Hadamard Representations: Augmenting Hyperbolic Tangents in RL

Anonymous authors
Paper under double-blind review

Abstract

Activation functions are one of the key components of a deep neural network. The most commonly used activation functions can be classed into the category of continuously differentiable (e.g. tanh) and piece-wise linear functions (e.g. ReLU), both having their own strengths and drawbacks with respect to downstream performance and representation capacity through learning (e.g. measured by the number of dead neurons and the effective rank). In reinforcement learning, the performance of continuously differentiable activations often falls short as compared to piece-wise linear functions. We provide insights into the vanishing gradients associated with the former, and show that the dying neuron problem is not exclusive to ReLU's. To alleviate vanishing gradients and the resulting dying neuron problem occurring with continuously differentiable activations, we propose a Hadamard representation. Using deep Q-networks, proximal policy optimization and parallelized Q-networks in the Atari domain, we show faster learning, a reduction in dead neurons and increased effective rank.

1 Introduction

Out of all activation functions, the Rectified Linear Unit (ReLU) (Fukushima, 1969; Nair & Hinton, 2010) and its variants (Xu et al., 2015; Klambauer et al., 2017) have emerged as the most widely used and generally best-performing activation functions up until this day (Jarrett et al., 2009; Goodfellow et al., 2016). The strength of the ReLU activation lies in its ability to naturally avoid vanishing gradients when used in deeper networks, in contrast to the continuously differentiable activation functions, such as the sigmoid and the hyperbolic tangent (Glorot & Bengio, 2010).

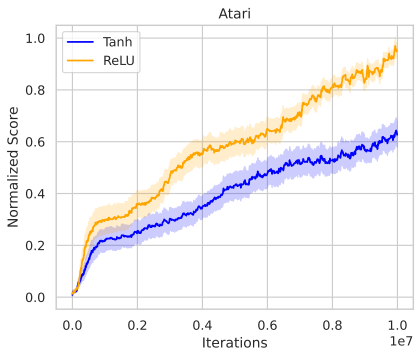


Figure 1: Normalized performance training DQN in the Atari domain where the nonlinear activation function of the representation, defined as the final hidden layer, is either ReLU or tanh. A strong performance discrepancy due to a relatively small architectural change can be observed.

A common drawback of using the ReLU activation is its limited expressivity in the context of shallow networks (see Fig. 2), as well as the phenomenon known as the dying ReLU problem (He et al., 2015; Lu et al., 2019). As training progresses, the number of dying ReLU's tend to increase, resulting in a dying network and loss of network capacity (Dubey et al., 2022).

In reinforcement learning (RL) (Sutton & Barto, 2018), the dying neuron phenomenon is more prevalent than in supervised learning due to the use of non-stationary targets (Sokar et al., 2023). However, even though training results in a large number of dying ReLU's (Gulcehre et al., 2022; Sokar et al., 2023), the ReLU function still remains the most popular activation for performance reasons (Henderson et al., 2018). Similar to supervised learning (Teney et al., 2024), continuously differentiable activation functions such as the hyperbolic tangent are therefore not commonly used in RL (see Fig. 1). However, one might argue that their symmetrical, bounded shape and smooth

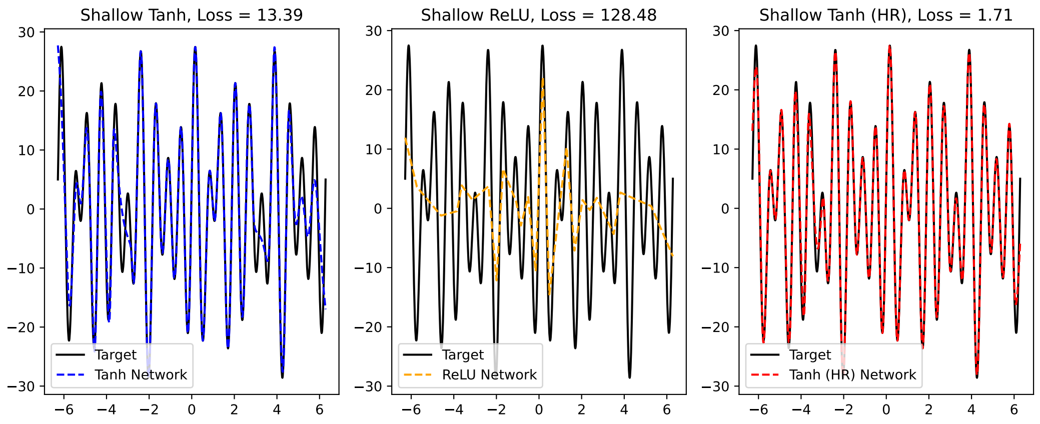


Figure 2: A regression of three shallow neural network architectures on a random complex sinusoidal function ($y = 10 * torch.sin(7 * x) + 15 * torch.sin(10 * x) + 5 * torch.cos(5 * x)$). The Tanh (HR) network emerges as the strongest function approximator, even while having less trainable parameters (501 vs 601 for Tanh & ReLU). To make a fair comparison, the Tanh and ReLU networks have one single hidden layer of size 200, while the Tanh (HR) network has a hidden layer of size 100. For the Tanh (HR) network however, we use two parallel linear layers preceding the hidden layer in order to be able to use the single hidden layer as the Hadamard product of two activations (see Section 3). For experiments comparing deeper networks, we refer the reader to Appendix C.1.

gradient landscape offer optimization advantages that the ReLU lacks. Recent findings also indicate that a hidden layer activated by a hyperbolic tangent displays a high effective rank and thus a high layer expressivity (Kumar et al., 2021; Gulcehre et al., 2022). Despite being a theoretically sound candidate, its lack of success in RL has not been thoroughly investigated.

We therefore aim to provide insights into the hyperbolic tangent's suboptimality, revealing that the vanishing gradient problem leads to dying neurons and under-utilization of the full network capacity. Based on these insights, we mitigate said effects by augmenting the original architecture. Specifically, we focus our research on the activations of an encoder, defined as the hidden layer, and provide an alternative to the conventional parameterization of this representation. Our contributions can be summarized as follows:

- We show that, in reinforcement learning, dying hyperbolic tangents are a phenomenon of a similar scale as the dying ReLU problem, but have a more profound effect on performance.
- A Hadamard representation (HR) is proposed, defining a latent representation as the Hadamard product of two separate, individually parameterized activation vectors.
- We empirically show that, without hyperparameter tuning or the use of auxiliary losses, using Hadamard representations yields notable performance gains in multiple algorithms in the Atari domain, and reveal how it decreases dying neurons and increases the internal representations' effective rank.

2 Preliminaries

We consider an agent acting within its environment, where the environment is modeled as a discrete Markov Decision Process (MDP) defined as a tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$. $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the environment's transition function, $R : \mathcal{S} \times \mathcal{A} \to \mathcal{R}$ is the environment's reward function and $\gamma \in [0, 1)$ is the discount factor. A replay buffer B is used to store visited states $s_t \in \mathcal{S}$ that were followed by actions $a_t \in \mathcal{A}$ and resulted in the rewards $r_t \in \mathcal{R}$ and the next states s_{t+1}. One entry in B contains a tuple of past experience (s_t, a_t, r_t, s_{t+1}). The agent's goal is to learn a policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes the expectation of the discounted return $V^\pi(s) = \mathbb{E}_\tau[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \mid s_t = s]$, where τ is a trajectory following the policy π.

Furthermore, we examine the setting where a high-dimensional state ($s_t \in \mathbb{R}^v$) is compressed into lower-dimensional activations $z_t \in \mathcal{Z} = \mathbb{R}^w$ where we call $\mathcal{Z}$ the representation space with $w \leq v$. This is done by means of a neural network encoding $e : \mathcal{S} \to \mathcal{Z}$ where e represents the encoder.

3 Augmenting Hyperbolic Tangents

Continuously differentiable activations such as the hyperbolic tangent ($\tanh$) and the sigmoid (σ) activations are fundamentally different than the ReLU or its piece-wise linear descendants, which are non-symmetric and have a large part of the input space mapped to zero (leading to sparsity). The hyperbolic tangent and the sigmoid output values in the ranges $[-1, 1]$ and $[0, 1]$, respectively. These functions are defined as $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ and $\sigma(x) = \frac{1}{1+e^{-x}}$.

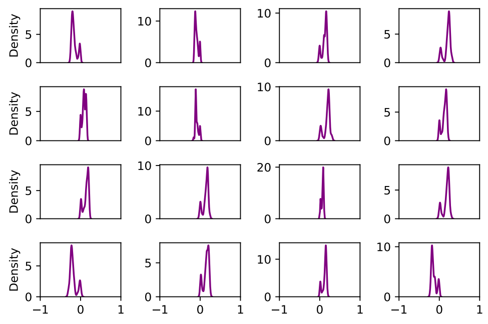
Both functions have the advantage of being differentiable everywhere, as well as being bounded. Furthermore, the sigmoid is well suited for output probabilities, while the tanh is convenient when requiring a zero-centered symmetrical output. However, both functions exhibit the vanishing gradient problem for saturating activations (Glorot & Bengio, 2010; Goodfellow et al., 2016).

Dying Hyperbolic Tangents

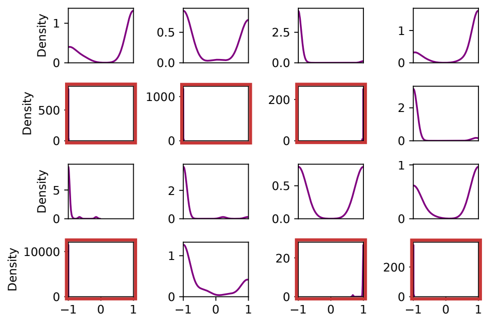
Although common literature has focused on the dying ReLU problem (He et al., 2015; Lu et al., 2019; Gulcehre et al., 2022; Sokar et al., 2023), we find that hidden layers activated by hyperbolic tangents similarly show strong dying neuron behavior. When using any activation function in a deep neural network, a single neuron $\alpha_i, i \in \mathbb{R}^w$, with w the layer dimension, is saturated or dying if:

$$\alpha_i \approx \Omega \ \forall \ s_t \in B \tag{1}$$

Where Ω represents the saturation value and s_t is an observation in buffer B. In practice, a mini-batch of observations is evaluated instead of the whole dataset in B. For the hyperbolic tangent, given that it is an asymptotic function near its saturation point, an approximate equality must be considered, as the classification of its saturation will always remain qualitative ($|\alpha_i| \neq 1 \ \forall \ s_t \in B$) . To approximate the condition given in Eq. 1, the amount of dying hyperbolic tangents is calculated by using a kernel density estimation (KDE) (Silverman, 1986) on the activations $\alpha_i, i \in \mathbb{R}^w$ of each individual neuron in the activation layer. In order to visualize activations in a hidden layer, a fixed subset of the KDE's of the neurons α_i is taken. A clear visualization of dying hyperbolic tangents during training in the Atari Breakout environment can be seen by analyzing sixteen individual neuron KDE's in Fig. 3.



(a) Tanh - 10^6 iterations



(b) Tanh - 10^7 iterations

Figure 3: Kernel Density Estimations (KDE) over a subset of 16 neurons in the compressed representation z_t after training DQN Mnih et al. (2015) in the Breakout environment using a hyperbolic tangent activation for z_t. Each neuron represents one dimension of the representation $z_t \in \mathbb{R}^{512}$. Red outlines represent dying neurons, where a near infinite sized density spike occurs at either 1 or -1.

More details on the KDE calculation and dying neuron classification can be found in Appendix B.

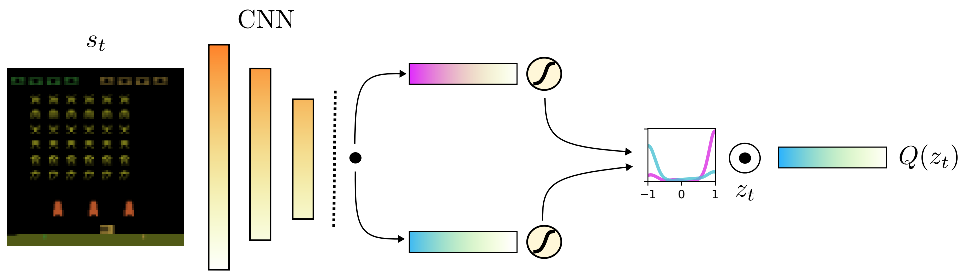


Figure 4: A visualisation of the Hadamard Representation architecture combined with the nature DQN architecture applied on a snapshot of the 'SpaceInvaders' Atari environment (Mnih et al., 2015). Vertical and horizontal bars represent convolutional and linear layers, respectively. A parallel hidden layer providing independently parameterized activations is integrated, where the Hadamard product of the two hidden layers represents the final latent representation z_t used for downstream learning.

Hadamard Representations (HR)

As Fig. 3 indicates that the activation of z_t with a hyperbolic tangent leads to saturation and dying neurons, an augmentation of the representation architecture is proposed. In the conventional encoder setting, the networks' final hidden layer is defined as $z^{enc}(x) = f(A_1x + B_1)$, with A_i and B_i representing weight and bias parameters, the function $f()$ representing a nonlinear activation function while x is the set of activations from the previous layer. In order to reduce the information dependence on a single set of neurons, we propose using a Hadamard representation that augments the original representation with a parallel representation layer z^*. This can be interpreted as using a single highway layer with a closed carry gate (Srivastava et al., 2015), or as an augmented version of the Gated Linear Unit (GLU) (Dauphin et al., 2017). The final representation is defined as the Hadamard product between the aforementioned activations $z(x) = z^{enc}(x) \cdot z^*(x)$, where $z^*(x) = f(A_2x + B_2)$. A visualization of the proposed architecture can be found in Fig. 4.

