Learning to Serve and Bounce a Ball

Sandra Amend **Gregor Gebhardt**
Technische Universität Darmstadt

Abstract

In this paper we investigate learning the tasks of ball serving and ball bouncing. These tasks display characteristics which are common in a variety of motor skills. To learn the required motor skills for these tasks the robot uses Relative Entropy Policy Search which is a state of the art method in Policy Search Reinforcement Learning. Our experiments show that REPS does not only converge consistently to good solutions, but also robust solutions.

1 Introduction

Learning motor skills similar to those of human beings poses a challenging task for robots. They are difficult and non-trivial to learn, but are necessary for performing complex tasks under varying conditions.

There are several common aspects to motor skills: Motor skills are often target oriented. For example reaching movements are often directed to specific objects. Motor skills are often also constrained by time. As an example, a ball has to be caught before it hits the ground. Some tasks such as walking require periodic motor skills, which involve repeatedly performing similar movements. Most motor skills also require feedback to compensate for errors. For example when writing we try to keep a constant pressure on the pen.

In this paper we will take a closer look at the motor skills of serving and bouncing a ball. Ball serving requires the robot to hit a dropped ball to a desired target location on the ground. Ball bouncing involves repeatedly hitting the ball into the air while keeping it centered above the paddle. They are good representatives of the common aspects of motor skills mentioned above. Ball serving is target oriented and time dependent while ball bouncing enhances serving with the aspect of feedback and a periodic movement. A detailed description of the tasks follows in Section 2. For learning these tasks we will use the state of the art method Relative Entropy Policy Search (REPS) which is explained in Section 2.4.1. Furthermore we will compare REPS to Finite Differences and evaluate the robustness of our solution in Section 4. The benefit of learning a feedback controller for ball bouncing will also be evaluated.

2 Background

Before we come to the methods we used to learn the ball bouncing task, we will first describe the setup of the ball bouncing environment we used for the experiments and then show how we modeled the movements of the robot arm.

2.1 The Experiment Setup

For the experiments we used a Barrett WAM Arm with seven degrees of freedom with a table tennis racket attached as the end effector. The robot arm was simulated using the *SL Simulation and Real-Time Control Software Package* (Schaal, 2007). The kinematic configuration of the Barrett WAM is shown in Figure 1. For our experiments we only actuated the 4th and the 7th joint. The 4th joint was used to perform a stroke movement, which will be explained in detail in the next section, and the 7th joint was used to control the motion of the ball in the x-direction.

In the ball serving task the ball was initially dropped from the ceiling above the table tennis racket. The robot has to perform a hitting motion to redirect the ball to a target location on the ground. The target locations were specified as part of the task. Given the target location x_g and the location where the ball landed x_b the reward for the ball serving task was given by

$$R = -\|x_g - x_b\|^2$$

In the ball bouncing task the ball was also dropped from the ceiling. Instead of a target on the ground the ball had to be hit back to the initial starting height above the racket. In this way the robot and ball return to the same state as at the start of the movement, and the action can be performed repeatedly. Given that

the peak of the balls trajectory is at x_p and the initial position was at x_i the reward function for the ball bouncing task was given by

$$R = -\|x_i - x_p\|^2 - \|\dot{x}_p\|^2$$

For the ball bouncing task the ball position was limited to the x-z plane. The learned controller should also be robust to disturbances in the x direction.

