ADAPTING TO REWARD PROGRESSIVITY VIA SPECTRAL REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

In this paper we consider reinforcement learning tasks with *progressive rewards*; that is, tasks where the rewards tend to increase in magnitude over time. We hypothesise that this property may be problematic for value-based deep reinforcement learning agents, particularly if the agent must first succeed in relatively unrewarding regions of the task in order to reach more rewarding regions. To address this issue, we propose *Spectral DQN*, which decomposes the reward into frequencies such that the high frequencies only activate when large rewards are found. This allows us to balance the training loss, so that it gives more even weighting to regions of the task with very different reward scales. In two domains with extreme reward progressivity, where standard value-based methods struggle significantly, Spectral DQN is able to make much farther progress. Moreover, when evaluated on a set of six standard *Atari* games that do not overtly favour the approach, Spectral DQN's performance remains competitive, demonstrating that it is not overfit to its target problem. Interestingly, the latter experiments also suggest that Spectral DQN may have broader applicability to sparse reward tasks.

1 INTRODUCTION

In decision making tasks involving compounding returns, such as stock market investing, it is common for the rewards received by the agent to increase in magnitude over time. An investor that starts out with little capital will achieve relatively small profits or losses early on, but if they are successful, their capital – and hence their potential profits and losses – will gradually increase. This property also arises in many games settings. For example, in the *Atari* game *Video Pinball*, the player can increase a "bonus multiplier" that increases the score paid per bumper hit. Since the bonus multiplier does not reset until the player dies, the rewards typically increase with time. In this paper, we refer to any task exhibiting this phenomenon as a *progressive reward task*.

We hypothesise that reward progressivity may be problematic for value-based deep reinforcement learning agents. Our rationale is that the temporal difference errors that arise under such algorithms typically scale with the magnitude of the training targets. Accordingly, an agent ought to learn more accurate value predictions (in percentage error terms) in high value regions of the task. While this is rational in one sense – generally, it is more important to make accurate decisions in high-stakes situations – it is potentially harmful on progressive reward tasks. This is because, on progressive reward tasks, *it may be necessary for the agent to perform well in relatively unrewarding regions before it can reach more rewarding regions*. For example, in stock market investing, an investor can only reach states where they have large capital if they first perform well with small capital. Note too that the rewards need not be strictly progressive for this problem to arise; all that is needed is for the rewards to be progressive over some extended period.

Algorithms that clip the reward, such as *DQN* (Mnih et al., 2015), are less susceptible to this problem, because they do not perceive increases in reward magnitude beyond the clipping point. However, it is straightforward to construct examples where reward clipping masks the optimal solution. For example, in *Bowling*, clipping the rewards to [-1, 1] makes bowling a strike appear no better than hitting a single pin (Pohlen et al., 2018). While subsequent approaches make learning feasible without reward clipping (van Hasselt et al., 2016; Pohlen et al., 2018), we show in Section 3 that these methods are negatively impacted by strong reward progressivity. Motivated by this, we seek an

algorithm that is capable of learning from unclipped rewards, while mitigating the negative impact that large rewards have on the agent's ability to learn less rewarding regions of the task.

To meet this aim, we propose *Spectral DQN*. Under this approach, rewards are decomposed into a multidimensional *spectral reward*, where each component is bound to [-1, 1]. The upper frequencies of the spectrum activate only when large magnitude rewards are received. For example, in *Video Pinball*, the lower frequencies activate on any score, while the upper frequencies only activate when the player has accumulated a large bonus multiplier. The bounded nature of the spectral rewards allows the agent to learn expected *spectral returns* without instability arising from large training targets. Meanwhile, the full, unclipped expected return can be recovered by summing across the return spectrum. (The full, technical definitions of these terms are provided in Section 4). In terms of addressing reward progressivity, a key advantage of this approach is that it allows us to balance the training loss and prevent the upper frequencies from receiving undue weight.

To test whether this approach helps mitigate the impact of reward progressivity, we perform two set of experiments. First, we apply Spectral DQN to two domains that we specifically designed to exhibit strong reward progressivity. While previous methods struggle on these tasks, failing to learn almost anything at all in the more extreme domain, Spectral DQN performs markedly better, making considerable progress on both tasks. Next, we apply our approach to a less constructed set of tasks; namely, 6 standard *Atari* games. Only some of these games exhibit reward progressivity, and none of them nearly so strongly as the extreme domains from earlier. While Spectral DQN's performance is less impressive here (it still outperforms both benchmarks overall, though not in all games), the experiments validate that the approach is not overfit to extreme cases. Moreover, these experiments highlight an unintended side benefit of the approach: It allows large, sparse rewards to be propagated aggressively, without any appreciable trade-off.

2 Previous Approaches to Handling Reward Variability

In the *Atari* domain, where the DQN algorithm was first evaluated, the agent receives a reward equal to the score increase at each frame. However, since score magnitudes vary greatly across games, DQN clips all rewards to $[-1, +1]$. This constrains the size of the training targets so that it is easier to find a step size that performs well across the entire suite of games. While this heuristic turns out to perform well in *Atari*, a clear downside of the approach is that the agent becomes unable to distinguish large rewards from small rewards.

The *Pop-Art* algorithm (van Hasselt et al., 2016) offers a more principled solution, by training from the unclipped reward, while normalising the training targets to have zero mean and unit variance. To ensure that the network's predictions are preserved when the normalisation parameters are updated, the output of the final layer is scaled and shifted by an offsetting amount.

