < \infty$.
2. $\gamma \in [0, 1)$.
3. The Q-values are stored in a lookup table.
4. Each state-action pair is visited infinitely often.
5. Each \( Q^k \) receives an infinite number of updates, for all \( k = 1, \ldots, K \).

6. \( 0 \leq \alpha_t(s, a) \leq 1, \sum \alpha_t(s, a) = \infty \), and \( \sum \alpha_t^2(x) < \infty \) w.p. 1. Moreover, \( \alpha_t(s, a) = 0, \forall (s, a) \neq (s_t, a_t) \).

7. \( \text{Var}(R(s, a)) < \infty, \forall s, a \)

**Proof.** Let \( k, j \in \{1, \ldots, K\} \) are randomly picked with \( k \neq j \). Apply Lemma 2 by letting \( P_t = \{Q^1_t, Q^2_t, \ldots, Q^K_t\}, s_0, a_0, \alpha_0, r_1, s_1, \ldots, s_t, a_t\}, X = S \times A, \Delta_t = Q^k_t - Q^j_t \), and \( F_t(s_t, a_t) = r_t + \gamma Q^k_t(s_{t+1}, a^*) - Q^j_t(s_t, a_t) \), where \( a^* = \arg \max_a Q^k(s, a) \). The first two conditions of Lemma 2 hold immediately from conditions 1 and 6 of Lemma 3 respectively. And since condition 7 of Theorem 3 gives us the bounds for the variance of rewards, the fourth condition of Lemma 2 holds.

To show the third condition of Lemma 2, we write
\[
F_t(s_t, a_t) = r_t + \gamma Q^k_t(s_{t+1}, a^*) - Q^j_t(s_t, a_t)
\]
which we define \( F'_t = r_t + \gamma Q^k_t(s_{t+1}, a^*) \) as the estimated \( Q \)-value for \((s, a)\) under the standard (single) Q-learning. Since the convergence of standard Q-learning in finite MDP is well-known, i.e., \( \text{Cross estimation converges in the limit, given finite and ergodic MDP.} \)

To show the third condition of Lemma 2, we write
\[
F_t(s_t, a_t) = F'_t(s_t, a_t) + \gamma c_t
\]
in which we define \( F'_t = r_t + \gamma Q^k_t(s_{t+1}, a^*) \) as the estimated \( Q \)-value for \((s, a)\) under the standard (single) Q-learning. While the convergence of standard Q-learning in finite MDP is well-known, i.e., \( \mathbb{E}[F'_t|P_t] \leq \gamma \|\Delta_t\| \), it suffices to show that \( c_t = Q^j_t(s_{t+1}, a^*) - Q^k_t(s_{t+1}, a^*) \to 0 \), so that the condition on the expected contraction of \( F_t \) holds.

Let \( \Delta^k_t(s_t, a_t) = Q^k_t(s_t, a_t) - Q^j_t(s_t, a_t) \). It is important to note that at each step, the choice of \( j, k \) is random, all with equal probability \( p_{jk} = p_{kj} = 1/K \). Consider the case that \( Q^k \) is updated using \( Q^j \) at time \( t \), the update of \( \Delta^k_t \) is
\[
\Delta^k_{t+1}(s_t, a_t) = r_t + \gamma Q^k_t(s_{t+1}, a^*) - Q^j_t(s_t, a_t)
\]
Or, again with probability \( p_{kj} = 1/K \), we use \( Q^k \) to update \( Q^j \), in this case we have
\[
\Delta^j_{t+1}(s_t, a_t) = r_t + \gamma Q^k_t(s_{t+1}, a^*) - Q^j_t(s_t, a_t)
\]
Otherwise, this particular \((j, k)\) pair is not selected at time \( t \), and the update of \( \Delta^k \) is then zero. Then
\[
\mathbb{E}[\Delta^k_{t+1}|P_{t+1}] = (1 - 2p_{jk})\mathbb{E}[\Delta^k_t|P_t]
\]
Clearly \( \mathbb{E}[\Delta^k_{t+1}|P_{t+1}] \) converges to 0 since the coefficient on the R.H.S. is less than 1. Therefore we have shown that \( c_t \to 0 \) since \( \Delta^k_{t+1} \to 0 \) in expectation and \( j, k \) are randomly chosen. It in turn ensures condition 3 of Lemma 2 holds, which completes our proof.

Finally, we rephrase Theorem 3 as follows:

**Proposition 4.** Cross estimation converges in the limit, given finite and ergodic MDP.