# ANS: Adaptive Network Scaling for Deep Rectifier Reinforcement Learning Models

**Yen-Yu Chang**
Department of Electrical Engineering
yenyu@stanford.edu

## Abstract

This work provides a thorough study on how reward scaling can affect performance of deep reinforcement learning agents. In particular, we would like to answer the question that *How does reward scaling affect non-saturating ReLU networks in RL?* This question matters because ReLU is one of the most effective activation functions for deep learning models. We also propose an Adaptive Network Scaling framework to find a suitable scale of the rewards during learning for better performance. We conducted empirical studies to justify the solution.

## 1    Introduction

Deep reinforcement learning (RL) has achieved tremendous success in a variety of domains such as recommendation systems (Chen et al. 2018; Dulac-Arnold et al. 2015) and board games (Silver et al. 2017). However, training a robust DRL agent is generally considered as non-trivial. Several factors such as network architecture (activation functions, number of layers, learning rate), codebases (Duan et al. 2016), random seed, and reward scale (Henderson et al. 2017) are shown to have significant impact on the overall performance of RL agents. This paper focuses on the *reward scaling* problem as reported in (Henderson et al. 2017; Gu et al. 2016; Duan et al. 2016), in which it is discovered that different reward scales can result in performance change by a large margin, transforming an unlearnable task into a learnable task. Given activation functions like *sigmoid* or *tanh*, the problem of saturation (Glorot and Bengio 2010; Vincent, de Brébisson, and Bouthillier 2015; Montavon, Orr, and Müller 1998) seems to be the reason why large reward scales cause performance drop since it is harder for bounded output to handle targets of large scale. One plausible remedy is to choose the Rectified Linear Unit (ReLU) (Nair and Hinton 2010) since it is not a saturating activation; that is,

$$\lim_{x \to \infty} \text{ReLU}(x) = +\infty.$$

A question therefore arises, *How does ReLU react to different scales of reward?* The answer does matter because ReLU is faster and the consequent sparse representations

have shown remarkable performance in training deep neural networks (Glorot, Bordes, and Bengio 2011). We would also like to study how the reward scaling affects the *dying ReLU* problem, and how some potential solutions such as Leaky-ReLU (Maas, Hannun, and Ng 2013) and ELU (Clevert, Unterthiner, and Hochreiter 2015) can be affected by the scaling of rewards. One of the major conclusion in this paper is that different from the intuition that smaller reward scale seems to be better (which is usually true for *sigmoid* or *tanh*), for ReLU a larger scale on rewards generally leads to a better performance. In a nutshell, we would like to answer the following questions in this work.

Q1. What is the difference between adjusting learning rate and reward scaling?

Q2. How is ReLU networks affected by reward scales and whether it is related to the level of dying ReLU?

Q3. Does non-saturating Leaky-ReLU and ELU improve the performance of RL by avoiding the dying-ReLU problem and how do they react when different reward scales are applied?

Q4. Since reward scale matters, how to find appropriate reward scales yielding better ReLU-based DRL models?

Section 2 gives answers to Q1, Q2, and Q3. We have shown that the change of learning rate actually shows opposite effect with the change of reward scale in terms of dying ReLU. We further show that the influence of reward scaling is significant – unlearnable tasks can become learnable when appropriate scale is adapted. Finally, the performance of ELU and leaky-ReLU can be greatly changed with different reward scales as well. Based on the empirical evidence obtained in our experiments, we solve Q4 by proposing the *adaptive network scaling* (ANS) framework, which enables ReLU networks to find the proper scale for reward targets and quickly update the parameters accordingly, without the need to retrain RL models after scaling.

### 1.1    Preliminaries

RL defines a class of algorithms solving problems modeled as a Markov Decision Process (MDP). An MDP consists of a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the space of state $s$, $\mathcal{A}$ is the space of action $a$, $\mathcal{T}$ is the transition function with probability distribution $\Pr(s'|s, a)$, $\mathcal{R}(s, a)$ is the reward function

that outputs a scalar feedback given action $a$ made at state $s$, and $\gamma$ is the discount factor.

In actor-critic framework, the actor is the policy function $\pi_\theta(a|s)$, which is used to react to environments and collect samples of experience $(s, a, r)$. Those samples are used to update the critic, the value function $Q_\pi(s, a)$ that outputs the estimate of cumulative reward of current policy $\pi_\theta$,

$$Q_\pi(s_t, a_t) = E_{s_{t+1}, a_{t+1}, \dots} \left[ \sum_{i=0}^{\infty} \gamma^i r(s_{t+i}) \right]. \quad (1)$$

The output of value function can be further used to update policy function, which aims to maximize the expected return $J(\theta)$. Policy gradient method (Sutton et al. 2000; Kakade 2002) is usually preferable to update the policy in actor-critic methods. The policy gradient can be written as

$$\nabla_\theta J(\theta) = E_{\tau \sim \pi_\theta(\tau)} \left[ \nabla_\theta \log \pi_\theta(\tau) r(\tau) \right], \quad (2)$$

where $\pi_\theta$ is the policy function and $\tau$ is the trajectories generated from the interaction between $\pi$ and the environment. Here the cumulative reward $r(s, a)$ is sometimes replaced with Q-value function $Q(s, a)$ or advantage function,

$$A(s, a) = Q(s, a) - V(s) \quad (3)$$

so as to reduce the variance of policy gradient.

Neural networks (Rosenblatt 1962) is a powerful function approximator trained with back-propagation. Different optimizers such as Stochastic Gradient Descent (SGD) (Rumelhart, Hinton, and Williams 1985) or Adam (Kingma and Ba 2014) are adapted to stabilize the parameter update, which can further improve the outcome of the learning process. SGD updates the parameters with shuffled order of instances and usually incorporates Nesterov momentum (Nesterov 1983) to prevent from taking a big jump in the direction of the updated gradient. On the other hand, Adam optimizer considers the first and the second moment of the gradient sequence with bias correction for parameter update

$$\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}}, \quad (4)$$

where $\hat{m}_t$ and $\hat{v}_t$ is the unbiased first and the second moment of gradients respectively.

## 2   Influence of Reward Scaling

Reward scaling indicates the change in MDP from $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$ to $(\mathcal{S}, \mathcal{A}, \mathcal{T}, c\mathcal{R}, \gamma)$, where $c \in \mathbb{R}^+$. When it comes to reward scaling, the first question is usually – what is the difference between scaling learning rate and scaling reward function (Q1)? By considering the gradient update with loss function $\mathcal{L}(f(\theta), y)$, we have

$$\theta_t \leftarrow \theta_{t-1} - \alpha \nabla_f \mathcal{L}(f(\theta), y) \nabla_\theta f(\theta). \quad (5)$$

If mean-square-error (MSE) loss is used, the update rule becomes

$$\theta_t \leftarrow \theta_{t-1} - \frac{2\alpha}{N} \sum_i (f_\theta(x_i) - y_i) \nabla_\theta f_\theta(x_i). \quad (6)$$

By scaling learning rate, the magnitude of the update term is scaled equally, and the signs of the gradients remain the same. However, if the target $y$ is scaled, the distance between the target and the estimate is changed instead of being scaled, which brings different sign distribution from the one without scaling. It is possible to scale the target $y$ so as to obtain a more informative gradient distribution. It can also potentially prevent the zig-zagging phenomena (Zisserman 2013) by making gradients have different signs (i.e. if the gradients are of the same sign, the parameters of neural networks are updated to the same direction, which may cause overshooting and require another update with opposite direction). Such zig-zagging results in optimization in an inefficient manner. On the other hand, if inappropriate reward scale is applied, the performance drops drastically as shown in (Henderson et al. 2017).

The other aspect that reward scaling affects the learning performance are the *saturation* problem (Glorot and Bengio 2010; Vincent, de Brébisson, and Bouthillier 2015; Montavon, Orr, and Müller 1998) and the *dying ReLU* problem. For activation functions like *sigmoid* or *tanh*, the output of the value is bounded and the gradients at these regions are almost zero. By chain rule, the gradient of the weight and bias in previous layers are diminished if the gradient of the activation is close to zero, which blocks the gradient signal from flowing to the subsequent layers. For ReLU, a large gradient may prevent neurons to output values larger than zero for all instances. Consequently, the gradient passed through the neurons becomes zero, resulting in the *dying ReLU* problem. Even though dying ReLU is not completely irreversible [1], it may take large amount of time to recover and thus slow down the learning process.