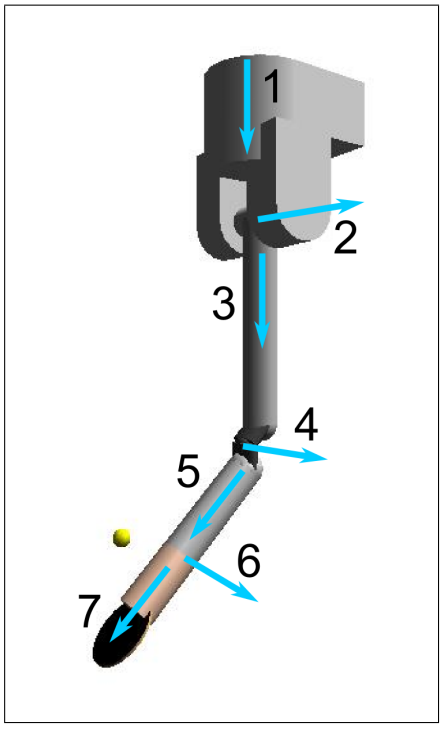

Figure 1: The kinematic configuration of the 7-DOF Barrett WAM Arm. The blue arrows depict the rotation axes of the joints. For our experiments we actuated the 4th and the 7th joint, while the other joints were kept in their zero position.

2.2 Parametric Representation of a Stroke

The first and most intuitive parametric representation of the stroke was a simple sinusoidal movement in the 4th joint (elbow) of the robot. The parameters of this movement consisted only of the amplitude A of the sine and its period T. The trajectory of the 4-th joint for this movement is outlined in Figure 2. However, this representation did not lead to good results.

The second approach we followed was a slightly more sophisticated representation. That part of the movement, in which the ball is hit, is still sinusoidal trajectory with amplitude A and period T. Additionally we introduced a delay phase with the parameter d before, and a return phase after the stroke phase. So the total movement is composed of a delay phase, a sinusoidal

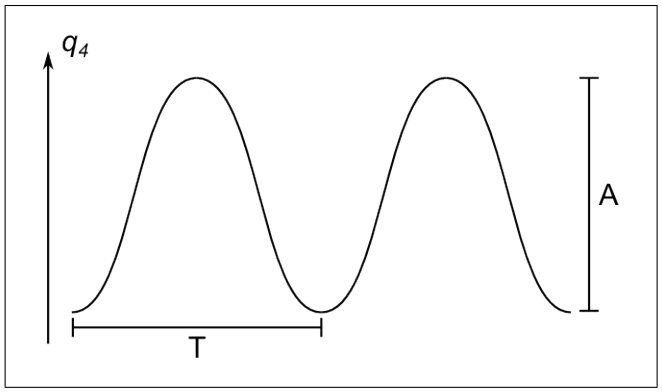

Figure 2: First approach of a parametric representation of a stroke: sinusoidal trajectory of joint 4 (elbow) with amplitude A and period T.

hitting phase, and a fixed return phase in which the robot arm returns to its initial position. The delay phase starts when Ball has reached its peak position. The trajectory of the elbow joint of this representation is depicted in Figure 2.

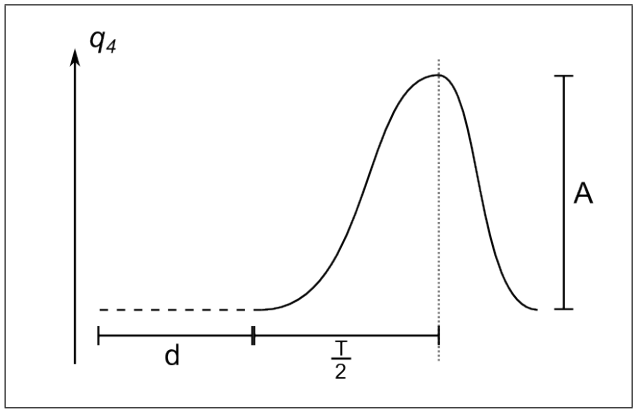

Figure 3: Second approach of a parametric representation of a stroke: The movement consists of three phases: 1) a delay phase with parameter d, in which the arm rests at its initial position. 2) the hitting phase in which the arm follows a sinusoidal trajectory with amplitude A and period T in joint 4 to hit the ball.

We will show in Section 4, that we were able to achieve good results with this representation.

