An Alternative Softmax Operator for Reinforcement Learning Reproducing Task

Hsin-En, Su, Wei-Shiang Chen, Shang-Hsuan Yang
Department of Computer Science
National Chiao Tung University
{littlecrazymouse.cs06, wschen.st06g, sandy861003.eo05}@nctu.edu.tw

1 Problem Overview

There is a fundamental tension in decision making between choosing the action that has the highest expected utility and avoiding "starving" all the other actions. This issue is often referred to as the "exploration-exploitation dilemma" Thrun [1992].

In reinforcement learning, an approach to tackle this problem is to use softmax policies for action selection. There is a popular softmax function called Boltzmann softmax which is defined as:

$$\mathrm{boltz}_\beta(\mathbf{X}) = \frac{\sum_{i=1}^{n} x_i e^{\beta x_i}}{\sum_{i=1}^{n} e^{\beta x_i}}$$

Boltzmann softmax is controlled by a hyperparameter β, and is often chosen as the softmax function for various task. However, the paper showed that Boltzmann softmax could lead to value-based methods such as SARSA non-convergence, i.e., the algorithm will never converge if an improper β value is selected.

Because of the misbehavior of Boltzmann softmax, the paper first analyzed the problem of Boltzmann softmax, proposed properties that a softmax function should meet, and presented a new softmax function, which met all of the recommended properties.

2 Background

An ideal softmax should met the following properties:

2.1 Properties

Property 1. **Maximization**

A softmax function is used in reinforcement problems to avoid starvation, which can be viewed as a relaxation of function **max**. Thus a good softmax function should still be able to simulate the behavior of maximization, i.e., there should be a parameter setting that allows it to approximate maximization.

Property 2. **Non-expansion**

A Markov decision process, or MDP, is specified by the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma \rangle$, where $\mathcal{S}$ is the set of states and $\mathcal{A}$ is the set of actions. The functions $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ denote the reward and transition dynamics of the MDP. Finally, $\gamma \in [0, 1)$, the discount rate, determines the relative importance of immediate reward as opposed to the rewards received in the future.

A typical approach to finding a good policy is to estimate how good it is to be in a particular state — the state value function. The value of a particular state s given a policy π and initial action a is written $Q_\pi(s, a)$. We define the optimal value of a state-action pair $Q^\star(s, a) =$

Preprint. Under review.

$\max_\pi Q_\pi(s,a)$. It is possible to define $Q^\star(s,a)$ recursively and as a function of the optimal value of the other state-action pairs:

$$Q^\star(s,a) = \mathcal{R}(s,a) + \sum_{s' \in \mathcal{S}} \gamma \mathcal{P}\left(s,a,s'\right) \max_{a'} Q^\star\left(s',a'\right)$$

Bellman equations, such as the above, are at the core of many reinforcement-learning algorithms such as Value Iteration. The algorithm computes the value of the best policy in an iterative fashion:

$$\hat{Q}(s,a) \leftarrow \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}\left(s,a,s'\right) \max_{a'} \hat{Q}\left(s',a'\right)$$

Regardless of its initial value, $\hat{Q}$ will converge to Q^* Littman & Szepesvári (1996) generalized this algorithm by replacing the max operator by any arbitrary operator $\otimes$ resulting in the generalized value iteration (GVI) algorithm with the following update rule:

$$\hat{Q}(s,a) \leftarrow \mathcal{R}(s,a) + \gamma \sum_{s' \in \mathcal{S}} \gamma \mathcal{P}\left(s,a,s'\right) \bigotimes_{a'} \hat{Q}\left(s',a'\right)$$

Algorithm 2 GVI algorithm

Input: initial $\hat{Q}(s,a)\ \forall s \in \mathcal{S}\ \forall a \in \mathcal{A}$ and $\delta \in \mathcal{R}^+$
repeat
  diff $\leftarrow 0$
  for each $s \in \mathcal{S}$ **do**
    for each $a \in \mathcal{A}$ **do**
      $Q_{copy} \leftarrow \hat{Q}(s,a)$
      $\hat{Q}(s,a) \leftarrow \sum_{s' \in \mathcal{S}} \mathcal{R}(s,a,s')$
        $+ \gamma \mathcal{P}(s,a,s') \bigotimes \hat{Q}(s',.)$
      diff $\leftarrow \max\left\{\text{diff}, |Q_{copy} - \hat{Q}(s,a)|\right\}$
    end for
  end for
until diff $< \delta$

However, the convergence of GVI algorithm to a unique fixed point is gauranteed only if the operator $\bigotimes$ is *non-expansion* with respect to the infinity norm:

$$\left| \bigotimes_a \hat{Q}(s,a) - \bigotimes_a \hat{Q}'(s,a) \right| \leq \max_a \left| \hat{Q}(s,a) - \hat{Q}'(s,a) \right|$$

for any $\hat{Q}, \hat{Q}'$ and s.

