Adaptive Tuning of Temperature in Mellowmax using Meta-Gradients

Seungchan Kim

Department of Computer Science, Brown University
Submitted in partial fulfillments of the requirements
for the Master's of Science

Advisor: George Konidaris

May 2020

Abstract

Mellowmax is a recently proposed alternative softmax operator that has been used in deep reinforement learning in the context of action-selection, reducing overestimation, and removing the need for a target network. However, Mellowmax has an additional temperature parameter that must be tuned to obtain optimal results. Previous work relied on grid-search, which is comptuationally expensive, and fixes a single temperature parameter setting for the entire learning process. We propose a novel online method that adaptively tunes the temperature parameter using meta-gradients. We show that we can use this method due to the differentiability of Mellowmax, and address practical issues on how to compute meta-gradients for temperature tuning in Mellowmax. Finally, we present the empirical results of our algorithm on a simple OpenAI gym domain, Acrobot.

1 Introduction

Mellowmax [1] is an alternative softmax operator for reinforcement learning with several interesting properties. Unlike the well-known Boltzmann softmax operator, Mellowmax has a non-expansion property that ensures convergence to a unique fixed point. Previous work has demonstrated other useful properties of Mellowmax, such as connections with entropy-regularization [2, 3] and convergent off-policy learning for non-linear function approximation [4]. More recently, it has been shown that, when combined with Deep Q-Network [5], Mellowmax yields comparable performance to the softmax Boltzmann operator in Atari games [6], reduces overestimation bias, and removes the need for a target network [7].

However, one limitation of Mellowmax is that it has a temperature hyperparameter, ω, which must be tuned to maximize performance on each task. Previous work [6, 7] has either relied on exhaustive grid-search to find optimal temperature values or reported algorithmic performance based on a small set of arbitrary temperature values. These approaches can be inefficient, especially in deep reinforcement learning settings where training takes a long time. Grid searches also have fundamental drawbacks, such as finding a single fixed temperature parameter setting, which may not perform as well as dynamically adjusting the parameter over the course of learning.

We propose a novel online method to adaptively tune the temperature parameter ω for deep reinforcement learning. We use meta-gradient reinforcement learning [8], a gradient-based meta learning algorithm that optimizes return by adapting its tunable meta-parameter. In this report, we address the technical issues regarding the implementations of meta-gradients, and show that we can adjust the temperature parameter using meta-gradient learning methods due to the differentiability of Mellowmax with respect to ω. We test the performance of this algorithm on Acrobot, a simple OpenAI gym control domain [9], and present preliminary results that our adaptive algorithm performs better than a fixed parameter identified by grid search.

2 Background

2.1 Mellowmax Operator

The Mellowmax operator [1] is an operator defined as:

$$mm_\omega(\mathbf{x}) = \frac{\log(\frac{1}{n}\sum_{i=1}^{n}\exp(\omega x_i))}{\omega}, \tag{1}$$

where $\mathbf{x}$ is an input vector of n real numbers, and ω is a temperature parameter. Mellowmax is a non-expansion, which ensures convergence to a unique fixed point. Another interpretation of this operator is as a smoothed version of max function : if $\omega \to \infty$, then $mm_\omega(\mathbf{x})$ becomes $\max(\mathbf{x})$, and if $\omega \to 0$, then $mm_\omega(\mathbf{x})$ becomes $mean(\mathbf{x})$. Any positive ω value between 0 and ∞ offers a continuous interpolation between max and mean. Thus, ω controls the degree of smoothness of the Mellowmax operator.