Pohlen et al. (2018) propose an alternative approach to handling unclipped rewards that reduces the variance of the training targets by applying a squashing function, h. The agent learns squashed action-values, $\tilde{Q}(s, a) = h(Q(s, a))$, via the transformed Bellman backup:

$$\tilde{Q}(s_t, a_t) \leftarrow \tilde{Q}(s_t, a_t) + \alpha\big[h\big(r + \gamma \max_{a'} h^{-1}(\tilde{Q}(s_{t+1}, a'))\big) - \tilde{Q}(s_t, a_t)\big] \tag{1}$$

The authors prove that the transformed Bellman operator remains a contraction provided that both h and h^{-1} are Lipschitz continuous and $\gamma < 1/L_h L_{h^{-1}}$ (where L_h and $L_{h^{-1}}$ are the respective Lipschitz constants). In their experiments, they use the following squashing function:

$$h(x) = sign(x)(\sqrt{|x| + 1} - 1) + \epsilon x \tag{2}$$

where $\epsilon > 0$ ensures that h^{-1} is Lipschitz continuous with $L_{h^{-1}} = 1/\epsilon$.

For the remainder of this paper, we refer to this approach as *target compression*. In the *Atari* domain, Pohlen et al. found target compression to perform significantly better than Pop-Art. It has since become the dominant method for handling unclipped rewards in *Atari*, with several recent large-scale agents (Kapturowski et al., 2019; Badia et al., 2020a;b), including the state-of-the-art *Agent57*, favouring target compression over Pop-Art.

3 DEMONSTRATING THE PROBLEM OF REWARD PROGRESSIVITY

While both Pop-Art and target compression address reward variance *across* tasks, making it easier to devise a single learning configuration that performs well across a task suite, they were not designed to address *intra-task* reward variability. As such, we posit that they are ill-suited to addressing reward progressivity, since it is an intra-task phenomenon. To test this claim, we introduce two domains that are designed to exhibit extreme reward progressivity, and thus exacerbate the issue: *Ponglantis* and *Exponential Pong*.

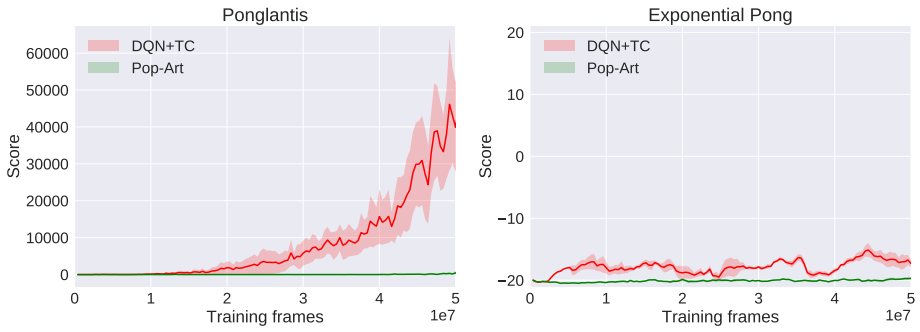
Ponglantis. In *Ponglantis*, the game starts out exactly the same as the *Atari* game *Pong*, where valued-based deep reinforcement learning methods are known to perform well (Mnih et al., 2015). The reward scale in *Pong* is small relative to most *Atari* games; the player receives a +1 reward for hitting the ball past the computer paddle, and a -1 reward if they concede a point. In *Ponglantis*, however, if the player manages to score 10 points, the task immediately switches to *Atlantis*, where the reward scale is roughly 1000 times larger. The *Atlantis* phase ends upon the lose of a single life.

Exponential Pong. The game *Exponential Pong* is identical to regular *Pong*, except that the rewards are multiplied by 2^n, where n is the player's current score.

We applied two agents to these games: Pop-Art and DQN+TC, where the latter is a version of DQN that we modified to use target compression and learn from the unclipped reward. The results of this experiment, averaged over 5 seeds, are shown in Figure 1. (For full experimental details, see the Appendix.) To simplify analysis of the *Exponential Pong* results, we have plotted the unexponentiated scores, i.e. the scores that the agents would have achieved under standard scoring in *Pong*. For context, the best possible score on standard *Pong* is +21, and the worst possible score is -21.

In *Exponential Pong*, which exhibits the more extreme reward progressivity (since the reward multiplier can potentially reach $2^{20} \approx 1$ million), both agents essentially fail, providing strong support for our hypothesis that reward progressivity can be damaging, at least in extreme cases.

In the more moderate case of *Ponglantis*, Pop-Art fails to make meaningful progress beyond the low-scoring *Pong* phase of the task. The likely reason for this is that as soon as the agent starts reaching *Atlantis* and receiving large rewards, the target scaling mechanism employed by Pop-Art starts dividing the training targets by a large factor, heavily dampening the much smaller learning signals arising from the *Pong* phase of the task. On the other hand, DQN+TC is eventually able to make some progress into the second, *Atlantis* phase. The fact that DQN+TC outperforms Pop-Art is consistent with our hypothesis; while Pop-Art sees the reward scale jump by a factor of $\approx$ 1000 upon reaching *Atlantis*, the training targets of DQN+TC jump by a much smaller factor, owing to the square root squashing function (Equation 2). A natural question to ponder then is whether DQN+TC might achieve much better performance in *Exponential Pong* by employing a logarithmic squashing function instead of a square root. Unfortunately though, as Pohlen et al. (2018) point out to a reviewer of their work (see `https://openreview.net/forum?id=BkfPnoActQ`), a logarithmic transform does not have a Lipschitz continuous inverse, and thus the transformed Bellman in this case is no longer guaranteed to be a contraction.



(a) *Ponglantis* (b) *Exponential Pong*

Figure 1: Mean episodic return for agents trained on *Ponglantis* and *Exponential Pong*.

4 SPECTRAL REINFORCEMENT LEARNING

Based on the finding of the previous section, it seems that what we require is an algorithm where: (1) The training targets are constrained in magnitude, no matter how large the reward scale grows. (2) Regions of the task with small rewards yield meaningfully large temporal difference errors, even once much larger rewards have been discovered. Note that Pop-Art fails to meet the second criterion, while target compression fails to meet both, albeit though it takes larger shifts in reward magnitude for the problems associated with reward progressivity to become apparent.

