DeepMellow: Removing the Need for a Target Network in Deep Q-Learning

Seungchan Kim, **Kavosh Asadi**, **Michael Littman** and **George Konidaris**
Brown University Department of Computer Science
{seungchan_kim, kavosh}@brown.edu, {mlittman, gdk}@cs.brown.edu

Abstract

Deep Q-Network (DQN) is an algorithm that achieves human-level performance in complex domains like Atari games. One of the important elements of DQN is its use of a target network, which is necessary to stabilize learning. We argue that using a target network is incompatible with online reinforcement learning, and it is possible to achieve faster and more stable learning without a target network when we use Mellowmax, an alternative softmax operator. We derive novel properties of Mellowmax, and empirically show that the combination of DQN and Mellowmax, but without a target network, outperforms DQN with a target network.

1 Introduction

Reinforcement learning (RL) is a general framework to study agents that make sequential decisions in an environment to maximize long-term utility. Recent breakthroughs in reinforcement learning have shown that reinforcement learning algorithms can train agents in high-dimensional complex domains when combined with deep neural networks. Deep Q-Network (or simply DQN) [Mnih *et al.*, 2015] was the first algorithm that successfully instantiated this combination; it yields human-level performance in high-dimensional large-scale domains like Atari video games [Bellemare *et al.*, 2013].

An important component of DQN is the use of a target network, which was introduced to stabilize learning. In Q-learning, the agent updates the value of executing an action in the current state, using the values of executing actions in a successive state. This procedure often results in an instability because the values change simultaneously on both sides of the update equation. A target network is a copy of the estimated value function that is held fixed to serve as a stable target for some number of steps.

However, a key shortcoming of target networks is that they move us farther from online reinforcement learning, a desired property in the reinforcement learning community [Sutton and Barto, 1998; van Seijen and Sutton, 2014]. The presence of a target network can also slow down learning due to delayed value function updates.

We propose an approach that reduces the need for a target network in DQN while still ensuring stable learning and good performance in high-dimensional domains. Our approach uses a recently proposed alternative softmax operator, Mellowmax [Asadi and Littman, 2017]. This operator has been shown to ensure convergence in learning and planning, has an entropy-regularization interpretation [Nachum *et al.*, 2017; Fox *et al.*, 2016; Neu *et al.*, 2017], and facilitates convergent off-policy learning even with non-linear function approximation [Dai *et al.*, 2018]. We additionally show that Mellowmax is convex regardless of the value of the temperature parameter, monotonically non-decreasing with respect to its temperature parameter, and that it alleviates the well-known overestimation problem associated with the max operator [van Hasselt, 2010]. These properties suggest that the use of Mellowmax in DQN mitigates the main cause of instability in online reinforcement learning with deep neural networks.

We test the performances of DeepMellow, the combination of Mellowmax operator and DQN, in two control domains (Acrobot and Lunar Lander) and two Atari domains (Breakout and Seaquest). Our empirical results show that DeepMellow achieves more stability than a version of DQN with no target network. We also show that, DeepMellow, which has no target network, learns faster than the original DQN with a target network, in these domains.

2 Background

RL problems are typically formulated as Markov Decision Processes (MDP) [Bellman, 1957], described by the tuple ($\mathcal{S}$, $\mathcal{A}$, $\mathcal{T}$, $\mathcal{R}$, γ). Here, $\mathcal{S}$ denotes the set of states and $\mathcal{A}$ denotes the set of actions. The function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]$ specifies the transition dynamics of the process; specifically, the probability of moving to a state s' after taking action a in state s is denoted by $T(s, a, s')$. Similarly, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is the reward function, with $\mathcal{R}(s, a, s')$ being the expected reward upon moving to state s' when taking action a in state s. Finally, the discount rate $\gamma \in [0, 1)$ determines the importance of immediate rewards relative to rewards received in the long term.

RL agents typically seek to find a policy $\pi : S \mapsto \Pr(A)$ that achieves high long-term reward. More formally, the long-term utility (or value) of taking an action in a state is defined as:

$$Q^{\pi}(s,a) := \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t, s_{t+1}) | s_0 = s, a_0 = a, \pi].$$

We can define the optimal Q value, meaning the value of a state–action pair under the best policy π^*, as follows:

$$Q^*(s,a) := \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, a_t, s_{t+1}) | s_0 = s, a_0 = a, \pi^*].$$

A fundamental result in MDPs is that Q^* can be written recursively:

$$Q^*(s,a) = \sum_{s' \in S} \mathcal{T}(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')] ,$$

in what are known as the Bellman equations [Bellman, 1957]. A generalization [Littman and Szepesvári, 1996] replaces the max operator with a generic operator $\bigotimes$ as follows:

$$Q(s,a) = \sum_{s' \in S} \mathcal{T}(s,a,s')[R(s,a,s') + \gamma \bigotimes_{a'} Q(s',a')].$$

The operator can be thought of as an action selector, which summarizes values over actions; different choices of the operator can lead to different solutions for Q.