Preventing Dying Neurons

Our key hypothesis is that the Hadamard representation can prevent saturation, hence alleviating vanishing gradients and dying neurons (See Eq. 1). To support our hypothesis, we investigate the derivative of a product of two functions. For the product of two arbitrary functions $g(x) \cdot h(x)$, the derivative is defined as $\underline{g'(x)h(x)} + \underline{g(x)h'(x)}$. In the context of using a sigmoid activation function for $f(x)$, the derivative of $z(x)$ becomes:

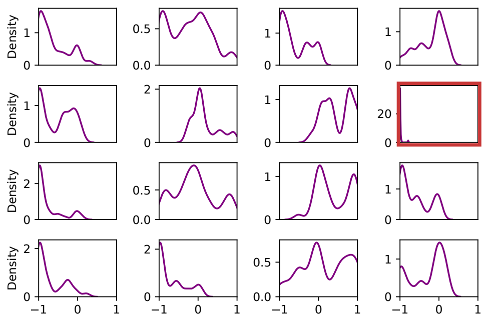
$$\begin{aligned} z'(x) = &\ \underline{A_1\sigma(A_1x + B_1)(1 - \sigma(A_1x + B_1))\sigma(A_2x + B_2)} \\ &+ \underline{A_2\sigma(A_1x + B_1)\sigma(A_2x + B_2)(1 - \sigma(A_2x + B_2))} \end{aligned}$$

If a neuron from $f(A_1x + B_1) = 0 \ \forall \ x$, the gradient of the product becomes 0 while if a positive saturation is experienced i.e. $f(A_1x + B_1) = 1 \ \forall \ x$, $z'(x)$ can remain nonzero. For a product of two hyperbolic tangent functions, the derivative is defined as:

$$z'(x) = \underline{A_1 \text{sech}^2(A_1x + B_1) \tanh(A_2x + B_2)} + \underline{A_2 \text{sech}^2(A_2x + B_2) \tanh(A_1x + B_1)}$$

In this context, sech^2 is the derivative of the hyperbolic tangent function. Unlike the sigmoid, the hyperbolic tangent saturates to nonzero values, ensuring that if and only if both parts are saturated, product saturation occours. Thus, when $g(x)$ saturates, $h(x)$ still keeps a non-trivial gradient in the product, providing a mechanism to avoid vanishing gradients. We visualize the kernel densities during training with a Hadamard representation in Fig. 5. The individual representations before taking the Hadamard product can be found in Appendix B.

Taking a more formal approximation of neuron collapse, we start by defining the probability of a single neuron saturating as $\mathbf{p}$. Furthermore, in the case of a sigmoid or hyperbolic tangent, we assume symmetric saturation probabilities to both ends, defining the probability of a neuron saturating to one end of the spectrum as $0.5\mathbf{p}$. Lastly, we make an independence assumption between two individual neurons. Under these assumptions, we show that interpreting a neuron as the product of two individual neurons can change saturation probabilities depending on the neuron's activation function.


(a) Tanh (HR) - 10^6 iterations

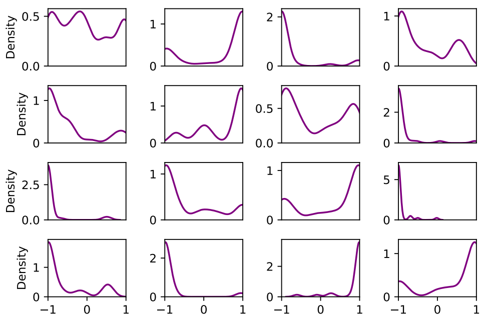

(b) Tanh (HR) - 10^7 iterations

Figure 5: Kernel Density Estimations (KDE) over a subset of 16 neurons in the compressed representation z_t after training DQN in the Breakout environment using a Hadamard representation (HR) with hyperbolic tangents. The Hadamard representation tends to quickly utilize the full range of the hyperbolic tangent while also mitigating dying neurons.

Hyperbolic Tangent: In the case of the hyperbolic tangent, product saturation only occurs if strictly both neurons are saturated. This results in a probability of $\mathbf{p} \cdot \mathbf{p} = \mathbf{p}^2$. Taking a product of hyperbolic tangent activated neurons thus reduces the probability of neuron saturation from $\mathbf{p}$ to $\mathbf{p}^2$.

Sigmoid: For the sigmoid function, product saturation occurs in two scenarios: Either one of the neurons is saturated towards zero or both neurons are saturated towards 1. The probability that a single neuron does not saturate towards zero is $(1 - 0.5\mathbf{p})$, and subsequently the probability that neither neuron saturates towards zero is $(1 - 0.5\mathbf{p})^2$. The probability that at least one of the two neurons saturates to zero is therefore $1 - (1 - 0.5\mathbf{p})^2 = \mathbf{p} - 0.25\mathbf{p}^2$. Adding the probability that both neurons saturate towards 1, which is $(0.5\mathbf{p})^2 = 0.25\mathbf{p}^2$, the final probability of the neuron product saturation is $\mathbf{p} - 0.25\mathbf{p}^2 + 0.25\mathbf{p}^2 = \mathbf{p}$. Taking a product of sigmoid activated neurons therefore does not reduce the probability of neuron collapse.

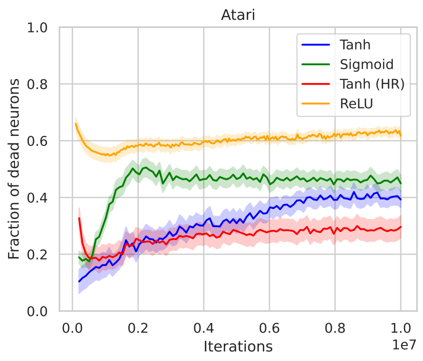
Rectified Linear-Unit: In the case of a ReLU activation, we also assume the probability of a single neuron dying to be $\mathbf{p}$. As we look at the product of two neurons, the probability that one of the two neurons does not saturate is therefore $1 - \mathbf{p}$, and the probability that both neurons do not saturate is $(1 - \mathbf{p})^2$. The probability that at least one neuron saturates is thus equal to $1 - (1 - \mathbf{p})^2 = 2\mathbf{p} - \mathbf{p}^2$. As the ReLU saturation results in strict zeroes, this results in the product also being zero. Taking a product of ReLU activated neurons therefore increases the final neuron saturation probability from $\mathbf{p}$ to $2\mathbf{p} - \mathbf{p}^2$. For an overview, we refer the reader to both Table 1 and the corresponding empirical evidence in Appendix C.3.

Table 1: Predicted dying neuron probabilities with and without a Hadamard representation.

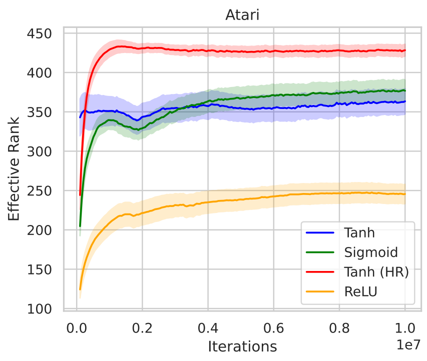
Activation Function	Dying Neuron Probability	Probability with HR	Difference
Hyperbolic Tangent	$\mathbf{p}$	$\mathbf{p}^2$	$-(\mathbf{p} - \mathbf{p}^2)$
Sigmoid	$\mathbf{p}$	$\mathbf{p}$	0
ReLU	$\mathbf{p}$	$2\mathbf{p} - \mathbf{p}^2$	$+(\mathbf{p} - \mathbf{p}^2)$

4 EXPERIMENTS

The main experiments and discussion are conducted with the value-based DQN algorithm (Mnih et al., 2015) in Atari, and complemented by an evaluation on the policy-based proximal policy optimization algorithm (PPO) (Schulman et al., 2017) and the parallelized Q-network (PQN) (Gallici et al., 2024). For more details on the PPO, DQN and PQN implementation, we refer the reader to Appendix A.1. In line with Gulcehre et al. (2022), for our qualitative experiments, a Hadamard Representation is only applied to the last hidden layer $z_t \in \mathbb{R}^{512}$. We perform qualitative experiments on 8 common, non-exploration driven Atari games. We aim to show the effect of a Hadamard representation on the representation's fraction of dying neurons, its effective rank and its downstream performance. For more information on the dying neuron and effective rank calculations, we refer the reader to Appendix. B.1 and Appendix. B.2, respectively. The performance is aggregated over 8 environments and normalized using the ReLU baseline scores. Finally, the PQN experiments are conducted with an HR on all hidden layers, showing the Median Human-Normalized scores on 51 Atari games.



(a) Dead Neurons



(b) Effective Rank

Figure 6: The average fraction of dead neurons (a) and the average effective rank (Kumar et al., 2021) (b) of the representation z_t when training DQN in the Atari domain for 10M iterations (40M frames). Similar to the well-known dying ReLU problem, hyperbolic tangent and sigmoid activations also exhibit strong dying neuron behavior during training. A Hadamard product of hyperbolic tangents reduces dead neurons in z_t and subsequently increases the effective rank of the representation.

Mitigating dying Hyperbolic Tangents

As mentioned in Section 3 and defined in Eq. 1, the number of dead neurons is equal to the amount of neurons that display the same saturated output for any given observation s_t. The amount of dead neurons over time when training on Atari can be seen in Fig. 6a. For the ReLU activation, around 60% of the neurons in the representation z_t die during training, while for the sigmoid and hyperbolic tangent activation this number is around 40%. When using a hyperbolic tangent Hadamard representation, a reduction in dead neurons as compared to using a single hyperbolic tangent can be observed. We credit this to the inherent ability of a Hadamard product of hyperbolic tangents to minimize long-term activation saturation, as explained in Section 3. Quantitative results of dead neurons can be found in Table 2, which is correlated with earlier predictions from Table 1.

Table 2: Dying neuron fractions with and without a Hadamard representation (HR).

Activation Function	**Dead Neuron Fraction**	**Dead Neuron Fraction (HR)**	**Difference**
Hyperbolic Tangent	0.39	0.30	-23%
Sigmoid	0.44	0.45	+2%
ReLU	0.62	0.73	+18%

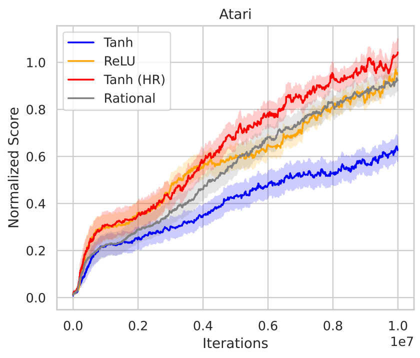
For more activation-specific dying neuron graphs, we refer the reader to Appendix C.3.

Increasing Effective Rank

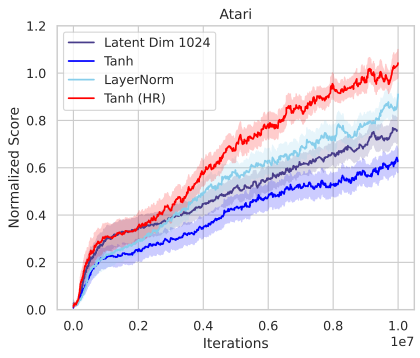
We additionally investigate the effective rank (Kumar et al., 2021) of the representation z_t during training, which can be seen in Fig. 6b. As observed by Gulcehre et al. (2022), a representation activated by a hyperbolic tangent or a sigmoid, already has a relatively high effective rank compared to a representation activated by a ReLU. Furthermore, similar to the results in our supervised learning experiments (see Fig. 2), using a Hadamard representation with hyperbolic tangents significantly improves the effective rank, which is strongly correlated to a network's 'expressivity'. Employing a Hadamard representation with ReLU activations significantly decreased the effective rank of the representation, as was expected.

Analyzing Performance in Atari

The influence of a Hadamard representation on downstream performance can be seen in Fig. 7. Correlating with the reduction in dying neurons and an increase in effective rank, a significant improvement over the standard hyperbolic tangent baseline is obtained, as well as an improvement over the default ReLU baseline. Furthermore, a comparison is made with the novel Rational (Delfosse et al., 2024) activation function as the activation in the final hidden layer. Fig. 7a shows that, although the Rational activation seems to be a stable learnable activation, it remains comparable to the ReLU.



(a) Normalized Score



(b) Ablations

Figure 7: Normalized performance with the standard deviation over the means in the Atari domain, after training DQN for 10M iterations (40M Frames). In (a), the Tanh (HR) significantly outperforms the baseline Tanh. The recently proposed Rational Activation (Delfosse et al., 2024) seems to have comparable performance to the ReLU in the 40M frame setting. In (b), several ablations including layer normalization (Ba et al., 2016) and an increased dimension of $z_t \in \mathbb{R}^{1024}$ with a reduced $\alpha = 5e^{-5}$ on hyperbolic tangent activations are shown. Note that no hyperparameter changes are done for the Hadamard Representation.

Examining further ablations in Fig. 8 shows that using a sophisticated piece-wise linear function such as the SELU activation (Klambauer et al., 2017) or using an addition rather than a product of hyperbolic tangents seems detrimental to performance. Furthermore, taking a product of 3 hyperbolic tangents (2HR) also appears to enhance performance, though there seems to be a negative effect in the early stages of training as compared to using a single Hadamard product. We hypothesize that this is the result of increased contracting behavior in the early stage of training due to increasing multiplication of hyperbolic tangent activations whose absolute values are < 1. Additional experiments combining the ReLU activation with a Hadamard representation can be found in Appendix C.4.

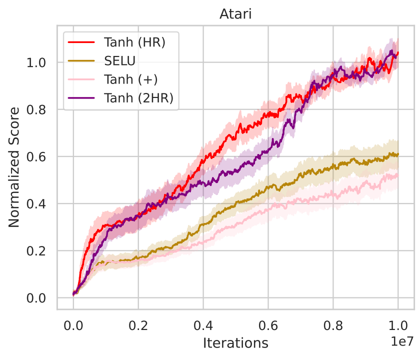
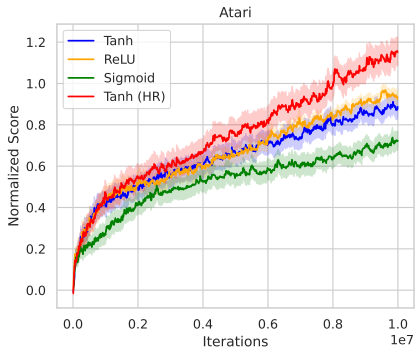


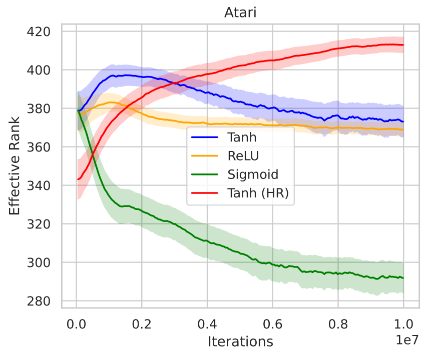
Figure 8: Additional ablations run on DQN in the Atari domain. Tanh (+) represents an addition rather than a Hadamard product, and Tanh (2HR) uses a triple Hadamard product.