In this section, we introduce a reinforcement learning approach that meets both of the above criteria by leveraging the idea of spectral decomposition. This algorithm does not in and of itself address reward progressivity; in fact, we show that it behaves identically to the standard *Q-learning* algorithm (Watkins, 1989), given certain caveats. However, the properties of this algorithm *facilitate* adaption to progressive rewards, and in Section 5 we present a deep learning version of the approach that is capable of learning in domains with extreme reward progressivity, such as *Expoential Pong*.

4.1 SPECTRAL DECOMPOSITION

Spectral decomposition is a broad term that arises in many different fields, but in this paper it refers to a specific method for decomposing real numbers into a sum of exponentially weighted components.

The intuition behind our method is illustrated in Figure 2. In this example, the number being spectrally decomposed is 6.5. The method begins by dividing the number line into buckets of width b^i, where $b > 0$ is the base of the decomposition and $i \in \mathbb{Z}_{\geq 0}$. Since the base here is 2, bucket 0 has width 1 and spans from 0 to 1, bucket 1 has width 2 and spans from 1 to 3, bucket 2 has width 4 and spans from 3 to 7, etc. The idea is that each bucket corresponds to a different frequency in the spectrum. The spectral decomposition is found by calculating the proportion of each bucket that is filled. Since the number 6.5 completely fills the first two buckets, then 87.5% of the next bucket, then none of the remaining buckets, the spectral decomposition of 6.5 to base 2 is (1, 1, 0.875, 0, 0, ...).

More generally, let r be a non-negative real number, and let r_b^{i+} denote the ith element of r's spectral decomposition to base b. The formula for calculating each element is:

$$r_b^{i+} = clamp\big((r - \tfrac{b^i - 1}{b-1})/b^i, 0, 1\big) \tag{3}$$

(See the Appendix for the derivation of this formula.) The value r can be recovered from the spectral decomposition by multiplying the bucket widths by the bucket fill proportions and summing:

$$r = \textstyle\sum_{i=0}^{\infty} b^i r_b^{i+} \tag{4}$$

This method can be extended to negative numbers by negating the sign on each element of the representation. Given any real number r, the formula for calculating each element then becomes:

$$r_b^{i} = sign(r) \times clamp\big((|r| - \tfrac{b^i - 1}{b-1})/b^i, 0, 1\big) \tag{5}$$

4.2 DECOMPOSING THE RETURN

The core idea behind our reinforcement learning approach is to apply spectral decomposition to the reward stream, and thus learn a spectral decomposition of the expected return. Before delving into

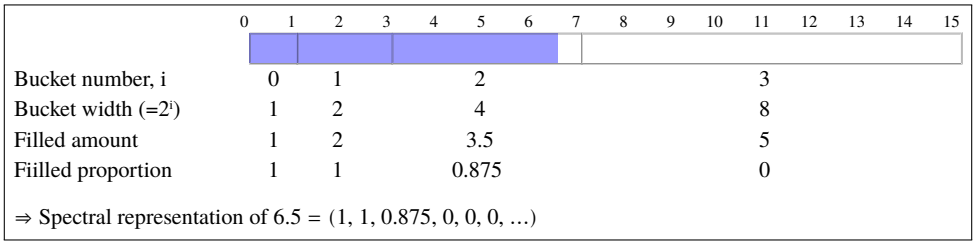


Figure 2: A worked example illustrating the spectral decomposition of the number 6.5.

			Spectral reward components			
Time step, t	γ^t	Reward, r	r_2^0	r_2^1	r_2^2	r_2^3
0	1	1	1	0	0	0
1	0.99	4	1	1	0.25	0
2	0.98	11	1	1	1	0.5
3	0.97	-4	-1	-1	-0.25	0
4	0.961	-10	-1	-1	-1	-0.375
Return		**2.254**				
Spectral returns			1.039	0.039	0.024	0.130
Weight, b^i			1	2	4	8
Spectral return $\times$ weight			1.039	0.078	0.098	1.039
Σ(Spectral return $\times$ weight)			**2.254**			

Table 1: An example of a return being decomposed into a weighted sum of spectral returns.

the technical details, see Table 1 for a worked example that highlights the intuition. The discount factor is $\gamma = 0.99$, and the return over the first five time steps is 2.254. The usual way of calculating the return is, of course, to multiply the rewards by γ^t then sum. The alternative, spectral approach is to first compute the spectral decomposition of the rewards according to Equation 5, then calculate the returns arising from each reward frequency. We refer to the decomposed rewards as *spectral rewards*, and to the decomposed returns as *spectral returns*. The overall return of 2.254 can be found by multiplying the spectral returns by their corresponding weights of b^i then summing.

To appreciate why we go to this extra effort in calculating the return, note that the spectral rewards are all bound to [-1, 1]. This in turn constrains the magnitude of the spectral returns to be no more than 1 / (1 - γ), which, again, is useful from a function approximation standpoint. Moreover, note that the relative sparsity of large rewards is made more apparent by this approach: As we progress from r_2^0 to r_2^3, the magnitudes of the spectral reward components decrease, with only the two largest magnitude rewards triggering a response in r_2^3.