As we mentioned earlier, the misbehavior of Boltzmann softmax comes from the fact that it is not a non-expansion Littman and Szepesvári [1996], and therefore it is possible that more than one distinct fixed points exist, making the algorithm bouncing between different fixed points.

Property 3. **Differentiable**

A softmax function is preferred to be differentiable so that it is possible to be used in a gradient-based optimization task.

2.2 Mellowmax

We advocate for an alternative softmax operator defined as follows:

$$mm_\omega(\mathbf{X}) = \frac{\log\left(\frac{1}{n}\sum_{i=1}^{n} e^{\omega x_i}\right)}{\omega}$$

And it can be proved that Mellowmax has all of the above properties, the proof is detail illustrated in the original paper, thus omitted here. Recall that Boltzmann softmax is not non-expansion since Mellowmax is non-expansion; thus, the paper claimed that Mellowmax might be a good alternative of Boltzmann softmax.

2.3 Maximum Entropy Mellowmax Policy

However, having such a softmax operator is not enough; we have to provide a probability distribution over the actions so that it can be used for action selection in algorithms like SARSA. This should satisfy that (1) a non-zero probability mass is assigned to each action, and (2) the resulting expected value equals the computed value.

In this section, we address the problem of identifying such a probability distribution as a maximum entropy problem-over all distributions that satisfy the properties above, pick the one that maximizes information entropy. We formally define the maximum entropy mellowmax policy of a state s as:

$$\pi_{\mathrm{mm}}(s) = \underset{\pi}{\operatorname{argmin}} \sum_{a\in\mathcal{A}} \pi(a \mid s)\log(\pi(a \mid s))$$

subject to

$$\begin{cases} \sum_{a\in\mathcal{A}} \pi(a \mid s)\hat{Q}(s,a) = \mathrm{mm}_\omega(\hat{Q}(s,.)) \\ \pi(a \mid s) \geq 0 \\ \sum_{a\in\mathcal{A}} \pi(a \mid s) = 1 \end{cases}$$

Note that this optimization problem is convex and can be solved reliably using any numerical convex optimization library.

If we solve the problem by the method of Lagrange Multipliers, we will see that the probability of taking action under the maximum entropy mellowmax policy has the form:

$$\pi_{mm}(a \mid s) = \frac{e^{\beta\hat{Q}(s,a)}}{\sum_{a\in\mathcal{A}} e^{\beta\hat{Q}(s,a)}} \quad \forall a \in \mathcal{A}$$

where β is a value for which:

$$\sum_{a\in\mathcal{A}} e^{\beta\left(\hat{Q}(s,a) - mm_\omega\hat{Q}(s,.)\right)}\left(\hat{Q}(s,a) - mm_\omega\hat{Q}(s,.)\right) = 0 \tag{1}$$

Note that $\pi_{mm}(a \mid s)$ has exactly the same form as Boltzmann softmax, the only difference is that when using Boltzmann softmax, β is fixed and pre-determined. However in the Maximum entropy Mellowmax policy, we compute β in every episode by solving **equation 1**.

3 Detailed Implementation

There are more than two experiments in the paper; however, we have selected the two most representative experiments to reproduce. The first experiment is conducted in the environment *Taxi-v2*, to show the effect of maximum entropy mellowmax policy π_{mm} in SARSA algorithm. The second is to show the effect of π_{mm} in the gradient-based method.

3.1 Taxi-v2

In this experiment, we compared between three different softmax function, *ϵ-greedy*, *Boltzmann softmax* and *Mellowmax softmax*. However, one thing to note is that we cannot find the exact environment *Multi-passenger Taxi* used in the paper, so we used *Taxi-v2* from OpenAI Gym instead.

Both tasks are focusing on picking up passengers at some pre-determined position and dropoff the passenger at another pre-determined position. Although their reward function and the *map* that the agent operates on are different, their main focus is the same; thus, we consider the difference between the two environments negligible.