Evaluating Hadamard representations on PPO

To evaluate results on a policy-based algorithm, additional experiments are run using the PPO algorithm (Schulman et al., 2017). For PPO, the internal architectural difference with DQN is that the compressed representation z_t precedes both a critic and an actor network, and thus receives policy and value gradients (see Appendix A.2). The results can be found in Fig. 9. Similarly to the results on DQN, the Tanh (HR) exhibits the highest effective rank and the best performance. The Tanh (HR) and the ReLU also seem to have consistent strong performance across both algorithms, whereas the pure hyperbolic tangent and sigmoid activations can be relatively unreliable across algorithms.



(a) Normalized Score



(b) Effective Rank

Figure 9: Normalized performance (a) and effective rank (b) in the Atari domain for 10M iterations (40M Frames) when training the PPO algorithm (Schulman et al., 2017).

Evaluating Hadamard representations on PQN

To more reliably test the performance gains one can achieve using Hadamard Representations, we evaluate their effects on the Parallelized Q-Network (PQN)in the full Atari suite without hard-exploration environments, for a total of 51 games with 5 seeds per game. PQN has recently emerged as a vectorized-friendly version of DQN, reporting both a speed up from DQN as well as better convergence (Gallici et al., 2024). We run tests implementing the Hadamard Representation in both all hidden layers and only the final linear hidden layer. The median human-normalized scores after training for 40M frames are shown in Fig. 10. We can clearly see a similar trend for the 51-game Atari suite, where using a Hadamard representation for all layers significantly increases the median performance as compared to the baseline PQN and CReLU (Abbas et al., 2023)

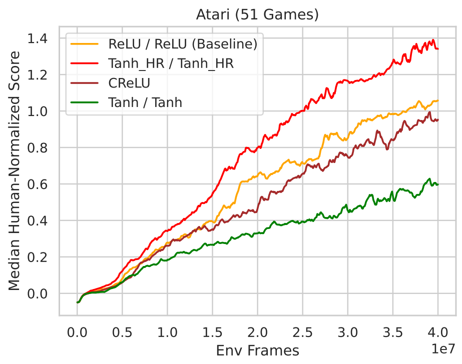


Figure 10: Median Human-Normalized scores on PQN in 51 Atari games for 5 seeds over 40M frames. Labels represent MLP / CNN activations. Employing a Hadamard Representation in the Convolutional layers as well as the MLP layer provides a significant performance improvement over the recent PQN baseline, as well as a CReLU (Abbas et al., 2023) implementation of PQN.

Hyperbolic Tangents turn Weights into Biases

Interestingly, it seems that the ReLU activations' performance is less correlated to its low effective rank and high amount of dying neurons than the continuously differentiable activations.

As also indicated by Teney et al. (2024), the exact reasons for the broad success of the ReLU activations are not yet fully understood. However, in the case of dying neurons, we aim to shed some light on this phenomenon.

When dying neurons occur in ReLU-activated layers, it basically prunes these neurons and the associated weights to the next layer. However, in hyperbolic tangent activated layers, dying neurons lead to an unintended phenomenon where weights associated with dead neurons effectively become biases.

Theorem 4.1. *When any set of neurons α^j in a hidden layer z^j collapses into nonzero values, the output to the next layer effectively changes from ($A^j z^j + B^j$) to ($A^j_{-*} z^j_{i-*} + B^{j+1} + A^j_* z^j_{i*}$), where $A^j_{-*} z^j_{-*}$ represent the active neurons multiplied by their corresponding forward-connected weights and $A^j_* z^j_{i*} = B^{j+1}_*$ represent the dead neurons, inducing a hidden bias.*

Proof. Take a set of neurons α^j_i and forward connected weights $\boldsymbol{w}^j_{\alpha_i}$ in layer z^j. The influence of these neurons on the next layer z^{j+1} is calculated as:

$$z^{j+1} = \sum_i \alpha^j_i \boldsymbol{w}^j_{\alpha_i} + B^{j+1}. \tag{2}$$

If the set of neurons dies and collapses into 0 ($\alpha^j_i = 0 \; \forall \; s \in \mathcal{S}$), which occurs when using ReLU activations, the influence on the next layer becomes 0, representing basic pruning:

$$\sum_i 0 \cdot \bar{\boldsymbol{w}}^j_{\alpha_i} = 0 \quad \forall s \in \mathcal{S}. \tag{3}$$

Where $\bar{\boldsymbol{w}}^j_{\alpha_i}$ is the set of weights connected to the dying neurons. However, for a hyperbolic tangent activation, if the set of neurons saturate into either -1 or 1 ($\alpha^j_i = \{1, -1\} \forall s \in \mathcal{S}$), the output is

$$\sum_i \{1, -1\} \cdot \bar{\boldsymbol{w}}^j_{\alpha_i} = B^{j+1}_* \quad \forall s \in \mathcal{S}. \tag{4}$$

As a result, the weight vector $\bar{\boldsymbol{w}}^j_{\alpha_i}$ corresponding to the dying activations only influences a bias B^{j+1}_* on the resulting hidden layer. Note that the bias B^{j+1}_* is constant for any input observation.

This emergent bias can hinder the optimization process by introducing unintended fixed contributions to a networks' next hidden layer or Q-values, reducing the flexibility of the network's representations and potentially reducing its performance in RL or supervised learning.

5 Related Work

Network Capacity in RL. Liu et al. (2019) investigated the need for sparse representations in the continuous control domain. Gulcehre et al. (2022) analyzed network expressiveness in RL by measuring the effective rank (Kumar et al., 2021) of the compressed representation, and found that hyperbolic tangent representations generally maintain high rank while not suffering strongly from rank decay as training continues. Related work used normalization techniques and action penalization to counteract high variance in pixel-based robotic control (Bjorck et al., 2022). Other work by Lyle et al. (2022) investigated capacity loss in RL and similarly found that, as training progresses, the inherent network capacity of RL algorithms decays. Further research by Nikishin et al. (2022) used network resets to counteract the primacy bias and Sokar et al. (2023) evaluated and mitigated the dying ReLU phenomenon in DQN, both operating in the sample efficiency setting. Nikishin et al. (2023) further studied plasticity injection for long-term training and Delfosse et al. (2024) applied rational activations (Molina et al., 2019) in RL to increase plasticity. Concurrent work by Dohare et al. (2024) used continual back-propagation to further alleviate plasticity loss. In another related direction, recent work has investigated network sparsity in RL, showing that a large part of network capacity might be unnecessary when training reinforcement learning (Arnob et al., 2021; Graesser et al., 2022; Sokar et al., 2022; Tan et al., 2023; Obando-Ceron et al., 2024). This provides further insights into why a ReLU can achieve strong performance despite resulting in a significant number of dead neurons.

Network Architecture in RL The origin of network optimization problems with hyperbolic tangents and sigmoids were empirically investigated by Glorot & Bengio (2010), where, according to the authors, a lot of mystery still surrounds the subject. Work by Srivastava et al. (2015) in supervised learning first looked at the idea of using products of hidden layers together with a 'gate' that determined the amount of information flow (Hochreiter & Schmidhuber, 1997). Using these ideas, the Resnet was invented (He et al., 2016) and also showed strong performance in combination with RL (Espeholt et al., 2018). Further work by Henderson et al. (2018) showed differences in RL performances over different network architectures and nonlinear activations. Work by Abbas et al. (2023) successfully applied ReLU concatenation (Shang et al., 2016) to improve continual learning while keeping a similar performance when training from scratch. Finally, recent work by Grooten et al. (2024) investigated raw pixel masking for distractions in RL using a parallel CNN input layer.

6 Conclusions and Discussion

This paper analyzed issues with continuously differentiable activations in RL and demonstrated that these activation functions also suffer from the dying neuron problem. Based on this analysis, we propose a novel representation architecture, the Hadamard Representation (HR), which enhances an encoder's final hidden layer by taking the Hadamard product with a parallel, independently parameterized activation layer. We further discussed and empirically showed that applying the Hadamard representation with hyperbolic tangents reduces the occurrence of dead neurons in the representation and increases layer expressiveness. In DQN, PQN and PPO, this approach significantly improved performance in Atari games compared to both standard representation parameterizations and merely increasing the representation dimension. Finally, we have given insights into the ability of the ReLU activation to be less affected the symptoms of dying neurons, as opposed to non-zero saturating activations such as the hyperbolic tangent. Future work could focus on further identifying the intricacies of the effects that different activation functions have on the resulting representation, in an attempt to push the potential of non piece-wise linear activations in reinforcement learning or even in a supervised learning setting (see Fig. 2). Also, we believe that an implementation of Hadamard-style architectures in a continual learning setting as in Abbas et al. (2023) or in Delfosse et al. (2024) could be promising.

7 Limitations

Using a Hadamard representation will double the incoming weights connected to a hidden layer. However, in Fig. 10, a comparison against the CReLU is shown, which equally doubles the network's parameters. Furthermore, recent work shows that a scaling baselines in Atari often leads to reduced performance (Obando-Ceron et al., 2024; Obando Ceron et al., 2024). Second, due to the vast amount of activation functions and the theoretical sound candidacy for the hyperbolic tangent, we have mostly focused our research on the combination of hyperbolic tangents. Further research into multiplicative representations could find even better activation combinations. A hyperparameter search could also give more insights into the strengths of the Hadamard Representation, since the baselines are specifically tuned for the ReLU activation. Finally, as this research performs experiments on DQN, PPO and PQN, integration of a Hadamard representation into more complex algorithms and architectures such as Rainbow (Hessel et al., 2018), Impala (Espeholt et al., 2018) or Schwarzer et al. (2021) would be interesting.

Reproducibility Statement

Constructing a Hadamard representation is a relatively straightforward process, of which the Pytorch pseudocode can be found in Appendix A.1. The DQN and PPO algorithms used are adopted from the compact `cleanrl` baselines (Huang et al., 2022). The PQN algorithm uses the implementation in the Github repository of Gallici et al. (2024). Finally, the details on the KDE calculation and the effective rank calculation can be found in Appendix B.1 and Appendix B.2.

REFERENCES

Zaheer Abbas, Rosie Zhao, Joseph Modayil, Adam White, and Marlos C. Machado. Loss of plasticity in continual deep reinforcement learning. In Sarath Chandar, Razvan Pascanu, Hanie Sedghi, and Doina Precup (eds.), *Proceedings of The 2nd Conference on Lifelong Learning Agents*, volume 232 of *Proceedings of Machine Learning Research*, pp. 620–636. PMLR, 22–25 Aug 2023. URL https://proceedings.mlr.press/v232/abbas23a.html.

Samin Yeasar Arnob, Riyasat Ohib, Sergey M. Plis, and Doina Precup. Single-shot pruning for offline reinforcement learning. *CoRR*, abs/2112.15579, 2021. URL https://arxiv.org/abs/2112.15579.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization, 2016. URL http://arxiv.org/abs/1607.06450. cite arxiv:1607.06450.

Johan Bjorck, Carla P Gomes, and Kilian Q Weinberger. Is high variance unavoidable in RL? a case study in continuous control. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=9xhgmsNVHu.

Yann N. Dauphin, Angela Fan, Michael Auli, and David Grangier. Language modeling with gated convolutional networks. In Doina Precup and Yee Whye Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 933–941. PMLR, 06–11 Aug 2017. URL https://proceedings.mlr.press/v70/dauphin17a.html.

Quentin Delfosse, Patrick Schramowski, Martin Mundt, Alejandro Molina, and Kristian Kersting. Adaptive rational activations to boost deep reinforcement learning. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=g90ysX1sVs.

Shibhansh Dohare, J. Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A. Rupam Mahmood, and Richard S. Sutton. Loss of plasticity in deep continual learning. *Nature*, 632 (8026):768–774, 2024. ISSN 1476-4687. doi: 10.1038/s41586-024-07711-7. URL https://doi.org/10.1038/s41586-024-07711-7.

Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. Activation functions in deep learning: A comprehensive survey and benchmark. *Neurocomputing*, 503:92–108, 2022. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2022.06.111. URL https://www.sciencedirect.com/science/article/pii/S0925231222008426.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1407–1416. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/espeholt18a.html.

Kunihiko Fukushima. Visual feature extraction by a multilayered network of analog threshold elements. *IEEE Transactions on Systems Science and Cybernetics*, 5(4):322–333, 1969. doi: 10.1109/TSSC.1969.300225.

Matteo Gallici, Mattie Fellows, Benjamin Ellis, Bartomeu Pou, Ivan Masmitja, Jakob Nicolaus Foerster, and Mario Martin. Simplifying deep temporal difference learning, 2024. URL https://arxiv.org/abs/2407.04811.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington (eds.), *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. URL https://proceedings.mlr.press/v9/glorot10a.html.

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. ISBN 9780262035613. URL https://books.google.co.in/books?id=Np9SDQAAQBAJ.

Laura Graesser, Utku Evci, Erich Elsen, and Pablo Samuel Castro. The state of sparse training in deep reinforcement learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 7766–7792. PMLR, 17–23 Jul 2022. URL `https://proceedings.mlr.press/v162/graesser22a.html`.

Bram Grooten, Tristan Tomilin, Gautham Vasan, Matthew E. Taylor, A. Rupam Mahmood, Meng Fang, Mykola Pechenizkiy, and Decebal Constantin Mocanu. Madi: Learning to mask distractions for generalization in visual deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS*, 2024.