Since the data are coming in an online fashion for RL, it is hard to apply the definition of dying-ReLU – for all input $x$ of the batch data, the output of neuron $n$ is negative. Consequently, here we define *pseudo-dying ReLU*, which is used to evaluate the tendency of neurons that may become dying.

**Definition 2.1** *For a set of collected samples $\mathcal{B}$ with $|\mathcal{B}| = B$, if*

$$f_n(f_p(\boldsymbol{x})) \le 0, \forall \boldsymbol{x} \in \mathcal{B}, \quad (7)$$

*where $f_p$ is the output of the previous layer and $f_n$ is the function of neuron $n$. For this case, we define neuron $n$ with ReLU as a pseudo-dying ReLU with respect to $\mathcal{B}$.*

**Proposition 2.1** *Consider $f_n(\boldsymbol{p}) = \boldsymbol{w}^\top \boldsymbol{p} + b$ and $\|\boldsymbol{p}\| = f_p(\mathcal{B})$ is normally distributed [2]. For the distribution of the angle between $\boldsymbol{w}$ and $\boldsymbol{p}$, we follow empirical distribution if necessary. If neuron $n$ is pseudo-dying with respect to $\mathcal{B}$, then the probability that $\mathcal{B} \cup \{\boldsymbol{x}_{B+1}\}$ does not cause pseudo-dying ReLU is upper bounded by*

---

[1] Technically, except for the first layer, the update of other parameters can change the distribution of the input of the dying neurons, which is possible to make them alive again.

[2] A more proper setting is to assume a truncated normal distribution. However, to simplified the derivation we use normal distribution and assume that the distribution is far enough from zero.

1. $|\boldsymbol{w}^\top \boldsymbol{p}_i| \geq |b|$ and $\boldsymbol{w}^\top \boldsymbol{p}_i < 0$

$$\Pr(\textit{not pseudo-dying ReLU}) \leq \frac{1}{2}\left[1 + erf\left(\frac{-1}{\sqrt{2B}}\right)\right]. \tag{8}$$

2. $|\boldsymbol{w}^\top \boldsymbol{p}_i| \leq |b|$ and $b < 0$

$$\Pr(\textit{not pseudo-dying ReLU})$$
$$\leq \frac{1}{2}\left[1 - erf\left(\sqrt{\frac{2(B-1)}{B}}\left(1 - \frac{\bar{\mu}}{|b|/\|\boldsymbol{w}\||\cos\theta_{min}|}\right)\right)\right]. \tag{9}$$

*For other cases,* $\Pr(\textit{not pseudo-dying ReLU}) = 0$.

The proof for Proposition 2.1 is in the appendix. The physical meaning of the proposition is that if neuron $n$ is currently pseudo-dying, then the probability that it is still pseudo-dying with subsequent samples is high.

In the following context, we use pseudo-dying ReLU as an indicator of the true dying ReLU. Using this indicator is computationally cheaper and more efficient than checking whether the neuron is really dying or not. Here we also define pseudo-dying ReLU ratio (PDRR). For layer $i$, we have,

$$\text{PDRR}_i = \frac{\text{number of pseudo-dying ReLU in layer } i}{\text{number of neurons in layer } i}. \tag{10}$$

It should be noted that, in actor-critic approaches, the value network is where we can differentiate between changing learning rate and reward scaling. If the target of value network is scaled, the subsequent gradients can have different sign distribution as mentioned instead of simply being scaled. For policy network, as shown in Equation 2, changing the scale of the estimate is the same as changing the magnitude of policy gradients. Especially, for RMSprop (Tieleman and Hinton 2012) and Adam (Kingma and Ba 2014) optimizers, as suggested in Equation 4, such scaling actually makes no difference because the division of the first moment to the squared root of the second moment cancels out the scale. In our experiments, to reduce variables that may affect experiment results, if the reward is scaled $c$ for the critic, we multiply the estimate from the critic with $1/c$ so that the magnitude of the estimate to the actor is the same.

## 2.1 ReLU Experiments

In this section, we aim to answer the following questions by conducting extensive experiments: (Q1) What is the difference between changing learning rate and changing reward scale in terms of cumulative rewards and PDRR? (Q2) How does ReLU network respond to different reward scales in terms of cumulative rewards and PDRR? (Q3) How is the performance of the variants of ReLU without dying-ReLU issue such as Leaky-ReLU and ELU? Are such activations insensitive to reward scales? To answer these questions, we compare different settings on DDPG (Lillicrap et al. 2015) and A2C (Mnih et al. 2016) to make sure the results are consistent with different actor-critic models. In this work, our experiments are built upon Mujoco environment (Todorov, Erez, and Tassa 2012), which is a physics engine for continuous control and provides multi-joint dynamics simulation.

**Background** DDPG extends the deterministic policy gradient to continuous action spaces with actor-critic approach and utilizes Ornstein-Uhlenbeck process for exploration. The actor is updated by the differential of the expected return from the start distribution $J$ with respect to the actor parameters:
$$\nabla_\theta J \approx E_{s_t \sim \rho^\beta}\left[\nabla_\theta Q(s,a)|_{s=s_t, a=\mu(s_t)}\right],$$
where $\rho^\beta$ is the state distribution following stochastic policy $\beta$. The critic is updated with expected return that is estimated from target networks:
$$y_t = r_t + \gamma Q'(s_{t+1}, \mu'(s_{t+1})),$$
where $Q'$ is the target critic and $\mu'$ is the target actor. To make the learning more stable, "soft" update is used to update target networks so that the output of the target networks will not change too fast.

The other model that we use here is advantage actor critic (A2C), which is a synchronous and batched version of asynchronous actor critic (A3C) (Mnih et al. 2016). It utilizes policy gradient as indicated in Equation 2 to update the policy. Advantage function $A(s,a)$ is used in place of reward $r(\tau)$ to scale the gradient $\nabla_\theta \log \pi_\theta$ because it obtains lower variance for gradient estimation. What makes A3C and A2C different from other actor-critic methods is that it accumulates gradients of workers and use them to update the global model.

Exponential Linear Unit (ELU) is similar to ReLU except for negative inputs,
$$\text{ELU}(x) = \max(0, x) + \min(0, \alpha(\exp(x) - 1)).$$
An interesting result of this form is that the gradient can be computed with addition:
$$\text{ELU}'(x) = \begin{cases} 1, & \text{if } x > 0 \\ \text{ELU}(x) + \alpha, & \text{otherwise.} \end{cases}$$
On the other hand, Leaky-ReLU allows a mild slope when input is negative,
$$\text{LeakyReLU}(x) = \max(0, x) + \alpha \min(0, x),$$
where $\alpha$ is usually set to a small value such as $0.01$.

In the following experiments, we use three-layer neural networks as function approximators. Since the two ReLU layers have similar patterns, we show ReLU in the second layer, which shows significant difference across variables. For clarity and conciseness, the most representative figures are presented, and we leave others in appendix.

**Results on Adjusting Learning Rates** Figures 1 shows how cumulative reward and PDRR change with respect to different learning rate. The results are not surprising as too large learning rate results in high PDRR, which impedes further performance improvement because of limited available computation units. It should be noted that although high PDRR leads to bad performance, the same correlation does not apply when PDRR is lower than a threshold. For $10^{-3}$ and $7 \cdot 10^{-4}$ learning rate, even though they have higher PDRR than $10^{-4}$ and $10^{-5}$, their returns are higher. It may be caused by the fact that it is possible for neural networks to have excessive computation capacity (Guss and Salakhutdinov 2018) for certain tasks; consequently, PDRR may pose no harm under such situation.