2.1 Deep Q-Network

In its basic form, given a sample $\langle s, a, r, s' \rangle$, online tabular Q-learning seeks to improve its approximate solution to the fixed-point of the Bellman equation by performing the following simple update:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\big(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big).$$

In domains with large state spaces, rather than learning a value for each individual state–action pair, algorithms must represent the Q-function using a parameterized function approximator. DQN, for example, uses a deep convolutional neural network [LeCun *et al.*, 2015] parameterized by weights θ. In this case, the value-function estimate is updated as follows:

$$\theta \leftarrow \theta + \alpha\big(r + \gamma \max_{a'} Q(s',a';\theta) - Q(s,a;\theta)\big) \nabla_\theta Q(s,a;\theta).$$

DQN employs two additional techniques, namely experience replay and the use of a separate target network, to stabilize learning and improve performance [Mnih *et al.*, 2015]. Experience replay stores samples $\langle s, a, r, s' \rangle$ in a buffer, randomly samples a minibatch, and performs the above update over that minibatch. This approach helps reduce the correlation between consecutive samples, which can otherwise negatively effect gradient-based methods.

The second modification, the target network, maintains a separate weight vector θ^- to create a temporal gap between the target action-value function and the action-value function that updates continually. The update is changed as follows:

$$\theta \leftarrow \theta + \alpha\big(r + \gamma \max_{a'} Q_T(s',a';\theta^-) - Q(s,a;\theta)\big) \nabla_\theta Q(s,a;\theta),$$

where the separate weight vector θ^- is synchronized with θ after some period of time chosen as a hyper-parameter. Using a separate target network makes divergence unlikely, because it adds a delay between the time that Q values are updated and the time that the target Q_T values are updated.

2.2 Motivation: Removing the Target Network

We aim to remove the target network for the following reasons. First, the target network in DQN violates online reinforcement learning, and hinders fast learning. Online learning enables real-time learning with streams of incoming data, continually updating the value functions without delays [Sutton and Barto, 1998; Rummery and Niranjan, 1994]. (Note that both using experience replay and a target network are deviations from online learning, but our focus is solely on the target network.) By eliminating the target network, we can remove the delays in the update of value functions, and thus support faster learning, as demonstrated in our experiments.

Secondly, having a separate target network doubles the memory required to store neural network weights. Thus, removing the target network contributes to the better allocation of memory resources. Lastly, we aim to develop simpler learning algorithms since they are easier to implement in practice and understand in theory. A target network is an extra complication added to Q-learning to make it work; removing that complication results in a simpler, and therefore, better algorithm.

3 New Properties of the Mellowmax Operator

Softmax operators have been found useful across many disciplines of science, including optimization [Boyd and Vandenberghe, 2004], electrical engineering [Safak, 1993], game theory [Gao and Pavel, 2017], and experimental psychology [Stahl II and Wilson, 1994].

In reinforcement learning, softmax has been used in the context of action selection to trade off exploration (trying new actions) and exploitation (trying good actions), owing to its cheap computational complexity relative to more principled approaches like optimism under uncertainty [Brafman and Tennenholtz, 2002; Strehl *et al.*, 2006] or Bayes-optimal decision making [Dearden *et al.*, 1998]. We use softmax in the context of value-function optimization, where we focus on the following softmax operator:

$$mm_\omega(x) := \frac{\log(\frac{1}{n}\sum_{i=1}^{n} \exp(\omega x_i))}{\omega}.$$

This recently introduced operator, called Mellowmax [Asadi and Littman, 2017], can be thought of as a smooth approximation of the max operator. Mellowmax can also be incorporated into the Bellman equation as follows:

$$Q(s,a) = \sum_{s' \in S} \mathcal{T}(s,a,s')[R(s,a,s') + \gamma mm_\omega Q(s,\cdot)].$$

In contrast to the more familiar Boltzmann softmax, it yields convergent behavior due to its non-expansion property. Recent research has also found an appealing entropy-regularization characterization of this operator [Nachum *et al.*, 2017; Fox *et al.*, 2016; Neu *et al.*, 2017].

Mellowmax has several interesting properties. In addition to being a non-expansion, the parameter ω offers an interpolation between max ($\omega \to \infty$) and mean ($\omega \to 0$) [Asadi and Littman, 2017]. In the next subsection, we develop two novel mathematical properties of this operator: convexity and monotonic non-decrease.