Caglar Gulcehre, Srivatsan Srinivasan, Jakub Sygnowski, Georg Ostrovski, Mehrdad Farajtabar, Matthew Hoffman, Razvan Pascanu, and Arnaud Doucet. An empirical study of implicit regularization in deep offline RL. *Transactions on Machine Learning Research*, 2022. ISSN 2835-8856. URL `https://openreview.net/forum?id=HFfJWx60IT`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '16, pp. 770–778. IEEE, June 2016. doi: 10.1109/CVPR.2016.90. URL `http://ieeexplore.ieee.org/document/7780459`.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press, 2018. ISBN 978-1-57735-800-8.

Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In Sheila A. McIlraith and Kilian Q. Weinberger (eds.), *AAAI*, pp. 3215–3222. AAAI Press, 2018. URL `http://dblp.uni-trier.de/db/conf/aaai/aaai2018.html#HesselMHSODHPAS18`.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. URL `http://jmlr.org/papers/v23/21-1342.html`.

Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato, and Yann LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pp. 2146–2153, 2009. doi: 10.1109/ICCV.2009.5459469.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pp. 971–980, 2017. URL `https://proceedings.neurips.cc/paper/2017/hash/5d44ee6f2c3f71b73125876103c8f6c4-Abstract.html`.

Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. In *International Conference on Learning Representations*, 2021.

Vincent Liu, Raksha Kumaraswamy, Lei Le, and Martha White. The utility of sparse representations for control in reinforcement learning. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019. ISBN 978-1-57735-809-1. doi: 10.1609/aaai.v33i01.33014384. URL `https://doi.org/10.1609/aaai.v33i01.33014384`.

Lu Lu, Yeonjong Shin, Yanhui Su, and George E. Karniadakis. Dying relu and initialization: Theory and numerical examples. *CoRR*, abs/1903.06733, 2019. URL `http://dblp.uni-trier.de/db/journals/corr/corr1903.html#abs-1903-06733`.

Clare Lyle, Mark Rowland, and Will Dabney. Understanding and preventing capacity loss in reinforcement learning. In *International Conference on Learning Representations*, 2022. URL `https://openreview.net/forum?id=ZkC8wKoLbQ7`.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2 2015. doi: 10.1038/nature14236.

Alejandro Molina, Patrick Schramowski, and Kristian Kersting. Padé activation units: End-to-end learning of flexible activation functions in deep networks. In *International Conference on Learning Representations*, 2019.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *ICML 2010*, pp. 807–814, 2010.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 16828–16847. PMLR, 17–23 Jul 2022.

Evgenii Nikishin, Junhyuk Oh, Georg Ostrovski, Clare Lyle, Razvan Pascanu, Will Dabney, and Andre Barreto. Deep reinforcement learning with plasticity injection. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 37142–37159. Curran Associates, Inc., 2023. URL `https://proceedings.neurips.cc/paper_files/paper/2023/file/75101364dc3aa7772d27528ea504472b-Paper-Conference.pdf`.

Johan Obando-Ceron, Aaron Courville, and Pablo Samuel Castro. In deep reinforcement learning, a pruned network is a good network. *arXiv preprint arXiv:2402.12479*, 2024.

Johan Samir Obando Ceron, Ghada Sokar, Timon Willi, Clare Lyle, Jesse Farebrother, Jakob Nicolaus Foerster, Gintare Karolina Dziugaite, Doina Precup, and Pablo Samuel Castro. Mixtures of experts unlock parameter scaling for deep RL. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 38520–38540. PMLR, 21–27 Jul 2024. URL `https://proceedings.mlr.press/v235/obando-ceron24b.html`.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017. URL `https://api.semanticscholar.org/CorpusID:28695052`.

Max Schwarzer, Ankesh Anand, Rishab Goel, R Devon Hjelm, Aaron Courville, and Philip Bachman. Data-Efficient Reinforcement Learning with Self-Predictive Representations. In *International Conference on Learning Representations, ICLR*, 2021.

Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In Maria Florina Balcan and Kilian Q. Weinberger (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 2217–2225, New York, New York, USA, 20–22 Jun 2016. PMLR. URL `https://proceedings.mlr.press/v48/shang16.html`.

Bernard W Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.

Ghada Sokar, Elena Mocanu, Decebal Constantin Mocanu, Mykola Pechenizkiy, and Peter Stone. Dynamic sparse training for deep reinforcement learning. In Lud De Raedt (ed.), *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pp. 3437–3443. International Joint Conferences on Artificial Intelligence Organization, 7 2022. doi: 10.24963/ijcai.2022/477. URL `https://doi.org/10.24963/ijcai.2022/477`. Main Track.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org, 2023.

Rupesh K Srivastava, Klaus Greff, and Jürgen Schmidhuber. Training very deep networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL `https://proceedings.neurips.cc/paper_files/paper/2015/file/215a71a12769b056c3c32e7299f1c5ed-Paper.pdf`.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL `http://incompleteideas.net/book/the-book-2nd.html`.

Yiqin Tan, Pihe Hu, Ling Pan, Jiatai Huang, and Longbo Huang. Rlx2: Training a sparse deep reinforcement learning model from scratch. *CoRR*, abs/2205.15043, 2023. URL `https://arxiv.org/abs/2205.15043`.

Damien Teney, Armand Mihai Nicolicioiu, Valentin Hartmann, and Ehsan Abbasnejad. Neural redshift: Random networks are not random functions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4786–4796, June 2024.

Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *CoRR*, abs/1505.00853, 2015. URL `http://arxiv.org/abs/1505.00853`.

A Implementation Details

A.1 Hyperparameters

To evaluate, 8 different Atari environments are tested, using 5 different random seeds. For the mean scores, we take the mean over the eight environments. Our normalized score is calculated according to our baseline, the original implementation using a ReLU activation.

All the hyperparameters used in our experiments for DQN and PPO, respectively, are as reported in `cleanrl` (Huang et al., 2022). The hyperparameters can be found in Table 1 and Table 2.

Table 3: DQN Hyperparameters

Hyperparameter	**Value**	**Description**
Learning Rate	1×10^{-4}	Learning rate for the optimizer
Discount Factor (γ)	0.99	Discount for future rewards
Replay Memory Size	1,000,000	Size of the experience replay buffer
Mini-batch Size	32	Number of samples per batch update
Target Network Update Frequency	1000	Update frequency for the target network
Initial Exploration	1.0	Initial exploration rate in ϵ-greedy
Final Exploration	0.1	Final exploration rate in ϵ-greedy
Final Exploration Frame	1,000,000	Frame number to reach final exploration
Exploration Decay Frame	1,000,000	Frames over which exploration rate decays
Action Repeat (Frame Skip)	4	Number of frames skipped per action
Reward Clipping	[-1, 1]	Range to which rewards are clipped
Input Dimension	84 x 84	Dimensions of the input image
Latent Dimension	512	Dimension of the latent representation
Input Frames	4	Number of frames used as input
Training Start Frame	80,000	Frame number to start training
Loss Function	Mean Squared Error	Loss function used for updates
Optimizer	Adam	Optimization algorithm used
Optimizer ϵ	10^{-5}	Adam Epsilon

Table 4: PPO Hyperparameters

Hyperparameter	**Value**	**Description**
Learning Rate	2.5×10^{-4}	Learning rate for the optimizer
Discount Factor (γ)	0.99	Discount factor for future rewards
Number of Steps	128	Number of steps per environment before update
Anneal LR	True	Whether to anneal the learning rate
GAE Lambda	0.95	Lambda parameter for GAE
Number of Minibatches	4	Number of minibatches to split the data
Update Epochs	4	Number of epochs per update
Normalize Advantage	True	Whether to normalize advantage estimates
Clipping Coefficient	0.1	Clipping parameter for PPO
Clip Value Loss	True	Whether to clip value loss
Entropy Coefficient	0.01	Coefficient for entropy bonus
Value Function Coefficient	0.5	Coefficient for value function loss
Maximum Gradient Norm	0.5	Maximum norm for gradient clipping
Target KL	None	Target KL divergence between updates
Latent Dimension	512	Dimension of the latent representation
Optimizer	Adam	Optimization algorithm used
Optimizer ϵ	10^{-5}	Adam Epsilon

Table 5: PQN Hyperparameters (Gallici et al., 2024)

Hyperparameter	**Value**	**Description**
Total Timesteps	10,000,000	Total timesteps for training
Timesteps for Decay	10,000,000	Timesteps for decay functions (epsilon and lr)
Number of Environments	128	Number of parallel environments
Steps per Environment	32	Steps per environment in each update
Number of Epochs	2	Number of epochs per update
Number of Minibatches	32	Number of minibatches per epoch
Starting Epsilon	1.0	Starting epsilon for exploration
Final Epsilon	0.001	Final epsilon for exploration
Epsilon Decay Ratio	0.1	Decay ratio for epsilon
Epsilon for Test Policy	0.0	Epsilon for greedy test policy
Learning Rate	0.00025	Learning rate
Learning Rate Linear Decay	True	Use linear decay for learning rate
Max Gradient Norm	10.0	Max gradient norm for clipping
Discount Factor (γ)	0.99	Discount factor for reward
Lambda (λ)	0.65	Lambda for generalized advantage estimation
Episodic Life	True	Terminate episode when life is lost
Reward Clipping	True	Clip rewards to range [-1, 1]
Frame Skip	4	Number of frames to skip
Max No-Ops on Reset	30	Max number of no-ops on reset
Test During Training	True	Run evaluation during training
Number of Test Envs	8	Number of environments used for testing

Hadamard Implementation

Constructing a Hadamard representation is a straightforward process that only requires an additional, parallel linear layer. Starting from the flattened convolutional features in DQN, the Pytorch pseudocode is defined as follows:

```
flattened_features = conv_features.flatten(1)
representation1 = nn.Tanh(linear1(flattened_features))
representation2 = nn.Tanh(linear2(flattened_features))
hadamard = representation1 * representation2
Q-values = linear_q(hadamard)
```

Reinforcement Learning

In DQN, the action a_t is chosen following an ϵ-greedy policy. With probability ϵ, a random action is selected, and with $(1-\epsilon)$, the action maximizing the Q-value is chosen. The target Y_t is defined as:

$$Y_t = r_t + \gamma Q'(z_{t+1}, \arg\max_{a \in \mathcal{A}} Q(z_{t+1}, a)), \tag{5}$$

where $Q'(z,a)$ denotes the target Q-network, an auxiliary network that stabilizes the learning by providing a stable target for $Q(z,a)$. The parameters of Q' are updated less frequently to enhance learning stability. The loss function for training the network is:

$$\mathcal{L}_Q = \left| Y_t - Q(z_t, a) \right|^2. \tag{6}$$

Proximal Policy Optimization (PPO) operates on a different principle, utilizing policy gradient methods for policy improvement. PPO seeks to update the policy by maximizing an objective function while preventing large deviations from the previous policy through a clipping mechanism in the objective's estimator. The clipped policy gradient loss L^{CLIP} is defined as:

$$L^{CLIP}(\theta) = \mathbb{E}\left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\right], \tag{7}$$

where $r_t(\theta)$ represents the ratio of the probabilities under the new policy versus the old policy, and $\hat{A}_t$ is the advantage estimate at timestep t. This clipped surrogate objective ensures gradual and stable policy updates.

A.2 PPO Architecture

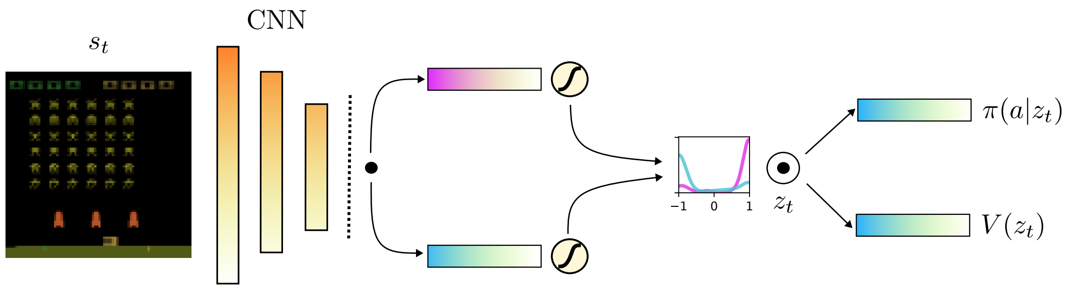
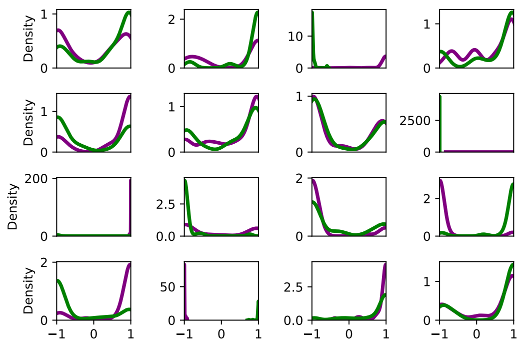


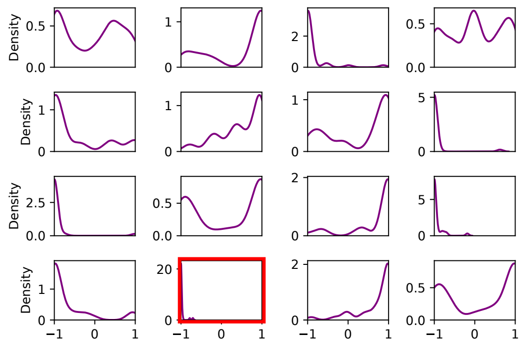
Figure 11: A visualisation of the Hadamard representation (HR) architecture combined with the PPO architecture applied on a snapshot of the 'SpaceInvaders' Atari environment. A parallel linear layer providing an independent representation is integrated, where the Hadamard product of the two parallel representations represents the final representation z_t.

B Kernel Density Estimations

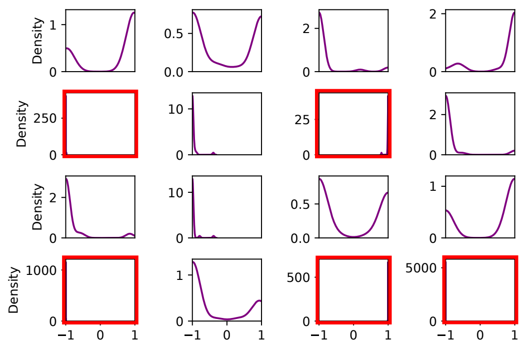
As discussed in Section 3, we hypothesize that the differences between a hyperbolic tangent with and without an HR are due to the increased ability of the product of hyperbolic tangents being able to negate dying neurons. We further see this phenomenom when plotting a random selection of neurons from both the mask and the base representation in Fig. 12a.