2.3 PD Control

Additionally to the stroke movement in joint 4 (elbow) of the robot, we learned a PD controller (proportional-derivative controller) for the 7th joint, to react on movements of the ball in x-direction. A PD controller has the two parameters k_p and k_d which are applied

to the error and the error's derivative, respectively:

$$u = k_p \cdot e + k_d \cdot \dot{e}. \tag{1}$$

The error e is the difference between a desired position x_d and the ball's position x_b, analogously the error's derivation is the difference between a desired velocity $\dot{x}_d$ and the ball's velocity $\dot{x}_b$ (all of them are only the x-components of positions and velocities):

$$u = k_p(x_d - x_b) + k_d(\dot{x}_d - \dot{x}_b). \tag{2}$$

For the ball bouncing task the desired position of the ball is $x_d = 0$, which is centered above the racket, and it has a velocity of $\dot{x}_d = 0$.

For the ball serving task the PD controller was not used. Instead the robot selected a fixed angle for the wrist joint throughout the trajectory. Using this joint it is able to hit the ball in different x directions.

2.4 The Learning Method

In reinforcement learning the general setup (Sutton and Barto, 1998) considers an *agent* that interacts with its *environment*. The actions taken by this agent are based on a Markov Decision Process (MDP). Hence, if the agent is in a state $s \in S$ it selects an action $a \in A$ using the *policy* $\pi(a|s)$. The agent then transfers to the next state s' with the transition probability $\mathcal{P}^a_{ss'} = p(s'|s,a)$. This transition yields a reward $r(s,a) = \mathcal{R}^a_s \in \Re$ for the agent.

The goal of reinforcement learning is now to find a policy that maximizes the expected reward of the agent

$$J(\pi) = \sum_{s,a} \mu^\pi(s)\pi(a|s)\mathcal{R}^a_s \tag{3}$$

Here, $\mu^\pi(s)$ denotes the probability ob the agent being in state s, the state distribution.

Policy search methods are one group of methods to maximize the expected reward by directly searching for an optimal policy. However, one drawback of most of these methods is they take only the experience of the most recent trials into account for computing the new policy. Hence, there is a loss of information from older policy evaluations during the policy improvement step.

2.4.1 Relative Entropy Policy Search

To circumvent the problem of loss of information when directly optimizing the policy, Peters et al. (2010) proposed the *relative entropy policy search* (REPS) method.

The objective function here is again the expected reward, which they want to be maximized. But additionally the *Kullback-Leibler divergence* or "*relative entropy* between the observed data distribution $q(s,a)$ and the data distribution $p^\pi(s,a) = \mu^\pi(s)\pi(a|s)$" (Peters et al., 2010) is constraint to an upper bound ε:

$$D(p^\pi||q) = \sum_{s,a} \mu^\pi(s)\pi(a|s) \log \frac{\mu^\pi(s)\pi(a|s)}{q(s,a)} \leq \varepsilon. \tag{4}$$

Together with the assumption of a *stationary state distribution* $\mu^\pi(s)$ and the constraint that probability distributions must sum to 1, they get the following problem statement:

Problem Statement. *The goal of relative entropy policy search is to obtain policies that maximize the expected reward $J(\pi)$ while the information loss is bounded, i.e.,*

$$\max_{\pi,\mu^\pi} J(\pi) = \sum_{s,a} \mu^\pi(s)\pi(a|s)\mathcal{R}^a_s \tag{5}$$

$$s.t. \qquad \varepsilon \geq \sum_{s,a} \mu^\pi(s)\pi(a|s) \log \frac{\mu^\pi(s)\pi(a|s)}{q(s,a)} \tag{6}$$

$$\sum_{s'} \mu^\pi(s')\phi_{s'} = \sum_{s,a,s'} \mu^\pi(s)\pi(a|s)\mathcal{P}^a_{ss'}\phi_{s'} \tag{7}$$

$$1 = \sum_{s,a} \mu^\pi(s)\pi(a|s) \tag{8}$$

Both μ^π and π are probability distributions and the features $\phi_{s'}$ of the MDP are stationary under policy π.

