Symphony: A Heuristic Normalized Calibrated Advantage Actor and Critic Algorithm in application for Humanoid Robots

Timur Ishuov[1], Michele Folgheraiter[2], Madi Nurmanov, Goncalo Gordo[3], Richárd Farkas[1], József Dombi[1]

[1]Department of Computer Algorithms and Artificial Intelligence, University of Szeged, Szeged, Hungary
[2]Department of Robotics and Mechatronics, Nazarbayev University, Astana, Kazakhstan
[3]Tinker AI Limited, London, UK
Emails: timur@inf.u-szeged.hu, michele.folgheraiter@nu.edu.kz, tinkerai.project@gmail.com
rfarkas@inf.u-szeged.hu, dombi@inf.u-szeged.hu

Abstract—In our work we not explicitly hint that it is a misconception to think that humans learn fast. Learning process takes time. Babies start learning to move in the restricted liquid area called placenta. Children often are limited by underdeveloped body. Even adults are not allowed to participate in complex competitions right away. However, with robots, when learning from scratch, we often don't have the privilege of waiting for dozen millions of steps. "Swaddling" regularization is responsible for restraining an agent in rapid but unstable development penalizing action strength in a specific way not affecting actions directly.

The Symphony, Transitional-policy[1] Deterministic Actor and Critic algorithm, is a concise combination of different ideas for possibility of training humanoid robots from scratch with Sample Efficiency, Sample Proximity and Safety of Actions in mind. It is no secret that continuous increase in Gaussian noise without appropriate smoothing is harmful for motors and gearboxes. Moreover, an added noise trail, when actions are clipped, can accumulate at the action limits, potentially increasing the frequency of maximum control values (torque, velocity, angles). Compared to Stochastic algorithms, we use a different approach, we set a limited parametric noise and adjust the strength of actions instead of noise. Promoting a reduced strength of actions, we safely increase entropy, since the actions are kind of immersed in weaker noise. When actions require more extreme values, actions rise above the weak noise and become more deterministic. Training becomes empirically much safer for both the environment around and the robot's mechanisms.

We use Fading Replay Buffer: using a fixed formula containing the hyperbolic tangent, we adjust the batch sampling probability: the memory contains a recent memory and a long-term memory trail. Fading Replay Buffer with increased Update-To-Data ratio (3) allows us to use Temporal Advantage when we improve the current Critic Network prediction compared to the exponential moving average. Temporal Advantage allows us to update Actor and Critic in one pass, as well as combine Actor and Critic in one Object (Class) and implement their Losses in one line. The Symphony algorithm is called Heuristic[2] and Calibrated, since the activation, loss, regularization functions were rewritten and parameters were customized during numerous experiments in order to bring safe model-free imitation-less Humanoid robot training without virtual simulations closer to reality.

[1]algorithm in between on and off policy due to balanced updates
[2]a hypothetical approach of combining different methods and using specific parameters based on intuition and experience to solve practical problems

Index Terms—sample-efficient, model-free, off-policy, advantage, reinforcement learning

I. Preliminary

We consider Agent or Actor that uses a Policy A_ϕ or Actor Network with parameters ϕ which at a Continuous Markov Decision Process (MDP) State (hereinafter referred as simply state) s_t produces a Continuous Vector of Actions (or actions) $a_t + \epsilon \in [-a_{max}, a_{max}]$. The Environment responds with reward r_t and the Agent appears at the Next MDP State (next state) s_{t+1}. We use Critic, Critic Network and Q Network terms interchangeably, while Critic's parameters are abbreviated by θ, and det or * abbreviation is used for values detached from the computation graph to compute gradients.

II. Introduction

The Symphony algorithm aims to solve two conflicting problems of Model-Free Imitation-Less Reinforcement Learning: Sample Efficiency and harmonious agent movements without jerks, which is achieved with the help of Sample Proximity and Safety of Actions. On-policy algorithms like Proximal Policy Optimization [1] (PPO) solved the problem of harmonious agent movements using a small "safe" gradient step, which leads to a larger number of close samples and better policy consequently. But PPO has proven itself in simulations, where you can parallelize the learning process over many virtual environments, namely Graphic Processing Units or Tensor Processing Units, and make a gradient step based on a larger number of roll-outs, so this process requires a careful transition from simulation to a real robot, though no bootstrapping (updating Online Q Network based on prediction of its delayed variant) is used. When it comes to training on a real robot from scratch, entropy regularization based off-policy sample-efficient algorithms like Soft Actor and Critic [2] (SAC) and its advanced derivatives are often used, though this family of algorithms can suffer from prediction inaccuracies of the TD equation; the regularization mitigates this issue to some degree.

To achieve even higher Sample-Efficiency of the SAC algorithm, recent studies (SAC-20, Randomized Ensembled Double Q learning or RedQ [3]) have concluded that it is possible to do a larger number of updates-per-frame. But the minimization between two prediction functions of the future aggregated reward min(Q1, Q2) [4] at an increased update frequency increases the probability of stagnation of the policy or Actor Critic at a local extremum due to the constantly decreasing step of min(Q1, Q2); Random Q network selection from Q network ensemble (RedQ algorithm) helps against this. The Aggressive Q learning with Ensembles [5] (AQE) algorithm instead of min(Q1, Q2) uses the idea of Truncated Quantile Critics [6] (TQC) which is about taking the average in a larger number of Q heads or nodes (Q distribution) after truncating several highest Q nodes. Truncation is necessary due to the overestimation and high variability of the average value (for Walker-2d this is up to half of the nodes). This step allows achieving highest scores. Another work (Reset [7] algorithm) suggests leaving the minimization but refreshing the weights of the last two layers to the default ones, periodic refresh allows using a higher update frequency, up to 128.

If we look at the work of Distributional Soft Actor and Critic [8], we can understand that the average value is a kind of "Pandora's box" for humanoid agent movements by increasing the variation and the number of attempts or roll-outs [9]. The point is that when we take the minimum of two functions for predicting the best actions, with time we create a very careful, cautious or reinsurer agent, but when we take the average, we give the agent the opportunity to be relatively optimistic in its predictions and make more mistakes, regarding humanoid agents, this means more falls. But when the agent falls more, this in turn leads to better experience, which results in more harmonious movements and more deterministic $Q = r_{done}$ in the batch sample, which in the Temporal Difference equation provides better predictions in the end. Unfortunately, this takes some time, while introducing a high instability in the learning process - the first version of Distributional Soft Actor and Critic still required several millions of steps. Most likely, because of this, the authors of this algorithm returned to the minimum between two Q distributions [10]

III. FOUNDATION

Symphony is an advantage based transitional-policy deterministic Actor-Critic algorithm with a limited[3] Normal Gaussian noise ($\epsilon \sim a_{max} * 1/e * \mathcal{N}(0.0, 1.0)$, where $\mathcal{N} \in [-e, e]$) added to produced actions a.