(a) Tanh (HR) - $5 \cdot 10^6$ iterations



(b) Final Tanh (HR) - $5 \cdot 10^6$ iterations



(c) Tanh (no HR) - $5 \cdot 10^6$ iterations

Figure 12: Kernel Density Estimations (KDE) over a subset of 16 neurons in the representations z_t^{enc} and z_t^* in (a), the resulting Hadamard product z_t in (b) and the representation z_t when training without an HR (c). These representations are obtained after training DQN in the 'Breakout' environment. Red outlines represent dead (collapsed) neurons. In (a), a closer look at neurons 3, 8 and 9 shows that when one of the representations saturates, the other is able to compensate, leading to a non-dead neuron in their product z_t in (b).

B.1 KDE calculation

Firstly, to stabilize the KDE computation and avoid singularity issues, a small noise ϵ, following a normal distribution, is added to each neuron's activations:

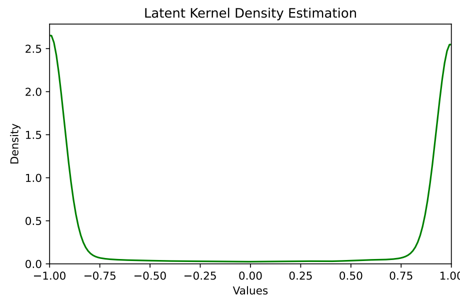
$$\alpha_i' = \alpha_i + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma^2)$$

where $\sigma^2 = 1 \times 10^{-5}$. The bandwidth for KDE, crucial for the accuracy of the density estimate, is calculated using Scott's rule, adjusted by the standard deviation of the jittered activations:

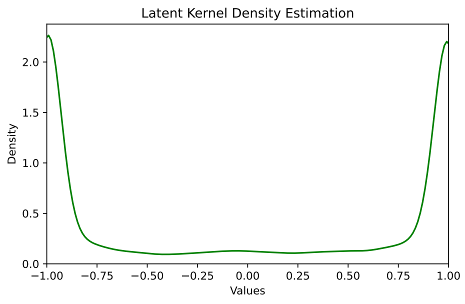
$$bw = n^{-\frac{1}{5}} \cdot \text{std}(\alpha_i')$$

where n is the number of samples in α_i. The density of activations is then estimated using a Gaussian kernel:

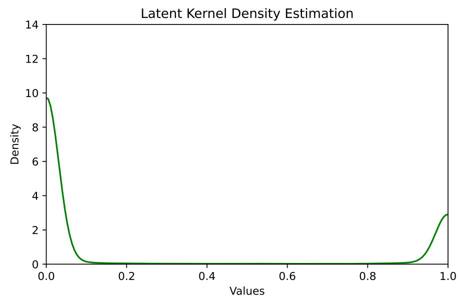
$$f(x) = \frac{1}{n \cdot bw} \sum_{j=1}^{n} K\left(\frac{x - \alpha_{ij}'}{bw}\right)$$



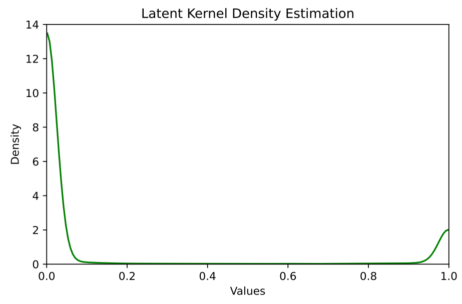
(a) Tanh - 10^7 iterations



(b) Tanh (HR) - 10^7 iterations



(c) Sigmoid - 10^7 iterations



(d) Sigmoid (HR) - 10^7 iterations

Figure 13: Kernel Density Estimations of the final representation z_t after training DQN for 10^7 iterations in the Breakout environment. A hyperbolic tangent Hadamard representation allows the representation to avoid strong saturation, keeping sufficient kernel density in the central sections of the hyperbolic tangent. As a sigmoid can saturate into zero, using a Hadamard representation remains less effective for preventing saturation, as any zero will lead to a Hadamard product of zero.

Here, K denotes the Gaussian kernel function. In order to finally determine if a neuron is dead, the maximum value of the estimated density function $f(x)$ is compared against a predefined threshold:

$$\max(f(x)) \geq \omega$$

where ω represents the predetermined threshold. In practice, after analyzing the individual neuron KDE's, using an ω of 20 provides a strong approximation of actual dead neurons.

B.2 Effective Rank calculation

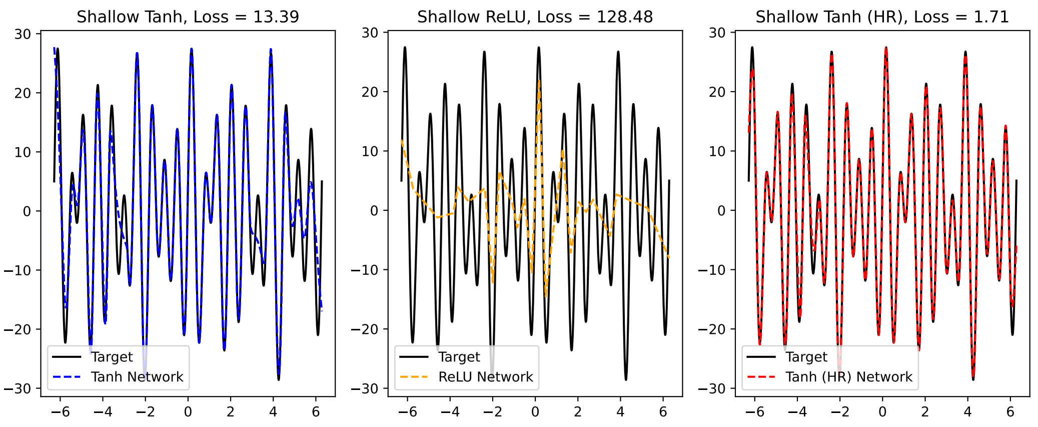
In line with Kumar et al. (2021), the effective rank of a feature matrix for a threshold δ ($\delta = 0.01$), denoted as $srank_\delta(\Phi)$, is given by $srank_\delta(\Phi) = \min \left\{ k : \frac{\sum_{i=1}^{k} \sigma_i(\Phi)}{\sum_{i=1}^{d} \sigma_i(\Phi)} \geq 1 - \delta \right\}$, where $\{\sigma_i(\Phi)\}$ are the singular values of Φ in decreasing order, i.e., $\sigma_1 \geq \cdots \geq \sigma_d \geq 0$. Intuitively, the effective rank of a feature matrix represents the number of "effective" unique components that form the basis for linearly approximating the resulting Q-values. The calculation in Python is done as follows:

```
def compute_rank_from_features(feature_matrix, rank_delta=0.01):
    sing_values = np.linalg.svd(feature_matrix, compute_uv=False)
    cumsum = np.cumsum(sing_values)
    nuclear_norm = np.sum(sing_values)
    approximate_rank_threshold = 1.0 - rank_delta
    threshold_crossed = (cumsum >= approximate_rank_threshold * nuclear_norm)
    effective_rank = sing_values.shape[0] - np.sum(threshold_crossed) + 1
    return effective_rank
```

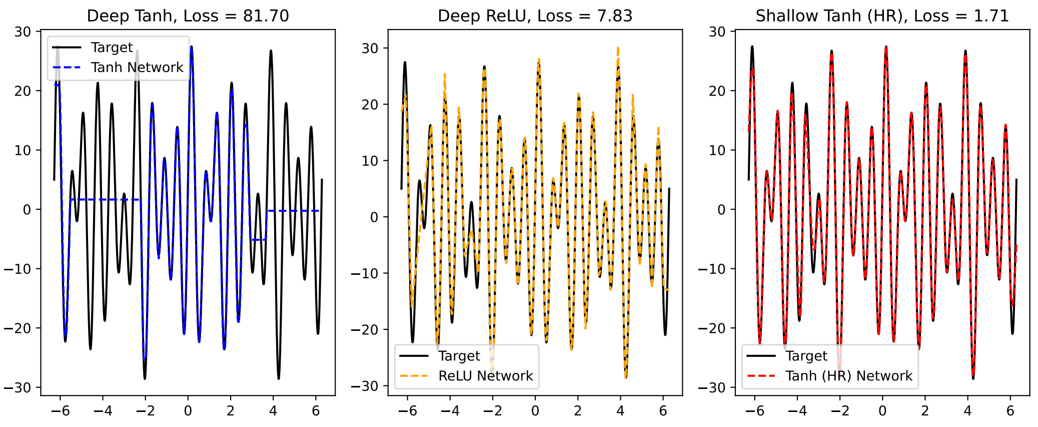
C ADDITIONAL EXPERIMENTS

C.1 SHALLOW AND DEEP FUNCTION APPROXIMATION

To further showcase the effect of activations on complex function approximation, we compare the single hidden layer Tanh (HR) network from Fig. 2 with a deep ReLU and Tanh network containing three hidden layers each. The comparison with shallow networks can be found in Fig. 14a and a comparison with deep networks can be found in Fig. 14b.

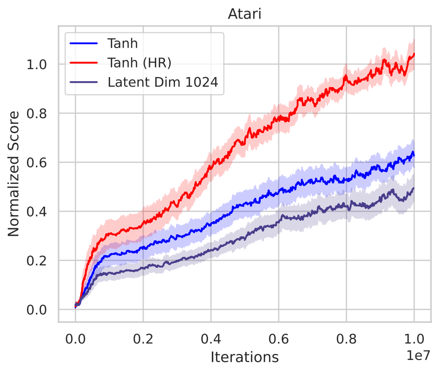


(a) Comparison of shallow networks for a nonlinear regression task. The Tanh and ReLU networks have a single hidden layer of 200 neurons, while the Tanh (HR) has a single hidden layer of 100 neurons but two preceding linear layers. The Tanh and ReLU networks have 601 parameters, while the Tanh (HR) network has 501 parameters. As found by Gulcehre et al. (2022), a shallow network activated by ReLU has a lower effective rank and consequently reduced network expressivity as compared to a Tanh activated network. Using a Hadamard representation, we achieve better function approximation while using less parameters.

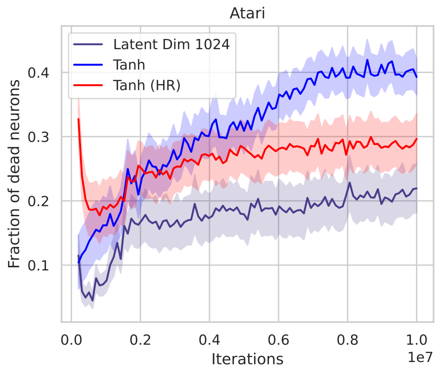


(b) Comparison of two deep networks and one shallow network for the same nonlinear regression task. The Tanh and ReLU networks have 3 hidden layers of 200 neurons each, while the Tanh (HR) network remains shallow. In line with common observations in deep learning, the ReLU activation thrives in deeper networks, in contrast to the Tanh activation. Interestingly, the shallow Tanh (HR) network still achieves better function approximation with only 0.6% of the deeper networks' parameters (81001 vs 501). No hyperparameter tuning or architecture search has been applied. Additional tests using deep Tanh (HR) networks gave similar function approximation as compared to the shallow Tanh (HR) network.

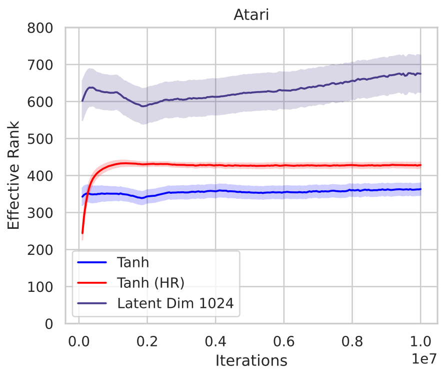
C.2 Increasing Representation Parameters



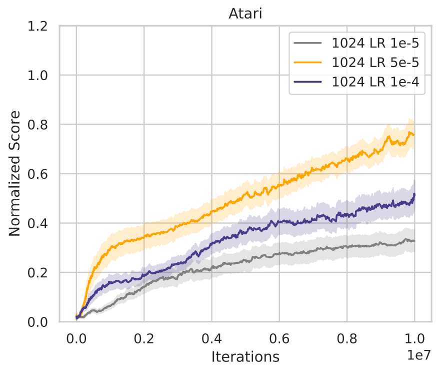
(a) Performance



(b) Dead Neurons



(c) Effective Rank

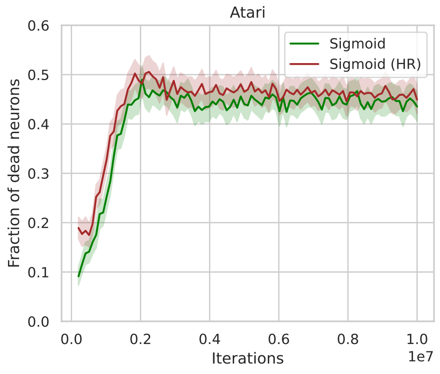


(d) Learning Rates $z_t \in \mathbb{R}^{1024}$

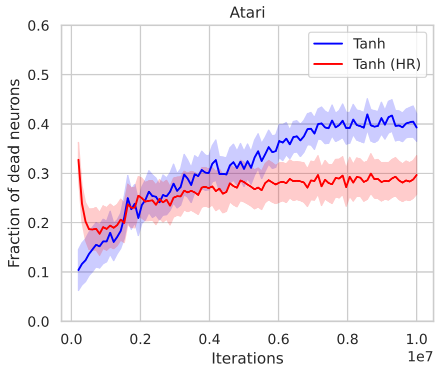
Figure 15: Comparison of a normal hyperbolic tangent (Tanh), a hyperbolic tangent with a higher representation dimension $z_t \in \mathbb{R}^{512 \to 1024}$ and a Hadamard representation using hyperbolic tangents. Comparisons are done on performance (a), the fraction of dead neurons (b), the effective rank of the representation z_t (c) and learning rates of the higher-dimensional latent. Naturally, increasing the representation dimension z_t increases the effective rank of the representation, but using a larger representation dimension is not always preferable as it often requires different hyperparameters, and can lead to reduced performance (Obando-Ceron et al., 2024; Obando Ceron et al., 2024).In (d), an ablation of learning rates shows that using a larger layer can sometimes prefer lower learning rates. However, it also shows that the improvement due to the Hadamard representation is likely not correlated with the parameter increase, as the Hadamard representation still significantly outperforms any of the 1024-dimensional latent state learning rate ablations.