4.3 Spectral Q-learning

To translate the above intuition into a concrete algorithm, we begin by defining the *spectral action-value function* as:

$$Q^\pi(s, a, i) = \mathbb{E}_\pi[\textstyle\sum_{k=t}^{\infty} \gamma^{k-t} r_{b,t}^i \mid a_t = a, s_t = s] \tag{6}$$

where $r_{b,t}^i$ denotes the ith element of the spectral reward at time t. As shown in the Appendix, it is straightforward to prove via the linearity of expectation that the standard action-value function can be recovered as the weighted sum:

$$Q^\pi(s, a) = \textstyle\sum_{i=0}^{\infty} b^i Q^\pi(s, a, i) \tag{7}$$

This leads to a natural extension of the Q-learning algorithm (Watkins, 1989), whereby the greedy action is found by maximising this expression with respect to the available actions, and $Q^\pi(s, a, i)$ is trained via the standard Bellman backup, except that the reward is replaced with the ith element of the spectral reward. For clarity, the full pseudocode for this algorithm, which we term *Spectral Q-learning*, is provided in the Appendix (see Algorithm 1).

An important practical limitation of this algorithm is that it is impossible to train over full, infinite spectrum; that is, one must choose a finite upper bound for i. However, for base b, observe that the first $N + 1$ frequencies fully capture all rewards up to magnitude $(b^{N+1} - 1)/(b - 1)$. (This follows from the fact that the bucket widths form a geometric series.) Due to the exponential nature of this term, it is possible to capture large magnitude rewards with a relatively small value of N. For example, with $b = 2$ and $N = 20$, the algorithm fully captures all rewards of magnitude up to 1 million, which is more than sufficient for *Atari*. Taking this idea one step further, we arrive at the following proposition, which is proved in the Appendix:

Proposition 1 *In the tabular setting, Spectral Q-learning behaves identically to standard Q-learning, provided that the rewards are bound in magnitude to no more than* $(b^{N+1} - 1)/(b - 1)$.

5 SPECTRAL DQN

In order to create a deep learning version of the Spectral Q-learning algorithm, we modify the DQN algorithm in an analogous way to which we modified the Q-learning algorithm in the previous section. We name the resultant algorithm *Spectral DQN*.

To parameterise all N frequencies of the spectral action-value function via a single network, we multiply the number of outputs on the standard DQN network by N. One crucial detail here is that, when calculating the greedy action, we "mute" all unused frequencies (i.e. frequencies that have never seen a non-zero reward) by multiplying by zero. This ensures that the action-value calculations do not blow out due to the exponential weights in Equation 7 being applied to random noise. This approach also allows us to set N to a conservatively large value, since it ensures that superfluous heads have no effect on the agent's policy.

For the training loss, we use a weighted sum of the temporal difference errors for each frequency:

$$\mathcal{L}(\theta) = \tfrac{1}{2}\sum_{i=0}^{N} w_i \Big[h\Big(r_b^i + \gamma \max_{a'} h^{-1}\big(\tilde{Q}(s',a',i;\theta^-)\big)\Big) - \tilde{Q}(s,a,i;\theta)\Big]^2 \tag{8}$$

where w_i is the error weight given to the ith frequency, s is a state sampled uniformly from the replay memory, a is the sampled action, s' is the next state, r is the reward (with spectral components r_b^i for $i \in \{0, \ldots, N\}$), θ are the parameters of the network being trained, and θ^- are the parameters of the periodically refreshed target network.

The most crucial detail here is the approach we take to weighting the loss for each frequency. Given that the ith frequency is weighted by b^i when calculating the overall action-values, it is tempting to set $w_i = b^i$. *However, this goes completely against one of the aims of our approach, which is to prevent the large reward frequencies from dominating the loss.* Therefore, rather than weighting by b^i, we strive to give all frequencies equal weighting. We do not simply set $w_i = 1$, since the training targets for the lower, denser reward frequencies will generally have greater variance, and we do not wish to overly emphasise them either. Instead, we set $w_i = 1/\sigma_i$, where σ_i is the standard deviation of the training targets for frequency i, calculated as a running average as in the Pop-Art algorithm.

Note that the loss also incorporates target compression, as per Equations 1 and 2. This is done to further equalise the treatment of the different frequencies. Even though the spectral rewards are already constrained to [-1, 1], we do not want the dense reward frequencies to have significantly larger training targets than sparse reward frequencies. This also removes one point of difference between Spectral DQN and the baseline DQN+TC, making comparison somewhat easier.

Finally, while we mute frequencies that have never seen a non-zero reward, a risk with this approach is that a single, large reward can cause a frequency to become unmuted, and the initial noisy outputs from that frequency may, when multiplied by b^i, disrupt the agent's policy in small reward regions of the task. To help mitigate this, we use an adaptive *mixed Monte Carlo update* (Bellemare et al., 2016), blending the training targets with a proportion of the Monte Carlo return for frequencies with extremely sparse rewards. This helps to quickly diminish the initial noisy outputs for frequencies that have hardly ever seen a non-zero reward. The Monte Carlo proportion for the ith frequency, β_i is set to decay with the standard deviation of the training targets, so that dense reward frequencies will be unaffected:

$$\beta_i = max(\beta_{max} - \sigma_i/k, 0) \tag{9}$$

where β_{max} is the maximum Monte Carlo proportion and k controls the annealing rate.

6 RETURNING TO THE EXTREME DOMAINS

To test how well Spectral DQN meets our aims, we first applied it to the two extreme domains from Section 3 – *Ponglantis* and *Exponential Pong* – where reward progressivity is clearly the key issue. In addition, to probe the importance of the frequency loss weighting, we included a variant of Spectral DQN with exponential loss weighting, i.e. $w_i = b^i$, which we argued against in the previous section. The results of this experiment, including the baseline results from Section 3, are shown in Figure 3, with each curve averaged over 5 seeds.

