A. Methods

A.1. Computing the spectral norm

Power Iteration. Computing the dominant singular value at each step would be expensive, therefore spectral normalisation is usually performed through power iteration. For each set of weights $\mathbf{W}_i \in \mathbb{R}^{N_i \times N_{i-1}} : i \in \mathcal{S}$ the corresponding left and right singular vectors are stored, and two extra matrix-vector multiplications for each normalised layer are performed at each forward pass (see Formulas 17, 18). At inference time there is no extra computational cost.

$$\mathbf{v} \leftarrow \mathbf{W}_i \mathbf{u}^{(t-1)}; \quad \alpha \leftarrow \|\mathbf{v}\|; \quad \mathbf{v}^{(t)} \leftarrow \alpha^{-1}\mathbf{v} \tag{17}$$

$$\mathbf{u} \leftarrow \mathbf{W}_i^\intercal \mathbf{v}^{(t)}; \quad \rho \leftarrow \|\mathbf{u}\|; \quad \mathbf{u}^{(t)} \leftarrow \rho^{-1}\mathbf{u} \tag{18}$$

For convolutional layers we adapt the procedure in (Gouk et al., 2020) and the two matrix-vector multiplications are replaced by convolutional and transposed convolutional operations.

Backpropagating through the norm. Since the parameters that are tuned during optimisation are the unnormalised weights $\mathbf{W}_i$, we investigated if there are any advantages in backpropagating through the power iteration step, i.e. considering the partial derivative $\frac{\partial \hat{\mathbf{W}}_i}{\partial \rho_i}\frac{\partial \rho_i}{\partial \mathbf{W}_i}$ when computing the gradient. Precisely, we verified if dropping the second term in the right-hand side of Formula 19 biases the gradient.

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_i} = \rho_i^{-1}\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{W}}_i} - \rho_i^{-2}\mathbf{u}\mathbf{v}^\intercal \left(vec\left(\frac{\partial \mathcal{L}}{\partial \hat{\mathbf{W}}_i}\right) vec\left(\mathbf{W}_i\right)\right) \tag{19}$$

In a batch of experiments performed on a subset of games of Atari we noticed no loss in performance when dropping the Jacobian from the power iteration.

A.2. Computing the norm of the Jacobians

Experiments in Sec. 5.2 used the maximum norm of the jacobians w.r.t. the inputs as an indirect metric for network function's smoothness. For each network we collected thousands of states (*on-policy*), computing the maximum euclidean norm of the jacobian w.r.t. the inputs: $\max_{i,\mathbf{x}} ||\frac{\partial q_i(\mathbf{x};\theta)}{\partial \mathbf{x}}||_2$. Note that we used the euclidean norm (and not the operator norm) to measure smoothness.

B. MinAtar experiments

Game selection. MinAtar (Young & Tian, 2019) benchmark is a collection of five games that reproduce the dynamics of Arcade Learning Environment (ALE) counterparts, albeit in a smaller observational space. Out of *Asterix, Breakout, Seaquest, Space Invaders* and *Freeway* we excluded the latter from all our experiments since all the agents performed essentially the same on this game.

Network architecture. All experiments on MinAtar are using a convolutional network with L_C convolutional layers with the same number of channels, a hidden linear layer, and the output layer. The number of input channels is game-dependent in MinAtar. All convolutional layers have a kernel size of 3 and a stride of 1. All hidden layers have rectified linear units. Whenever we vary depth we change the number of convolutional layers L_C, keeping the two linear layers. When the width is varied we change the width of both convolutional layers (e.g. 16/24/32 channels) and the penultimate linear layer. All convolutional layers are always identically scaled.

We list all the architectures used in various experiments described in this section in Table 6.

General hyper-parameter settings. In all our MinAtar experiments we used the same set of hyper-parameters returned by a small grid search around the initial values published by (Young & Tian, 2019). We list the values we settled on in Table 2. For the rest of this section we only mention how we deviate from this set of hyper-parameters and settings for each of the experiments that follow.

HYPER-PARAMETER	VALUE
discount γ	0.99
update frequency	4
target update frequency	4,000
starting ϵ	1.0
final ϵ	0.01
ϵ steps	250,000
ϵ schedule	linear
warmup steps	5,000
replay size	100,000
history length	1
cost function	MSE LOSS
optimiser	ADAM
learning rate η	0.00025
damping term ϵ	0.0003125
β_1, β_2	(0.9, 0.999)
validation steps	125000
validation ϵ	0.001

Table 2: MinAtar general hyper-parameter settings.