(a) PDRR
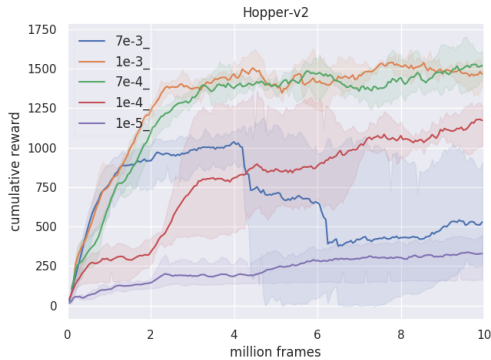


(b) Return

Figure 1: A2C learning rate comparison on Hopper-v2.

**Reward Scale**  In Figure 2 and 3, they show that reward scales can have tremendous influence on the performance of RL models with ReLU networks. With proper reward scales, the results can be changed from learning nothing to a well-trained model. Another fact worth noting is that large reward scale does not cause significant PDRR compared with other scales; on the contrary, 0.5 reward scale shows a tendency to cause higher PDRR as suggested in Figure 2c. Such phenomenon also reflects on cumulative reward as A2C with 0.5 reward scale is unable to converge to a satisfactory solution.

The phenomenon of small reward scale resulting in dying-ReLU can be explained as follows. Consider a layer with ReLU outputs a scalar as $\text{ReLU}(\boldsymbol{w}^\top \boldsymbol{x} + b) = y$. If the target $\hat{y}$ is scaled by $c < 1$ and $\boldsymbol{x}$ remains unchanged, then the positive outputs of $\boldsymbol{w}^\top \boldsymbol{x} + b$ should be reduced while the negative part is not affected (the value does not even matter). In this case, $b$ is updated to make $y$ smaller, which potentially increase the probability of getting trapped by dying ReLU. In contrast, reducing learning rate decreases the magnitude of parameter updates in all directions, which proportionally make dying ReLU less probable.

The results reveal that reward scales over certain unknown threshold usually allows RL models with ReLU to learn better. It can be caused by the fact that larger reward scales enlarge the difference between positive and negative returns and therefore the information of rewards becomes clearer. It can be confirmed by the outcome of the experiments that the influence of reward scaling on A2C is much larger than that on DDPG because A2C makes use of advantage function that has zero expected value with respect to actions. However, larger reward scale does not always mean better. Too large reward scale may introduce large gradients as well, which makes RL models unable to converge easily. As a result, to improve the performance of RL models, finding the reward scale that gives enough information of reward functions and meanwhile is not too large to converge to a good solution is where we can put effort.

**Leaky-ReLU and ELU**  We try variants of ReLU, Leaky-ReLU, and ELU to illustrate (1) how the performance of models with ELU or Leaky-ReLU changes with different reward scales, and (2) if the variants of ReLU without dying ReLU issue can outperform ReLU in RL problems. As Figure 4 suggests, ELU generally outperforms ReLU and Leaky-ReLU slightly; on the other hand, even though Leaky-ReLU does not suffer from dying ReLU, it still performs empirically worse than ReLU when it comes to cumulative rewards. Reward scaling is still an important factor to the performance of ELU and Leaky-ReLU. With proper reward scales, the RL models also converge to a much better solution.

**A Short Summary**  Learning rate and reward scale show utterly different pattern in terms of PDRR – larger learning rate can result in higher PDRR whereas the higher reward scale have lower PDRR. The recommendation drawn from our experiments is that ELU slightly outperforms ReLU but is also computationally more costly than ReLU, so there is a trade-off. Even though Leaky-ReLU allows information to flow when the input is negative, empirically its performance is worse than ReLU. Since the magnitude of reward functions depends on the design of the environment, it is hard to give suggestions on reward scales. In spite of that, it seems that for ELU, Leaky-ReLU, and ReLU, a slightly larger reward scale such as 5 or 10 usually poses no harm and potentially brings benefits such as better performance.

## 3 Adaptive Network Scaling

We have shown that reward scaling has the potential to improve the performance of RL models by a large margin. In this section, we propose the *Adaptive Network Scaling* (ANS) framework to answer the question: (Q4) How to efficiently find an appropriate reward scale to improve the performance? This question can be decomposed into two sub-problems: (1) How to find a suitable reward scale? and (2) How to scale the reward while learning and transfer the learned parameters instead of learning from scratch? For the first sub-problem, we propose a search strategy that requires few queries to find a satisfactory solution. For the second sub-problem, our solution is to transfer the model trained previously to the current reward scale so that retraining is not required. ANS is the combination of these two solutions.

### 3.1 Adaptive Search Strategy

Our previous experiments suggest that to achieve the best performance, the reward scale should be able to balance the

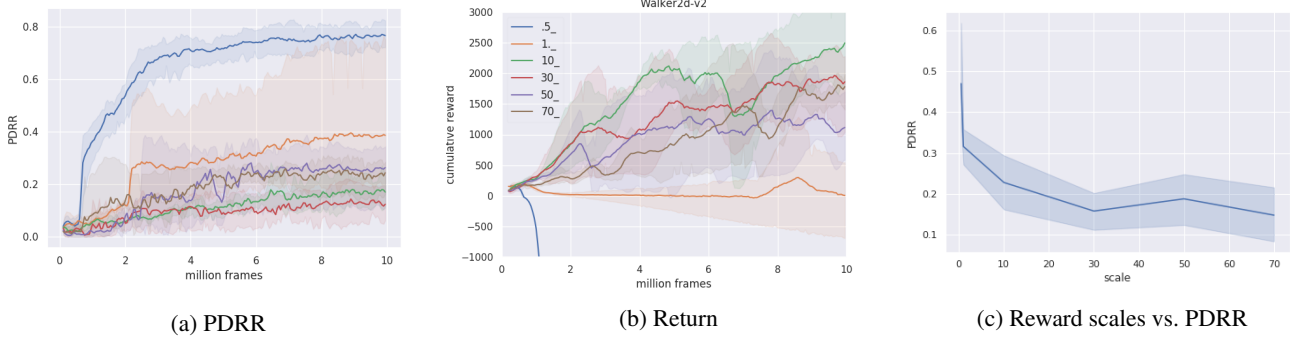| (a) PDRR | (b) Return | (c) Reward scales vs. PDRR |

Figure 2: A2C reward scales comparison on Walker2d-v2 and the correlation between reward scales and PDRR in five Mujoco environments.
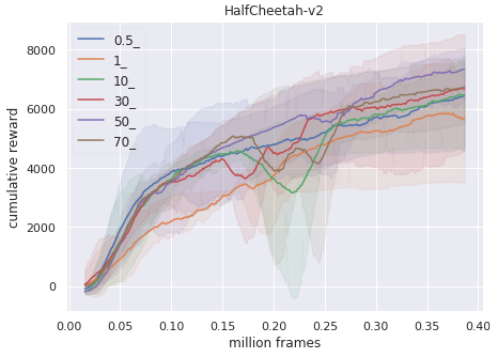


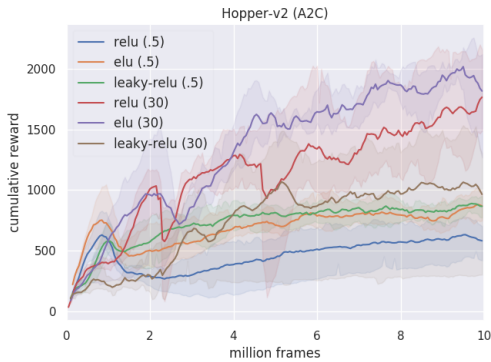Figure 3: DDPG reward scales comparison on HalfCheetah-v2.



Figure 4: Activation comparison on Hopper-v2 with A2C.

magnitude of gradients and the maximization of the diversity between positive and negative samples. We first assume that the landscape of reward scale search space is concave as indicated in Assumption 3.1.

**Assumption 3.1** *We assume that for some reward scale $s_x$ and $s_y$, and for any $\alpha \in [0, 1]$,*

$$g(\alpha s_x + (1 - \alpha)s_y) > \alpha g(s_x) + (1 - \alpha)g(s_y), \quad (11)$$

*where g is the function that indicates the highest cumulative reward that the RL model can achieve with this reward scale.*

Even though it is a strong assumption for the landscape,

we still need to be careful not to use too many queries while optimizing it since estimating $g(s)$ requires solving an MDP. To derive a near-optimal solution under Assumption 3.1, we propose an adaptive search strategy that can find a satisfactory scale efficiently and is easy to implement.