There are two clear findings here: (1) Spectral DQN is much better at handling extreme reward progressivity than the baselines, scoring much higher in *Ponglantis*, and making far more meaningful

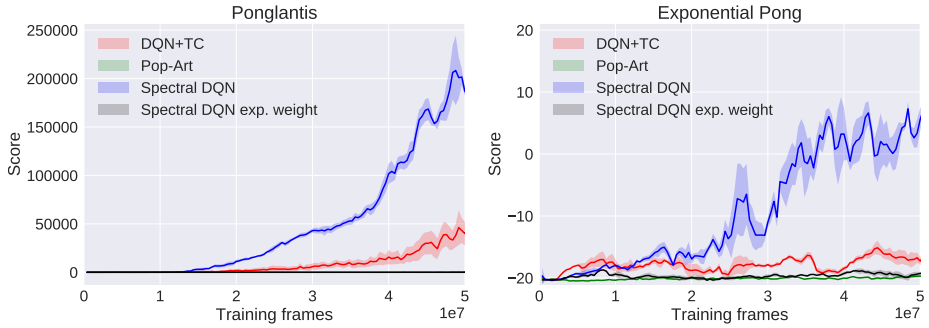


Figure 3: Mean episodic return for Spectral DQN vs. various baselines on the two extreme domains.

progress in *Exponential Pong*, eventually reaching a higher average score than the computer player. (2) The loss weighting term is of crucial importance, with the agent using exponential weights performing worse than even DQN+TC. The latter finding matches our expectations; as we noted in Section 3, target compression does at least somewhat mitigate the overemphasis given to large reward regions of the task, while Spectral DQN with $w_i = b^i$ gives heavy emphasis to such regions.

Even with the mixed Monte Carlo trick described above, tiny noise from the high frequencies does eventually start to interfere with Spectral DQN's policy in the early stages of *Exponential Pong*, which prevents it from pushing towards a perfect score of 21. Essentially, this comes down to a precision issue with the DQN architecture; it struggles to predict very near-zero values for certain inputs (in percentage accuracy terms) if it is also required to output large values elsewhere.

7 Experiments in Standard Atari Games

To validate that Spectral DQN's performance in *Ponglantis* and *Exponential Pong* does not come at the expense of performance in less constructed domains, we also applied it to 6 standard *Atari* games. Moreover, to validate that Spectral DQN did not succeed in the above experiments by somehow performing an implicit kind of reward clipping, we included 3 games where reward clipping is known to be detrimental, as per van Hasselt et al. (2016): *Bowling*, *Centipede* and *Ms. Pacman*. The results of these experiments are shown in Figure 4, with all curves averaged over 5 seeds.

While Spectral DQN's performance relative to the baselines is less impressive here, it remains strong overall. Though it is clearly beaten by DQN+TC in *Centipede*, it notably outperforms DQN+TC in *Bowling*, *Chopper Command* and *Video Pinball*. Moreover, the fact that it performs reasonably in *Bowling*, *Centipede* and *Ms. Pacman* shows that it is truly responding to the unclipped reward.

Since *Video Pinball* exhibits reward progressivity, as explained in the introduction, Spectral DQN's strong performance there fits well with our hypothesis. While *Ms. Pacman* also has progressive rewards – the player receives an increasing bonus for eating multiple ghosts with a single power pellet – these rewards come so close together that they are within the agent's temporal horizon at the start of an episode. As such, it may not matter if the agent effectively becomes blinded to small rewards. This illustrates an important point: For the issues associated with progressive rewards to truly come to the fore, the small and large reward regions of the task must be sufficiently separated.

One other possible advantage of Spectral DQN over DQN+TC in high scoring games is simply that its training targets are more constrained. In high scoring games, such as *Video Pinball* and *Chopper Command*, the expected returns can reach into the thousands, so even with target compression this may be problematic for DQN+TC. On the other hand, the only game where DQN+TC clearly outperforms Spectral DQN is *Centipede*, which is also high scoring. One possible explanation for this is that the reward structure in *Centipede* "confuses" Spectral DQN: In general, it is likely that Spectral DQN learns a positive association between the different frequencies of the reward (due to the fact that large rewards also trigger low frequencies). However, in *Centipede*, losing a life triggers a stream of consecutive small rewards. Therefore, associating small and large rewards may cause Spectral DQN to make faulty generalisations about the utility of dying.

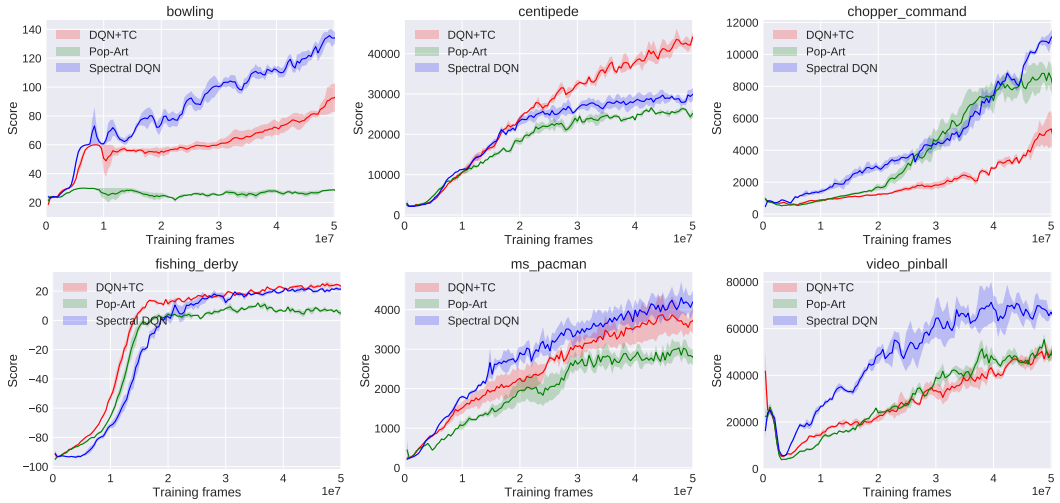


Figure 4: Mean episodic return for Spectral DQN vs. baselines on standard *Atari* games.