Symphony increases Temporal Advantage (TA) or plain difference between a currently predicted target Q value and exponential weighted moving average[4] [11] of target Q values (copy of Q values, Q^*, detached from the gradient computation) with heuristic $\alpha = 0.5..0.7$.

---

[3]The parametric standard deviation of $1/e$ scales Standard Gaussian noise's range to [-1, 1]

[4]the exponential weighted moving average of Q target value has some resembles with Approximate Value Function, though the latter uses return traces R for the average $Q_\pi(s, a) = (1 - \gamma)R_t(s, a) + \gamma V(s_t|\psi)$

TABLE I: Fixed weights for Target Critic's nodes
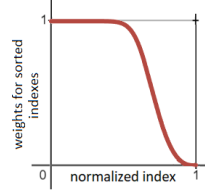
1) $i_n = \frac{\{1,2,3...384\}}{384}$

2) $w_{i_n} = tanh((\pi(1 - i_n))^e)$

3) $w_n = w_{i_n} / \sum_1^{384} w_{i_n}$

$$Q_{target} = \sum_{n=1}^{384}(w_n * q_n)$$

The Symphony's Objective Function can be expressed as:

$$\mathbb{E}_{s_t \sim \rho^\beta}[\nabla_a [Q_{target}(s_{t+1}, a(s_{t+1})) - Q_T] \nabla_\phi a(s_{t+1}|\phi)] \quad (1)$$

where $Q_T = \alpha \bar{Q}_{T-1} + (1 - \alpha) Q^*_{target}$.

In Symphony, instead of ensemble of 5-10 networks or only 1-2 Critics, we take 3 Critics each of which outputs 128 nodes and concatenate output nodes together producing a distribution of size 384. For the output of the Target Critic, we sort the distribution from minimum to maximum and multiply those values by fixed weights that are opposite to the Fading Replay Buffer probabilities (which will be discussed further) though for the length of 384. The weighted sum of 384 Critic's nodes becomes the output of the Target Critic.

Though, Target Critic based Actor's updates can slow down training due to Polyak Averaging [12] [13] via soft updates ($\theta^T \leftarrow \tau\theta + (1-\tau)\theta^T$) [14], Advantage or ΔQ gives us some independence from the actual Q values, and together with the increased update-to-data ratio $\mathcal{G}$ (Symphony uses 3 updates per frame) this method provides a "seamless" drastically cheap update of the Actor-Critic. For this we predict next action a'_{t+1} using next state and Online Actor ($a'_{t+1} = A_\phi(s_{t+1})$), then use next state and next action to produce next target Q value and its detached variant:

$$Q_{target}, Q^*_{target} = Q_{target}(s_{t+1}, a'_{t+1}) \quad (2)$$

which are used to update simultaneously Online Actor and Online Critic using Temporal Advantage and Temporal Difference [15], respectively:

$$-L_\phi(\frac{Q_{target} - Q_T}{|Q_T|}) + L_\theta(\widehat{r_t} + \gamma Q^*_{target} - Q_{online}) \quad (3)$$

Simultaneous update of Online Networks becomes possible when we unite Actor and Critic into a single Class, or unified Object consequently. We do not utilize Target Actor, while the Target Critic is updated using Polyak Averaging as previously mentioned.

Equation revision ...

One can notice $|Q_T|$ in the denominator of TA, which is used for Advantage normalization, while $\widehat{r_t}$ are normalized

rewards. Normalization is done using $\bar{r}_n$, a mean of absolute values of the exploratory rewards r_{exp} collected during $N^{exp} = 10,240$ exploration steps:

$$\bar{r}_n = \frac{1}{N_{exp}} \sum_{1}^{N_{exp}} |r_{exp}| \qquad (4)$$

Normalization is done after exploration and further using estimated $\bar{r}_n$.

... end of revision

As the Replay Buffer grows in size, sampling evenly with a small batch size (e.g., 32–64) may lead to Q-values predicted from one batch having a very weak correlation with Q predictions from the other. To address this, we use the previously mentioned exponential smoothing, an increased batch size of 384, and a Fading Replay Buffer, where sampling priorities p correspond to the normalized indexes of the transitions i (normalized by the capacity of the Replay Buffer, N_{rb}).

More precisely and preferably, we smoothed linear priorities with $tanh((\pi i)^e)$ giving a smooth transition (by applying π to the input of the hyperbolic tangent function, we compress the function's transition region closer to the 0..1 range) and plateau like shape for the latest experience (order e). To obtain probabilities that add up to 1.0 we further divide weights/priorities by a sum of all weights/priorities.

TABLE II: Probabilities for the Fading Replay Buffer

$$\text{1)} \ i_n = \frac{\{1,2,3...N_{rb}\}}{N_{rb}}$$

$$\text{2)} \ w_{i_n} = tanh((\pi i_n)^e)$$

$$\text{3)} \ p = \frac{w_{i_n}}{\sum_{1}^{N_{rb}} w_{i_n}}$$
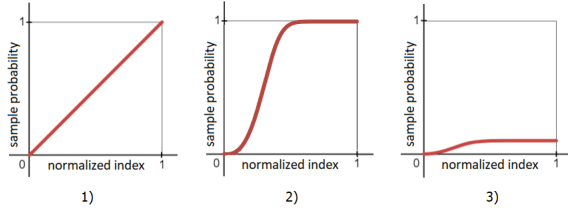
Finally, we repeat 10,240 Exploration transitions 50 times, creating a fully filled Replay Buffer of length 512,000 from the beginning. In this way we ensure that all Exploration transitions will be thoroughly processed before disappearing. After the Replay Buffer is filled, a new transition enters it at the last index. Then we do roll or shift left operation so that the first (oldest) transition appears at the last index and is ready to be overwritten. When we encounter a transition that contains $\widehat{r}_t = r_{done}$ at the oldest placeholder, we do shift left 2 positions instead of 1, so that Terminal points are never overwritten. More than that, outside of Replay Buffer, when agent learns to move without falling or approaches an artificial termination limit, e.g. 1000, at a timestep close to termination

(950) we zero actions a_t and leave only noise, so that the agent falls deliberately. This resembles human memory which remembers pivotal points and makes the Actor-Critic Network less dependent on Bootstrapping.