MinAtar Normalised Score. In our work we present many MinAtar experiments as averages over the four games we tested on. Since in MinAtar the range of the expected returns is game dependent we normalise the score. Inspired by the Human Normalised Score in (Mnih et al., 2015) we take the largest score ever recorded by a baseline agent in our experiments and use it to compute MNS $= 100 \times (\text{score}_{\text{agent}} - \text{score}_{\text{random}})/(\text{score}_{\text{max}} - \text{score}_{\text{random}})$. We can then use the resulting MinAtar Normalised Score whenever we need to report performance aggregates over the games.

GAME	MAX	RANDOM
Asterix	78.90	0.49
Breakout	122.88	0.52
Seaquest	93.91	0.09
Space Invaders	360.92	2.86

Table 3: MinAtar maximum and random scores used for computing the MinAtar Normalised Score.

B.1. Large optimiser hyper-parameter sweep

For the large optimiser hyper-parameter sweep we train a DQN agent with six different architectures as listed in Table 6. For each architecture we then trained on all combinations of optimisation hyper-parameters (from the lists below), both normalised and baseline agents, two seeds each. We generated values for the learning rate using a geometric progression in the following ranges:

Optimiser	Hyper-parameters
ADAM	η: $\{0.00001, ..., 0.00215\}$ ϵ: $\{0.00001, ..., 0.01\}$
RMSPROP	η: $\{0.00001, ..., 0.001\}$ ϵ: $\{0.00001, ..., 0.0316\}$ $\alpha \in \{0.95\}$ centered: yes
ADAM	η: $\{0.00001, ..., 0.001\}$

Table 4: Hyper-parameters ranges for the large optimiser sweep.

Figure 8 illustrates our findings. In the case of Adam optimiser we can see that normalising one layer does not degrade the performance of the baseline and allows for larger learning rates. A similar observation can be made for RMSprop and exploring learning rates larges than 0.001 should clarify to what degree the trend already visibile continues.

Figure 9 is a different visualisation of the same results. Here we sort the x-axis by the mean normalised score of an optimiser configuration for the DQN agents equipped with SN and plot both the performance of the baseline and

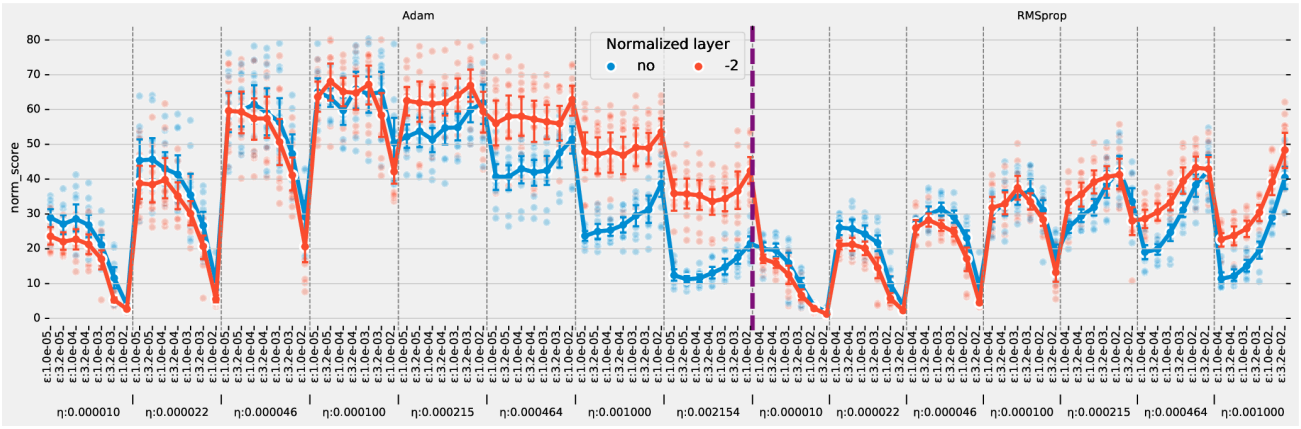

Figure 8: **Spectral Normalisation increases the range of effective optimisation settings.** Each configuration is used to train 6 models of different depth and width on four MinAtar games, two seeds each. Dots represent the average maximum normalised score over the four games.