One final, interesting observation is that the rewards in *Bowling* are small, and not progressive at all. In this case, Spectral DQN's strong performance is most likely attributable to its adaptive mixed Monte Carlo update, which may help it latch on to the large, delayed rewards for bowling strikes and spares, which are initially very sparse. It should be noted, however, that standard mixed Monte Carlo updates are known to perform disastrously in two of the games we tested (*Fishing Derby* and *Video Pinball*) (Ostrovski et al., 2017). The fact that Spectral DQN performed well in these games points to the *adaptive* nature of the mixed Monte Carlo updates as being key. Looking beyond this, the ability to apply different backup operators to different components of the return spectrum under Spectral DQN is an interesting direction for future work.

8 Discussion and Conclusion

In this paper, we brought attention to the phenomenon of reward progressivity, which poses a potential issue for value-based reinforcement learning agents; namely, it can cause an agent's performance to degrade on relatively unrewarding regions of a task once it discovers large rewards. To address this, we proposed Spectral DQN, which decomposes the expected return magnitudinally, allowing the losses arising from regions of the task with vastly different reward scales to be balanced. Our experiments showed this approach to be particularly effective in domains exhibiting extreme reward progressivity, while still yielding more than competitive performance elsewhere.

Our work fits into a broader line of research whereby reinforcement learning agents seek to predict quantities other than just the mean expected return (Jaderberg et al., 2016; Bellemare et al., 2017; Fedus et al., 2019). Of this related work, one of the most similar approaches to ours is that of Romoff et al. (2019), who learn the value function in a modular way by decomposing based on the discount factor. Another closely related line of work is distributional reinforcement learning (Bellemare et al., 2017; Dabney et al., 2018b;a), with the *C51* agent (Bellemare et al., 2017) in particular bearing quite a similar network architecture to ours. Finally, another, older line of work in this vein is decomposed RL (Russell & Zimdars, 2003; Sprague & Ballard, 2003; Van Seijen et al., 2017), whereby an agent is responsible for predicting only a portion of the action-value function.

There has been some conjecture that one advantage of combining distributional RL with function approximation is that it forces the agent to learn a richer representation (Bellemare et al., 2017; Lyle et al., 2019). The same argument may also apply to our approach. In particular, in some tasks, small rewards are of more informational value than their face value represents. For example, in *Video Pinball*, where rewards range into the thousands, the play field includes "spinners" that pay only one point per hit. Despite their negligible face value, hitting the spinners can dramatically change the velocity of the ball, so giving the lower frequencies more weight in the training loss may promote a stronger representation.

REFERENCES

Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, and Charles Blundell. Agent57: Outperforming the Atari Human Benchmark. *arXiv preprint arXiv:2003.13350*, 2020a. URL https://arxiv.org/pdf/2003.13350.pdf.

Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexander Pritzel, Andew Bolt, et al. Never Give Up: Learning Directed Exploration Strategies. In *Proceedings of the 2020 International Conference on Learning Representations (ICLR)*, 2020b. URL https://arxiv.org/pdf/2002.06038.pdf.

Marc Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying Count-Based Exploration and Intrinsic Motivation. In *Advances in Neural Information Processing Systems 29*, pp. 1471–1479. 2016.

Marc G. Bellemare, Will Dabney, and Rémi Munos. A distributional perspective on reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pp. 449–458, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

Will Dabney, Georg Ostrovski, David Silver, and Remi Munos. Implicit Quantile Networks for Distributional Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1096–1105, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018a. PMLR.

Will Dabney, Mark Rowland, Marc G Bellemare, and Rémi Munos. Distributional Reinforcement Learning with Quantile Regression. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018b.

William Fedus, Carles Gelada, Yoshua Bengio, Marc G Bellemare, and Hugo Larochelle. Hyperbolic Discounting and Learning over Multiple Horizons. *arXiv preprint arXiv:1902.06865*, 2019.

Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement Learning with Unsupervised Auxiliary Tasks. *arXiv preprint arXiv:1611.05397*, 2016. URL https://arxiv.org/pdf/1611.05397.pdf.

Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. Recurrent Experience Replay in Distributed Reinforcement Learning. In *Proceedings of the 2019 International Conference on Learning Representations (ICLR)*, 2019.

Clare Lyle, Marc G Bellemare, and Pablo Samuel Castro. A Comparative Analysis of Expected and Distributional Reinforcement Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4504–4511, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-Level Control Through Deep Reinforcement Learning. *Nature*, 518(7540):529–533, 2015.

Georg Ostrovski, Marc G. Bellemare, Aäron van den Oord, and Rémi Munos. Count-Based Exploration with Neural Density Models. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, volume 70, pp. 2721–2730. PMLR, 2017.

Tobias Pohlen, Bilal Piot, Todd Hester, Mohammad Gheshlaghi Azar, Dan Horgan, David Budden, Gabriel Barth-Maron, Hado Van Hasselt, John Quan, Mel Večerík, et al. Observe and Look Further: Achieving Consistent Performance on Atari. *arXiv preprint arXiv:1805.11593*, 2018.

Joshua Romoff, Peter Henderson, Ahmed Touati, Emma Brunskill, Joelle Pineau, and Yann Ollivier. Separating value functions across time-scales. volume 97 of *Proceedings of Machine Learning Research*, pp. 5468–5477, Long Beach, California, USA, 2019. PMLR.

Stuart J Russell and Andrew Zimdars. Q-Decomposition for Reinforcement Learning Agents. In *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 656–663, 2003.

Nathan Sprague and Dana Ballard. Multiple-Goal Reinforcement Learning with Modular Sarsa(0). 2003.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3 (1):9–44, 1988.

Hado P van Hasselt, Arthur Guez, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude. In *Advances in Neural Information Processing Systems 29*, pp. 4287–4295. Curran Associates, Inc., 2016.