3.1 Convexity and Monotonic Non-decrease

Claim 1: For any $\omega \geq 0$, $mm_\omega(x)$ is convex.

Proof: Our proof generalizes the proof by Boyd and Vandenberghe [2004]. Note that $\nabla^2 mm_\omega(x) = \frac{\omega}{(\mathbf{1}^\top z)^2}((\mathbf{1}^\top z)diag(z) - zz^\top)$ where $z_i = e^{\omega x_i}$. They showed that $\frac{1}{(\mathbf{1}^\top z)^2}((\mathbf{1}^\top z)diag(z) - zz^\top) \geq 0$, and thus, convexity holds as long as temperature $\omega \geq 0$.

Claim 2: For any $\omega \geq 0$ and any x, $mm_\omega(x)$ is non-decreasing with respect to ω.

Proof: Let $\omega_2 > \omega_1 > 0$.
We want to show that $mm_{\omega_2}(x) \geq mm_{\omega_1}(x)$:

$$\begin{aligned}
mm_{\omega_2}(x) &= \frac{\log \frac{1}{n}\sum_i e^{\omega_2 x_i}}{\omega_2} \\
&= \frac{\log \frac{1}{n}\sum_i e^{\omega_1 x_i \frac{\omega_2}{\omega_1}}}{\omega_2} \\
&= \frac{\log \frac{1}{n}\sum_i e^{(\omega_1 x_i)^{(\frac{\omega_2}{\omega_1})}}}{\omega_2}.
\end{aligned}$$

We now utilize the convexity of the function $f(z) = z^p$ for $z > 0$ and $p > 1$. For any such convex function, Jensen's inequality holds:

$$\frac{1}{n}\sum_i f(y_i) \geq f(\frac{1}{n}\sum_i y_i).$$

Substituting f with $y_i = e^{\omega_1 x_i}$ and $p = \frac{\omega_2}{\omega_1}$ we get:

$$\frac{1}{n}\sum_i e^{(\omega_1 x_i)^{(\frac{\omega_2}{\omega_1})}} \geq (\frac{1}{n}\sum_i e^{(\omega_1 x_i)})^{(\frac{\omega_2}{\omega_1})}.$$

Using the above inequality, we finally get:

$$\begin{aligned}
mm_{\omega_2}(x) &= \frac{\log \frac{1}{n}\sum_i e^{(\omega_1 x_i)^{(\frac{\omega_2}{\omega_1})}}}{\omega_2} \\
&\geq \frac{\log(\frac{1}{n}\sum_i e^{(\omega_1 x_i)})^{(\frac{\omega_2}{\omega_1})}}{\omega_2} \\
&= \frac{(\frac{\omega_2}{\omega_1})\log(\frac{1}{n}\sum_i e^{(\omega_1 x_i)})}{\omega_2} \\
&= \frac{\log(\frac{1}{n}\sum_i e^{(\omega_1 x_i)})}{\omega_1} = mm_{\omega_1}(x),
\end{aligned}$$

allowing us to conclude that $mm_\omega(x)$ is a non-decreasing function of ω.

4 DeepMellow

We now introduce DeepMellow, a new algorithm that uses the Mellowmax operator in DQN. We first present the theoretical basis of our algorithm (overestimation bias analysis of the Mellowmax operator), and then explain the details of the algorithm.

4.1 Alleviation of Overestimation

Previous work [van Hasselt, 2010] showed that standard Q-learning, which uses the $\max$ operator, suffers from an overestimation problem: note that due to Jensen's inequality and the convexity of max,

$$\mathbb{E}[\max \widehat{Q}] \geq \max \mathbb{E}[\widehat{Q}] \,.$$

Q-learning can overestimate the target due to noise in the estimator $\widehat{Q}$. In practice, this gap can be quite large. We hypothesize that using a separate target network keeps the target $\widehat{Q}$ constant for a while, and, in effect, removes the randomness from the target. In this case, both sides of the above inequality will be the same quantity: $\max \widehat{Q}$.

By the same argument, and as a corollary of the convexity argument of Mellowmax (Claim 1), Q-learning with Mellowmax also suffers from this overestimation problem. However, the magnitude of the overestimation is reduced by lowering the temperature parameter ω, as we argue next.