Furthermore, Mellowmax is differentiable with respect to ω, contrary to the non-differentiable max function:

$$\frac{\partial mm_\omega(\mathbf{x})}{\partial \omega} = \frac{1}{\omega^2}\{\frac{\omega\sum_{i=1}^{n}x_i\exp(\omega x_i)}{\sum_{i=1}^{n}\exp(\omega x_i)} - \log(\frac{1}{n}\sum_{i=1}^{n}\exp(\omega x_i))\}, \tag{2}$$

where $\mathbf{Q} = [Q(s,a_1), Q(s,a_2), ..., Q(s,a_n)]$ is an input vector of action-values (or Q-values) in Q-learning. Thus, Mellowmax can replace the max function in the Bellman update equation and be incorporated into Q-learning as follows:

$$Q_{new}(s,a) \leftarrow Q(s,a) - \alpha[r + \gamma mm_{\omega,a'}Q(s',a') - Q(s,a)]. \tag{3}$$

Using Equation 3, we can combine Deep Q-Network (DQN) and Mellowmax in the context of action-values summarizations. Since the Q-function is approximated with neural network, the Bellman equation of the combination of DQN and Mellowmax (or Mellowmax-DQN) becomes:

$$\theta \leftarrow \theta - \alpha[r + \gamma mm_\omega\widehat{Q}(s',a';\theta^-) - Q(s,a;\theta)]\nabla_\theta Q(s,a;\theta), \tag{4}$$

where θ is the weight vector of Q network, and θ^- is the weight vector of the target network $\widehat{Q}$.

Previous works investigated the properties of Mellowmax-DQN in various perspectives: Song et al.[6] compared the performance of Mellowmax-DQN with softmax-DQN and DQN; Kim et al.[7] focused on the role of Mellowmax in reducing the overestimation bias and removing the need for a separate target network $\widehat{Q}$ in DQN.

One limitation of using Mellowmax is the presence of an additional temperature parameter ω. This parameter must be tuned, because the performance of the algorithm largely depends on the choice of the temperature value. Previous works have performed grid-search to find the optimal temperature parameter from a set of possible values. However, this approach is problematic for three reasons: 1) the grid-search finds a single temperature value for the entirety of learning, but fixing the temperature may not be the best strategy, when the dynamic adjustment of the parameter is needed, 2) it requires a domain simulator for each parameter search, making it unusable for online interactions with a real environment and 3) it is computationally expensive, especially when using deep neural networks that take several hours to train. Therefore, the need for an adaptive, online method for tuning ω is acute.

2.2 Meta-Gradient Reinforcement Learning

In deep reinforcement learning, the value function is parameterized by θ, and the main goal of the agent is to update θ according to the update equation:

$$\theta' \leftarrow \theta - \alpha\frac{\partial J(\theta,\tau)}{\partial\theta} \tag{5}$$

where τ is the transition samples of (s,a,r,s') from experience replay buffer, J is loss function, and α is the learning rate. Meta-gradient reinforcement learning [8] sets additional meta-parameter η that characterizes the forms of return function. The agent aims to maximize rewards not only by updating the parameter θ but also by optimizing the form of return function parameterized by tunable meta-parameters η. Now the equation becomes:

$$\theta' \leftarrow \theta - \alpha\frac{\partial J(\theta,\tau,\eta)}{\partial\theta}. \tag{6}$$

Note that the term $-\alpha\cdot\partial J(\theta,\tau,\eta)/\partial\theta$ is the change of parameters of θ per single update step. Meta-gradient methods sample additional batches of transitions τ', and compute the derivative of meta-objective loss function J' with respect to the tunable meta-parameter η. Xu et al.[8] uses approximation to compute this derivative:

$$\frac{\partial J'(\tau',\theta',\overline{\eta})}{\partial\eta} = \frac{\partial J'(\tau',\theta',\overline{\eta})}{\partial\theta'}\frac{d\theta'}{d\eta}, \tag{7}$$

and updates the meta-parameter η using the following equation:

$$\eta' \leftarrow \eta - \beta \frac{\partial J'(\tau', \theta', \overline{\eta})}{\partial \theta'} \frac{d\theta'}{d\eta}. \tag{8}$$

Here, β is the meta-learning rate for η. For this method to work, the return function should be differentiable with respect to the meta-parameter η. Xu et al.[8] uses this method to adapt discount factor γ and bootstrapping parameter λ during the training. We note that the Mellowmax operator is differentiable with respect to its temperature parameter ω, and in the next section, explain how this method can tune the temperature parameter.