The Fading Replay Buffer is an intermediate step between On-policy Cache type Memory and Off-policy Experience Replay [16]. It smooths difference between both memories since actions that were done far in the past have a little relevance to the current policy, but it is also important to improve Q from those old "bad" transitions for better generalization. Also Fading Replay Buffer diminishes a transition problem of Experience Replay as a vanilla Experience Replay is a Stack type memory until it reaches its capacity then it is a Double-ended queue or a Standard Moving Average.

Temporal Advantage and Fading Replay Buffer introduce high sample efficiency which can lead to the fast growth in Q learning. However, if more closely related experiments lead to more harmonious agent movements but requires big amount of samples, then how to achieve Sample Proximity and Sample Efficiency if these two goals contradict each other?

It may sound unobvious but encouraging the agent not to use the full range of motion, "Swaddling" or Control Cost punishment, can help.

Before we learned to run, we learned to walk, before we learned to walk, we learned to stand. If a small child is not swaddled and not kept in a crib, he can cause a lot of trouble to himself and to the Environment. How is swaddling different from "caution" in predictions? In that variability is preserved, but in a small range. This means that before learning to run like Usain Bolt, our agent must learn to move its legs correctly.

If we look at the reward formation in the Mujoco Humanoid-v4 environment, we subtract the sum of squares of actions from the positive direction of speed. That is, we punish the agent for the power or strength of actions – stronger the actions much stronger the punishment, although the weight of this punishment (control cost) is 0.1, and the weight of the speed reward is 1.25, and the healthy state is 5. It turns out that the relative "Swaddling" is embedded in the reward. But the weight value for the control cost in Walker-2d is only 0.001. When we do a comparative analysis using these environments, the environments themselves are a black box for us, and we cannot influence the internal parameters of the environment (which does not apply to developing our own robot) What is "Swaddling" regarding the algorithm? In the simplest implementation, it is a direct soft suppression of bigger actions by adding control cost to the Actor's Loss Function:

$$ctrl\_cost = \beta \, a^2 \qquad (5)$$

where β is a temperature parameter, a - actions. a^2 in Loss function will reduce the strength of the actions depending on the β coefficient. The β's strength should be relatively small, otherwise we unlikely to learn deterministic moves. Adjusting β, we facilitate actions that can be viewed as buried in limited Gaussian Noise increasing Entropy Level.

The difference with a learnable standard deviation is that Noise Level is set constant, it will never go to unprecedented values.

With the introduction of a new harmonic activation function that naturally induces fluctuations in the output and loss functions that allow for a less strict penalization of stronger values, and using the principles behind these functions, instead of adding the control cost, we introduced a new approach called "Swaddling" which does not affect actions directly.

Harmonic activation functions are rarely used in Reinforcement Learning due to high level of instability. But if stabilized they can introduce infinite exploration in a more natural way than hard resetting neural networks introduced in Reset mechanism [7] as small input signal can cause some oscillatory effect (as in the muscular system of animals).

We use internal Sine wave function in between two Linear Layers both in Actor and Critic, which represents an approximation to a Fourier Series in the form of: $Asin(wx+b)+c$, where an absence of the Cosine part is to some extent covered by the presence of a learnable phase shift. A harmonic and non-linear nature of a Sine wave introduces instability and consequently a broader search for solutions in the learning process. We rectified those instabilities by a Swish or SiLU activation function with a weight factor of 1.5.

During experiments, we slightly decreased the scale of the Rectified Sine function, which showed better balance between exploration and exploitation. However, the best choice was to implement proportional scaling (σ) using a trainable vector s with a length equal to the hidden dimension (initialized uniformly between -$\frac{1}{\sqrt{dim}}$ and $\frac{1}{\sqrt{dim}}$).

After this vector is squashed between 0 and 1 using a sigmoid function, it is then used to proportionally scale the activation function. Apart from stabilization, rectification additionally provides pseudo-random harmonic dropouts at rectified areas, while noise produced in the form of harmonics is visually closer to the movements of living beings.

TABLE III: Rectified Sine or ReSine

1) $\sigma = \text{sigmoid}(s)$

2) $f(x) = \sigma \sin\left(\frac{x}{\sigma}\right)$

3) $F(x) = f(x) \text{ sigmoid}\left(\frac{1.5}{\sigma} f(x)\right)$

We discussed Rectified Huber Symmetric $x\,tanh(x)$ and Asymmetric $|x|\,tanh(x)$ Error (ReHSE, ReHAE) functions
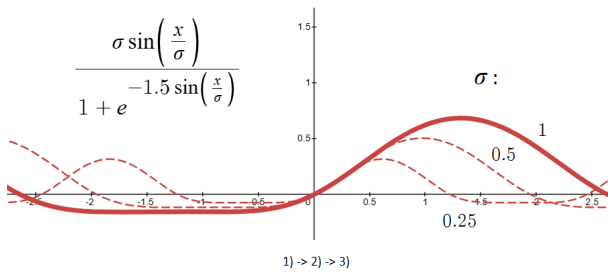
previously [17] [18]. x is the error between prediction and target for the Loss function (ReHSE) or prediction and baseline for the Advantage function (ReHAE).

They have almost quadratic relationships x^2 for relatively small values, but the functions strive for linearity for big values as tanh approaches 1 for bigger values. The difference with the Huber loss function is that transition happens gradually. Symmetric version ($x\,tanh(x)$) is used to increase stability in Critic learning, while faster learning can be achieved when the Asymmetric version ($|x|\,tanh(x)$) is used in conjunction with Advantage for Actor's updates. For the Advantage it is important to keep the sign intact.

Delta or Advantage utilizes gradual change from quadratic to linear relationship most of the time, whereas Q value prediction even if it starts from low values initially, rapidly goes to stronger values ($\gg 1.0$) for positive rewards. The benefit of quadratic relationship at smaller values is a dumping effect of small deterministic gradients. Additionally, we made ReHSE and ReHAE closer to the Huber representation by dividing the argument under tanh by 2: $xtanh(x/2)$ and $|x|tanh(x/2)$.

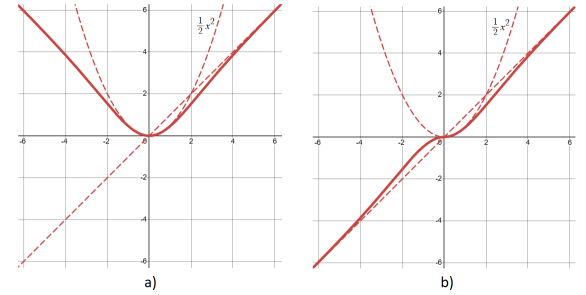Fig. 1: a) x*tanh(x/2) - Rectified Huber Symmetric Error, b) abs(x)*tanh(x/2) - Rectified Huber Asymmetric Error.