In case the reader is not familiar with the SARSA algorithm, let $\mathcal{X}$ be all the possible state, and $\mathcal{A}$ be all possible actions, the main purpose of SARSA is to compute the function $Q : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$, that represent the reward of state-action pair in the environment, by performing sampling and bootstrapping, we implemented the following algorithm in python:

Algorithm 1 SARSA: Learn function $Q : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$

Require:
Sates $\mathcal{X} = \{1, \ldots, n_x\}$
Actions $\mathcal{A} = \{1, \ldots, n_a\}$, $A : \mathcal{X} \Rightarrow \mathcal{A}$
Reward function $R : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$
Black-box (probabilistic) transition function $T : \mathcal{X} \times \mathcal{A} \to \mathcal{X}$
Learning rate $\alpha \in [0, 1]$, typically $\alpha = 0.1$
Discounting factor $\gamma \in [0, 1]$
$\lambda \in [0, 1]$: Trade-off between TD and MC

1: **procedure** SARSA($\mathcal{X}$, A, R, T, α, γ, λ)
2: Initialize $Q : \mathcal{X} \times \mathcal{A} \to \mathbb{R}$ arbitrarily
3: **while** Q is not converged **do**
4: Select $(s, a) \in \mathcal{X} \times \mathcal{A}$ arbitrarily
5: **while** s is not terminal **do**
6: $r \leftarrow R(s, a)$
7: $s' \leftarrow T(s, a)$
8: **Calculate π based on Q (i.e. epsilon-greedy, Boltmann softmax policy or Mellowmax softmax policy)**
9: $a' \leftarrow \pi(s')$
10: $Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot (r + \gamma Q(s', a'))$
11: $s \leftarrow s'$
12: $a \leftarrow a'$
return Q

We substitute the action selection (line 8 in **Algorithm 1**) with all three different softmax functions and compare their result. The algorithm stops updating when all 20000 episodes are finished or when the Q-table converges.

3.2 Lunar Lander

Our second experiment uses the exact environment as the paper did, which is the Lunar lander from OpenAI Gym. We experimented only on Boltzmann softmax and π_{mm}, since ϵ-greedy is not differentiable. We follow the description in the paper to implement the policy, which is a neural network with a hidden layer comprised of 16 hidden units with ReLU activation functions followed by a second layer with 16 units and softmax activation functions. When performing softmax, we use Boltzmann softmax with a fixed β if Boltzmann softmax policy is being evaluated; otherwise, we compute the proper β for π_{mm} as mentioned in **section 2**.

We use REINFORCE to train the network, and the different parameter settings of β and ω is evaluated, both ranging from $0 \sim 1$. The original paper runs the experiment with 40000 episodes; however, we set our training to a maximum of 3000 episodes, so the training with stop either when the network converges, or when the maximum episode of 3000 is reached. We will discuss about the reason we do so in **section 3.3**.

3.3 Difficulties

There are several main difficulties when reproducing the experiments.

Difficulty 1. **Overflow**

When solving the **equation 1**, there's a risk of overflow when computing $e^{\beta\left(\hat{Q}(s,a)-mm_\omega \hat{Q}(s,\cdot)\right)}$, if the exponent is too large. To avoid the overflow, we normalized $\hat{Q}(s,a)$ when computing β. Note that this is an ad hoc choice, and this might affect the good properties of the Mellowmax. However, we didn't discover any misbehavior from the experiment result so far, the performance still match our expectation.

Difficulty 2. **Computation Overhead**

We mentioned that we only run 3000 episodes instead of 40000 episodes in our **Lunar Lander** experiment; this is due to the extra overhead when solving the **equation 1**. We will further illustrate the problem in the next section by comparing the computation time of two different softmax function. Despite this being a difficulty for us, we believe that this problem should have been mentioned in the original paper, and consider this actually a good discover of this reproducing task.

4 Empirical Evaluation

In this section, we will first analyze the result of Taxi-v2, and then the result of Lunar lander.

4.1 Taxi-v2

We graph out the mean returns of three different softmax functions with different parameter settings in **figure 1.**

As we can see, both ϵ-greedy and Mellowmax softmax performs quite well. However, in the case of Boltzmann softmax, the performance degrades severely as the β becomes larger. This experiment result proved the fact that Boltzmann softmax can misbehave if the improper β is given. Recall that Mellowmax policy $\pi_{mm}(a \mid s)$ recomputes the β value for each episode, so in theory $\pi_{mm}(a \mid s)$ should be able to avoid such misbehavior, and this theory is also proven by this experiment.