At the beginning of the strategy, we choose to multiply the reward scale with $c_{inc} > 1$. If the "performance" is better than the previous scale, we continue to multiply the reward scale with the same $c_{inc}$ until the improvement stops. ANS then considers the case as overshooting and multiply the reward scale with $c_{dec}$ slightly less than one to refine the result and so on. It should be noted that the near-optimal scale can be found even when the optimal scale is less than one.

To derive the performance of the current scale, we leverage exponential moving average with bias correction (Kingma and Ba 2014),

$$m_t \leftarrow \beta m_{t-1} + (1 - \beta)R_t$$
$$\hat{m}_t \leftarrow m_t/(1 - \beta^t),$$

where $R_t$ is the return of the $t$-th episodes and $\hat{m}_t$ is the unbiased estimate of the mean. We record the highest unbiased estimate of mean $\hat{m}_{max}$. If it is not updated in $T$ episodes, then ANS considers this case as getting trapped by a local optimum and $\hat{m}_{max}$ is treated as the performance of the current scale.

The main advantage of this search method is that we do not need many queries of $g$ to find a satisfactory solution. In addition, if the optimal reward scale is bounded by $s_{max}$ and one, we have a bounded number of steps $n_{max}$,

$$n_{max} \leq \lceil \log_{c_{inc}} s_{max} \rceil - \lfloor \log_{c_{dec}} c_{inc} \rfloor. \quad (12)$$

Empirically, ANS requires less than six steps to achieve competitive performance of the best reward scale in our ReLU experiments and the computation cost is much lower than grid searching for the best scale. We leave the detailed pseudo-code in the appendix.

### 3.2 Network Scaling

When the reward function is scaled, the target of state-action value $Q(s, a)$ and the state value $V(s)$ are scaled equally because they are exponentially weighted sum of rewards. Since

learning from scratch is time consuming, here we introduce *network scaling*, which allows us to preserve scaled outputs of the function,

$$cf_{W',b'}(\boldsymbol{x}) = f_{W,b}(c\boldsymbol{x}). \qquad (13)$$

To achieve Equation 13, we utilize the scaling property of ReLU,

$$c \cdot \text{ReLU}(\boldsymbol{x}) = \text{ReLU}(c\boldsymbol{x}). \qquad (14)$$

The ReLU network can be expressed as

$$f_{W,b}(\boldsymbol{x}) = \boldsymbol{W}_n(...(\boldsymbol{W}_2(\boldsymbol{W}_1\boldsymbol{x}' + \boldsymbol{b}_1)) + \boldsymbol{b}_2)...) + \boldsymbol{b}_n, \qquad (15)$$

where we omit ReLU in each layer for clarity and conciseness. If we scale each parameter,

$$f_{W',b'}(\boldsymbol{x}) = \boldsymbol{W}'_n(...(\boldsymbol{W}'_2(\boldsymbol{W}'_1\boldsymbol{x} + \boldsymbol{b}'_1) + \boldsymbol{b}'_2)...) + \boldsymbol{b}'_n \qquad (16)$$

$$= c_n\boldsymbol{W}_n(...(c_1\boldsymbol{W}_1\boldsymbol{x} + r_1\boldsymbol{b}_1)...) + r_n\boldsymbol{b}_n \qquad (17)$$

by obeying the following constraints, we are able to preserve the scaled output as Equation 13,

$$\prod_{i=1}^{t} c_i = r_t, \text{ for } t = 1, ..., n \qquad (18)$$

$$b_n = c. \qquad (19)$$

We consider $c_i = \sqrt[n]{c}$ in the following derivation and our experiments. However, it is straightforward to extend to different values of $c_i$, which can be determined by the condition of each layer.

The change of the weights and the corresponding gradients are,

$$\boldsymbol{W}_i^{\text{new}} \leftarrow \sqrt[n]{c}\boldsymbol{W}_i \qquad (20)$$

$$\nabla_{\boldsymbol{W}_i^{\text{new}}}\mathcal{L} \leftarrow c^{\left(2-\frac{1}{n}\right)}\nabla_{\boldsymbol{W}_i}\mathcal{L}. \qquad (21)$$

For each bias, we have similar result,

$$\boldsymbol{b}_i^{\text{new}} \leftarrow \sqrt[n]{c^i}\boldsymbol{b}_i \qquad (22)$$

$$\nabla_{\boldsymbol{b}_i^{\text{new}}}\mathcal{L} \leftarrow c^{\left(2-\frac{i}{n}\right)}\nabla_{\boldsymbol{b}_i}\mathcal{L}. \qquad (23)$$

We may consider reward scaling as a transfer approach that provides an initialization closed to the new solution space, which saves a lot of time to train RL models from scratch. The result with and without network scaling is shown in Figure 5. Note that unless care is taken, manipulating parameters of neural networks can result in instability in the first few epochs of training; consequently, we strictly clip the gradients by imposing maximal norm on the policy network after scaling and gradually relax the norm, which prevents the trained policy network from being misled by the scaled value network at the beginning.

## 3.3 Related Work

To the best of our knowledge, Pop-Art (van Hasselt et al. 2016) is the only framework that directly deals with reward scales. Their work is mainly for value-based RL approaches while we focus on actor-critic approaches, which is capable
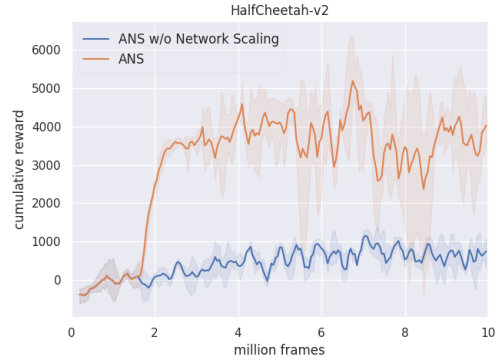


Figure 5: Comparison of ANS with and without network scaling on A2C.

of achieving competitive performances (Duan et al. 2016) such as DDPG (Lillicrap et al. 2015), PPO (Schulman et al. 2017), and A2C (Mnih et al. 2016). The main difference between Pop-Art and our approach is that ANS finds the best reward scale in terms of learning performance whereas Pop-Art maintains a normalized reward distribution, which may not guarantee to improve the outcome of learning.

Pop-Art considers the neural network as

$$f(x) = \boldsymbol{\Sigma}g_{\theta,\boldsymbol{W},\boldsymbol{b}}(x) + \boldsymbol{\mu} = \boldsymbol{\Sigma}(\boldsymbol{W}h_\theta(x) + \boldsymbol{b}) + \boldsymbol{\mu}, \qquad (24)$$

where $g_{\theta,\boldsymbol{W},\boldsymbol{b}}$ is a normalized function and $h_\theta$ is a parameterized (non-linear) function. To preserve the output of the unnormalized function $f$ while maintaining the normalized function $g_{\theta,\boldsymbol{W},\boldsymbol{b}}$, that is,

$$f_{\theta,\boldsymbol{\Sigma},\boldsymbol{W},\boldsymbol{b},\boldsymbol{\mu}}(x) = f_{\theta,\boldsymbol{\Sigma}_{\text{new}},\boldsymbol{W}_{\text{new}},\boldsymbol{b}_{\text{new}},\boldsymbol{\mu}_{\text{new}}}, \qquad (25)$$

they apply incremental update to derive zero mean and unit variance with $\boldsymbol{\Sigma}_{\text{new}}$ and $\boldsymbol{b}_{\text{new}}$. Additionally, the output can be preserved by the following update

$$\boldsymbol{W}_{\text{new}} = \boldsymbol{\Sigma}_{\text{new}}^{-1}\boldsymbol{\Sigma}\boldsymbol{W} \qquad (26)$$

$$\boldsymbol{b}_{\text{new}} = \boldsymbol{\Sigma}_{\text{new}}^{-1}(\boldsymbol{\Sigma}\boldsymbol{b} + \boldsymbol{\mu} - \boldsymbol{\mu}_{\text{new}}). \qquad (27)$$

They claim that their approach is able to deal with the problem that the magnitude of value target can change over time. However, even though maintaining the magnitude to a predetermined range is a straightforward approach to stabilizing the learning process, empirically, the performance is not generally good – the performance drops in nearly fifty percent of the Atari games with Pop-Art approach as reported. This can be caused by the fact that it is hard to determine appropriate values that the mean and variance should be scaled to. Conceptually, we claim that the scaling method should consider the training status such as improvement in cumulative reward, which is the ultimate objective of RL.