The expression $tanh(x/2)$ can also be used for a sparser squashing of actions before adding noise. And when noise is introduced, a more linear squashing using $tanh(x)$ instead of action clipping can help prevent noise accumulation at the action limits. At the action edges, Gaussian noise will become slightly skewed toward the center with a smaller distribution due to the applied tanh function. Considering action regularization, both processes can be combined into the representation shown in Figure 2, where $N = 1/e*\mathcal{N}(0.0, 1.0), \mathcal{N} \in [-e, e]$.
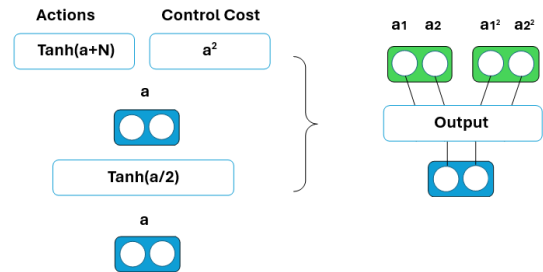

Fig. 2: "Control Cost" regularization.

Unfortunately, the direct relationship between "Control Cost" regularization and actions can impose stronger constraint to the learning process, thus the temperature β should be infinitely small.

Decoupling action regularization from actions

We can use a relatively similar approach that we used in proportional scaling for the ReSine activation function. We can limit the action scaling using the limiting vector s. In this case, the Actor Network will output two independent parameters a and s, both with the action dimension. We can process s through the same $tanh(x/2)$ function. The only thing we need to do is to find mean over absolute values of vector s (the average counteracts overfitting), Figure 3.
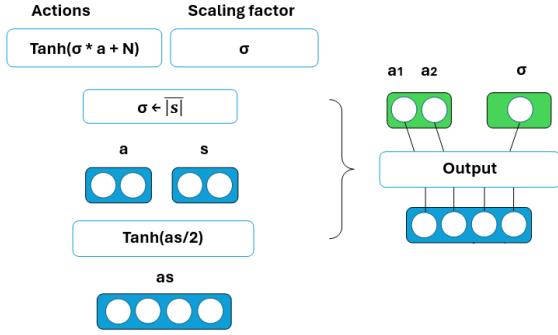
Fig. 3: Action decoupling process

For regularization of σ, we can use inverse tanh (ATanh) with argument $\sigma^{1/\beta}$ creating a barrier regularization for values approaching 1.0. Under regularization ATanh strives for its minimum value of 0.0, while β responds for creation of a symmetric plateau around 0.0. To differentiate it from the control cost (a^2) we referred to it as "Swaddling" (Ω)

We must multiply output actions by a_{max}, for simplicity we omitted this step in the figure. A factor 2 is used for Swaddling, as $2\,ATanh(x)$ is inverse for $Tanh(x/2)$.
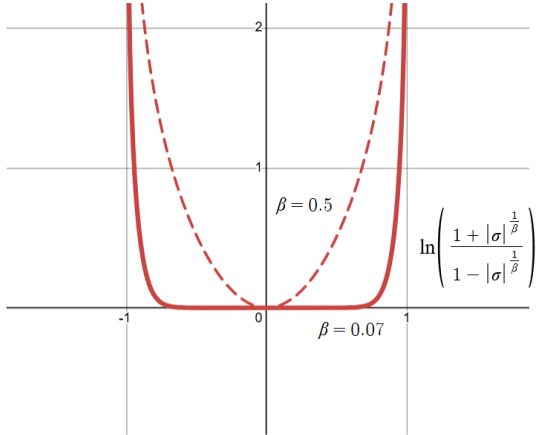

Fig. 4: Swaddling Regularization with β as temperature.

Adjusting β

When the degree $\beta = 0.5$, the function $\Omega(x) = \ln\left(\frac{1+x}{1-x}\right)$ (where $x \leftarrow |\sigma|^{1/\beta}$) resembles a hyperbola near 0.0, while it approaches +inf as σ approaches 1.0. When we decrease β we give a broader range for our scaling factor to operate with, e.g. $\beta = 0.07$.

To increase entropy, the β value should suppress scaling factor values above the standard deviation of noise, but prevent the scaling factor from rolling toward 0.0. To determine β more precisely, we can use a nontrivial approach by adding a helper function $\omega(x) = x\,ln(x)$. This function's minimum is reached at $x = \frac{1}{e}$, precisely the value of the parametric standard deviation. However, if we add this function with the β parameter to align it with ATanh, then for a strong beta, the minimum will shift closer to the center, while we want the scaling factor to float around the standard deviation. The combined equation is difficult to solve mathematically for β, but heuristically, we see that values $\beta \leq 0.05$ provide the correct match (for values smaller than 0.05, the scale value for the function's extremum coincides with 1/e):
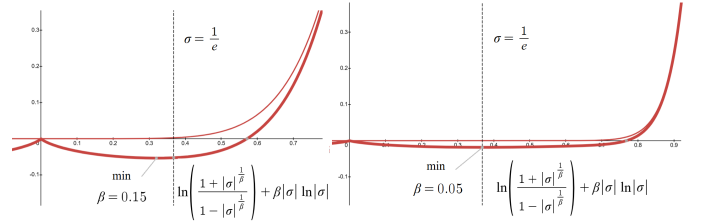

Fig. 5: Approximating β value

For adjustable Beta, it is possible to make it trainable. We repeated the previous approach where the Actor Network outputs several parameters, now σ and β additionally to actions, except we do not find means of absolute values of vectors s and b because σ and β are to balance each other to some extent, though they are clipped between $1e-3$ and $1.0 - 1e-3$:
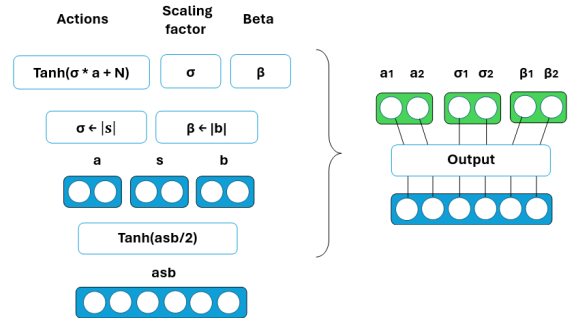