3 Learning the Stroke

As we described in Section 2.2, the stroke movements are abstracted using a parametric description. Due to this simplification we do not have any states – or, from another point of view, we are always in the same state, when executing the movement. Thus we can simplify the REPS problem statement to the following form:

Problem Statement. *Maximize the expected reward $J(\pi)$ while the loss of information is bounded, i.e.,*

$$\max_{\pi} J(\pi) = \sum_{a} \pi(a)\mathcal{R}^a \tag{9}$$

$$s.t. \qquad \varepsilon \geq \sum_{a} \pi(a) \log \frac{\pi(a)}{q(a)} \tag{10}$$

$$1 = \sum_{a} \pi(a) \tag{11}$$

From this simplified problem we can then derive the policy update

$$\pi(a) = \frac{q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right)}{\sum_a q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right)}, \tag{12}$$

with the Lagrangian parameter η which we obtain from the minimization of the dual function

$$g(\eta) = \eta \log \left(\sum_a q(a) \exp \left(\frac{\mathcal{R}^a}{\eta} \right) \right) + \eta\varepsilon. \tag{13}$$

As we are using policy iteration the data distribution $q(a)$ is inherent in the distribution of the samples from the old policy. Hence the sample-based policy update becomes a weighted maximum likelihood estimation with weight w_i for the i-th sample:

$$w_i = \frac{\exp\left(\frac{r_i}{\eta}\right)}{\sum_{i=1}^N \exp\left(\frac{r_i}{\eta}\right)}, \tag{14}$$

where r_i is the reward received by the i-th sample of the policy. The dual function becomes then

$$\hat{g}(\eta) = \eta \log \left(\frac{1}{N} \sum_{i=1}^N \exp\left(\frac{r_i}{\eta}\right) \right) + \eta\varepsilon, \tag{15}$$

with the number of samples N. The learning algorithm is outlined in Algorithm 1.

Policy Iteration with Adapted REPS

Input: maximal information loss ε, initial policy $\pi_0(a)$, number of iterations N, number of samples M.

for $k \leq N$ **do**

Sampling:: Draw M samples a_i from the policy $\pi_k(a)$.

Critic: *Evaluate policy*

foreach *sample* $a_i, i \in \{1, \ldots, M\}$ **do**

Perfrom experiment with sample a_i to obtain reward r_i.

Compute the Dual Function:

$$\hat{g}(\eta) = \eta \log \left(\frac{1}{N} \sum_{i=1}^N \exp\left(\frac{r_i}{\eta}\right) \right) + \eta\varepsilon$$

Compute the Dual Function's Derivative:

$$\frac{\partial \hat{g}(\eta)}{\partial \eta} = \log \left(\frac{1}{N} \sum_{i=1}^N \exp\left(\frac{r_i}{\eta}\right) \right) - \frac{1}{\eta} \frac{\sum_{i=1}^N r_i \exp\left(\frac{r_i}{\eta}\right)}{\sum_{i=1}^N \exp\left(\frac{r_i}{\eta}\right)} + \varepsilon$$

Optimize: $\eta^* = \texttt{fmin\_BFGS}(\hat{g}, \partial\hat{g}, \eta_0)$.

Actor: *Improve policy*

New Policy is weighted maximum likelihood estimation of samples with weights

$$w_i = \frac{\exp\left(\frac{r_i}{\eta^*}\right)}{\sum_{i=1}^N \exp\left(\frac{r_i}{\eta^*}\right)}$$

Algorithm 1: Policy Iteration with adapted REPS. `fmin_BFGS` stands for the Broyden-Fletcher-Goldfrab-Shannon (BFGS) optimization method.

4 Experiments

To evaluate REPS for learning motor skills we ran three experiments. In the first experiment we compared the performance of REPS with that of finite differences with RPROP on the ball serving task. In the second experiment we evaluated the effect of different values for ε on the performance of REPS. In the third experiment the robot learns ball bouncing with a feedback controller to compensate for errors.