## 3.4 Experiments

In our experiment, we use DDPG and A2C models to show how ANS and Pop-Art influence the performance. For performance evaluation, we follow (Henderson et al. 2017) to use final average across 5 trials of returns across the last 100

| Algorithm | HalfCheetah-v2 | Hopper-v2 | Walker2d-v2 | Swimmer-v2 | Ant-v2 |
|---|---|---|---|---|---|
| DDPG (ReLU) | 7944.54 | 264.19 | 705.96 | - | - |
| DDPG (ELU) | 7899.17 | 795.83 | 1256.20 | - | - |
| DDPG (Sigmoid) | 7730.29 | 1173.10 | 695.21 | - | - |
| DDPG (Tanh) | 1802.56 | 808.24 | 723.77 | - | - |
| DDPG+Pop-Art | 1273.33 | 287.92 | 356.08 | - | - |
| DDPG+ANS | **9393.24** | **1233.73** | **1643.47** | - | - |
| A2C (ReLU) | 1132.85 | 1341.84 | 44.81 | 11.80 | -15.77 |
| A2C (ELU) | 1280.25 | 1489.04 | -1086.8 | 27.14 | -15.73 |
| A2C (Sigmoid) | 832.32 | 1494.26 | 155.51 | -3.46 | -13.25 |
| A2C (Tanh) | 622.83 | 1495.83 | 140.67 | 25.12 | -17.84 |
| A2C+Pop-Art | -455.57 | 207.99 | -572.62 | -16.27 | -148.40 |
| A2C+ANS | **3689.64** | **1533.48** | **1760.07** | **83.00** | **1000.57** |

Table 1: Results of different frameworks applied to RL models. Final average across 5 trials of returns across the last 100 trajectories after $N$ frames.



(a) Swimmer-v2 (A2C)  (b) HalfCheetah-v2 (A2C)  (c) Walker2d-v2 (DDPG)
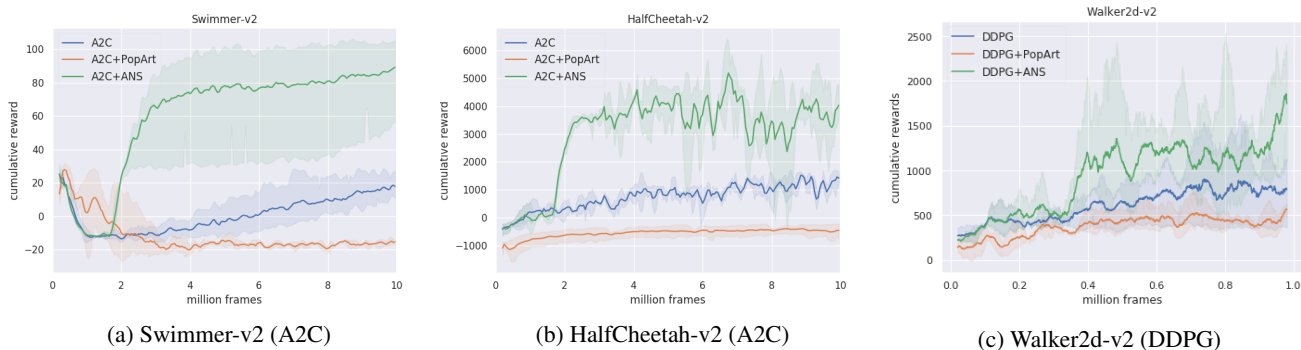
Figure 6: Performance comparison between original models, models with Pop-Art, and models with ANS.

trajectories after $N$ frames. It should be noted that it is possible for ANS to terminate before $N$ frames and we continue to train the model until $N$ frames are reached.

Since A2C undergoes 16 environments at the same time, we use 100 updates as tolerance $T$. For DDPG, 100-episode tolerance is adapted for Walker2d-v2 and Hopper-v2, and 500-episode tolerance is used for HalfCheetah-v2. Similarly, the total number of frames $N$ for A2C is 10 million. For DDPG, we use one million frames for Walker2d-v2 and Hopper-v2, and two million frames for HalfCheetah-v2. Different number of frames and tolerance are adapted because DDPG keeps improving within one million frames on HalfCheetah-v2, which makes it difficult to apply ANS. We fix $\beta = 0.9$, $c_{inc} = 8.0$ and $c_{dec} = 0.9$ to avoid cumbersome parameter tuning. The performance of Swimmer-v2 and Ant-v2 in DDPG is not included since they can hardly be trained even with reward scaling.

As indicated in Figure 6, our experiments show that the performance of ReLU is competitive with other activations although it is faster to compute. Besides, RL models equipped with Pop-Art even perform worse than original models. It indicates that unit variance and zero mean may not be the appropriate setting for ReLU networks. The other reason could be that Pop-Art is at first crafted for value-based RL approaches such as Dueling DQN (Wang et al. 2015) instead of actor-critic models. On the other hand, ANS reaches competitive performance with the proper reward scales in ReLU experiments. The most important thing is that we have no prior information about how the reward scale should be and ANS is able to find it with limited number of frames. ANS also outperforms models with ELU, sig-

moid and tanh when no reward scaling is applied. The advantages of ReLU networks with ANS over them are threefold: (1) It is easier to compute and therefore the learning process is faster. (2) It is able to preserve the sparsity property of ReLU. (3) The performance is better by a large margin. Another observation is that in most of the cases, ANS starts to shrink the reward scale after continuously increasing till reaches 64, which agrees with our previous experiments that the optimal scale is less than 64.

## 4 Conclusion

Here is the brief summary of the contribution of this work

- We conduct thorough experiments and provide analysis to show that ReLU is a competitive activation function to used for training DRL, but its performance can be influenced by the scale of the rewards.

- The idea of pseudo-dying ReLU is proposed for DRL models to evaluate the dying-ReLU situation, and empirically we have shown that it is correlated with the scale of the rewards as well as the final cumulative rewards.

- We propose the adaptive network scaling framework, which can be incorporated with actor-critic approaches with ReLU networks to efficiently find a suitable reward scale of better performance. Empirically, we show that ANS improves the performance with respect to the original models by a large margin.

Future works include the extension to analyze the effects of reward scaling to saturating activations such as *sigmoid* and *tanh*, and design a more general strategy to find suitable reward scale for different activation functions.

# References

Chen, S.-Y.; Yu, Y.; Da, Q.; Tan, J.; Huang, H.-K.; and Tang, H.-H. 2018. Stabilizing reinforcement learning in dynamic environment with application to online recommendation. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 1187–1196. ACM.

Clevert, D.-A.; Unterthiner, T.; and Hochreiter, S. 2015. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*.

Duan, Y.; Chen, X.; Houthooft, R.; Schulman, J.; and Abbeel, P. 2016. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, 1329–1338.

Dulac-Arnold, G.; Evans, R.; van Hasselt, H.; Sunehag, P.; Lillicrap, T.; Hunt, J.; Mann, T.; Weber, T.; Degris, T.; and Coppin, B. 2015. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv:1512.07679*.

Glorot, X., and Bengio, Y. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 249–256.

Glorot, X.; Bordes, A.; and Bengio, Y. 2011. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, 315–323.

Gu, S.; Lillicrap, T.; Ghahramani, Z.; Turner, R. E.; and Levine, S. 2016. Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*.

Guss, W. H., and Salakhutdinov, R. 2018. On characterizing the capacity of neural networks using algebraic topology. *arXiv preprint arXiv:1802.04443*.

Henderson, P.; Islam, R.; Bachman, P.; Pineau, J.; Precup, D.; and Meger, D. 2017. Deep reinforcement learning that matters. *arXiv preprint arXiv:1709.06560*.