Fig. 6: β as trainable parameter

$2\,ATanh(x)$ is also used to regularize Beta, with the function argument being $\beta^{1/0.5}$ or β^2. A separate helper function for Beta is not needed, since β, by the defined

property, should converge closer to 0.0. While scaling factor σ participates in Q value learning through produced actions, β mostly is a resisting factor. Sending β without power of 2 will result in a higher gradient value directed towards 0.0 which quickly diminishes β; while sending β with a higher order in opposite decreases the gradient and imposes a stronger obstacle for Q value learning through increased β. On the one hand the helper function already steers Beta toward 1.0 (function's minimum), on the other the additional hyperbola shaped regularization suppresses its value (we detach β from the gradient computation for the original Swaddling Function):
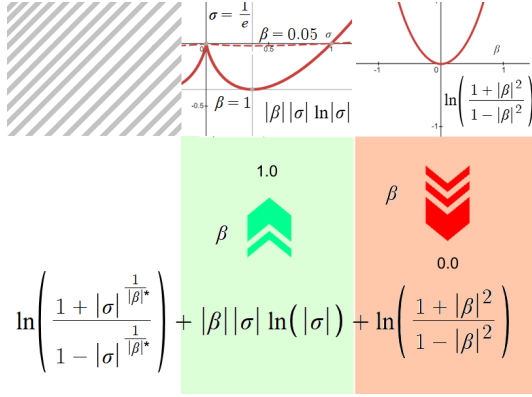
Fig. 7: Full Swaddling Regularization

Hyper-parameters matter

In Symphony algorithm we rigidly preset some constants that can also effect the final performance.

The reciprocal of the golden ratio for the smoothing factor of the TA's exponential moving average $\alpha = \frac{1}{\phi} = \frac{\sqrt{5}-1}{2} \approx 0.62$, reward normalization factor $\bar{r}_n = \overline{|r_{exp}|}$ calculated after $N_{exp} = 10,240$ exploratory steps, which are repeated 50 times to produce a memory of 512,000 length from the beginning.

The Fading Replay Buffer's capacity of $N_{rb} = 50\,N_{exp}$, decay factor $\gamma = 0.99$, delay constant $\tau = 0.005$, Update-To-Data $\mathcal{G} = 3$.

The Actor-Critic network's Optimizer is AdamW [19] [20] with the learning rate $\alpha_{lr} = 1\mathrm{e}{-4}$, less sharp gradient with $\beta 1, \beta 2 = (\alpha, 1 - \tau) \approx (0.62, 0.995)$ and weight decay $\lambda = 0.01$. Performance orientation: we removed Adam's [19] bias correction as it is not essential in the Reinforcement Learning setup; the weight decay complements the main gradient value without a significant alteration of AdamW's principles:

$$\theta_t \leftarrow \theta_{t-1} - \alpha_{lr}\left(m_t/(v_t + \epsilon) + \lambda\,\theta_{t-1}\right) \quad (6)$$

We use Layer Normalization [21] after the first fully connected layer. The hidden layer size h_{dim} is $N_{rb} \div 1000 = 512$ for Actor-Critic inner networks. Each Critic Network has $h_{dim} \div 4 = 128$ output nodes. The batch-size $\mathcal{B} = 384$, the same as the number of concatenated output nodes of the Critic Network.

Each step during and after Exploration we set new seeds for the libraries we work with (namely, PyTorch, numpy, random).

Symphony: Pseudo-code

Initialize Actor-Critic Network (A_ϕ, Q_θ);
Initialize Target Critic Network Q_{θ^T} ($\theta^T \leftarrow \theta$);
Initialize Fading Replay Buffer $\mathcal{F}(N_{rb})$;
Set default $\alpha_{lr}, \gamma, \tau, \mathcal{B}, \mathcal{G}$. Set $\alpha = \frac{1}{\phi}$;
After N_{exp} estimate $\bar{r}_n$ and normalize r_{exp};
Fill $\mathcal{F}$ by repeating N_{exp} transitions 50 times,
  update priorities in $\mathcal{F}$;
foreach *episode* **do**
  foreach *step of episode* **do**
    Take an action a_t, observe r_t, s_{t+1};
    Store i transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{F}$;
    Sample $\mathcal{B}$ (s_t, a_t, r_t, s_{t+1}) transitions
      using priorities from $\mathcal{F}$;
    foreach $\mathcal{G}$ *updates* **do**

$$a'_{t+1}, \sigma_{t+1}, \beta_{t+1} = A_\phi(s_{t+1})$$

$$Q_{\theta^T}, Q^*_{\theta^T} = Q_{\theta^T}(s_{t+1}, a'_{t+1})$$

$$Q_T = \alpha\,\bar{Q}_{T-1} + (1 - \alpha)\,Q^*_{\theta^T}$$

$$L_{\phi,\theta} = -\mathrm{ReHAE}\big(\tfrac{Q_{\theta^T} - Q_T}{|Q_T|}\big)$$

$$+ \mathrm{ReHSE}(\widehat{r}_t + \gamma Q^*_{\theta^T} - Q(s,a))$$

$$+ \Omega\omega\,(\sigma_{t+1}, \beta_{t+1})$$

$$\phi \leftarrow \phi - \alpha_{lr} \frac{\partial \bar{L}_{\phi,\theta}}{\partial\,\phi}$$

$$\theta \leftarrow \theta - \alpha_{lr}\,\frac{\partial \bar{L}_{\phi,\theta}}{\partial\,\theta}$$

$$\theta^T \leftarrow (1 - \tau)\theta^T + \tau\theta$$

    end foreach
    Change random seeds;

  end foreach
end foreach

$'$:	predicted actions	$*$:	detached values
$\bar{r}_n$:	$\overline{\|r_{exp}\|}$	$\widehat{r}_t$:	$r_t/\bar{r}_n$
ReHSE(x):	$x\,tanh(x/2)$	ReHAE(x):	$\|x\|\,tanh(x/2)$
$\Omega(x)$:	$ln(\frac{1+x}{1-x})$	$\omega(x)$:	$x\,ln(x)$
$\Omega\omega(x,\kappa)$:	$\Omega\,(x^{1/\kappa^*}) + \kappa\,\omega(x) + \Omega\,(\kappa^2)$		

Under ReHSE, ReHAE, $\Omega\omega$ functions mean is calculated over batch of size $\mathcal{B}$. Standard Gradient Descent used as a shorthand for customized AdamW optimizer;