C.3 Validating dying neuron probability derivations

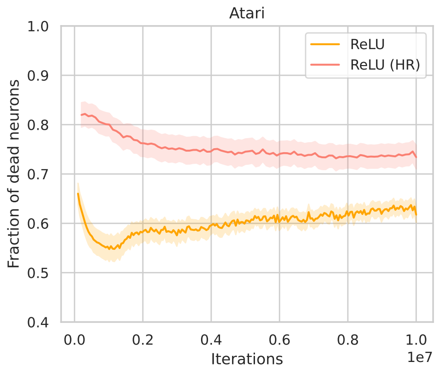
As discussed in Section 3, the effect of using a Hadamard representation strongly depends on the activation function. These derivations are empirically validated by the results in Fig. 16. In practice, since a neural network prefers symmetry, a sigmoid saturates slightly faster to 0 than to 1. This could explain the very slight increase in dead neurons when using an HR with activations. Note that, since we use neuron independence assumptions in our theoretically calculated dying neuron probabilities, the empirical results differ in magnitude from the theoretical predictions.



(a) Sigmoid with and without HR



(b) Tanh with and without HR

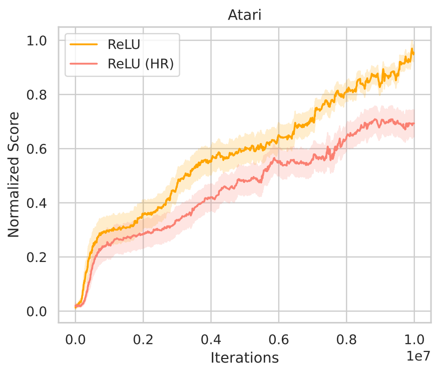


(c) ReLU with and without HR

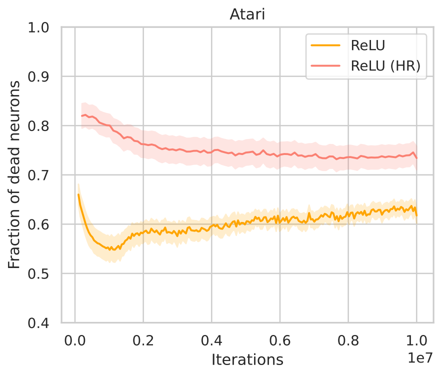
Figure 16: By evaluating the effect of an HR on dying neurons through the lens of probability theory, we predicted that only the hyperbolic tangent benefits in this metric. Specifically, only a hyperbolic tangent was speculated to have a decrease in dying neurons. Using an HR with sigmoid activations would have no notable difference, and for an HR with ReLU activations an increase in dead neurons was expected. This empirically validates our hypotheses in Section 3.

C.4 ReLU activated Hadamard representation

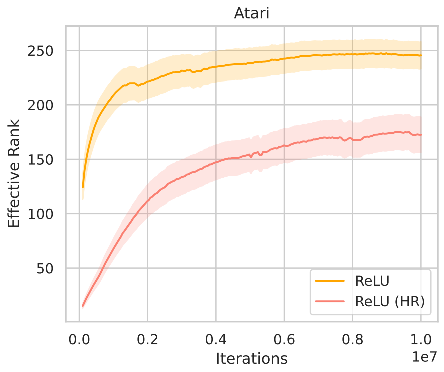
Additional Atari experiments are provided comparing a ReLU activation with and without an HR. The normalized scores, dying neurons and the effective rank during training can be seen in fig. 17.



(a) Normalized Score



(b) Dead Neurons



(c) Effective Rank

Figure 17: As a Rectified Linear Unit creates sparse representations, it does not benefit from using an HR, since the final representation will consist of the Hadamard product between two sparse representations. Therefore, a decrease in both performance and effective rank and an increase in dead neurons can be expected.

C.5 Hyperbolic Tangent Convolutional Activation

We additionally run tests in DQN on 8 Atari games. The baseline ReLU is used, except for the convolutional activations which are changed to a hyperbolic tangent.

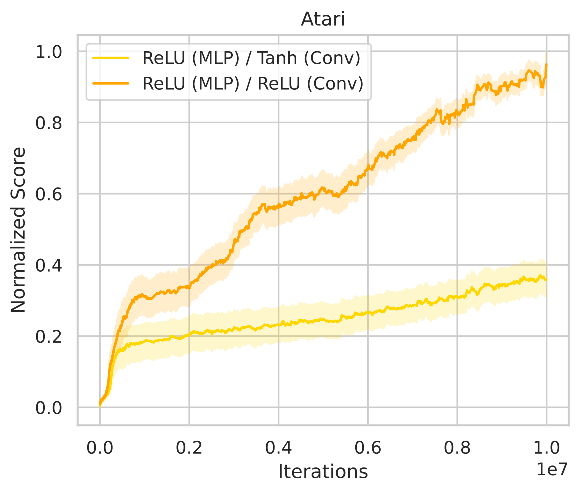


Figure 18: Baseline normalized performance in the Atari domain for 10M iterations (40M Frames) on DQN.

It is evident that using non-Hadamard hyperbolic tangents in the convolutional layers can lead to only a fraction of the performance of the ReLU-activated baseline.

D Atari

D.1 DQN Evaluation Details

We normalize performance with respect to the ReLU baseline in `cleanrl` Huang et al. (2022). The minimum and maximum score of the ReLU baseline are taken for each environment, and the normalized score for each environment is calculated as follows:

$$\text{Normalized Score} = \frac{\text{Score} - \text{Min Score}}{\text{Max Score} - \text{Min Score}} \tag{8}$$

where *Score* refers to the raw performance score of the model being evaluated, *Min Score* is a single value representing the lowest score obtained by the ReLU baseline (usually equivalent to random policy or even slightly worse), and *Max Score* is a single value representing the highest score achieved by the ReLU baseline in the same environment. To average, we sum the normalized scores for every run and take the mean.

The more official Human-Normalized Score, as referenced in Mnih et al. (2015), is calculated similarly but using human and random performance benchmarks:

$$\text{Human-Normalized Score} = \frac{\text{Score} - \text{Random Score}}{\text{Human Score} - \text{Random Score}} \tag{9}$$

where *Human Score* and *Random Score* refer to the scores recorded by human players and random agents, respectively. Calculating our performance according to the Human-Normalized Score leads to the plot seen in Fig. 19. Due to taking a subset of the Atari domain in DQN and PPO, the VideoPinball environment is extremely dominant in the Human-Normalized Score calculation. For a more realistic comparison of the methods, we therefore decided to use baseline-normalized scores in the main paper.

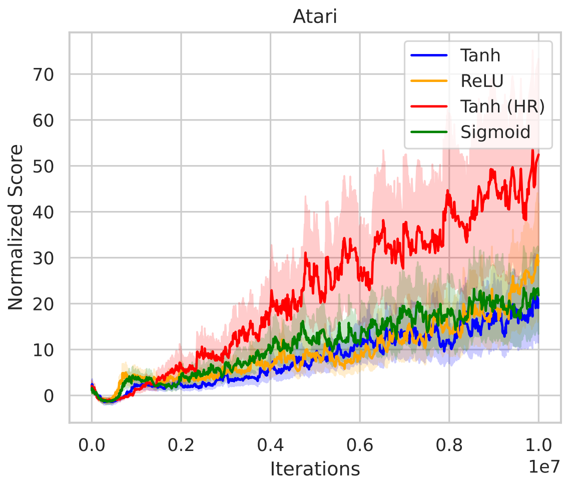


Figure 19: Human-Normalized performance (in multiples) with the standard deviation over the means in the Atari domain for 10M iterations (40M Frames).

D.2 Individual Environment Scores

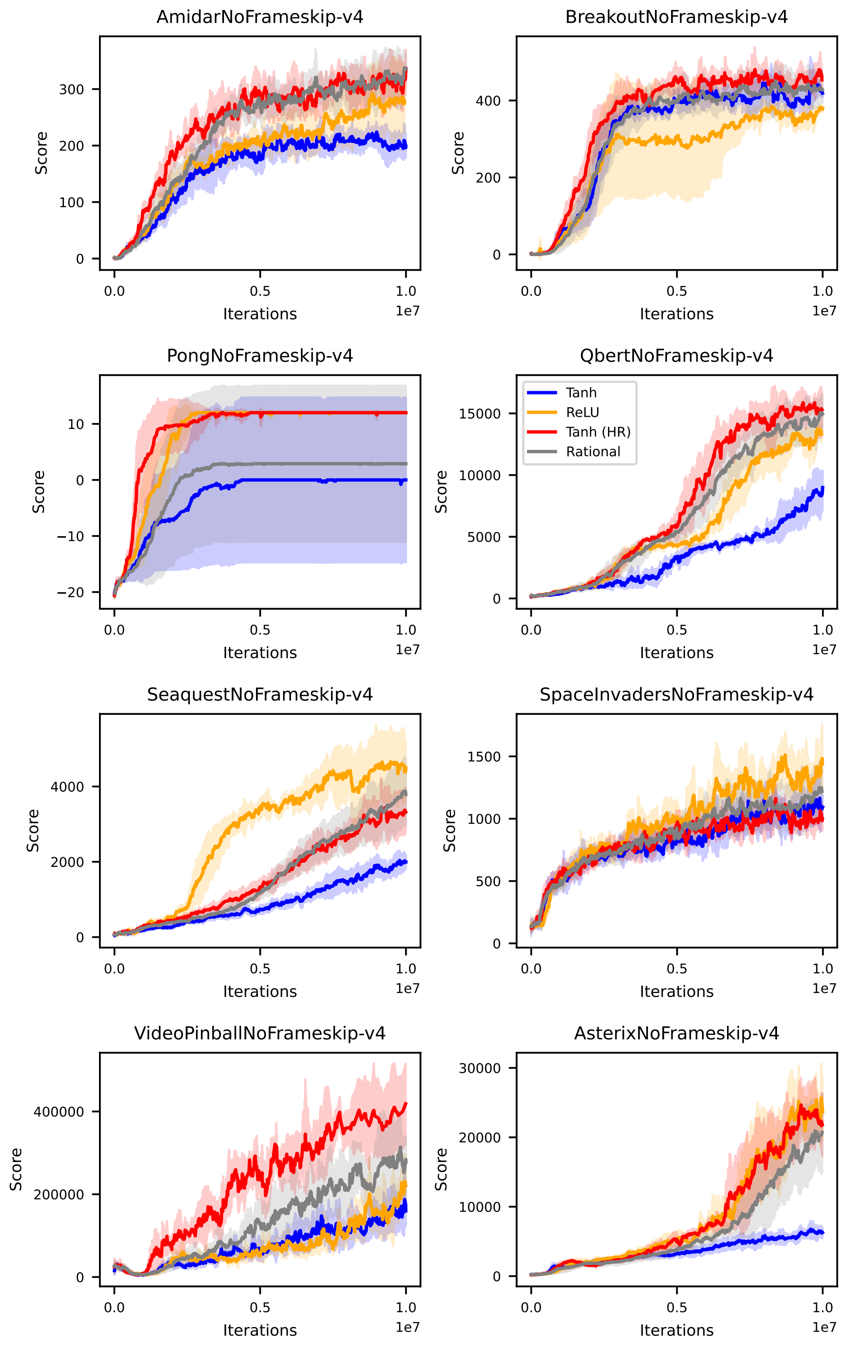


Figure 20: DQN Performance comparison on the individual Atari Environments. Plotted lines represent the mean taken over 5 seeds, with the standard deviations expressed as the shaded region. Eight popular Atari games in the RL community that are non-exploration driven were chosen due to computational limitations. No other games were tested.