Kakade, S. M. 2002. A natural policy gradient. In *Advances in neural information processing systems*, 1531–1538.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

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

Maas, A. L.; Hannun, A. Y.; and Ng, A. Y. 2013. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, 3.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, 1928–1937.

Montavon, G.; Orr, G. B.; and Müller, K.-R. 1998. Tricks of the trade.

Nair, V., and Hinton, G. E. 2010. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, 807–814.

Nesterov, Y. 1983. A method for unconstrained convex minimization problem with the rate of convergence o (1/k^ 2). In *Doklady AN USSR*, volume 269, 543–547.

Rosenblatt, F. 1962. Principles of neurodynamics.

Rumelhart, D. E.; Hinton, G. E.; and Williams, R. J. 1985. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science.

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

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017. Mastering the game of go without human knowledge. *Nature* 550(7676):354.

Sutton, R. S.; McAllester, D. A.; Singh, S. P.; and Mansour, Y. 2000. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, 1057–1063.

Tieleman, T., and Hinton, G. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* 4(2):26–31.

Todorov, E.; Erez, T.; and Tassa, Y. 2012. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, 5026–5033. IEEE.

van Hasselt, H. P.; Guez, A.; Hessel, M.; Mnih, V.; and Silver, D. 2016. Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems*, 4287–4295.

Vincent, P.; de Brébisson, A.; and Bouthillier, X. 2015. Efficient exact gradient update for training deep networks with very large sparse targets. In *Advances in Neural Information Processing Systems*, 1108–1116.

Wang, Z.; Schaul, T.; Hessel, M.; Van Hasselt, H.; Lanctot, M.; and De Freitas, N. 2015. Dueling network architectures for deep reinforcement learning. *arXiv preprint arXiv:1511.06581*.

Zisserman. 2013. C25 optimization. [accessed 23-August-2018].

# ANS: Adaptive Network Scaling for Deep Rectifier Reinforcement Learning Models – Appendix

**Yen-Yu Chang**
Department of Electrical Engineering
yenyu@stanford.edu

## 1 Proof of Proposition 2.1

**Proposition.** *Consider $f_n(\boldsymbol{p}) = \boldsymbol{w}^\top \boldsymbol{p} + b$ and $\|\boldsymbol{p}\| = f_p(\mathcal{B})$ is normally distributed [1]. For the distribution of the angle between $\boldsymbol{w}$ and $\boldsymbol{p}$, we follow empirical distribution if necessary. If neuron $n$ is pseudo-dying with respect to $\mathcal{B}$, then the probability that $\mathcal{B} \cup \{\boldsymbol{x}_{B+1}\}$ is not pseudo-dying ReLU is upper bounded by*

*1. $|\boldsymbol{w}^\top \boldsymbol{p}_i| \geq |b|$ and $\boldsymbol{w}^\top \boldsymbol{p}_i < 0$*

$$\Pr(\textit{not pseudo-dying ReLU}) \leq \frac{1}{2}\left[1 + erf\left(\frac{-1}{\sqrt{2B}}\right)\right]. \tag{1}$$

*2. $|\boldsymbol{w}^\top \boldsymbol{p}_i| \leq |b|$ and $b < 0$*

$$\Pr(\textit{not pseudo-dying ReLU})$$
$$\leq \frac{1}{2}\left[1 - erf\left(\sqrt{\frac{2(B-1)}{B}}\left(1 - \frac{\bar{\mu}}{|b|/\|\boldsymbol{w}\|\,|\cos\theta_{min}|}\right)\right)\right]. \tag{2}$$

*For other cases,* $\Pr(\textit{not pseudo-dying ReLU}) = 0$.

*Proof.* If neuron $n$ is a pseudo-dying ReLU with respect to $\mathcal{B}$, we have

$$f_n(\boldsymbol{p}_i) = f_n(f_p(\boldsymbol{x}_i)) < 0, \forall \boldsymbol{x}_i \in \mathcal{B}. \tag{3}$$

We consider two cases that satisfy Equation 3

1. $|\boldsymbol{w}^\top \boldsymbol{p}_i| \geq |b|$ and $\boldsymbol{w}^\top \boldsymbol{p}_i < 0$
   With Cauchy-Schwarz inequality, we have $\|\boldsymbol{p}_i\| \geq |b|/\|\boldsymbol{w}\|$. Therefore, the probability of sampling an instance that is not dying ReLU is

$$\Pr(\text{not pseudo-dying ReLU})$$
$$= \Pr(\|\boldsymbol{p}_i\| < \frac{|b|}{\|\boldsymbol{w}\|}) \tag{4}$$
$$= \frac{1}{2}\left[1 + \text{erf}\left(\frac{|b|/\|\boldsymbol{w}\| - \bar{\mu}}{\bar{\sigma}\sqrt{2}}\right)\right], \tag{5}$$

[1] A more proper setting is to assume a truncated normal distribution. However, to simplified the derivation we use normal distribution and assume that the distribution is far enough from zero.

where $\bar{\sigma}$ and $\bar{\mu}$ are the estimate standard deviation and the estimate mean respectively. The estimate variance is bounded by

$$\bar{\sigma}^2 = \frac{1}{(B-1)}\sum(\|\boldsymbol{p}_i\| - \bar{\mu})^2 \tag{6}$$
$$\leq \frac{1}{(B-1)}\left[B(B-1)(\frac{|b|}{\|\boldsymbol{w}\|} - \bar{\mu})^2\right] \tag{7}$$
$$= B(\frac{|b|}{\|\boldsymbol{w}\|} - \bar{\mu})^2. \tag{8}$$

Therefore,

$$\Pr(\text{not pseudo-dying ReLU}) \leq \frac{1}{2}\left[1 + \text{erf}\left(\frac{-1}{\sqrt{2B}}\right)\right]. \tag{9}$$

2. $|\boldsymbol{w}^\top \boldsymbol{p}_i| \leq |b|$ and $b < 0$
   In this case, it should be noted that if $\boldsymbol{p}_i$ and $\boldsymbol{w}$ are orthogonal, then the probability of being not pseudo-dying ReLU is zero. We further consider the case that $|\cos\theta| \geq |\cos\theta_{\min}|$. Similarly, we have $\|\boldsymbol{p}_i\| \leq |b|/\|\boldsymbol{w}\|\,|\cos\theta_{\min}|$.

$$\Pr(\text{not pseudo-dying ReLU})$$
$$= \Pr(\|\boldsymbol{p}_i\| > |b|/\|\boldsymbol{w}\|\,|\cos\theta_{\min}|) \tag{10}$$
$$= \frac{1}{2}\left[1 - \text{erf}\left(\frac{|b|/\|\boldsymbol{w}\|\,|\cos\theta_{\min}| - \bar{\mu}}{\bar{\sigma}\sqrt{2}}\right)\right]. \tag{11}$$

Since the samples $\|\boldsymbol{p}_i\|$ are bounded by zero and $|b|/\|\boldsymbol{w}\|\,|\cos\theta_{\min}|$, the variance is bounded by

$$\bar{\sigma}^2 \leq \frac{B}{(B-1)}\frac{(|b|/\|\boldsymbol{w}\|\,|\cos\theta_{\min}|)^2}{4}. \tag{12}$$

Therefore,

$$\Pr(\text{not pseudo-dying ReLU})$$
$$\leq \frac{1}{2}\left[1 - \text{erf}\left(\sqrt{\frac{2(B-1)}{B}}\left(1 - \frac{\bar{\mu}}{|b|/\|\boldsymbol{w}\|\,|\cos\theta_{\min}|}\right)\right)\right]. \tag{13}$$

$\square$

### 1.1 Adaptive Network Scaling

Please refer to Algorithm 1 for the details of ANS.

**Algorithm 1** Adaptive Network Scaling

---

Given tolerance $T$, increasing ratio $c_{\text{inc}}$ and decreasing ratio $c_{\text{dec}}$

Initialize $t = 0$, $t_{\text{stop}} = 0$, $m_t = 0$, $R_{\text{prev}} = -\infty$, $\hat{m}_{\text{max}} = -\infty$ and *reverse* = False.