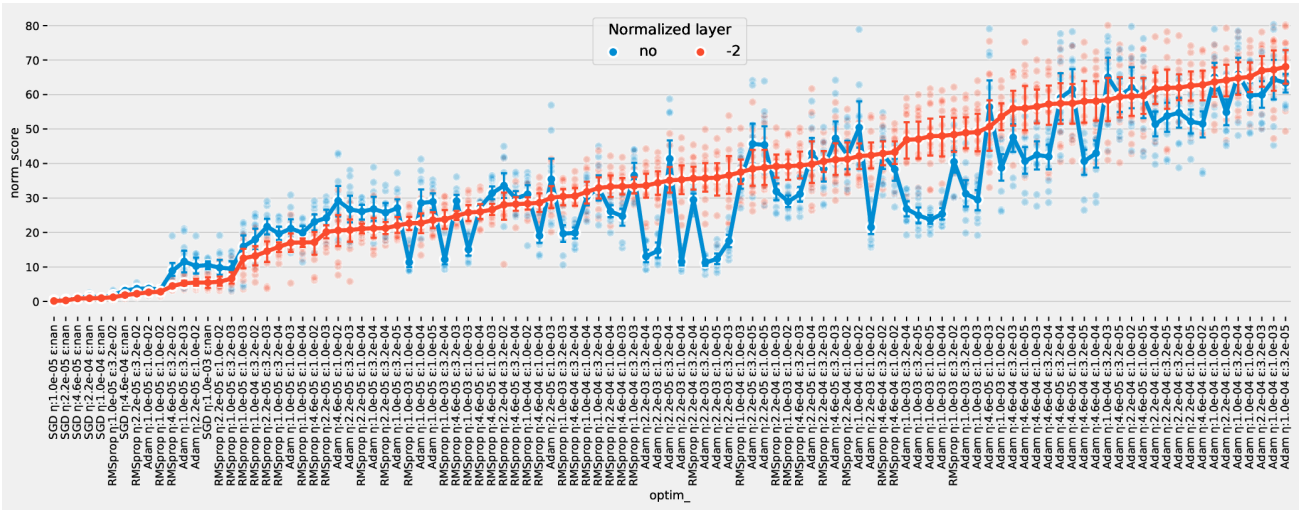

Figure 9: **Spectral Normalisation improves on the baseline on a wide range of optimisation settings.** Each optimisation setting is used to train 6 models of different depths and widths on four MinAtar games, two seeds each. Dots represent the average MinAtar Normalised Score achieved over four games. We sort by the mean performance of the SN experiment and show the baseline stays mostly under this curve in the region of high performance optimiser configurations.

the normalised experiments. We show that the performance of the normalised agent generally outperforms the baseline, especially in the right half of the plot corresponding with higher performance agents.

B.2. Effect on model capacity

For understanding the effect of SN on model capacity we train DQN agents with 12 different model sizes of three different depths and four different widths. We apply SN on various layer subsets and report in Figure 12 the MinAtar Normalised Score averaged over the four games. All the other parameters remain the same as described at the beginning of this section. Each resulting game-architecture combination was trained on 10 seeds.

B.3. Smoothness and performance are weakly correlated

In Figure 14 we plot the peak performance and the norm of the Jacobian for each of the seeds in the experiment we discuss in Section B.3 instead of averages.

Computing a correlation measure for all the normalisation schemes in the experiment is complicated by the fact that any selection we make affects the correlation we want to measure. Limiting ourselves to just the baseline and DQN[-2], the normalisation scheme we have shown repeatedly that it does not hurt performance, we computed the Spearman rank-order correlation we report in Table 5.