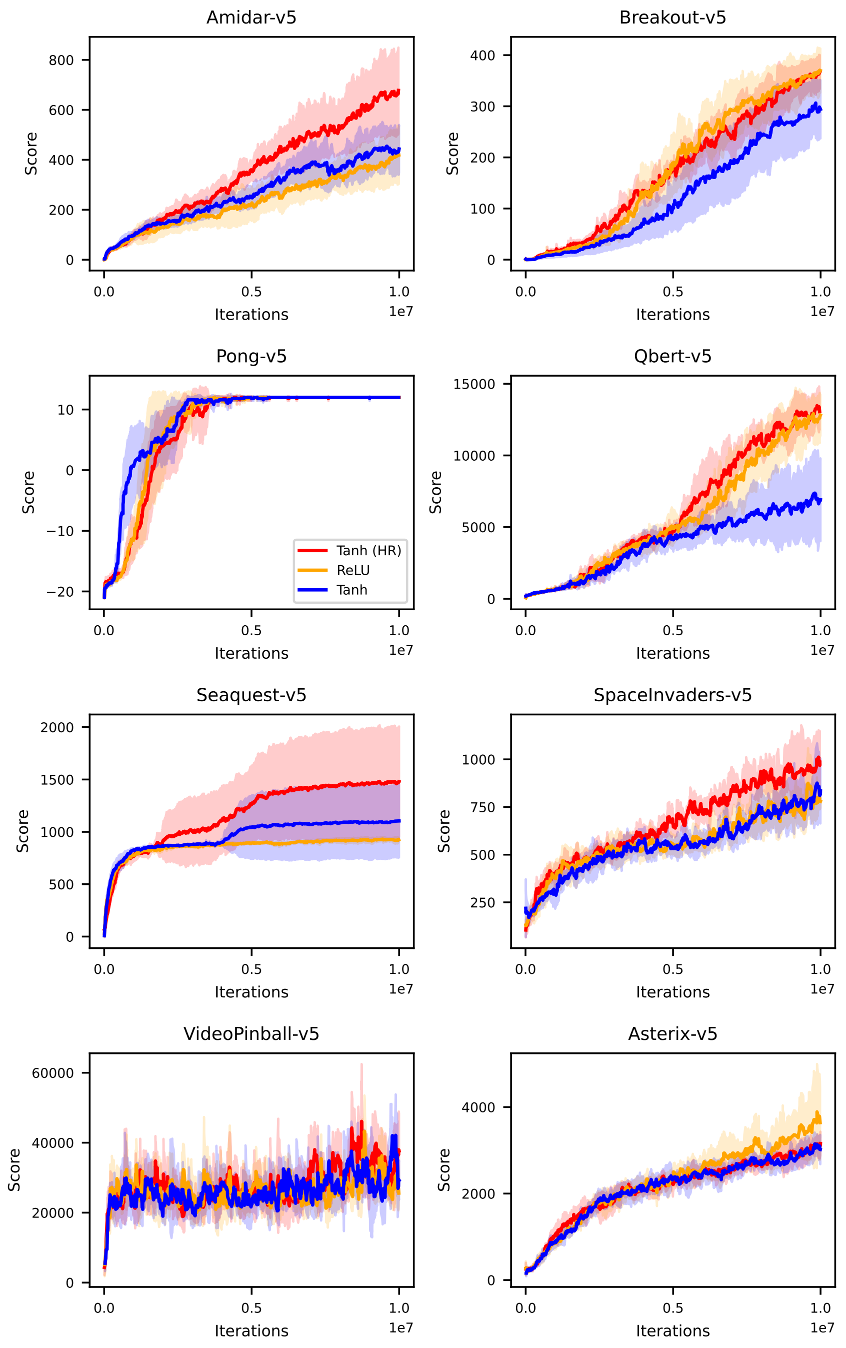


Figure 21: PPO Performance comparison on the individual Atari Environments. Plotted lines represent the mean taken over 5 seeds, with the standard deviations expressed as the shaded region. Eight popular Atari games in the RL community that are non-exploration driven were chosen due to computational limitations. No other games were tested.

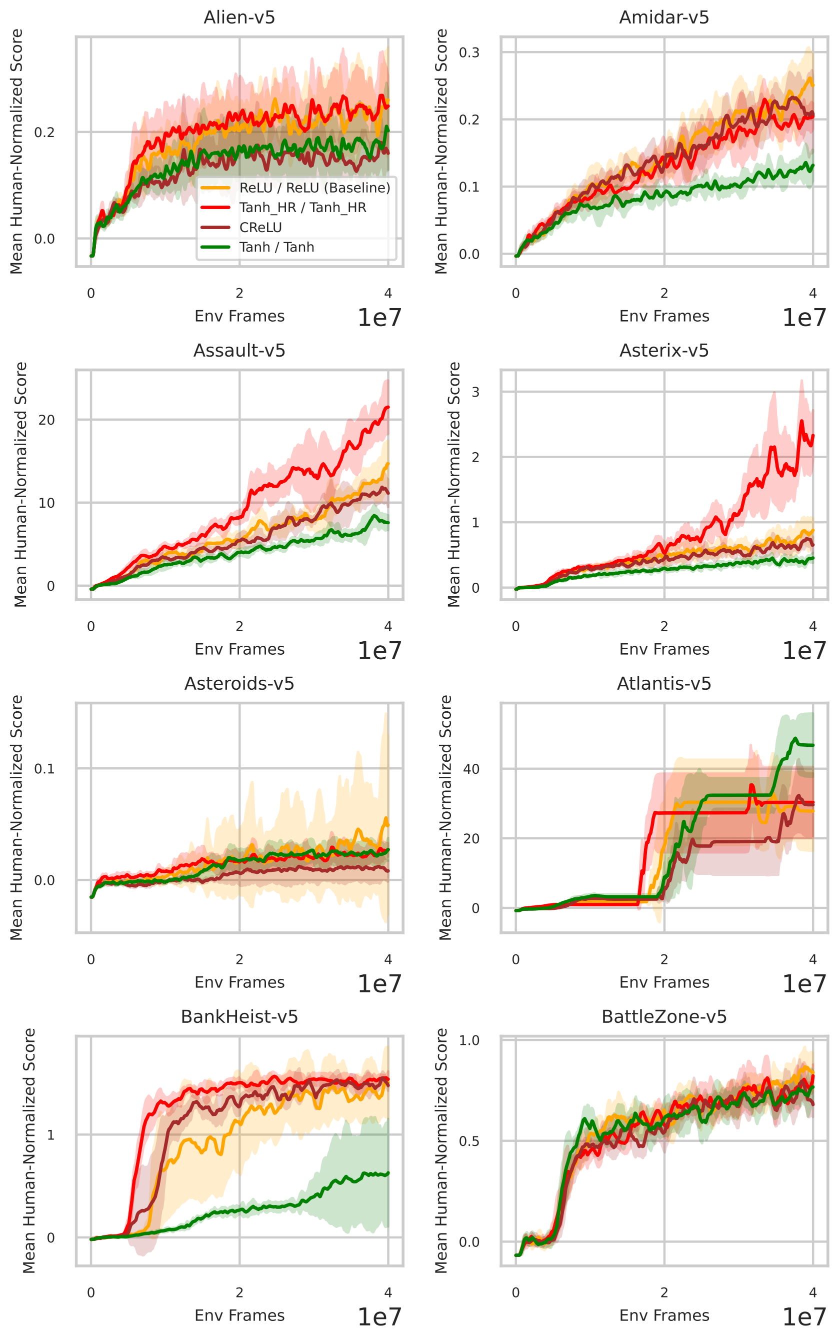


Figure 22: PQN Performance comparison on the individual Atari Environments. Labels represent MLP / CNN activations. Plotted lines represent the mean taken over 5 seeds, with the standard deviations expressed as the shaded region.

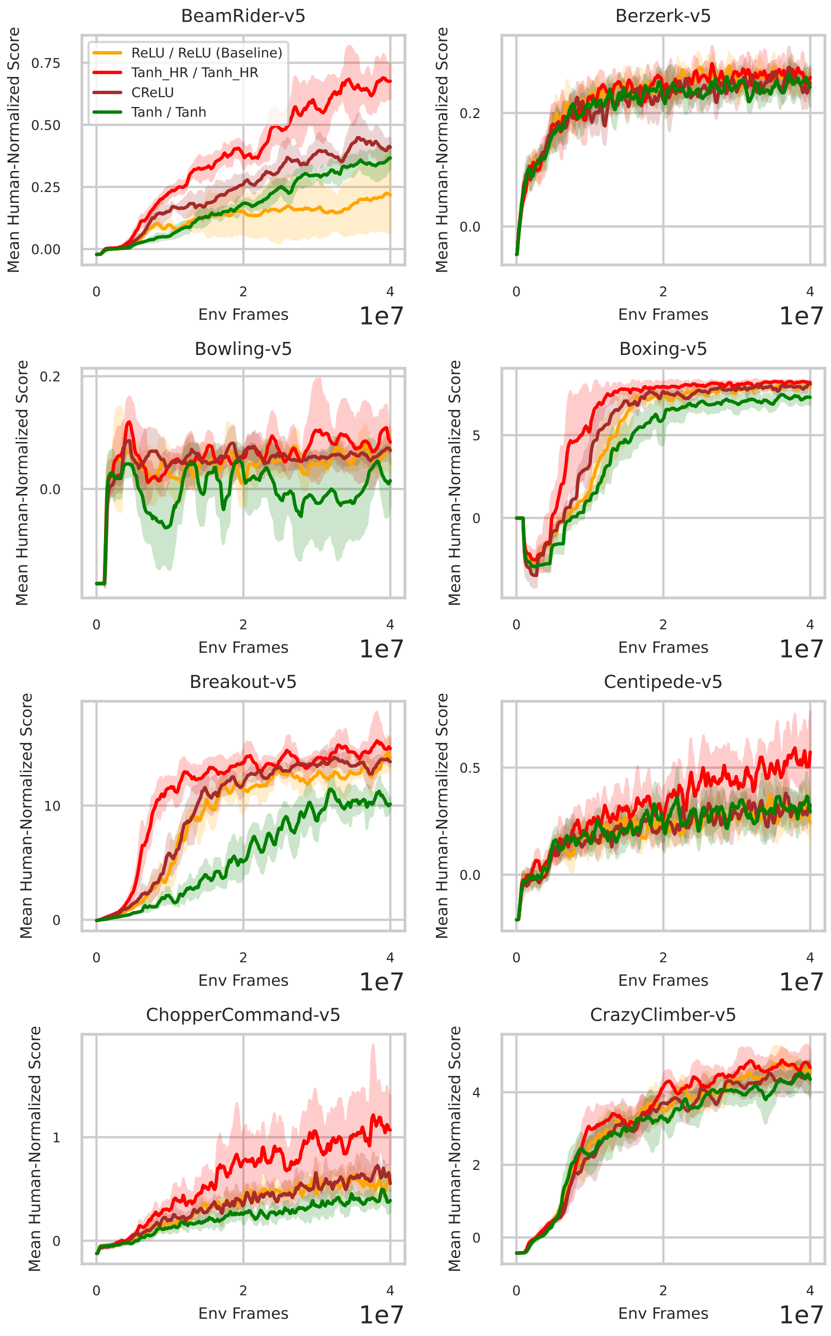


Figure 23: PQN Performance comparison on the individual Atari Environments. Labels represent MLP / CNN activations. Plotted lines represent the mean taken over 5 seeds, with the standard deviations expressed as the shaded region.

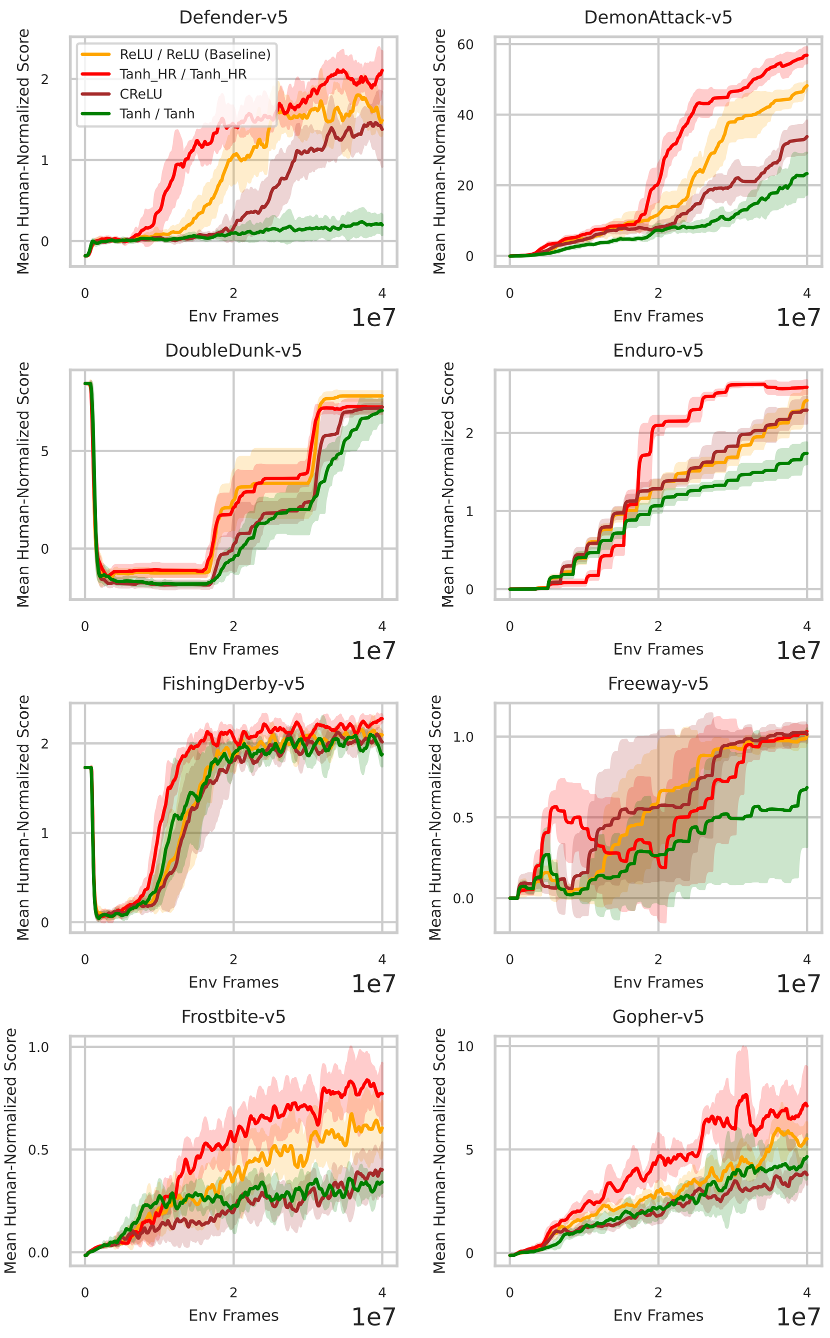


Figure 24: PQN Performance comparison on the individual Atari Environments. Labels represent MLP / CNN activations. Plotted lines represent the mean taken over 5 seeds, with the standard deviations expressed as the shaded region.