**for** each episode **do**
    $t \leftarrow t + 1$
    $t_{\text{stop}} \leftarrow t_{\text{stop}} + 1$
    $m_t \leftarrow \beta m_{t-1} + (1-\beta)R_t$ ▷ Update estimate of the mean of the return
    $\hat{m_t} \leftarrow m_t/(1-\beta^t)$
    **if** $\hat{m}_t > \hat{m}_{max}$ **then**
        $\hat{m}_{max} \leftarrow \hat{m}_t$
        $t_{\text{stop}} = 0$
    **end if**

    **if** $t_{\text{stop}} > T$ **then** ▷ Check if we should increase or decrease the scale
        **if** *reverse* and $\hat{m}_{\text{max}} \leq R_{\text{prev}}$ **then**
            **break**
        **else if** *reverse* and $\hat{m}_{\text{max}} > R_{\text{prev}}$ **then**
            scale the reward function with $c_{\text{dec}}$
            *NetworkScaling*$(g_{W,b}, c_{\text{dec}})$
        **else if** not *reverse* and $\hat{m}_{\text{max}} \leq R_{\text{prev}}$ **then**
            scale the reward function with $c_{\text{dec}}$
            *NetworkScaling*$(g_{W,b}, c_{\text{dec}})$
            *reverse*←True
        **else**
            scale the reward function with $c_{\text{inc}}$
            *NetworkScaling*$(g_{W,b}, c_{\text{inc}})$
        **end if**
        $R_{\text{prev}} \leftarrow \hat{m}_{\text{max}}$
        reinitialize $t, t_{\text{stop}}, m_t, \hat{m}_{\text{max}}$
    **end if**
**end for**

---

## 2 ReLU Experiments

We present all the experiment results here. For conciseness, we put the legend in the last figure.

### 2.1 A2C

**Learning Rate** Please refer to Figure 1.

**Reward Scale** Please refer to Figure 2 to 6 for details.

**ELU, Leaky-ReLU and ReLU** Please refer to Figure 7.

### 2.2 DDPG

**Learning Rate** Please refer to Figure 8.

**Reward Scale** Please refer to Figure 9 to 11 for details.

**ELU, Leaky-ReLU, ReLU** Please refer to Figure 12.



(a) ReLU of layer 1

(b) ReLU of layer 2

(c) cumulative reward

Figure 1: A2C learning rate comparison on Hopper-v2.



(a) ReLU of layer 1

(b) ReLU of layer 2

(c) cumulative reward
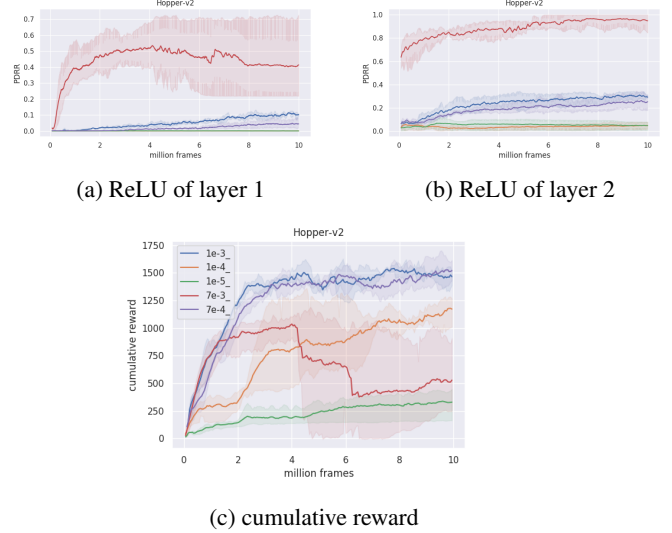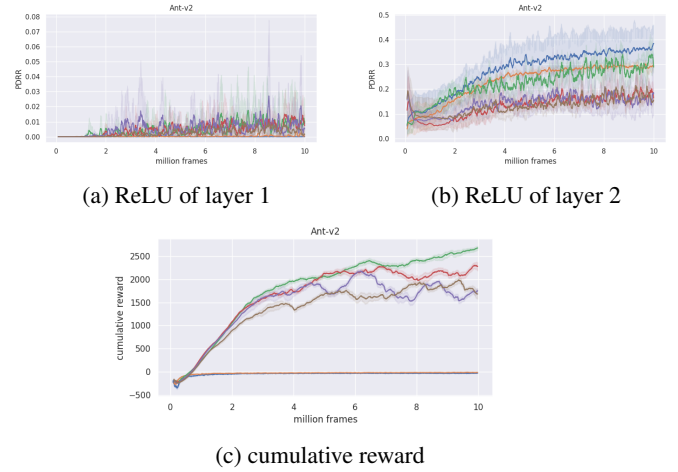
Figure 2: A2C on Ant-v2 with different reward scales.



(a) ReLU of layer 1

(b) ReLU of layer 2

(c) cumulative reward
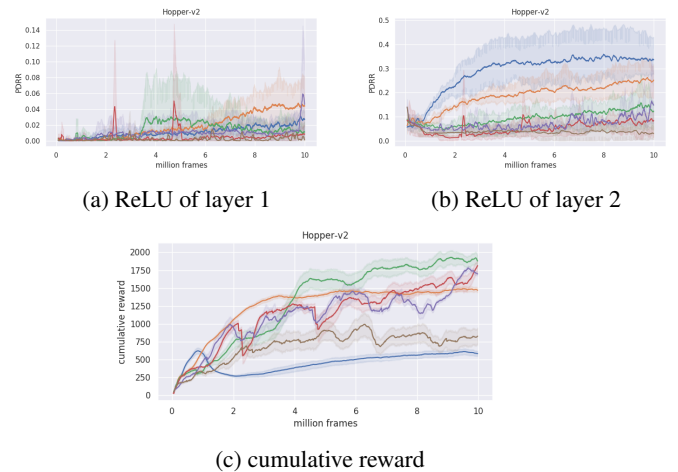
Figure 3: A2C on Hopper-v2 with different reward scales.

(a) ReLU of layer 1

(b) ReLU of layer 2

(c) cumulative reward
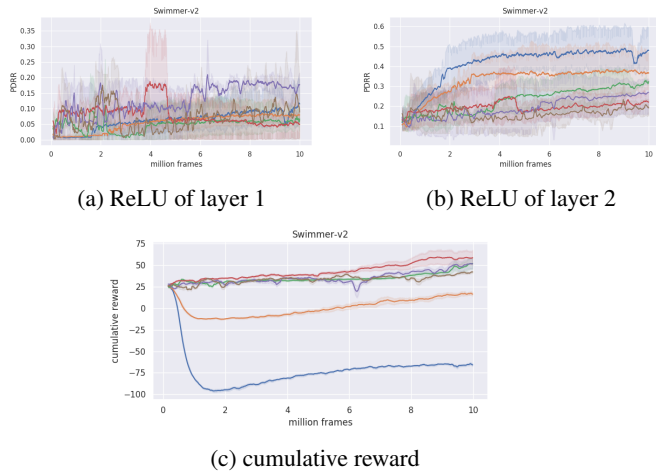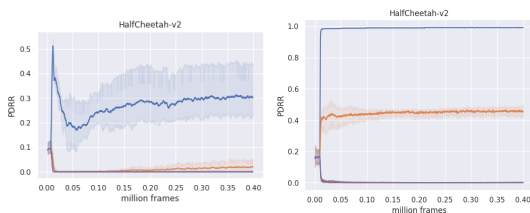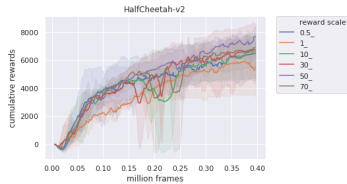
Figure 4: A2C on Swimmer-v2 with different reward scales.
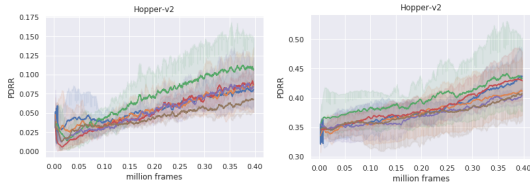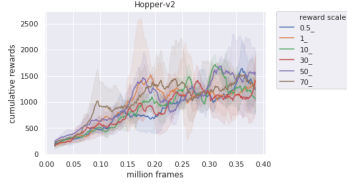