4.1 Comparison with Finite Differences

In this experiment the robot was given the task of serving the ball to three different locations:

$$x_g = [0, -1.5]$$

$$x_g = [1, -2]$$

$$x_g = [2, -1]$$

Each target location was evaluated three times with both methods. For comparison the robot used REPS and finite differences with RPROP. For REPS the ε was set to 1. For finite differences with RPROP the initial step size was 0.005. The step size was increased by a factor of 1.2 when the gradient direction stayed the same and decreased by a factor of 0.5 when the gradient flipped. Both methods were given 50 iterations for each task and 15 samples per iteration. The results of the experiments are shown in Figure 4.

As can be seen by the plots both methods started with the same performance as they were initialized with the same parameters. However REPS quickly converged to the target location. In comparison the finite differences method converged more slowly. The solutions found by finite differences tended to perform worse than those of REPS. This is due to finite differences often ending in a local maximum where it simply lets the ball drop to the ground. At this point there is no longer a gradient and the method stops learning. This experiment shows that REPS is more robust to getting stuck in such a local maximum.

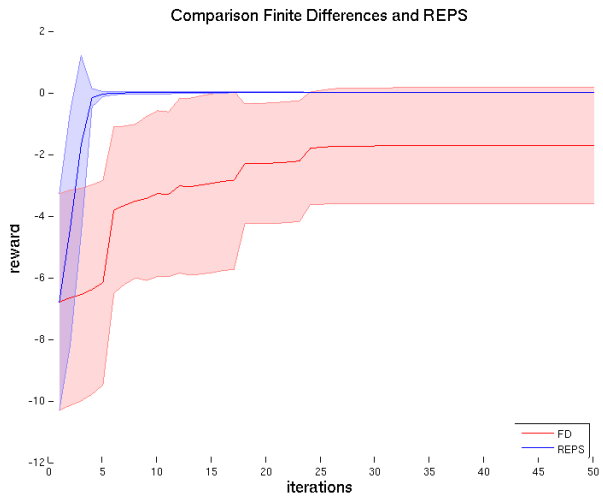

Figure 4: Comparison Finite Differences with RPROP and REPS

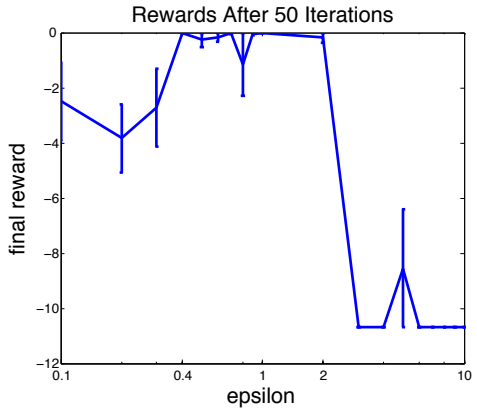

Figure 5: Effect of different ε values

4.2 Effect of different values for ε

In the second experiment we investigated the effects of changing the main parameter of the REPS method. The evaluation was again performed using the ball serving task. In this case the ball was always served to the same target location.

$$x_g = [-2, -2]$$

As a performance measure we looked at the rewards achieved after 50 iterations. Again each iteration consisted of 15 samples. Each ε value was tested five times. The results of the experiment are shown in Figure 5.

As can be seen the REPS algorithm is able to consistently obtain good final rewards when using ε values in the range 0.4 to 2. Using higher ε values seems to have led to numerical instabilities and the method does not learn. For smaller ε values the performance gradually decreases. The wide peak of performance indicates that the performance of REPS is not sensitive to ε and can be easily tuned.

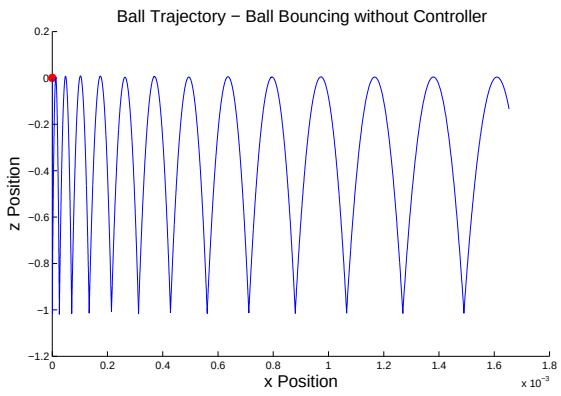

Figure 6: Ball trajectory without feedback controller. A small error accumulates and would eventually lead to the ball falling off the paddle.