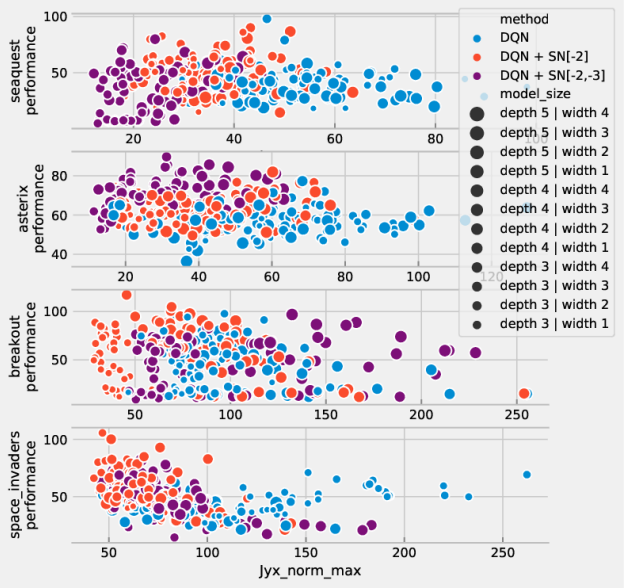

Figure 14: **Applying SN on a layer subset does not consistently produce smoother networks.** Often normalising a subset of the network's layers makes the network less smooth than the baseline while performance improves still. Each point in the graph represents the maximum performance achieved by a single seed. Detailed view of Fig. 5

.

GAME	SPEARMAN RANK
Asterix	-0.129
Breakout	-0.199
Seaquest	-0.151
Space Invaders	-0.453

Table 5: Correlation between the norm of the Jacobian and peak performance for each game.

B.4. Other regularisation methods

As briefly touched upon in Section 5.2, we investigated whether other regularisation methods imposing smoothness constraints can have similar effects on the agent's performance. To this end we ran experiments with both Gradient Penalty (GP) and Batch Normalisation (BN) on several architectures (Table 6).

Batch Normalisation. For each of the architectures we employed Batch Normalisation (BN) after the ReLU activation of every convolutional or linear layer except the output.

Gradient Penalty. We did extensive experimentation and penalty coefficient tuning for Gradient Penalty regularisation. Specifically we tried penalising the norm of the sum of the gradients of all actions (the way GP is usually implemented in other domains), regularising the expected norm of each Q-value with respect to the state and also regularising the norm of the gradient of the Q-value associated with the optimal action. In all cases we swept through a wide range of penalty coefficients λ with various degree of success. In Figure 16 we report the results of the best setting we could identify.

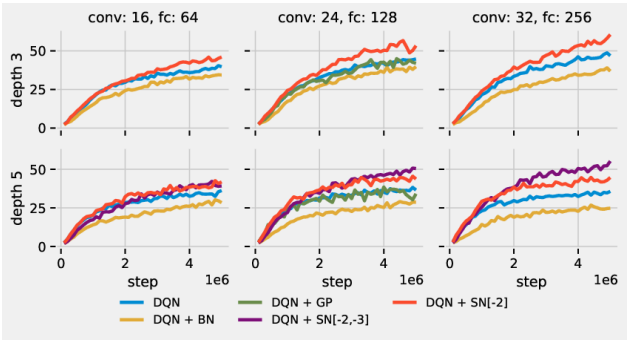

Figure 16: **Regularization does not recover Spectral Normalization performance.** Performance on MinAtar games of SN, GP and BN. Each line is an average over normalized scores of each game. Ten seeds for each configuration.

Relaxations to 1-Lipschitz normalisation. We run a small experiment with a three layer network to investigate the relaxation introduced by (Gouk et al., 2020): $\hat{\mathbf{W}}_i = \mathbf{W}_i / \max(\lambda_i, \|\mathbf{W}_i\|_2)$. Figure 10 shows that for increased λ_i values (which we keep equal for every layer) we are able to get good performance even when normalising all the layers of the network, further confirming our initial observation that controlling the amount of regularisation is important to achieving optimal performance and that achieving 1-Lipschitz functions is not critical in this setup. The increased computation required when approximating ρ for all the layers and the addition of one hyper-parameter per layer determined us to not pursue this setup further.

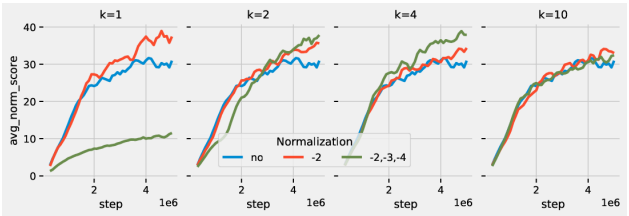

Figure 10: **Relaxing the 1-Lipschitz condition recovers the performance for fully normalised networks**. Average MinAtar Normalised Score of SN with different target Lipschitz constants. Each line is an average over normalised scores of 4 games $\times$ 10 seeds.