After the Replay Buffer is filled with the Exploratory data, a new transition enters it at the last index. Then we do roll or shift left operation so that the first (oldest) transition appears at the last index and is ready to be overwritten. When we encounter a transition that contains $\widehat{r}_t = r_{done}$ at the oldest placeholder, we do shift left 2 positions instead of 1. At a step
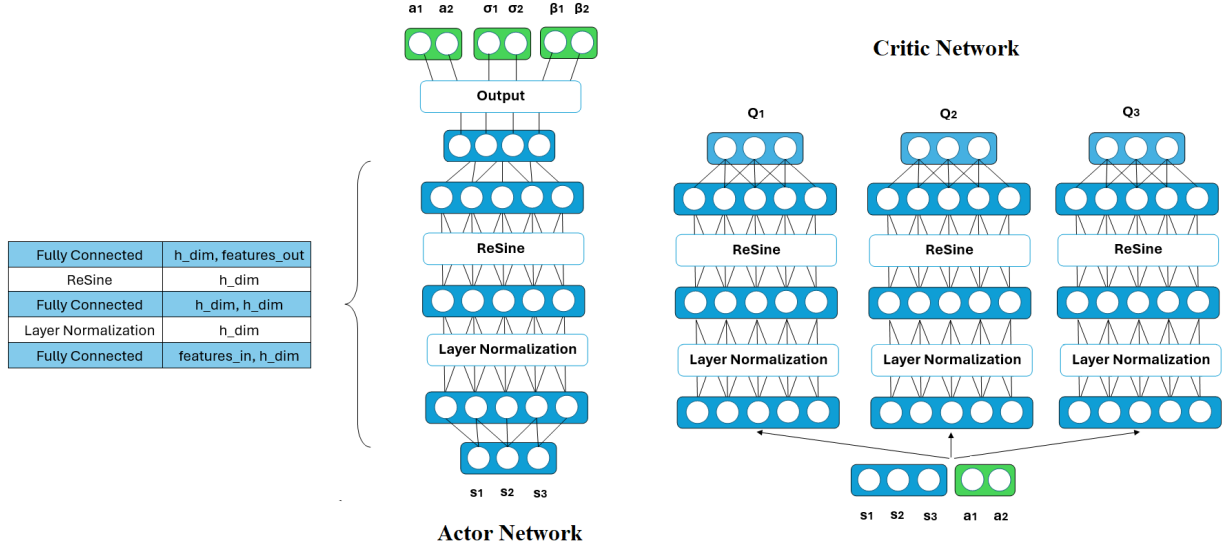
Fig. 8: Architecture of the Actor-Critic network

The table within the figure:

Fully Connected	h_dim, features_out
ReSine	h_dim
Fully Connected	h_dim, h_dim
Layer Normalization	h_dim
Fully Connected	features_in, h_dim

close to the artificial termination limit we zero scaled actions and leave only noise.

An important adjustment to increase speed of training is to put Replay Buffer to the same device the network's learning is accomplished: we created State, Action, Reward, Done, Next State placeholders with size (N_{max}, feature dimension). It is much more computationally efficient to put single instance of each feature to the device placeholder than to sample a bigger batch and put it from one device to another at each training step, not to mention that instances can be accessed just by list of indexes.
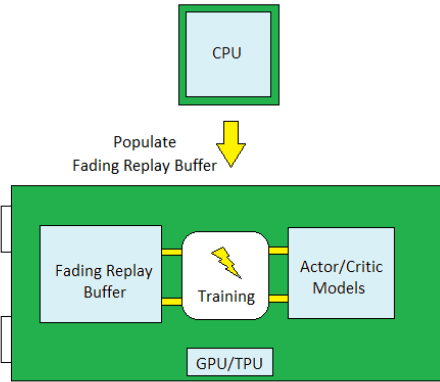
Fig. 9: Representation of Device occupation and training

Even though we used GPU: NVidia RTX 3000 series, CPU: AMD Ryzen 5..9. for testing, to decrease computational time we recommend performing a PyTorch's trivial just-in-time (JIT) compilation on new modules converting them to C++ optimized graphs including Loss Functions which are processed as modules with forward pass.

All these hyperparameters and measures constitute the default implementation of the algorithm, *Symphony-S3*. Other configurations are possible, which result in different associated parameters. One can unlink the correspondences between parameters completely for related purposes, though we do not recommend a Replay Buffer size lower than 7680*50=384,000.

Sample-efficient version, (SE)

We made a Sample-efficient configuration by decreasing the noise level $\epsilon \sim a_{max} * 1/\pi * \mathcal{N}(0.0, 1.0)$, where $\mathcal{N} \in [-\pi, \pi]$.

Sample-Proximity and Safety, (S2)

It is possible to shift focus from Sample-Efficiency to Sample-Proximity and Safety just by increasing the noise level or sending β with a higher order than 2.0 which will eventually punish strong action values harder. However, our Full Swaddling Function was balanced while the parametric noise value of 1/e perfectly suits it and already at the highest level. Layer Normalization and Resine Activation Function provide indirect generalization for the Input and Hidden Layers, respectively, whereas the Output Layer was left untouched. Original Dropout Layer [22] did not suit as it would zero some nodes distorting final distribution.

Last but not least we added a specific implementation of *Gradient Dropout* [23], which instead of zeroing some nodes as in Dropout Q Functions (DroQ) [24], does not calculate their gradients during backpropagation. This can be achieved simply by (mask is an original dropout mask, x^* - detached values):

$$mask * x + (1 - mask) * x^* \qquad (7)$$

Gradient Dropout after Fully Connected Output Layer in Actor and Critic with a dropout probability of 50% was the last

missing ingredient. We, additionally, decreased the learning speed: $\alpha_{lr} = \frac{1}{2}e-4$ and increased initial generalization: $N_{exp} = 20,480$, $N_{rb} = 25\,N_{exp} = 512,000$.

Embedded devices (ED), Baseline Model

A configuration for less computationally powerful devices like Jetson Series: $N_{exp} = 7680$, $N_{rb} = 384,000$, $h_{dim} = 384$, each Critic outputs 96 nodes, $\mathcal{B} = 288$. Data is stored in Fading Replay Buffer as float16, converted to float32 during the sample retrieval process.