One small thing to note that in the original paper, ϵ-greedy performas terribly. However, in our experiment, ϵ-greedy performs pretty well. Moreover, ϵ-greedy actually also have the shortest converge time. We suspect this is due to the difference between the environment Taxi-v2 and Multi-passenger Taxi.

4.2 Lunar Lander

In the subsection we want to discuss the performance of Mellowmax, as well as the computation cost of it.

4.2.1 Performance

We graph the running reward and different parameter settings in **figure 2.** We can see that in Boltzmann softmax, the performance is unstable, this is especially notable when β is 7. On the other hand, unlike the result of Boltzmann softmax, maximum entropy mellowmax has a much smoother graph. This is yet another proof that using mellowmax over Boltzmann softmax can result in a much more stable result; however, this stabilization is not free, we will next compare the computation cost of two softmax function.

4.2.2 Computation Cost

We graph the running time of two different softmax function in **figure 3.** We can see that the computation time of mellowmax is extremely longer than Boltzmann softmax. Note that this gap of computation time will only increase as the number of episodes grows since, for every single episode, we have to compute a proper β for mellowmax softmax. As mentioned, we have limited our experiments to 3000 episodes, but the original experiment requires 40000 episodes. This gap is already huge when computing 3000 episodes, and this is the reason why we have to limit the number of episodes to such a small number.

Now if we look closer into **figure 1** and **figure 2**, we can see that if the β of Boltzmann softmax is selected at its best performance, the result of both softmax function is pretty similar. Since the

computation overhead is so large, why don't we simply perform the grid search on β? Maybe it's even more efficient to perform a grid search once than compute the β for every episode in mellomax policy.

5 Conclusion

As we have shown, mellowmax softmax indeed can stabilize the result of Boltzmann softmax. However, there are two questions that we would like to raise for future research:

- The computation cost of mellowmax policy is high; this computation overhead arises from solving the **equation 1.** We wonder if there's a possible way to relax the optimization problem, so that instead of addressing the equation correctly, we can make an approximation, thus reduce the computation overhead.
- Besides the fact that ϵ-greedy is not differentiable, is there any other advantage that mellowmax has over it? As shown in **figure 1,** we see that their performance has no significant difference.

References

Sebastian Thrun. The role of exploration in learning control. 1992.

Michael L. Littman and Csaba Szepesvári. A generalized reinforcement-learning model: Convergence and applications. In *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, ICML'96, page 310–318, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc. ISBN 1558604197.