4.3 Learning Ball Bouncing

In the final experiment the robot was given the task to robustly perform the ball bouncing task. First the robot learned a standard hitting movement where the ball was dropped from the initial position as usual. To learn this hitting motion REPS was used with 50 iterations, $\varepsilon = 1$ and 15 samples per iteration. The learned movement was concatenated into a periodic motion according to the time it took the ball to get back to the initial position. The resulting ball trajectory can be seen in Figure 6.

A small error accumulated in the x direction which would eventually result in the ball falling off the paddle. We therefore also learned a PD controller based on the balls position and velocity in the x direction as already described in section (2.3). To learn the gains of the PD controller we again used REPS for 50 iterations. The resulting controller was evaluated by initially dropping the ball at 2.5 cm increments across the width of the racket. In each trial the robot successfully managed to bounce the ball back to center of the racket and keep bouncing it there. As an additional test we evaluated the situation where the ball is thrown near the edge of the paddle with a horizontal speed of 1.25 m/s. The resulting ball trajectory is shown in Figure 7.

The figure shows that even in this extreme situation the learned controller was still able to compensate for the error and successfully perform the ball bouncing task.

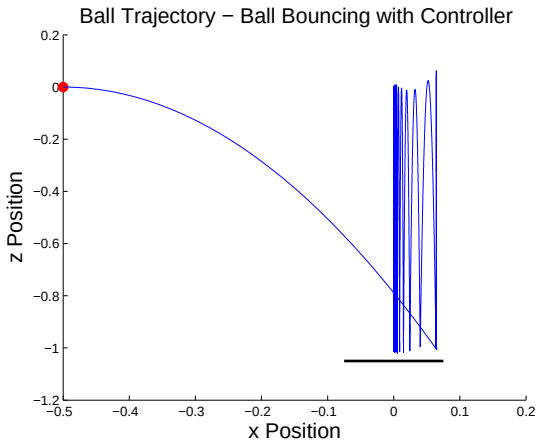

Figure 7: Ball trajectory with feedback controller. The ball is thrown onto the paddle with a speed of 1.25 m/s. The width of the paddle is marked with a black line. The robot is able to slowly bring the ball to the center of the paddle.

5 Conclusion

In this paper we investigated using the Relative Entropy Policy Search algorithm for learning robot motor skills. We looked at the tasks of ball serving and ball bouncing. Our experimental results show that REPS is robust to getting stuck in local maxima. REPS is based on bounding the information loss between policies by a value ε. We discovered that the performance of the algorithm is not sensitive to this parameter which means it can be easily set. Using the REPS algorithm the robot was able to learn to serve the ball to various target locations and robustly perform the ball bouncing task. The learned controller was even able to compensate for the ball being thrown onto the paddle with a horizontal velocity of 1.25 m/s.

In the future we would like to use a controller based on inverse kinematics, such that the paddle can easily be rotated around the x axis. Using such a controller the PD feedback controller could also be learned for the y direction similar to how the robot learns it now in the x direction. We also plan to investigate using the state dependent version of REPS to directly learn to serve the ball to different target locations.

References

C. Daniel, G. Neumann, and J. Peters. Hierarchical relative entropy policy search. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS 2012)*, 2012a.

C. Daniel, G. Neumann, and J. Peters. Learning concurrent motor skills in versatile solution spaces. In *Proceedings of the International Conference on Robot Systems (IROS)*, 2012b.

Jan Peters, K Mülling, and Yasemin Altun. Relative entropy policy search. *National Conference on Artificial Intelligence*, 2010.