IV. EXPERIMENTS

We set our experiments in Humanoid-v4, OpenAI's Gymnasium (Mujoco) Environment. Earlier versions of the Mujoco Humanoid environment, e.g. Humanoid-v1, had a maximum scale of action within -1 and 1, and due to the complexity of agent training, the maximum scale decreased to [-0.4, 0.4]. We returned the limit of the scale value to 1.0, since we have internal regularization. We did 10 random seed experiments for following models: Symphony-S3, Symphony-SE, Symphony-S2, Symphony-ED (Baseline). Symphony-S2 uses 20,480 exploratory steps, while others use 10,240. Total number of steps $3 * 10^6$, the episode limit was 1000 steps. We took readings in two ways, step-wise, when we sampled 25 trials each 2500 steps to estimate return and episode-wise to read an average σ value in batch (Exploratory steps and episodes were omitted on the graphs).

As was said for the Baseline model we choose Symphony-ED which to some extent close to Soft-Actor and Critic algorithm in performance but still generates alternating leg movements, though upper-body can stay underdeveloped, variance is relatively low:
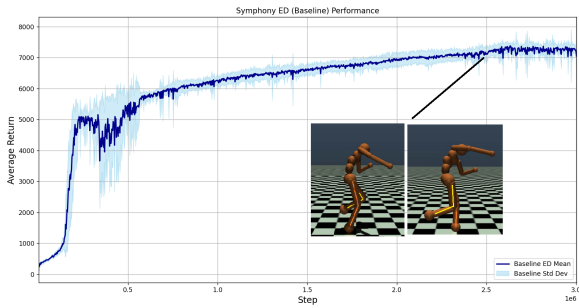

Fig. 10: Symphony-ED Return step-wise (orange lines depict supporting leg at a point in time).

In contrast, Sample-Efficient version, Symphony-SE develops a better gait, but suffer from high volatility especially when the agent's speed increases. It would be wise to decrease learning rate after some point.
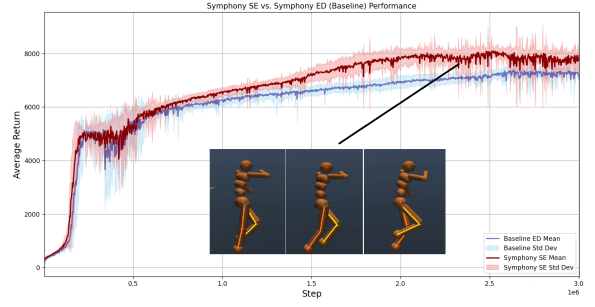

Fig. 11: Symphony-SE Return step-wise (orange lines depict supporting leg at a point in time)

.

Symphony-S3 shows better development through entire training, though as S3 is very close to SE model, it still suffers of higher variance at higher speeds. The same technique of decreasing learning rate can work here as well.
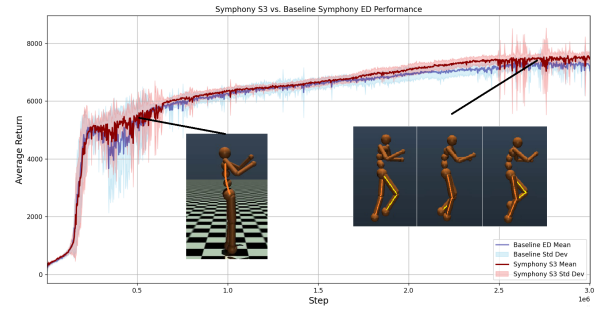

Fig. 12: Symphony-S3 Return step-wise (orange lines depict supporting leg at the moment)

.

Symphony-S2 solves most problems of SE and S3 algorithms in exchange of sample-efficiency, however, we can see that the average Return shows positive dynamics, while for the Baseline model it might hit plateau.
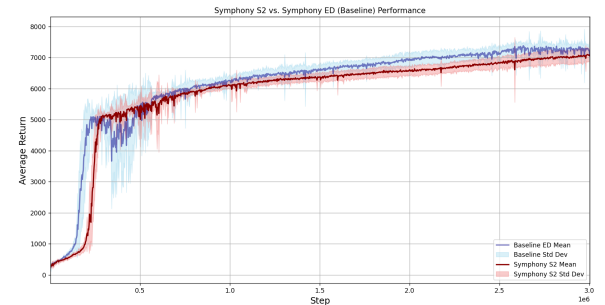

Fig. 13: Symphony-S2 Return step-wise

.

The diagram below shows scaling factor behavior episode-wise. For each algorithm, we trimmed the values for the first episode when the number of steps reached $3 * 10^6$. It can be

seen that the Scaling factor initially varies greatly (least of all for Symphony-S2), but then decreases closer to $\frac{1}{e}$, and slowly increases from this value.
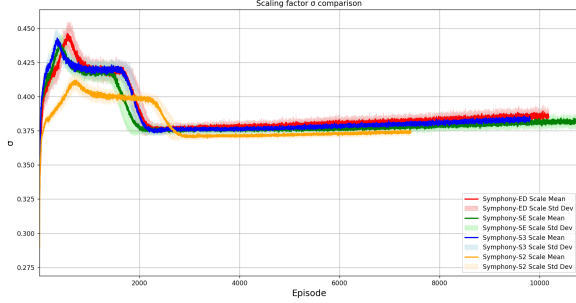

Fig. 14: Scaling Factor comparison.

.

We were interested in finding out the average value for the episode when $3 * 10^6$ steps are reached. A preliminary conclusion can be drawn that, better generalization and slower learning rate result in better prediction as agent falls less:

TABLE IV: Episode number when the $3 * 10^6$ steps were reached

Version	Average episode
ED	11530.5 ± 1132.8
SE	12433.5 ± 1055.0
S3	11411.5 ± 966.0
S2	$\mathbf{8284.1 \pm 753.7}$

Symphony-S2's accurate gait during training
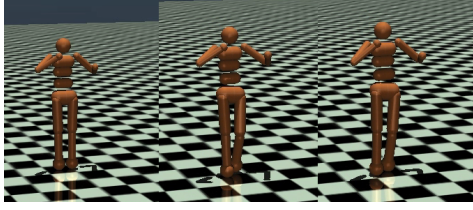
However, this does not imply that top scores are reached faster. To develop human-like movements one may need to sacrifice Sample-Efficiency to some extent.
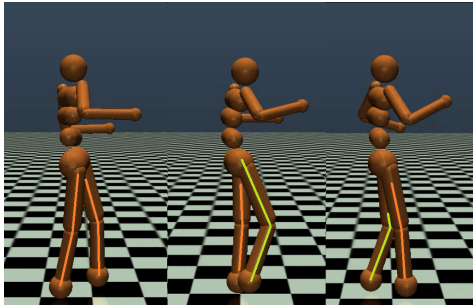

Fig. 15: Depiction when agent uses both hands and legs for balance during training

TABLE V: Step when the specific threshold of average Return is reached

Avg. Return	The Symphony version			
	ED	SE	S3	S2
5000	227k	**217k**	240k	*285k*
6000	782k	**657k**	682k	*880k*
7000	2,060k	**1,392k**	1,895	*2,768k*

Even with all this, there's no 100% guarantee that there will be absence of developmental defects. In our world, external obstacles often play an additional regulatory role. Following the logic of gradual human learning, we recommend starting training without obstacles, then gradually increasing the difficulty, filling a new Fading Replay Buffer with new exploratory data. While, for continuous learning with terminal cases to prevent overflow, we recommend subtracting a small value of ϵ from the "done=1.0" feature, and then shifting it by 2 positions. This will ensure that after the second cycle, the terminal transition disappears unable to pass the next "done equals 1.0" check. We also recommend reducing the learning rate ($\alpha_{lr} = 1e{-}5$) for continuous learning.