For this analysis, we assume that $\widehat{Q}$ is an unbiased estimate of Q as assumed by previous work. We further assume that $\widehat{Q}$ values are uncorrelated. We wish to find the following gap:

$$\text{bias}\big(mm_\omega(\widehat{Q})\big) = \mathbb{E}[mm_\omega(\widehat{Q})] - mm_\omega(Q).$$

We begin with a second-order Taylor expansion of Mellowmax as a good approximation for convex functions:

$$\begin{aligned}
&mm_\omega(y) - mm_\omega(x) \\
&\approx \nabla mm_\omega(x)^\top(y-x) + \frac{1}{2}(y-x)^\top \nabla^2 mm_\omega(x)(y-x).
\end{aligned}$$

Replacing x and y with Q and $\widehat{Q}$, we get:

$$\begin{aligned}
&mm_\omega(\widehat{Q}) - mm_\omega(Q) \\
&= \nabla mm_\omega(Q)^\top(\widehat{Q}-Q) + \frac{1}{2}(\widehat{Q}-Q)^\top \nabla^2 mm_\omega(Q)(\widehat{Q}-Q).
\end{aligned}$$

Taking the expectation from both sides, we notice that the left hand side is exactly the quantity we are looking for, namely the bias under Mellowmax:

$$\begin{aligned}
&\mathbb{E}[mm_\omega(\widehat{Q}) - mm_\omega(Q)] \\
&\quad = \mathbb{E}[mm_\omega(\widehat{Q})] - mm_\omega(Q) = \text{bias}\big(mm_\omega(\widehat{Q})\big).
\end{aligned}$$

Thus:

$$\begin{aligned}
\text{bias}\big(mm_\omega(\widehat{Q})\big) &= \mathbb{E}[\nabla mm_\omega(Q)^\top(\widehat{Q}-Q)] \\
&+ \frac{1}{2}\mathbb{E}[(\widehat{Q}-Q)^\top \nabla^2 mm_\omega(Q)(\widehat{Q}-Q)] \\
&= \nabla mm_\omega(Q)^\top \mathbb{E}[\widehat{Q}-Q] \\
&+ \frac{1}{2}\mathbb{E}[(\widehat{Q}-Q)^\top \nabla^2 mm_\omega(Q)(\widehat{Q}-Q)] \\
&= \frac{1}{2}\mathbb{E}[(\widehat{Q}-Q)^\top \nabla^2 mm_\omega(Q)(\widehat{Q}-Q)] \\
&= \frac{1}{2}\sum_i \frac{\partial^2 mm_\omega(Q)}{\partial(Q_i)^2}\mathbb{E}[(\widehat{Q}_i - Q_i)^2] \\
&= \frac{1}{2}\sum_i \frac{\partial^2 mm_\omega(Q)}{\partial(Q_i)^2}\text{Var}[\widehat{Q}_i].
\end{aligned}$$

Algorithm 1 DeepMellow

```
Procedure DeepMellow($\omega$)
Initialize experience replay memory $D$
Initialize action-value function $Q$ with random $\theta$
~~Initialize target function $Q_T$ with initial weights $\theta^- = \theta$~~
for episode = 1 to $M$ do
    Initialize sequence $s_1$ and preprocess $\phi_1 = \phi(s_1)$
    for t= 1 to T do
        Select a random action $a_t$ with probability $\epsilon$
        Otherwise, select $a_t = \arg\max_a Q(\phi_t, a; \theta)$
        Execute action $a_t$ and observe reward $r_t$
        Set a new state $s_{t+1}$ and preprocess $\phi_{t+1}$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
        Sample random batch $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
        if episode terminates at step $j + 1$ then
            set $y_j = r_j$
        else
            ~~set $y_j = r_j + \gamma \max Q_T(\phi_{j+1}, a'; \theta^-)$~~
            set $y_j = r_j + \gamma \underset{a'}{mm_\omega} Q(\phi_{j+1}, a'; \theta)$
        end if
        Perform gradient descent on $\{y_j - Q(\phi_j, a_j; \theta)\}^2$
        ~~Every C steps, copy $Q_T = Q$~~
    end for
end for
    (Crossed-out texts denote modifications.)
```

We make two observations about the bias quantity. First, the amount of bias relates to the variance of the estimator. If the estimator $\widehat{Q}$ can perfectly estimate Q with one sample (no variance), then there will also be no bias. Second, note that

$$\begin{aligned}\frac{\partial^2 mm_\omega(Q)}{\partial(Q_i)^2} &= \frac{\omega e^{\omega Q_i} \sum_i e^{\omega Q_i} - w e^{\omega Q_i} e^{\omega Q_i}}{\sum_i e^{\omega Q_i} \sum_i e^{\omega Q_i}} \\ &= \omega x - \omega x^2 \\ &= \omega x(1-x) \text{ where } w > 0 \text{ and } 0 < x < 1.\end{aligned}$$

Here, x denotes $e^{\omega Q_i} / \sum_i e^{\omega Q_i}$. We see that the bias is always positive and monotonically increases with ω.