Harm Van Seijen, Mehdi Fatemi, Joshua Romoff, Romain Laroche, Tavian Barnes, and Jeffrey Tsang. Hybrid Reward Architecture for Reinforcement Learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 30*, pp. 5392–5402. 2017.

Christopher John Cornish Hellaby Watkins. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, 1989.

A APPENDIX

In this appendix, we provide further details about our experimental setup and prove the theoretical claims made in Section 4 of the main text.

EXPERIMENTAL SETUP

We injected stochasticity into the *Atari* environment via the no-ops regime used in the Nature DQN paper (Mnih et al., 2015). Rather than conducting periodic evaluations of each agent with a small exploration constant, we simply report training performance in all graphs, since this arguably provides a less variable measure of progress. All curves in the paper were calculated by averaging over 5 seeds. Shaded regions represent bootstrapped single standard deviation confidence intervals.

Hyperparameters specific to Spectral DQN are listed in Table 2, while the hyperparameters common to all agents are listed in Table 3.

To ensure that the gradients remain steady under Spectral DQN as additional frequencies come online, we normalise the temporal difference errors by dividing by a running average of their L2 norm. This made it easier to find a value for the Adam regularisation constant that performed well across all tasks. To ensure fair comparison against the benchmark agents, we also normalised their TD errors.

DERIVATION OF THE SPECTRAL DECOMPOSITION FORMULA

Recall from Section 4.1 that the spectral representation of a number is found by laying buckets of exponentially increasing width on the number line, then calculating the proportion of each bucket that is filled.

To calculate the proportion of the ith bucket that is filled by a non-negative number r, note that the sum of all the preceding buckets' widths is equal to:

$$\sum_{k=0}^{i-1} b^k = \frac{b^i - 1}{b-1} \tag{10}$$

Therefore, the amount leftover to fill the ith bucket is:

$$r - \frac{b^i - 1}{b-1} \tag{11}$$

To calculate the proportion of the ith bucket that is filled, we must divide this quantity by the width of the ith bucket (which is equal to b^i) then clamp the result to the allowable range of values:

$$\text{Filled proportion} = clamp\big((r - \frac{b^i - 1}{b-1})/b^i, 0, 1\big) \tag{12}$$

Finally, to convert this into an odd function so that the representation of $-r$ is equal to the negated representation of r, we replace r with $|r|$ and multiply the result by $sign(r)$. This yields the formula for the ith element of the spectral representation:

$$r_b^i = sign(r) \times clamp\big((|r| - \frac{b^i - 1}{b-1})/b^i, 0, 1\big) \tag{13}$$

Note that a key difference between base 2 spectral representations and binary representations is that spectral representations are, loosely speaking, better distance-preserving. For example, the binary representations of 7 and 8 are bitwise very different (0111 and 1000 respectively), whereas the spectral representations are (1, 1, 1, 0, 0, ...) and (1, 1, 1, 0.125, 0, ...). Intuitively, the latter representation ought to be better suited to generalising function approximation.

Hyperparameter	Value
Base of spectral decomposition, b	2.0
Number of frequencies, N	20
Mixed Monte Carlo β_{max}	0.1
Mixed Monte Carlo k	1.0

Table 2: Hyperparameters specific to the Spectral DQN agent.

Hyperparameter	Value
Optimiser	Adam
Adam learning rate	2.5×10^{-5}
Adam regularisation constant	0.005 / 32
Discount factor	$0.99^{1/3}$
Multi-step n	3
Final exploration ϵ	0.01
Reward clipping	None
Life loss signal used	True

Table 3: Hyperparameters shared between all agents.

PROOF OF EQUATION 7

Starting with the standard definition of the action-value function, then replacing the reward at each time step with its spectral decomposition, we have:

$$Q^\pi(s,a) = \mathbb{E}_\pi[\textstyle\sum_{k=t}^{\infty} \gamma^{k-t} r_k \mid a_t = a, s_t = s] \tag{14}$$

$$= \mathbb{E}_\pi[\textstyle\sum_{k=t}^{\infty} \gamma^{k-t} \sum_{i=0}^{\infty} b^i r_{b,k}^i \mid a_t = a, s_t = s] \tag{15}$$

Since there is no interaction between the summation indices, we can rearrange (15) into:

$$Q^\pi(s,a) = \mathbb{E}_\pi[\textstyle\sum_{i=0}^{\infty} b^i \sum_{k=t}^{\infty} \gamma^{k-t} r_{b,k}^i \mid a_t = a, s_t = s] \tag{16}$$

Finally, using the linearity of expectation, we can take the expectation operator inside and write:

$$Q^\pi(s,a) = \textstyle\sum_{i=0}^{\infty} b^i \mathbb{E}_\pi[\sum_{k=t}^{\infty} \gamma^{k-t} r_{b,k}^i \mid a_t = a, s_t = s] \tag{17}$$

$$= \textstyle\sum_{i=0}^{\infty} b^i Q^\pi(s,a,i) \tag{18}$$

SPECTRAL Q-LEARNING ALGORITHM

Pseudocode for the Spectral Q-Learning algorithm is provided in Algorithm 1.

PROOF OF PROPOSITION 1

Proposition 1 *In the tabular setting, Spectral Q-learning behaves identically to standard Q-learning, provided that the rewards are bound in magnitude to no more than* $(b^{N+1} - 1)/(b - 1)$.