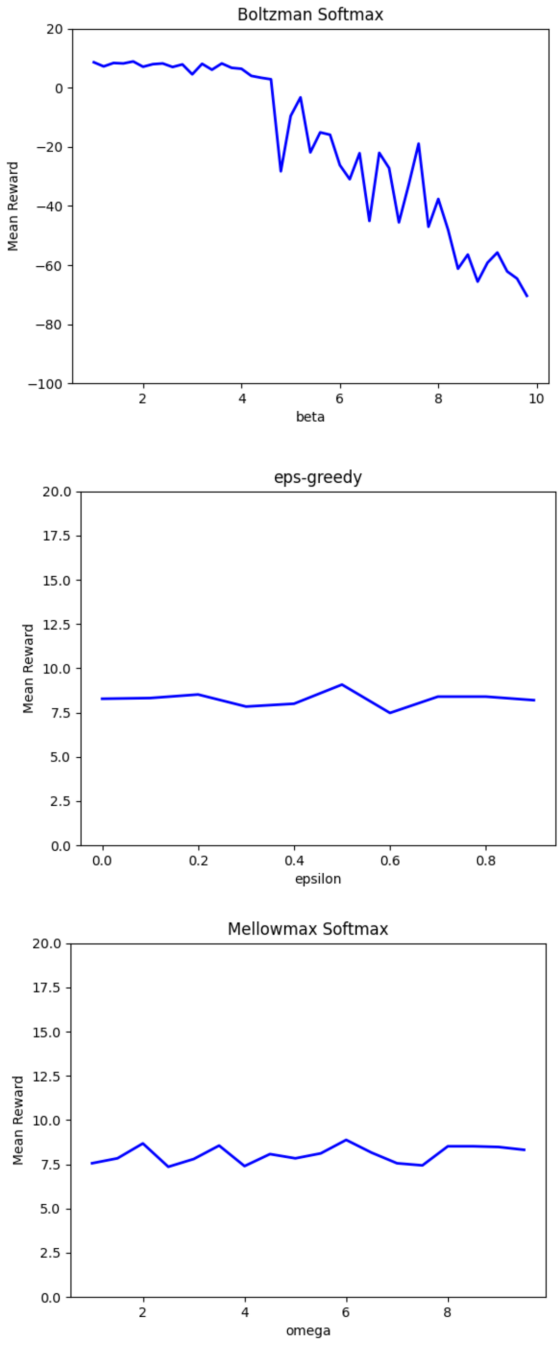

Figure 1: Mean reward of different policies, where x-axis represents the parameter setting of the softmax function. We can see the performance degrade of Boltzmann softmax as the β becomes larger.

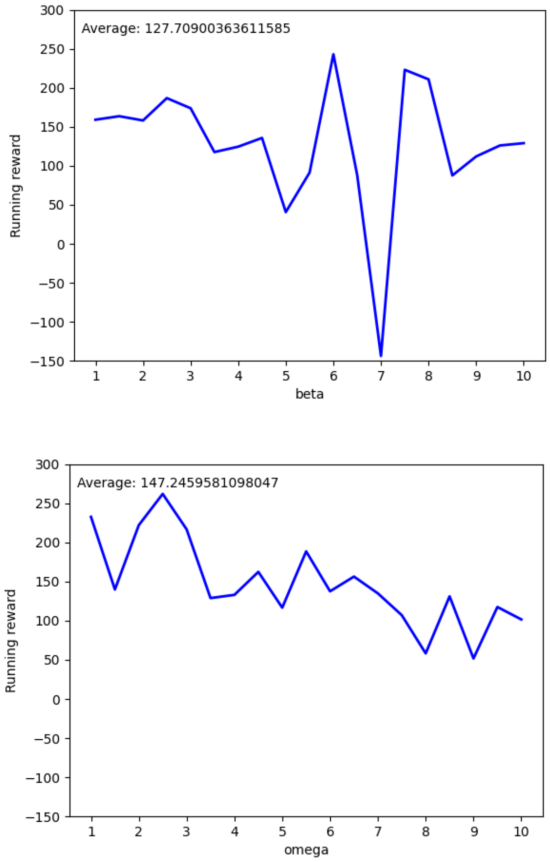

Figure 2: Mean reward of different policies, where x-axis represents the parameter setting of the softmax function. **Top:** Boltzmann softmax, **Bottom:** Maximum entropy mellowmax. We can see that Boltzmann softmax is unstable under different parameter settings.

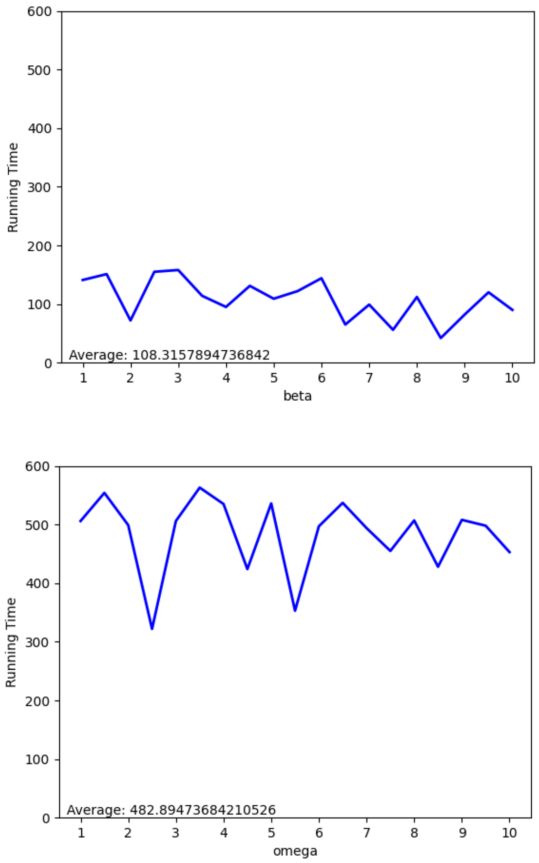

Figure 3: Computation time of different policies, where x-asix represents the parameter setting of the softmax function. **Top:** Boltzmann softmax, **Bottom:** Maximum entropy mellowmax. It is obvious that mellowmax requires a much longer computation time than Boltzmann softmax.