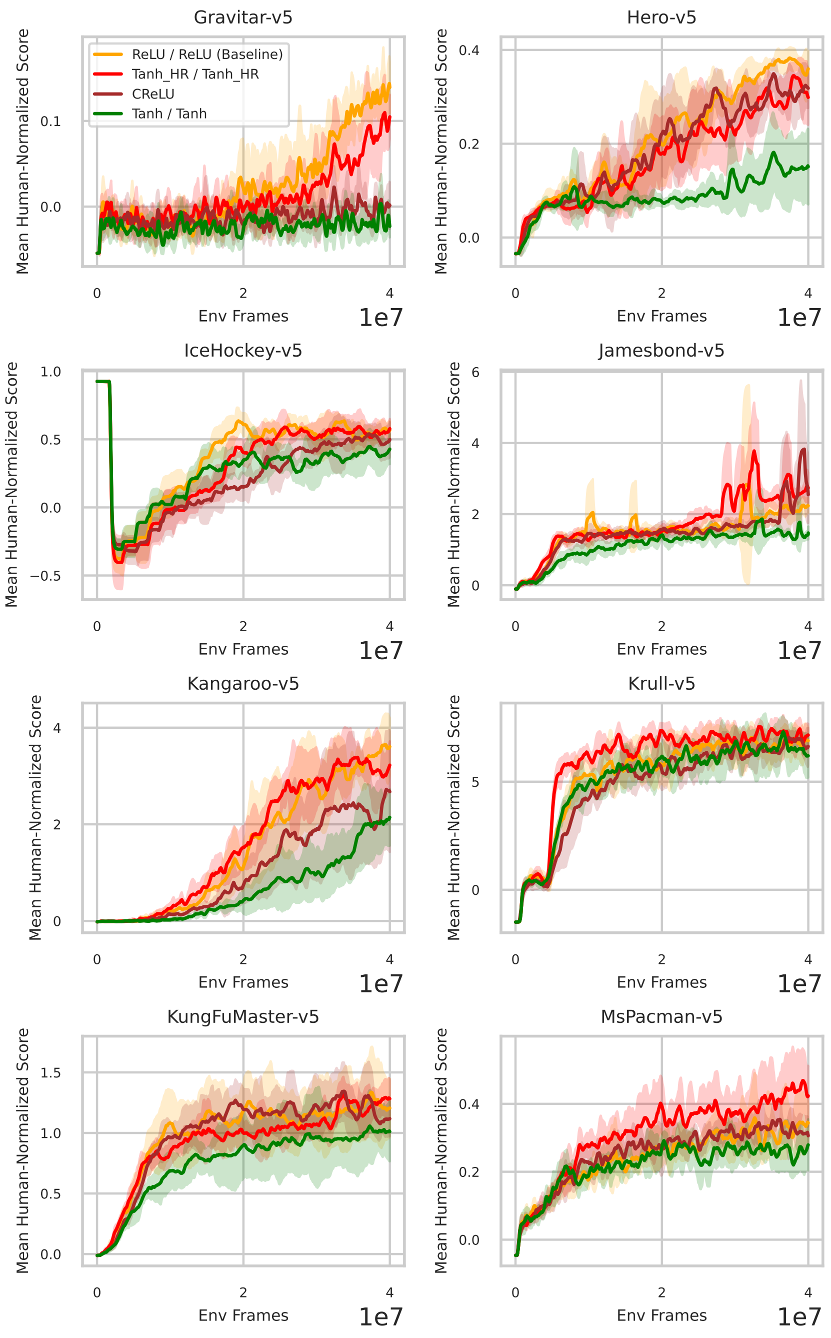


Figure 25: PQN Performance comparison on the individual Atari Environments. Labels represent MLP / CNN activations. Plotted lines represent the mean taken over 5 seeds, with the standard deviations expressed as the shaded region.

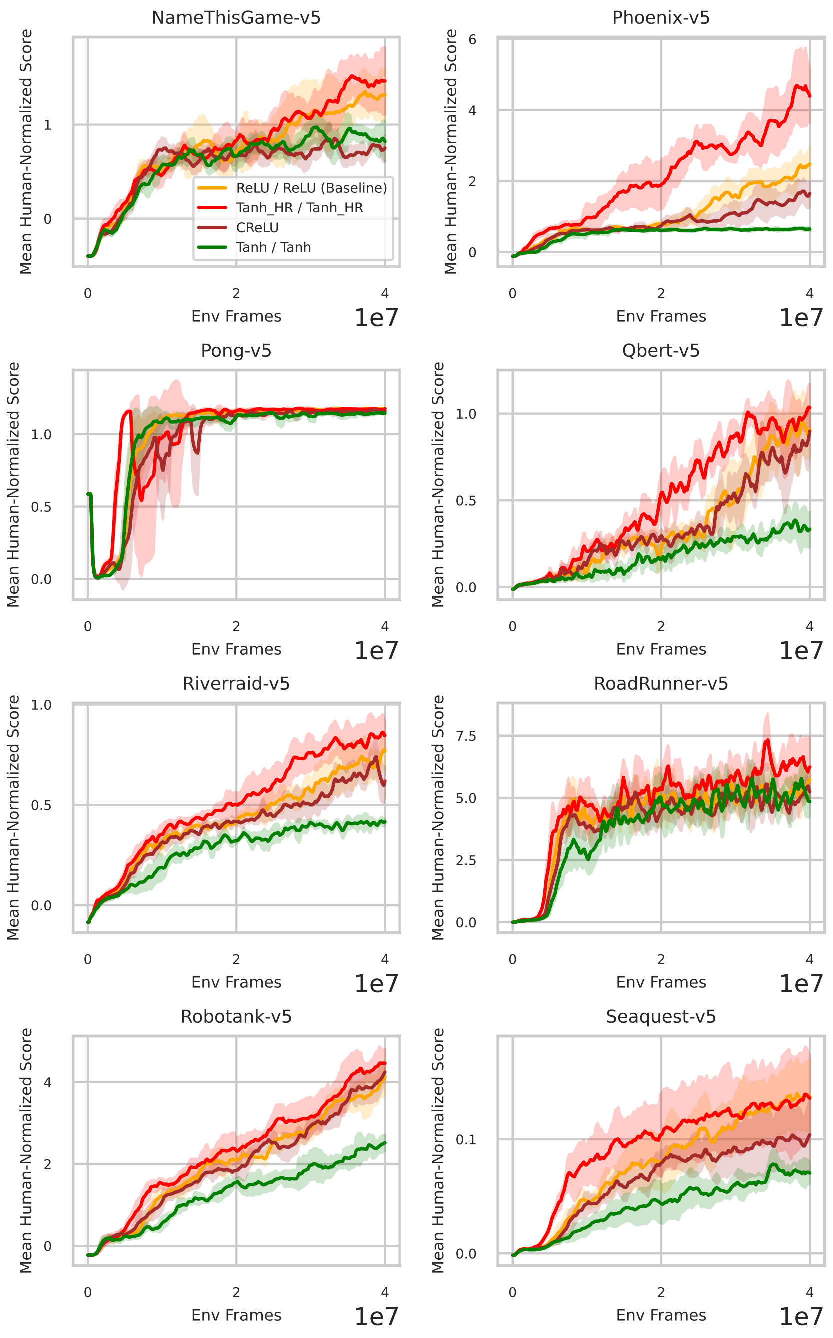


Figure 26: PQN Performance comparison on the individual Atari Environments. Labels represent MLP / CNN activations. Plotted lines represent the mean taken over 5 seeds, with the standard deviations expressed as the shaded region.

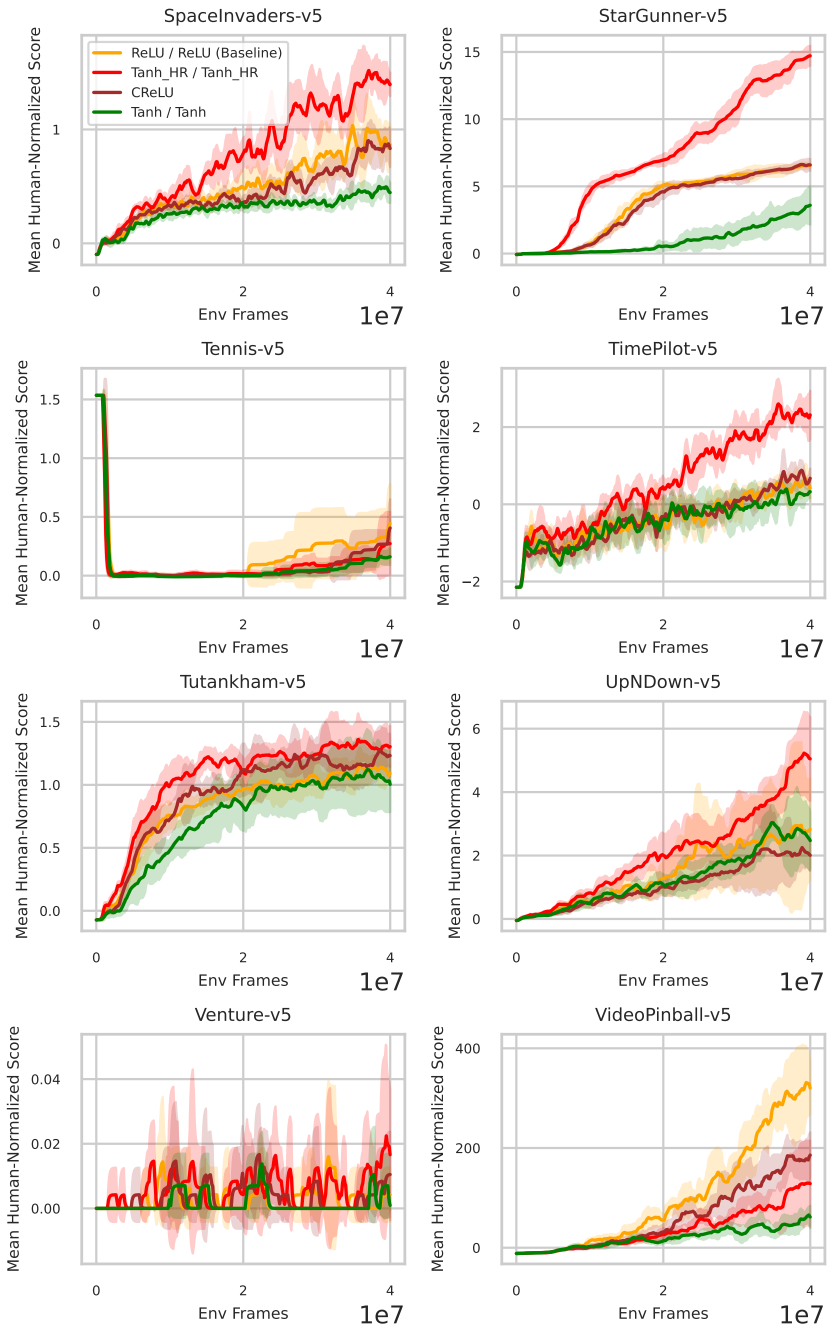


Figure 27: PQN Performance comparison on the individual Atari Environments. Labels represent MLP / CNN activations. Plotted lines represent the mean taken over 5 seeds, with the standard deviations expressed as the shaded region.

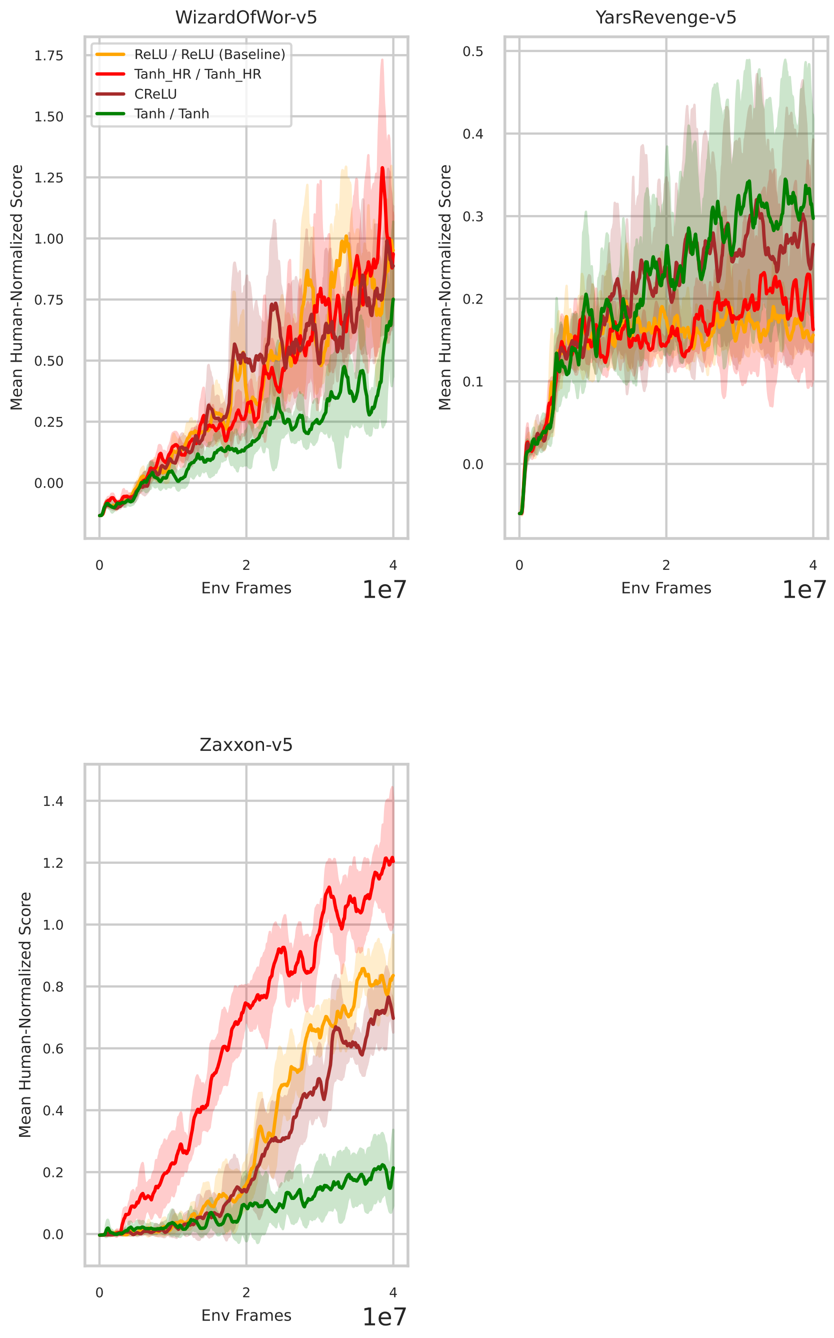


Figure 28: PQN Performance comparison on the individual Atari Environments. Labels represent MLP / CNN activations. Plotted lines represent the mean taken over 5 seeds, with the standard deviations expressed as the shaded region.