Algorithm 1 Spectral Q-Learning

var: the base of the spectral decomposition, b
var: the number of spectral components trained, N
var: the learning rate, α

Initialise $Q(s,a,i)$
for each episode **do**
    Sample s from start distribution
    while s is not terminal **do**
        Choose a according to ϵ-greedy policy derived from $Q(s,a) = \sum_{i=0}^{N} b^i Q(s,a,i)$
        Take action a, observe r, s'
        $a'^* = \arg\max_{a'} \sum_{i=0}^{N} b^i Q(s',a',i)$
        for $i \in \{0, \ldots, N\}$ **do**
            $Q(s,a,i) \leftarrow Q(s,a,i) + \alpha[r_b^i + \gamma Q(s',a'^*,i) - Q(s,a,i)]$
        $s \leftarrow s'$

We prove this statement via induction. The first point to observe is that, provided the action-value estimates of the two algorithms agree at a given point in time, their ϵ-greedy policies at that time will also be equivalent. Therefore, assuming that the action-value functions agree at $t = 0$ (e.g. they are both initialised to zero), all we need to prove is the inductive step, i.e. that agreement in action-values at time t implies agreement in action-values at time $t + 1$.

To distinguish between the action-value estimates of the two algorithms at different time steps, we use the following notation:

- Let $\hat{Q}(s, a; t)$ denote the action-value estimates under standard Q-learning after t updates.
- Let $Q(s, a; t) = \sum_{i=0}^{N} b^i Q(s, a, i; t)$ denote the action-values estimates under Spectral Q-learning after t updates.

For the induction step, we need to show that $Q(s, a; t + 1) = \hat{Q}(s, a; t + 1)\ \forall s \in \mathcal{S}, a \in \mathcal{A}$ given that $Q(s, a; t) = \hat{Q}(s, a; t)\ \forall s \in \mathcal{S}, a \in \mathcal{A}$.

To begin, note that the standard Q-learning backup expressed under the above notation is:

$$\hat{Q}(s, a; t + 1) \leftarrow \hat{Q}(s, a; t) + \alpha\big[r + \gamma \hat{Q}(s', a'^*; t) - \hat{Q}(s, a; t)\big] \tag{19}$$

where $a'^* = \arg\max_{a'} \hat{Q}(s', a'^*; t)$ is the greedy action for the next state.

Since we assume equal action-values at time t, Spectral Q-learning will calculate the same greedy action for s'. Hence, the backup for each frequency of the spectral action-value function is:

$$Q(s, a, i; t + 1) \leftarrow Q(s, a, i; t) + \alpha\big[r_b^i + \gamma Q(s', a'^*, i; t) - Q(s, a, i; t)\big] \tag{20}$$

After performing this update on each frequency, the function at time $t + 1$ becomes:

$$\begin{aligned}
Q(s, a; t + 1) &= \textstyle\sum_{i=0}^{N} b^i \Big[Q(s, a, i; t) + \alpha\big[r_b^i + \gamma Q(s', a'^*, i; t) - Q(s, a, i; t)\big]\Big] && (21)\\
&= \textstyle\sum_{i=0}^{N} b^i Q(s, a, i; t) + \sum_{i=0}^{N} \alpha b^i \big[r_b^i + \gamma Q(s', a'^*, i; t) - Q(s, a, i; t)\big] && (22)\\
&= Q(s, a; t) + \textstyle\sum_{i=0}^{N} \alpha b^i \big[r_b^i + \gamma Q(s', a'^*, i; t) - Q(s, a, i; t)\big] && (23)
\end{aligned}$$

Since the induction base case assumes that $\hat{Q}(s, a; t) = Q(s, a; t)\ \forall s \in \mathcal{S}, a \in \mathcal{A}$, we can rewrite this as:

$$\begin{aligned}
&Q(s, a, i; t + 1)\\
&\quad= \hat{Q}(s, a; t) + \alpha \textstyle\sum_{i=0}^{N} b^i \big[r_b^i + \gamma Q(s', a'^*, i; t) - Q(s, a, i; t)\big] && (24)\\
&\quad= \hat{Q}(s, a; t) + \alpha\big[\textstyle\sum_{i=0}^{N} b^i r_b^i + \gamma \sum_{i=0}^{N} b^i Q(s', a'^*, i; t) - \sum_{i=0}^{N} b^i Q(s, a, i; t)\big] && (25)\\
&\quad= \hat{Q}(s, a; t) + \alpha\big[\textstyle\sum_{i=0}^{N} b^i r_b^i + \gamma Q(s', a'^*; t) - Q(s, a; t)\big] && (26)\\
&\quad= \hat{Q}(s, a; t) + \alpha\big[\textstyle\sum_{i=0}^{N} b^i r_b^i + \gamma \hat{Q}(s', a'^*; t) - \hat{Q}(s, a; t)\big] && (27)
\end{aligned}$$

Now, all that remains to establish equality with the standard Q-learning update in (19) is to show that $\sum_{i=0}^{N} b^i r_b^i = r$. That is, we need to show that the partial sum over indices 0 to N fully captures the reward, or, equivalently, that $r_b^i = 0$ for all $i >= N + 1$.

From the spectral decomposition formula (13) and the proposition's statement that the rewards are bound to no more than $(b^{N+1} - 1)/(b - 1)$, we have:

$$\begin{aligned}
|r_b^i| &= clamp\big((|r| - \tfrac{b^i - 1}{b - 1})/b^i, 0, 1\big) && (28)\\
&\leq clamp\big((\tfrac{b^{N+1} - 1}{b - 1} - \tfrac{b^i - 1}{b - 1})/b^i, 0, 1\big) && (29)\\
&= clamp\big((\tfrac{b^i(b^{N+1-i} - 1)}{b - 1})/b^i, 0, 1\big) && (30)
\end{aligned}$$

Now, if $i >= N + 1$, it implies that $(b^{N+1-i} - 1) \leq 0$, so the whole expression in (30) becomes 0.

Thus $\sum_{i=0}^{N} b^i r_b^i = \sum_{i=0}^{\infty} b^i r_b^i = r$ and the proof is complete. ■