4.2 DeepMellow

The target network is just a copy of the action-value function that is updated with a delay, and it can serve as a stable target between updates. We note that the analysis in the previous subsection provides an explanation for how a target network improves Q-learning—keeping the target network fixed reduces the variance of estimator $\widehat{Q}$. As we showed above, the variance of the estimator is connected to the amount of bias, so using a target network results in a bias reduction. Our analysis suggests that Mellowmax reduces overestimation bias, and thus reduces the need for a target network.

DeepMellow replaces the max operator in DQN with the Mellowmax operator, as in the framework of generalized MDPs [Littman and Szepesvári, 1996]. As described in Algorithm 1, DeepMellow further differs from DQN, as it does not use a separate target action-value function Q_T, and thus does not copy the action-value function every C steps.

Parameters	Acrobot	Lunar Lander	Atari
learning rate	10^{-3}	10^{-4}	0.00025
neural network	MLP	MLP	CNN
layers	3	3	4
neurons per layer	300	500	–
update frequency	100	200	10000
number of runs	100	50	5
processing unit	CPU	GPU	GPU

Table 1: Experimental details for each domain.

5 Experiments and Results

We tested DeepMellow in two control domains (Acrobot, Lunar Lander) and two Atari games (Breakout, Seaquest). We used multilayer perceptrons (MLPs) for the control domains, and convolutional neural networks (CNNs) for Atari games. The parameters and neural network architectures for each domain are summarized in Table 1. For the Atari game domains, we used the same CNN as Mnih *et al.* [2015].

Target network update frequency is a crucial factor in our experiments. In DQN, while the real action-value function is updated every iteration, the target network is only updated every C iterations—we call C the target network update frequency.[1] When $C > 1$, the target network is updated with a delay. On the other hand, setting $C = 1$ means that, after every update, the target network is copied from the real action-value function immediately; we will use $C = 1$ as a synonym for eliminating the separate target network.

Choice of Temperature Parameter ω

The temperature parameter ω of DeepMellow is an additional parameter that should be tuned. Both too large and too small ω values are bad—as ω increases, Mellowmax behaves more like a max operator, so there is no advantage to using it. With an ω close to zero, Mellowmax behaves like a mean operator, resulting in persistent random behaviors. Intermediate values yield better performances than the two extremes, but the beginning and end of the "reasonable performance range" of the parameter varies with domain. To find the optimal ranges of the ω parameter for each domain, we used a grid search method, as we did for other hyperparameters. We empirically found that simpler domains (Acrobot, Lunar Lander) require relatively smaller ω values while large-scale Atari domains require larger values. For Acrobot and Lunar Lander, our parameter search set was $\omega \in \{1, 2, 5, 10\}$. For Breakout and Seaquest, we tested $\omega \in \{100, 250, 1000, 2000, 3000, 5000\}$ and $\omega \in \{10, 20, 30, 40, 50, 100, 200\}$, respectively. An adaptive approach to choosing this parameter can benefit DeepMellow, but we leave this direction for future work.

5.1 DeepMellow vs DQN without a Target Network

We first compared DeepMellow and DQN in the absence of a target network (or target network update frequency $C = 1$). The control domain results are shown in the Figure 1

[1] Though we use the term "frequency", "period" might be more apt.