3 Computing Meta-Gradients for Mellowmax

In the case of Mellowmax-DQN, we set the loss objective function J as below:

$$J(\tau, \theta, \omega) = \frac{1}{2}[r_t + \gamma mm_\omega \widehat{Q}(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta)]^2. \tag{9}$$

Here τ represents the minibatch samples for the update (s_t, a_t, r_t, s_{t+1}) tuples, θ are the Q-function parameters, α is a learning rate, and ω is the meta-parameter that we want to tune. At every iteration, the Q-learning algorithm will update θ in a direction that minimizes $J(\tau, \theta, \omega)$.

We define another meta-objective function J', using the same form of loss function as before:

$$J(\tau', \theta', \overline{\omega}) = \frac{1}{2}[r'_t + \gamma mm_{\overline{\omega}} \widehat{Q}(s'_{t+1}, a'; \theta^-) - Q(s'_t, a'_t; \theta')]^2. \tag{10}$$

After each iteration of Equation 9, θ is updated to θ', and we use the updated θ' to compute the meta-objective J'. We also use another independent batch sample τ' (denoted as $(s'_t, a'_t, r'_t, s'_{t+1})$ tuples) and the reference meta-parameter value $\overline{\omega}$. Our goal is to adapt the value of ω such that it optimizes the meta-objective function J', using the equation below:

$$\frac{\partial J'(\tau', \theta', \overline{\omega})}{\partial \omega} = \frac{\partial J'(\tau', \theta', \overline{\omega})}{\partial \theta'} \frac{d\theta'}{d\omega}. \tag{11}$$

The term $d\theta'/d\omega$ can be approximated:

$$\frac{d\theta'}{d\omega} = \frac{d}{d\omega}(\theta - \alpha \frac{\partial J(\tau, \theta, \omega)}{\partial \theta}) \approx \frac{d}{d\omega}(-\alpha \frac{\partial J(\tau, \theta, \omega)}{\partial \theta}). \tag{12}$$

In deep learning libraries like PyTorch, it's easy to obtain $\frac{\partial J(\tau,\theta,\omega)}{\partial \theta}$ (e.g. gradients of parameters). If we use a simple SGD optimizer, it is not hard to compute $-\alpha \frac{\partial J(\tau,\theta,\omega)}{\partial \theta}$, because we can just linearly scale the gradients with learning rate. However, since we use advanced optimizers like RMSprop [10] or Adam [11], computing $-\alpha \frac{\partial J(\tau,\theta,\omega)}{\partial \theta}$ becomes a non-trivial task. For this work, we used RMSprop to update θ, so we kept track of exponentially decaying running average and decaying learning rate to compute accurate $-\alpha \frac{\partial J(\tau,\theta,\omega)}{\partial \theta}$.

Next, recall that $J(\tau, \theta, \omega)$ is $\frac{1}{2}[r_t + \gamma mm_\omega \widehat{Q}(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta)]^2$, and

$$-\alpha \frac{\partial J(\tau, \theta, \omega)}{\partial \theta} = \alpha[r_t + \gamma mm_\omega \widehat{Q}(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta)] \nabla_\theta Q(s_t, a_t; \theta). \tag{13}$$

Thus,

$$\frac{d}{d\omega}(-\alpha \frac{\partial J(\tau, \theta, \omega)}{\partial \theta}) = \frac{d}{d\omega}[r_t + \gamma mm_\omega \widehat{Q}(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta)] \alpha \nabla_\theta Q(s_t, a_t; \theta) \tag{14}$$

$$= \frac{d}{d\omega}[\gamma mm_\omega \widehat{Q}(s_{t+1}, a; \theta^-)] \alpha \nabla_\theta Q(s_t, a_t; \theta). \tag{15}$$