B.5. Adaptability to changing dynamics

We noticed that in most experiments on MinAtar the DQN agent reaches its peak performance within the standard 5M steps and then it plateaus. Most of these training curves end in flat performance regimes. In contrast, when SN is used on a single hidden layer, not only that agents surpass the baseline performance, but they also show continuous improvement with no signs of plateauing. We therefore asked what happens if we extend the training period and trained agents for 15M steps. As anticipated, our *long* training experiments show that SN agents show steep learning curves even after large numbers of steps (Fig. 11), supporting our claim that SN yields a better adapting optimiser.

B.6. Spectral Schedulers

For the experiments with the schedulers proposed in Sec. 5.4 we used the four estimator architectures in Table 6. We detail the MinAtar Normalised Score for various subsets of layers considered in the comparison in Figure 18. In a single subplot the lines represent the performance of the baseline, SN, and spectral schedulers, all using the same spectral radii. Observe that DIVOUT has a close behaviour to that of SN not only on average, but also on a case base. In contrast, the MULEPS and DIVGRAD optimisers converge even when all hidden layers are normalised.

Experiment	No of conv layers	Conv width	FC width
Large optimisation sweep (B.1)	1	24	128
	2	24	128
	3	24	128
	4	24	128
	2	32	256
	3	32	256
Regularisation (B.4)	1	16	64
	3	16	64
	1	24	128
	3	24	128
	1	32	256
	3	32	256
Spectral schedulers (B.6) and long training run (B.5)	1	24	128
	2	24	128
	1	32	256
	2	32	256

Table 6: Architecture sets used in the experiments described in this section.