CONCLUSION

In our work, we traded off a fast convergence rate for the sake of eliminating jerky motions and convergence to suboptimal policies during training. We acknowledge that our results might not achieve state-of-the-art benchmarks in any single direction as the Symphony algorithm is the fruit of painstaking work balancing between 3 edges - Sample Efficiency, Sample Proximity and Safety of Actions (regarding extreme action values) for practical applications. To describe the Symphony Algorithm in a nutshell, it can be considered as a biological organism where all parts complement each other: disabling one weakens another. Even though the update-to-data ratio is 3 it is mostly used for coherent understanding of the distribution shift. We conducted hundreds of experiments before finding suitable fixed parameters for humanoid robots. In the next article, we'd like to conduct a full analysis of each component separately, we are preparing an ablation study as a separate project.
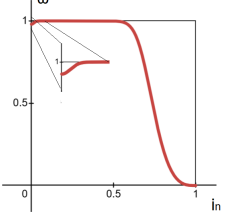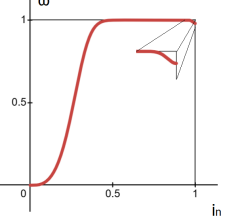
REFERENCES

[1] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[2] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[3] X. Chen, C. Wang, Z. Zhou, and K. Ross, "Randomized ensembled double q-learning: Learning fast without a model," *arXiv preprint arXiv:2101.05982*, 2021.

[4] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[5] Y. Wu, X. Chen, C. Wang, Y. Zhang, and K. W. Ross, "Aggressive q-learning with ensembles: Achieving both high sample efficiency and high asymptotic performance," *arXiv preprint arXiv:2111.09159*, 2021.

[6] A. Kuznetsov, P. Shvechikov, A. Grishin, and D. Vetrov, "Controlling overestimation bias with truncated mixture of continuous distributional quantile critics," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5556–5566.

[7] E. Nikishin, M. Schwarzer, P. D'Oro, P.-L. Bacon, and A. Courville, "The primacy bias in deep reinforcement learning," in *International conference on machine learning*. PMLR, 2022, pp. 16 828–16 847.

[8] X. Ma, L. Xia, Z. Zhou, J. Yang, and Q. Zhao, "Dsac: Distributional soft actor critic for risk-sensitive reinforcement learning," *arXiv preprint arXiv:2004.14547*, 2020.

[9] M. G. Bellemare, W. Dabney, and R. Munos, "A distributional perspective on reinforcement learning," in *International conference on machine learning*. PMLR, 2017, pp. 449–458.

[10] J. Duan, W. Wang, L. Xiao, J. Gao, and S. E. Li, "Dsac-t: Distributional soft actor-critic with three refinements," *arXiv preprint arXiv:2310.05858*, 2023.

[11] S. W. Roberts, "Control chart tests based on geometric moving averages," *Technometrics*, vol. 1, no. 3, pp. 239–250, 1959. [Online]. Available: https://www.tandfonline.com/doi/abs/10.1080/00401706.1959.10489860

[12] D. Ruppert, "Efficient estimations from a slowly convergent robbins-monro process," 02 1988.

[13] B. T. Polyak and A. B. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM Journal on Control and Optimization*, vol. 30, no. 4, pp. 838–855, 1992. [Online]. Available: https://doi.org/10.1137/0330046

[14] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[15] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, Aug 1988. [Online]. Available: https://doi.org/10.1007/BF00115009

[16] L. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine Learning*, vol. 8, pp. 293–321, 1992. [Online]. Available: https://api.semanticscholar.org/CorpusID:3248358

[17] T. Ishuov, Z. Otarbay, and M. Folgheraiter, "A concept of unbiased deep deterministic policy gradient for better convergence in bipedal walker," in *2022 International Conference on Smart Information Systems and Technologies (SIST)*, 2022, pp. 1–6.

[18] T. Ishuov and M. Folgheraiter, *How to make Robot move like a Human with Reinforcement Learning*, A. Qo'shboqov, Z. Otarbay, and M. Nurmanov, Eds. Independently published, 10 2023. [Online]. Available: https://www.amazon.com/dp/B0CKYWHPF5

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. [Online]. Available: https://arxiv.org/abs/1412.6980

[20] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2019. [Online]. Available: https://arxiv.org/abs/1711.05101

[21] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[22] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *arXiv preprint arXiv:1207.0580*, 2012.

[23] H.-Y. Tseng, Y.-W. Chen, Y.-H. Tsai, S. Liu, Y.-Y. Lin, and M.-H. Yang, "Regularizing meta-learning via gradient dropout," 2020. [Online]. Available: https://arxiv.org/abs/2004.05859

[24] T. Hiraoka, T. Imagawa, T. Hashimoto, T. Onishi, and Y. Tsuruoka, "Dropout q-functions for doubly efficient reinforcement learning," *arXiv preprint arXiv:2110.02034*, 2021.

APPENDIX

Symphony-$\tilde{S}2$

For practical use, we decided to stick with Symphony-S2, with some modifications. We greatly increased Gradient Dropout to 75% dropout level. A slight "dimple" was added to the modified hyperbolic tangent curve using a Gaussian function with a small amplitude (0.02) and reduced width (0.02). When using new weights for Target Critic, we reduce the importance of minimum values, while when using them for Fading Replay Buffer, our algorithm will not jump towards the latest transitions immediately.

TABLE VI: Fixed weights

a) for Target Critic	b) for Fading Replay Buffer
$tanh((\pi(1-i_n))^\pi) -$	$tanh((\pi i_n)^\pi) -$
$-0.02e^{-(i_n/0.02)^2}$	$-0.02e^{-((i_n-1)/0.02)^2}$

order π compensates for decreased plateau length