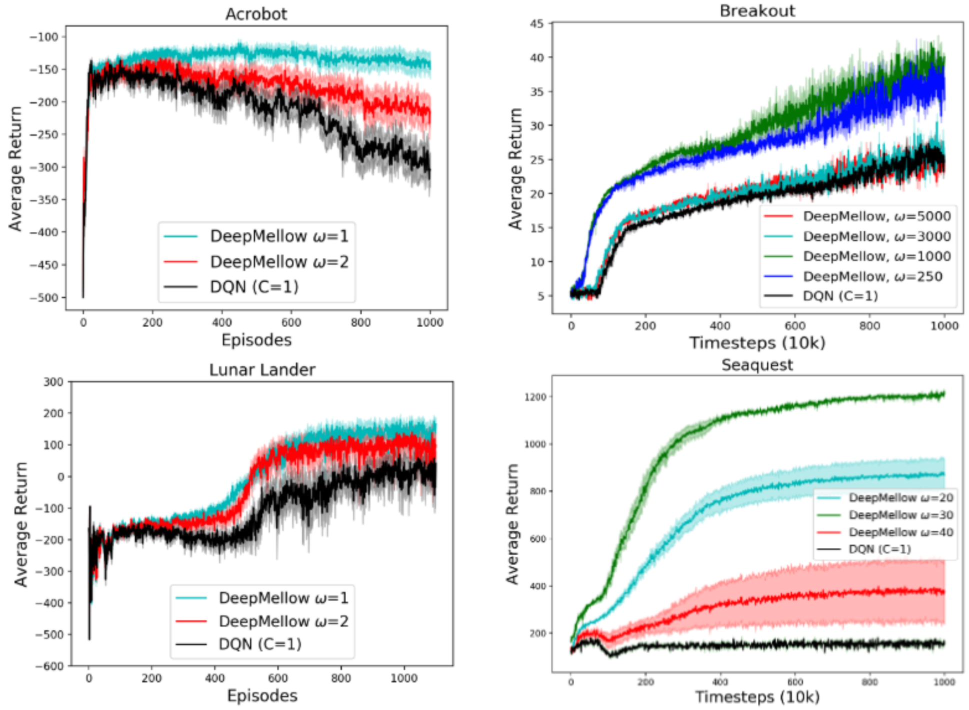

Figure 1: The performance of DeepMellow (no target network) and DQN (no target network) in control domains (left) and Atari games (right). DeepMellow outperforms DQN in all domains, in the absence of target network. Note that the best performing temperature ω values vary across domains.

(left). In Acrobot, DeepMellow achieves more stable learning than DQN—without a target network, the learning curve of DQN goes upward fast, but soon starts fluctuating and fails to improve towards the end. By contrast, DeepMellow (especially with temperature parameter $\omega = 1$) succeeds. Similar results are observed in Lunar Lander. DeepMellow ($\omega \in \{1, 2\}$) achieves more stable learning and higher average returns than DQN without a target network.

In order to quantify the performances in each domain, we computed the sum of areas under the curves for DeepMellow and DQN (for the first 1000 episodes; y-axis lower bound is -500). Setting the areas under the curves of DeepMellow ($\omega = 1$) as 100, the areas under the curves of DeepMellow ($\omega = 2$) and DQN were 88.9% and 78.7%, respectively, in Acrobot. Similarly, in Lunar Lander domain, the areas under the curves of DeepMellow($\omega = 2$) and DQN were 93.4% and 79.2% of that of DeepMellow ($\omega = 1$). In both domains, DeepMellow($\omega = 1$) performed best, followed by DeepMellow($\omega = 2$), and then by DQN.

We also compared the performances of DeepMellow and DQN in two Atari games, Breakout and Seaquest. We chose these two domains because the effects of having a target network are known to be different in each domain [Mnih *et al.*, 2015]. In Breakout, the performance of DQN does not differ significantly with and without a target network. On the other hand, Seaquest is a domain that shows a significant performance drop when the target network is absent. Thus, these two domains are two contrasting examples for us to see whether DeepMellow obviates the need for a target network.

Figure 1 (right) shows the performances of DeepMellow and DQN in these games. We used similar methods to quantify their performances (computing the areas under the curves), and the results were as follows: in Breakout, compared with the areas under the curves of DeepMellow($\omega = 1000$), those of DeepMellow($\omega = 250, 3000, 5000$) and DQN were 93.1%, 68.5%, 67.1%, and 64.5%, respectively. In Seaquest, the performance gaps widened as expected: the areas under the curves of DeepMellow ($\omega = 20, 40$) and DQN were 69.5%, 31.1%, and 14.9% of that of DeepMellow($\omega = 30$).

DeepMellow performed better than DQN without a target network in both Breakout and Seaquest; especially in Seaquest, the performance gap was substantial. Also, note that there are intermediate ω values that yield best performances of DeepMellow in each domain. ($\omega = 30$ is better than $\omega = 20$ or $\omega = 40$ in Seaquest; $\omega = 1000$ is better than $\omega = 250$ or $\omega = 3000, 5000$ in Breakout.) These results are consistent with the property of the Mellowmax operator that both too large (behaving like max operator) and too small (entailing random action-selection) ω values degrade the performance.