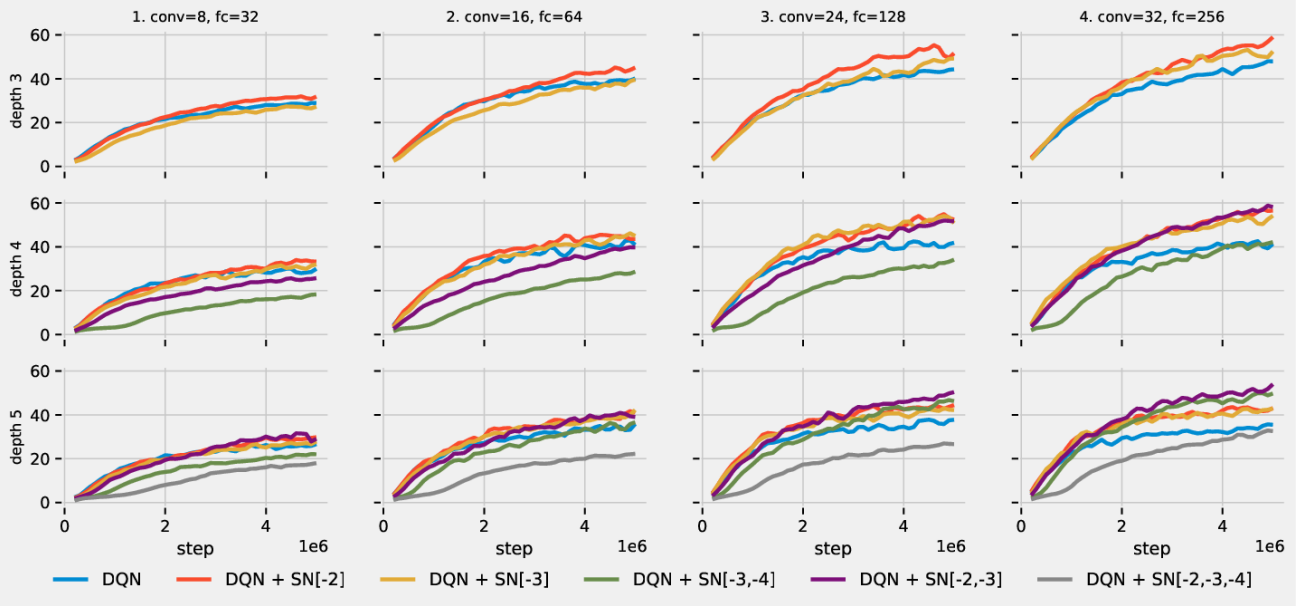

Figure 12: **Spectral Normalization shows gains for all model sizes.** Looking at the baseline (— DQN), we observe two performance regimes on MinAtar: for shallow, depth 3 models, performance increases with the width of the model; for deeper models performance generally stagnates with increasing depth and width. In both regimes applying SN on individual (—, —) or multiple (—) layers improves upon the baseline suggesting a regularisation effect we could not reproduce with other regularisation methods. Notice that the strong regularisation resulted from applying SN to input layers (—) or too many layers (—) can however degrade performance. Each line is an average over normalized scores of 4 games × 10 seeds. Detailed view of Fig. 4.

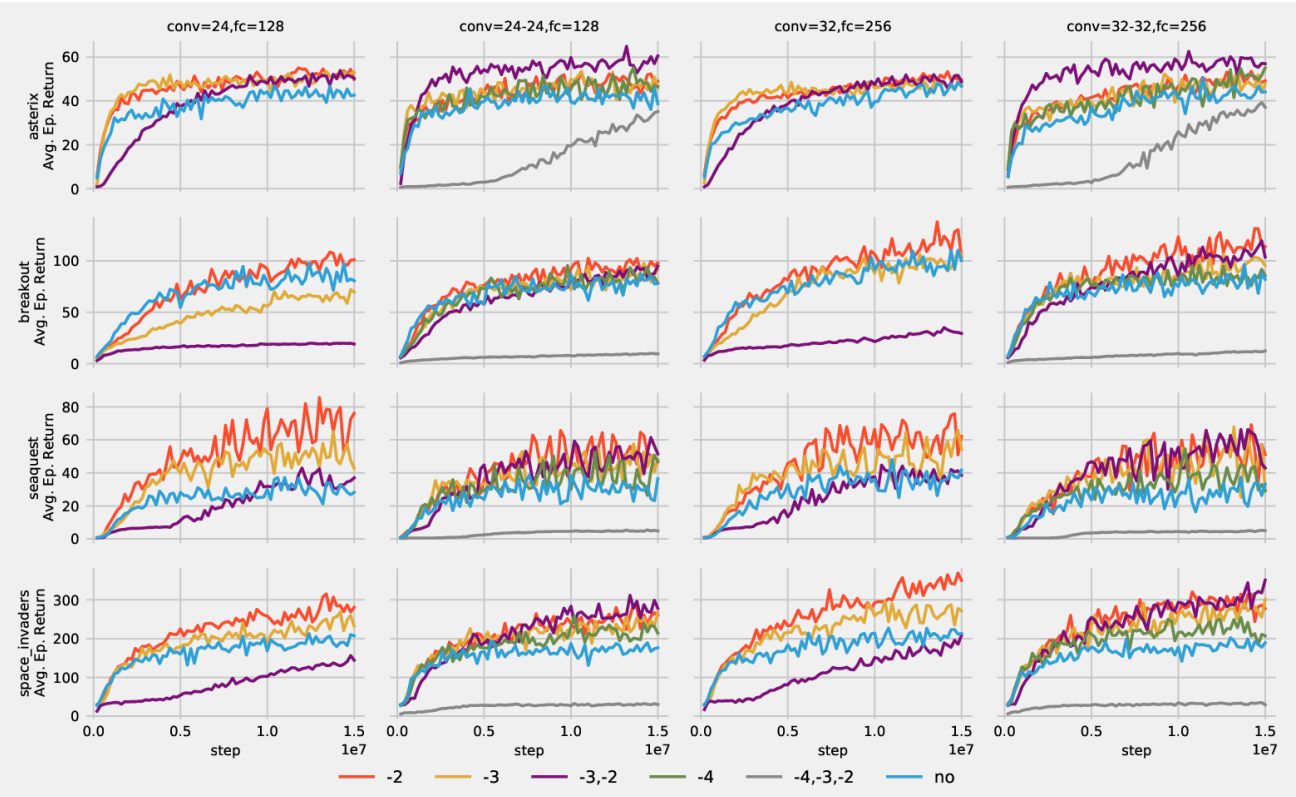

Figure 11: Performance curves of DQN agents using four different architectures trained for 15M steps. This plot shows that baseline plateaus, while spectrally normalised variants generally don't. See Fig. 13 for plots of the spectral radii for the same experiments.