The term $\frac{d}{d\omega}[\gamma mm_\omega \widehat{Q}(s_{t+1}, a; \theta^-)] = \gamma mm'_\omega \widehat{Q}$ can be directly computed using the equation 2. Also, note that it is hard to directly compute the term $\alpha \nabla_\theta Q(s, a; \theta)$. What we do instead is to use importance sampling:

$$\frac{d}{d\omega}(-\alpha \frac{\partial J(\tau, \theta, \omega)}{\partial \theta}) = -\alpha \frac{\partial J(\tau, \theta, \omega)}{\partial \theta} \cdot \frac{\gamma mm'_\omega \widehat{Q}}{[r + \gamma mm_\omega \widehat{Q}(s_{t+1}, a; \theta^-) - Q(s_t, a_t; \theta)]}. \tag{16}$$

After we approximate the term $d\theta'/d\omega$ using the equation 16, we now compute $\frac{\partial J'(\tau', \theta', \overline{\omega})}{\partial \theta'}$. This is fairly easy to obtain; we can simply retrieve the gradients of θ' parameters for this step. Combining these two, we can finally get $\frac{\partial J'(\tau', \theta', \overline{\omega})}{\partial \omega}$ as in the Equation 11, and update the temperature using:

$$\omega' \leftarrow \omega - \beta \frac{\partial J'(\tau', \theta', \overline{\omega})}{\partial \omega}. \tag{17}$$

In Algorithm 1, we present the pseudocode for updating the temperature parameter of Mellowmax using meta-gradients.

Algorithm 1 Adaptive Mellowmax Deep Q-Network with Meta-Gradient

Initialize temperature parameter $\omega = \omega_0$, and set a reference temperature parameter $\overline{\omega}$
Initialize experience replay memory D, set learning rate α, set meta-learning rate β
Initialize action-value network Q with random weights θ and target network $\widehat{Q}$ with $\theta^- = \theta$
for episode = 1 to M **do**
  Initialize the starting state s_1
  for t= 1 to T **do**
    Select a random action a_t with probability ϵ. Otherwise, $a_t = \arg\max_a Q(s_t, a; \theta)$
    Execute action a_t, observe reward r_t, and obtain the next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in D
    Sample random batch $\tau = (s_j, a_j, r_j, s_{j+1})$ from D for value function update
    Sample another random batch $\tau' = (s_k, a_k, r_k, s_{k+1})$ from D for meta-parameter update
    Compute the objective function $J = \frac{1}{2}[r_j + \gamma mm_\omega \widehat{Q}(s_{j+1}, a'; \theta^-) - Q(s_j, a_j; \theta)]^2$
    Update parameters: $\theta' \leftarrow \theta - \alpha \frac{\partial J(\theta)}{\partial \theta}$
    Compute the meta-objective
    $J'(\tau', \theta', \overline{\omega}) = \frac{1}{2}[r_k + \gamma mm_{\overline{\omega}} \widehat{Q}(s_{k+1}, a'; \theta^-) - Q(s_k, a_k; \theta')]^2$
    Update meta-parameter $\omega' \leftarrow \omega - \beta \frac{\partial J'(\tau', \theta', \overline{\omega})}{\partial \omega}$
    Every C steps, copy $\widehat{Q} = Q$
  end for
end for

4 Empirical Results

Now we present the empirical results on Acrobot, a simple control domain in OpenAI gym. We compare the performance of Adaptive Mellowmax-DQN with DQN and Mellowmax-DQN (fixed temperature). We also present the evolution curve of temperature over the training. For the Q-value function approximation, we used 3 linear dense layers of 300 units, each with ReLU activation. The learning rate for θ parameter is $\alpha = 0.001$, and we used RMSprop optimizer. The meta learning rate for ω is $\beta = 0.001$, and we used SGD optimizer. The sizes of minibatch and metabatch are 16 and 8, respectively. We set the target network update frequency $C = 1$ for the Mellowmax-DQN and $C = 100$ for DQN as in previous work.

First, we observed that the performance of Mellowmax-DQN depends on the choice of temperature parameter ω in the fixed settings. As shown in the upper plots of Figure 1, Mellowmax-DQN with $\omega = 1, 2$ performs almost as well as DQN, but $\omega = 5, 10$ performs poorly. However, in the adaptive settings, the Mellowmax-DQN with meta-gradients consistently shows good performance, regardless of initial ω_0.