5.2 DeepMellow vs DQN with a Target Network

In the previous section, we showed that DeepMellow outperforms DQN without a target network. The next question that naturally arises is whether DeepMellow without

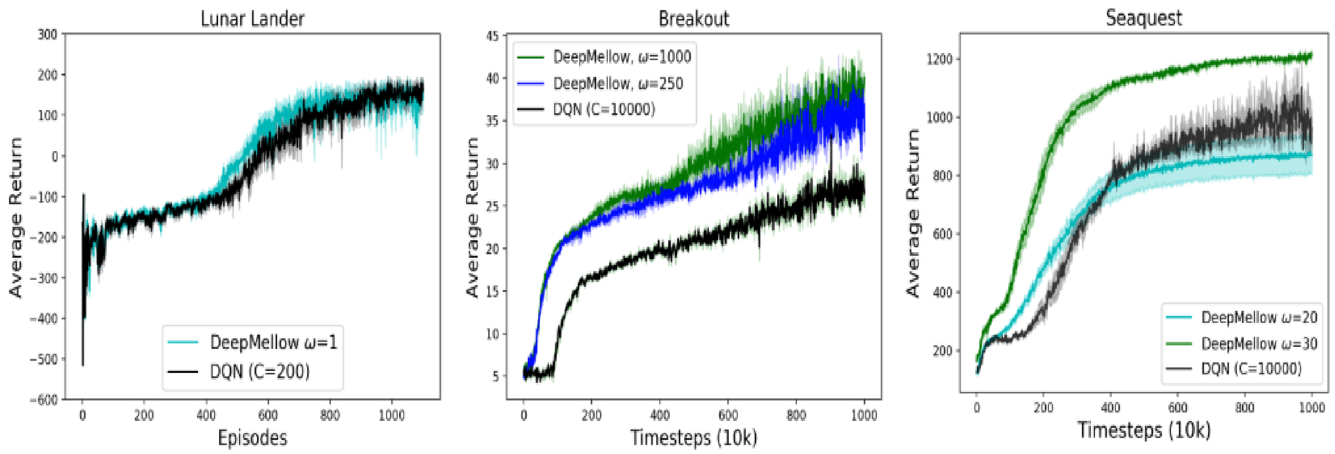

Figure 2: Performances of DeepMellow (no target network) and DQN (with a target network). If tuned with an optimal temperature parameter ω value, DeepMellow learns faster than DQN with a target network.

a target network performs even better than DQN *with* a target network.

To see if DeepMellow has an advantage over DQN with a target network, we compared their performances, focusing on their learning speed. Our prediction is that DeepMellow will learn faster than DQN, because DQN's updates are delayed ($C > 1$), and DeepMellow is likely to react faster to the environment.

As shown in Figure 2, DeepMellow *does* learn faster than DQN in Lunar Lander, Breakout, and Seaquest domains. In Acrobot (not shown), there was no significant difference, because both algorithms learned very quickly. In Lunar Lander, DeepMellow reaches a score of 0 at episode 517 on average, while DQN reaches the same point around episode 561 on average. (DeepMellow is 8% faster than DQN in reaching to the zero-score.)

In Breakout and Seaquest, DeepMellow learns faster than DQN and achieves higher performance, if tuned with an optimal ω parameter. In Breakout, DeepMellow ($\omega = 1000$) reaches the score of 15 and 20 at timestep 55×10^4 and 95×10^4, while DQN reaches them at timestep 153×10^4 and 503×10^4. Similarly, in Seaquest, DeepMellow ($\omega = 30$) reaches the score of 600 and 800 at timestep 140×10^4 and 193×10^4, while DQN reaches them at 296×10^4 and 406×10^4. We also observed that, in both Atari games, DeepMellow achieves higher scores than DQN. In Breakout, at timestep 1000×10^4, the scores of DeepMellow ($\omega = 1000$) and DQN are 38 and 26, respectively, which means that the score of DeepMellow at this timepoint is 42.6% higher than DQN. Similarly in Seaquest, at timestep 800×10^4, the scores of DeepMellow ($\omega = 30$) and DQN are 1205 and 962, respectively. At this timepoint, DeepMellow achieves a score that is 25.2% higher than DQN.

6 Conclusion and Future Work

We have introduced a new algorithm, DeepMellow, that can learn stably without the use of a target network. DeepMellow replaces the max operator in DQN with the Mellowmax operator. We proved new mathematical properties of the Mellowmax operator (convexity, monotonic non-decrease, and mitigation of overestimation) and explained how we can reduce the instability of deep Q-learning using Mellowmax. This increased stability reduces the need for a target network, speeding up learning.

Our empirical results show that DeepMellow outperforms DQN, both when DQN does and does not have a target network. In control domains and Atari games, DeepMellow (with temperature parameter tuned to each domain) showed more stability, higher performance, and faster learning than DQN. The improvement is likely due to the elimination of delays between action-value function updates.

One limitation of this work is that we used an exhaustive grid-search method to choose an optimal ω parameter. This approach can be inefficient, especially when researchers deal with large-scale domains with deep neural network settings. Developing a new method that adaptively chooses this parameter for each domain remains an open research problem.