(a) ReLU of layer 1

(b) ReLU of layer 2

(c) cumulative reward
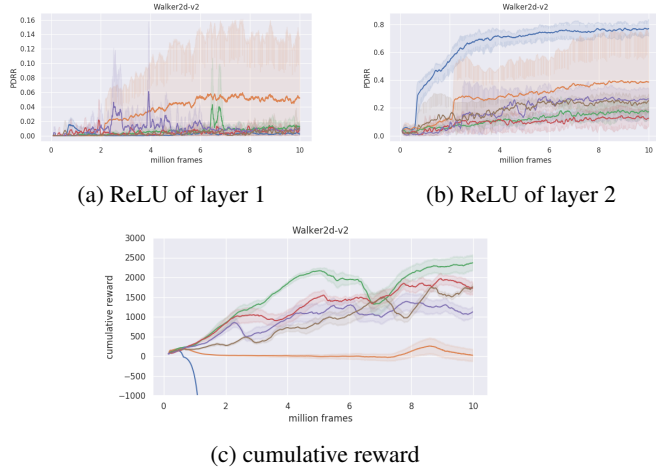
Figure 5: A2C on Walker2d-v2 with different reward scales.



(a) ReLU of layer 1

(b) ReLU of layer 2

(c) cumulative reward
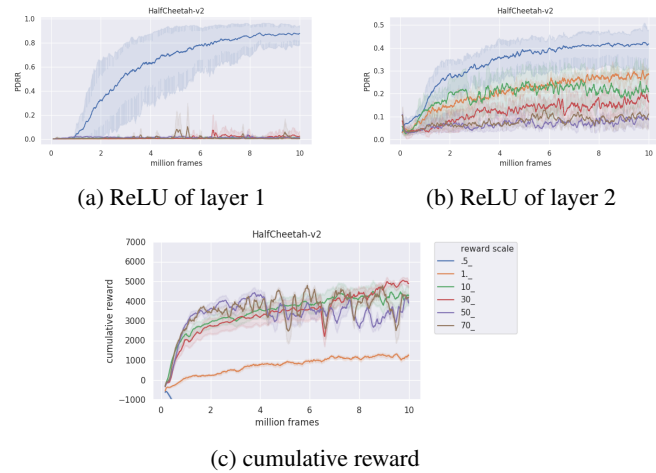
Figure 6: A2C on HalfCheetah-v2 with different reward scales.



(a) Scale 0.5.

(b) Scale 1.0.

(c) Scale 10.

(d) Scale 30.

(e) Scale 50.

(f) Scale 70.
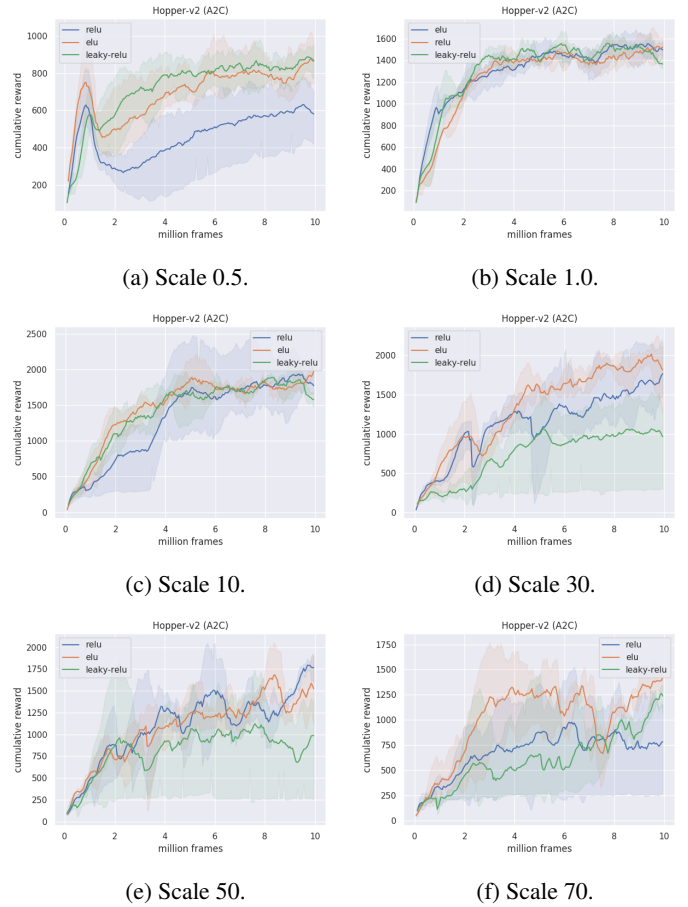
Figure 7: A2C activations comparison with different reward scales.



(a) ReLU of layer 1

(b) ReLU of layer 2

(c) cumulative reward
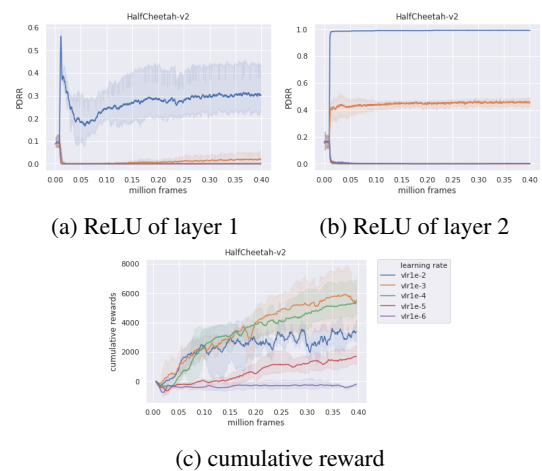
Figure 8: DDPG learning rate comparison on HalfCheetah-v2.

(a) ReLU of layer 1



(b) ReLU of layer 2



(c) cumulative reward

Figure 9: DDPG on HalfCheetah-v2 with different reward scales.
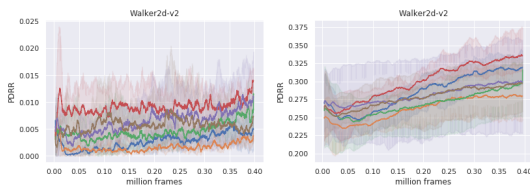


(a) ReLU of layer 1



(b) ReLU of layer 2



(c) cumulative reward

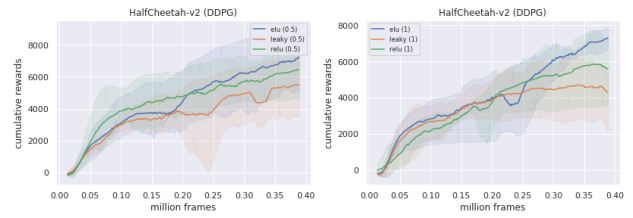Figure 10: DDPG on Hopper-v2 with different reward scales.



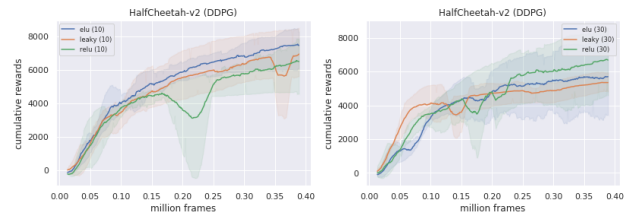(a) ReLU of layer 1



(b) ReLU of layer 2



(c) cumulative reward

Figure 11: DDPG on Walker2d-v2 with different reward scales.
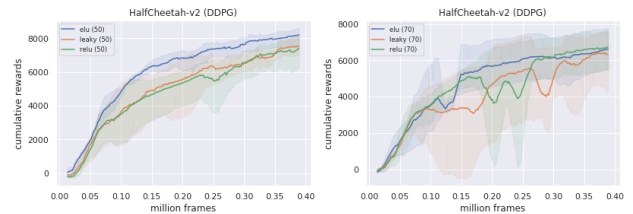


(a) Scale 0.5.



(b) Scale 1.0.



(c) Scale 10.



(d) Scale 30.



(e) Scale 50.



(f) Scale 70.

Figure 12: DDPG activations comparison with different reward scales.