In the Figure 2, we plotted the performance of Adaptive Mellowmax-DQN with meta-gradients (initial $\omega_0 = 1, 2, 5$), and compared them with their counterparts (fixed temperature $\omega = 1, 2, 5$). The adaptive algorithm performs better than the original algorithm with fixed settings. On the right side of the figure, we plotted the evolution curves of temperature over the training.

5 Discussion and Future Directions

Mellowmax is a useful softmax operator in deep reinforcement learning. However, the choice of temperature parameter is a non-trivial task, and often requires extensive search for the parameter. We proposed a novel, online adaptive method that tunes the temperature parameter of Mellowmax using the idea of meta-gradients. In the simple control domain, Acrobot, we showed that this adaptive algorithm yields better results than the previous algorithm with fixed temperatures. We raise important questions in this work and discuss the directions for the future research.

- **Does this adaptive algorithm perform better than previous algorithm with fixed temperature?** At least in the simple control task, the Adaptive Mellowmax-DQN with meta-gradients shows better performance than the Mellowmax-DQN with fixed temperature settings. It also shows comparable, or even better, performance than DQN.

- **Does this algorithm work in large-scale domains?** Our next step is to test this method on large-scale Atari game domains. Our previous findings are that, in several domains including Seaquest, Mellowmax-DQN with carefully chosen fixed-temperature outperforms DQN and shows comparable performance with Double-DQN [12]. It is worthwhile to investigate how well our adaptive algorithm works in large-scale Atari games, and whether it can outperform state-of-the-art deep Q-learning algorithms.

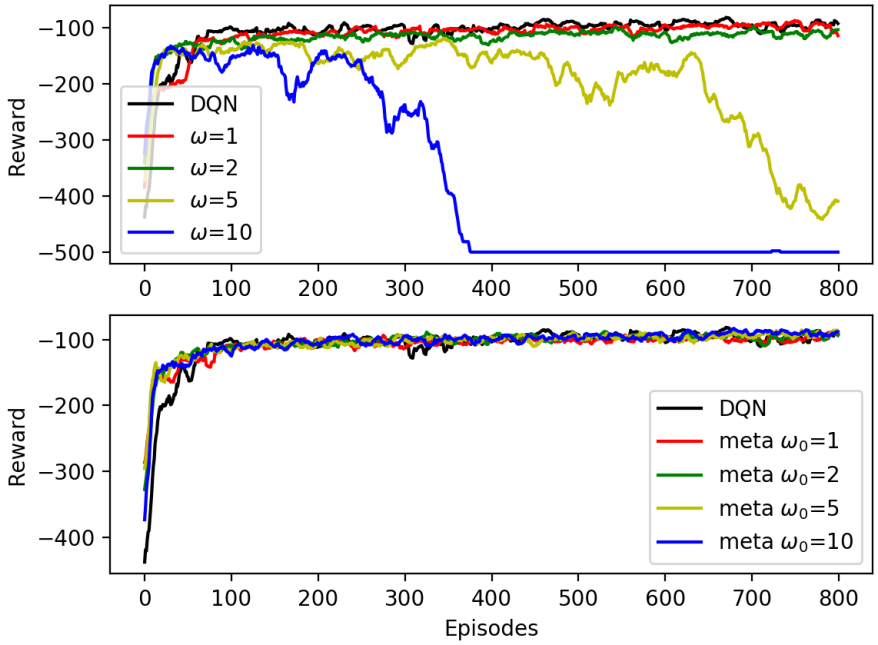

Figure 1: Upper: comparison of performance between DQN and Mellowmax-DQN with fixed temperature settings $\omega = 1, 2, 5, 10$. Lower: comparison of performance between DQN and Mellowmax-DQN with adaptive temperature settings, with initial $\omega_0 = 1, 2, 5, 10$.