As a final remark, we note that DeepMellow is closer to true online reinforcement learning, because using a target network deviates from online learning. One possible future direction of research is to reduce the need for an experience replay, which is another deviation from online learning, and achieve full online reinforcement learning in complex settings. Another future research direction is to compare the DeepMellow algorithm with other variants of DQN, such as Double-DQN [van Hasselt *et al.*, 2016] or Duel-DQN [Wang *et al.*, 2016].

Acknowledgements

This research was supported in part by the National Science Foundation, under grant number 1717569.

References

[Asadi and Littman, 2017] Kavosh Asadi and Michael L. Littman. An Alternative Softmax Operator for Reinforcement Learning. In *Proceedings of the 34th International Conference on Machine Learning*, pages 243–252, 2017.

[Bellemare *et al.*, 2013] Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The Arcade Learning Environment: An Evaluation Platform for General Agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

[Bellman, 1957] Richard Bellman. A Markovian Decision Process. *Journal of Mathematics and Mechanics*, 6(5), 1957.

[Boyd and Vandenberghe, 2004] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge university press, 2004.

[Brafman and Tennenholtz, 2002] Ronen I. Brafman and Moshe Tennenholtz. R-MAX—A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. *Journal of Machine Learning Research*, 3:213–231, 2002.

[Dai *et al.*, 2018] Bo Dai, Albert Shaw, Lihong Li, Lin Xiao, Niao He, Zhen Liu, Jianshu Chen, and Le Song. SBEED: Convergent Reinforcement Learning with Nonlinear Function Approximation. In *Proceedings of the International Conference on Machine Learning*, pages 1133–1142, 2018.

[Dearden *et al.*, 1998] Richard Dearden, Nir Friedman, and Stuart Russell. Bayesian Q-learning. In *AAAI/IAAI*, pages 761–768, 1998.

[Fox *et al.*, 2016] Roy Fox, Ari Pakman, and Naftali Tishby. Taming the Noise in Reinforcement Learning via Soft Updates. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, 2016.

[Gao and Pavel, 2017] Bolin Gao and Lacra Pavel. On the Properties of the Softmax function with Application in Game Theory and Reinforcement Learning. *arXiv preprint arXiv:1704.00805*, 2017.

[LeCun *et al.*, 2015] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep Learning. *nature*, 521(7553):436, 2015.

[Littman and Szepesvári, 1996] Michael L Littman and Csaba Szepesvári. A Generalized Reinforcement-Learning Model: Convergence and Applications. In *ICML*, volume 96, pages 310–318, 1996.

[Mnih *et al.*, 2015] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level Control through Deep Reinforcement Learning. *Nature*, 518(7540):529–533, 2015.

[Nachum *et al.*, 2017] Ofir Nachum, Mohammad Norouzi, Kelvin Xu, and Dale Schuurmans. Bridging the Gap Between Value and Policy Based Reinforcement Learning. In *Advances in Neural Information Processing Systems*, pages 2775–2785, 2017.

[Neu *et al.*, 2017] Gergely Neu, Anders Jonsson, and Vicenç Gómez. A Unified View of Entropy-Regularized Markov Decision Processes. *arXiv preprint arXiv:1705.07798*, 2017.

[Rummery and Niranjan, 1994] G. A. Rummery and M. Niranjan. On-line Q-learning Using Connectionist Systems. Technical Report CUED/F-INFENG/TR 166, Cambridge University Engineering Department, 1994.

[Safak, 1993] Aysel Safak. Statistical Analysis of the Power Sum of Multiple Correlated Log-Normal Components. *IEEE Transactions on Vehicular Technology*, 42(1):58–61, 1993.

[Stahl II and Wilson, 1994] Dale O Stahl II and Paul W Wilson. Experimental Evidence on Players' Models of Other Players. *Journal of Economic Behavior & Organization*, 25(3):309–327, 1994.

[Strehl *et al.*, 2006] Alexander L Strehl, Lihong Li, Eric Wiewiora, John Langford, and Michael L Littman. PAC Model-Free Reinforcement Learning. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 881–888, 2006.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning - An Introduction*. MIT Press, 1998.

[van Hasselt *et al.*, 2016] Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2094–2100, 2016.

[van Hasselt, 2010] Hado van Hasselt. Double Q-learning. In *Advances in Neural Information Processing Systems 23*, pages 2613–2621, 2010.

[van Seijen and Sutton, 2014] Harm van Seijen and Rich Sutton. True Online TD (lambda). In *Proceedings of the International Conference on Machine Learning*, pages 692–700, 2014.

[Wang *et al.*, 2016] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. Dueling Network Architectures for Deep Reinforcement Learning. In *Proceedings of the 33nd International Conference on Machine Learning*, pages 1995–2003, 2016.