Stefan Schaal. The sl simulation and real-time control software package. *Processing*, pages 1–94, 2007.

Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 0262193981.

Derivation of adapted REPS

Lagrangian of the program in Equations (9-11):

$$\begin{aligned} L &= \sum_a \pi(a)\mathcal{R}^a + \eta\left(\varepsilon - \sum_a \pi(a)\log\frac{\pi(a)}{q(a)}\right) \\ &\quad + \lambda\left(1 - \sum_a \pi(a)\right) \\ &= \sum_a \pi(a)\left[\mathcal{R}^a - \eta\log\frac{\pi(a)}{q(a)} - \lambda\right] + \eta\varepsilon + \lambda \end{aligned} \tag{16}$$

Differentiate the Lagrangian with respect to $\pi(a)$:

$$\begin{aligned} \frac{\partial L}{\partial \pi(a)} &= \left(\mathcal{R}^a - \eta\log\frac{\pi(a)}{q(a)} - \lambda\right) + \pi(a)\left(-\eta\frac{1}{\pi(a)}\right) \\ &= \mathcal{R}^a - \eta\log\frac{\pi(a)}{q(a)} - \lambda - \eta \end{aligned} \tag{17}$$

Set to zero and solve for $\pi(a)$:

$$\eta\log\frac{\pi(a)}{q(a)} = \mathcal{R}^a - \lambda - \eta \tag{18}$$

$$\pi(a) = q(a)\exp\left(\frac{\mathcal{R}^a}{\eta} - \frac{\lambda}{\eta} - 1\right) \tag{19}$$

$$= q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right)\exp\left(-1 - \frac{\lambda}{\eta}\right) \tag{20}$$

Since we require that $\sum_a \pi(a) = 1$, we can sum up both sides of Equation (20) over a and obtain:

$$1 = \sum_a q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right)\exp\left(-1 - \frac{\lambda}{\eta}\right)$$

$$1 = \exp\left(-1 - \frac{\lambda}{\eta}\right)\sum_a q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right)$$

$$\exp\left(-1 - \frac{\lambda}{\eta}\right) = \left(\sum_a q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right)\right)^{-1} \tag{21}$$

If we insert Equation (21) into Equation (20) we get:

$$\pi(a) = \frac{q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right)}{\sum_a q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right)} \tag{22}$$

We can now replace $\pi(a)$ in the Lagrangian (Equation (16)) using Equation (22) and obtain the dual function:

$$\begin{aligned} L = &\sum_a \frac{q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right)}{\sum_a q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right)} \\ &\cdot \left[\mathcal{R}^a - \eta\log\frac{q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right)}{q(a)\sum_a q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right)} - \lambda\right] \\ &+ \eta\varepsilon + \lambda \end{aligned} \tag{23}$$

For the sake of limited space the following equations show the simplification of only the term in the big square brackets of Equation (23):

$$\begin{aligned} &\left[\mathcal{R}^a - \eta\log\frac{q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right)}{q(a)\sum_a q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right)} - \lambda\right] \\ &= \left[\mathcal{R}^a - \eta\frac{\mathcal{R}^a}{\eta} + \eta\log\sum_a q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right) - \lambda\right] \\ &= \left[\eta\log\sum_a q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right) - \lambda\right] \end{aligned} \tag{24}$$

As the term in Equation (24) is now not dependent on the running variable of the outer summation in Equation (23) anymore, we can pull it out of that sum and thus

$$\sum_a \frac{q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right)}{\sum_a q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right)} = 1. \tag{25}$$

Applying the simplifications of Equations (24) and (25) to the Lagrangian in Equation (23), we obtain the dual function as follows:

$$\begin{aligned} g(\eta) &= \eta\log\sum_a q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right) - \lambda + \eta\varepsilon + \lambda \\ &= \eta\log\sum_a q(a)\exp\left(\frac{\mathcal{R}^a}{\eta}\right) + \eta\varepsilon \end{aligned} \tag{26}$$