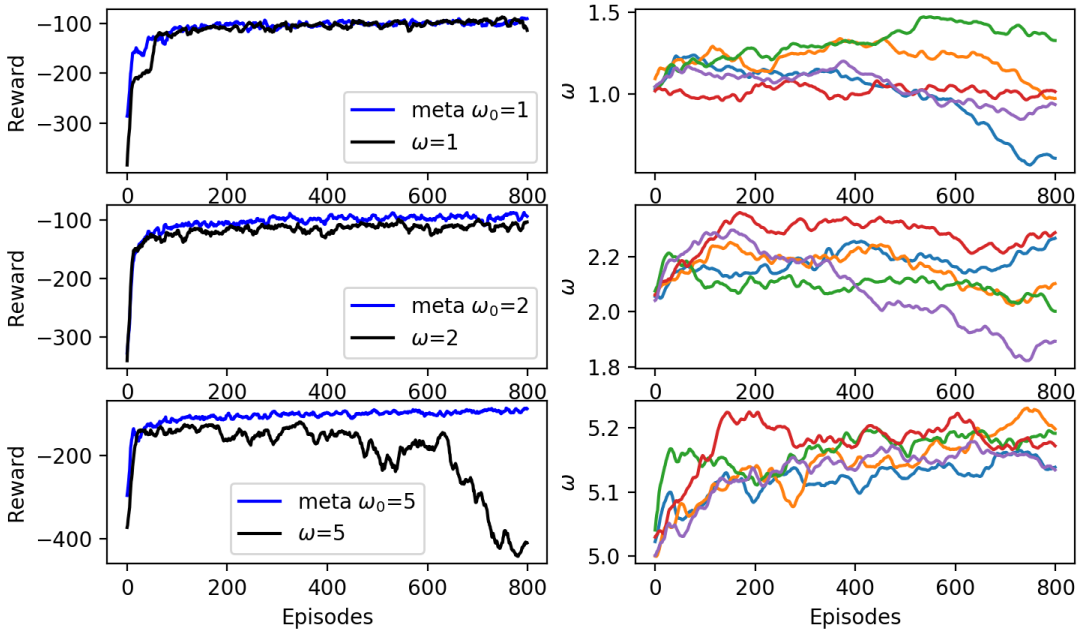

Figure 2: Comparison of Mellowmax-DQN between fixed and adaptive temperature settings, with initial $\omega_0 = 1, 2, 5$. Right side of figures are the evolution curves of ω in adaptive settings (five independent runs).

- **Does the choice of initial temperature ω_0 matter?** Our empirical results show that the performance of the adaptive algorithm is not very sensitive to the choice of initial temperature. Having said that, it is also worth noting that the temperature does not fluctuate drastically; the final temperature at later episodes are bounded near the initial temperature. This might be problematic in large-scale domains like Atari games, where the search space for temperature is wider.
- **How sensitive is this to the choice of meta-learning rate β?** We only tested with the meta-learning rate $\beta = 0.001$, but choosing the right meta-learning rate is important. If the meta-learning rate is too large, it might result in the divergence of temperature; if the meta-learning rate is too small, the tuning effect will be small. It is also worthwhile to use advanced optimizers like ADAM [11] (we used SGD for this work) that can effectively set the meta-learning rate over time.

References

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

[2] 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.*

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

[4] 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*

[5] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G.Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, StigPetersen, 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.*

[6] Zhao Song, Ronald Parr, and Lawrence Carin. Revisiting the Softmax Bellman Operator: New Benefits and New Perspective. *In Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA, pages 5916–5925, 2019.*

[7] Seungchan Kim, Kavosh Asadi, Michael L. Littman, and George Konidaris. DeepMellow: Removing the Need for a Target Network in Deep Q-Learning. *In Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao,China, August 10-16, 2019, pages 2733–2739, 2019.*

[8] Zhongwen Xu, Hado van Hasselt, and David Silver. Meta-Gradient Reinforcement Learning. *In Advances in Neural Information Processing Systems 31: Annual Conference on NeuralInformation Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.,pages 2402–2413, 2018.*

[9] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang,and Wojciech Zaremba. OpenAI Gym. *CoRR, abs/1606.01540, 2016.*

[10] Geoffrey Hinton. Neural Networks for Machine Learning - Lecture 6a - Overview of mini-batch gradient descent. 2012

[11] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *In 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May7-9, 2015, Conference Track Proceedings, 2015*

[12] 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.*