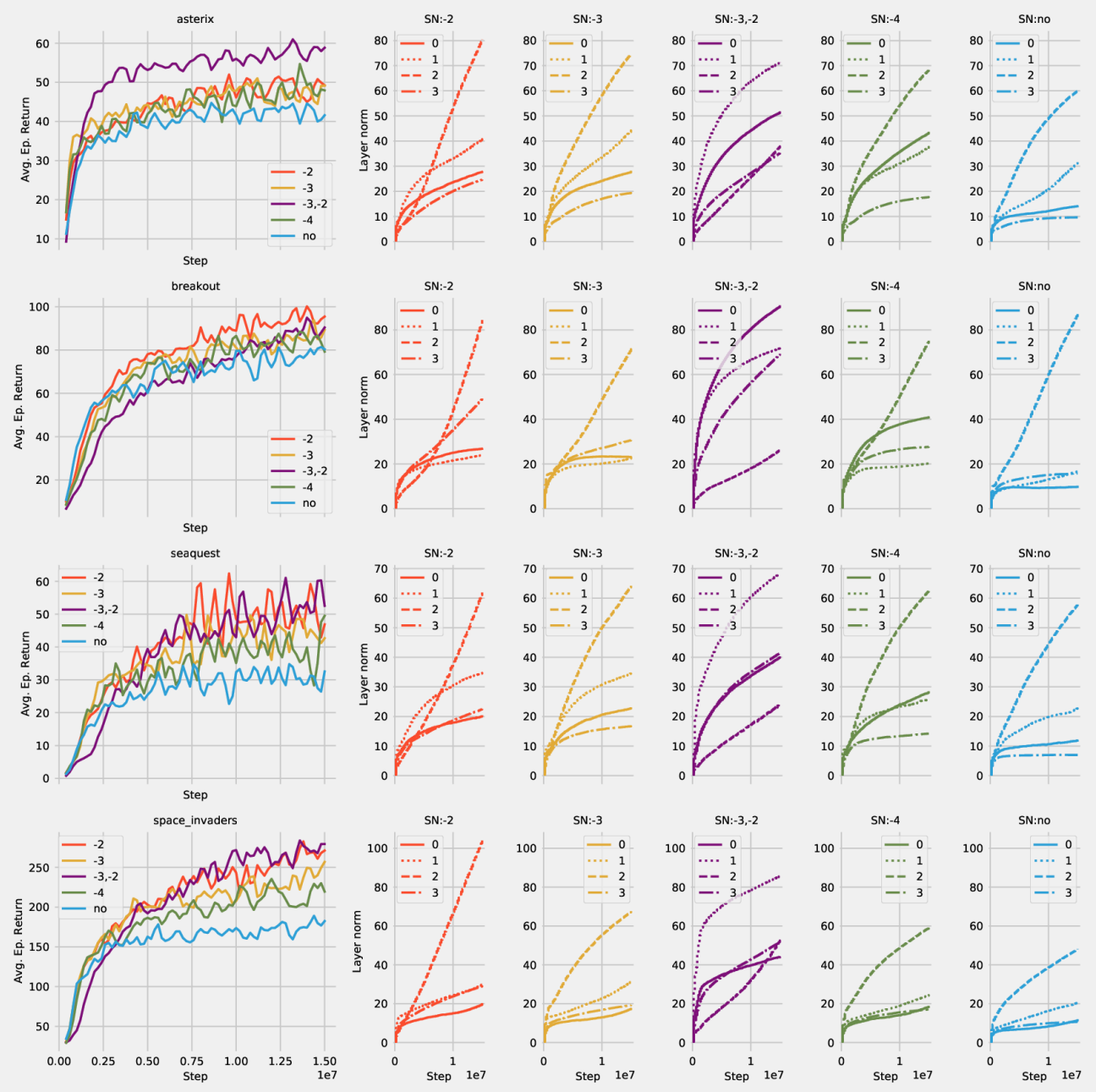

Figure 13: All spectral radii for the 15M experiment on MinAtar using a 4-layer architecture (conv=24-24,fc=128). Colors code the subsets of layers that are normalised (consistent with the rest of the document), while line styles code the four layers. Note how the penultimate layer has the largest spectral norm across all normalisation variants. 10 seeds.

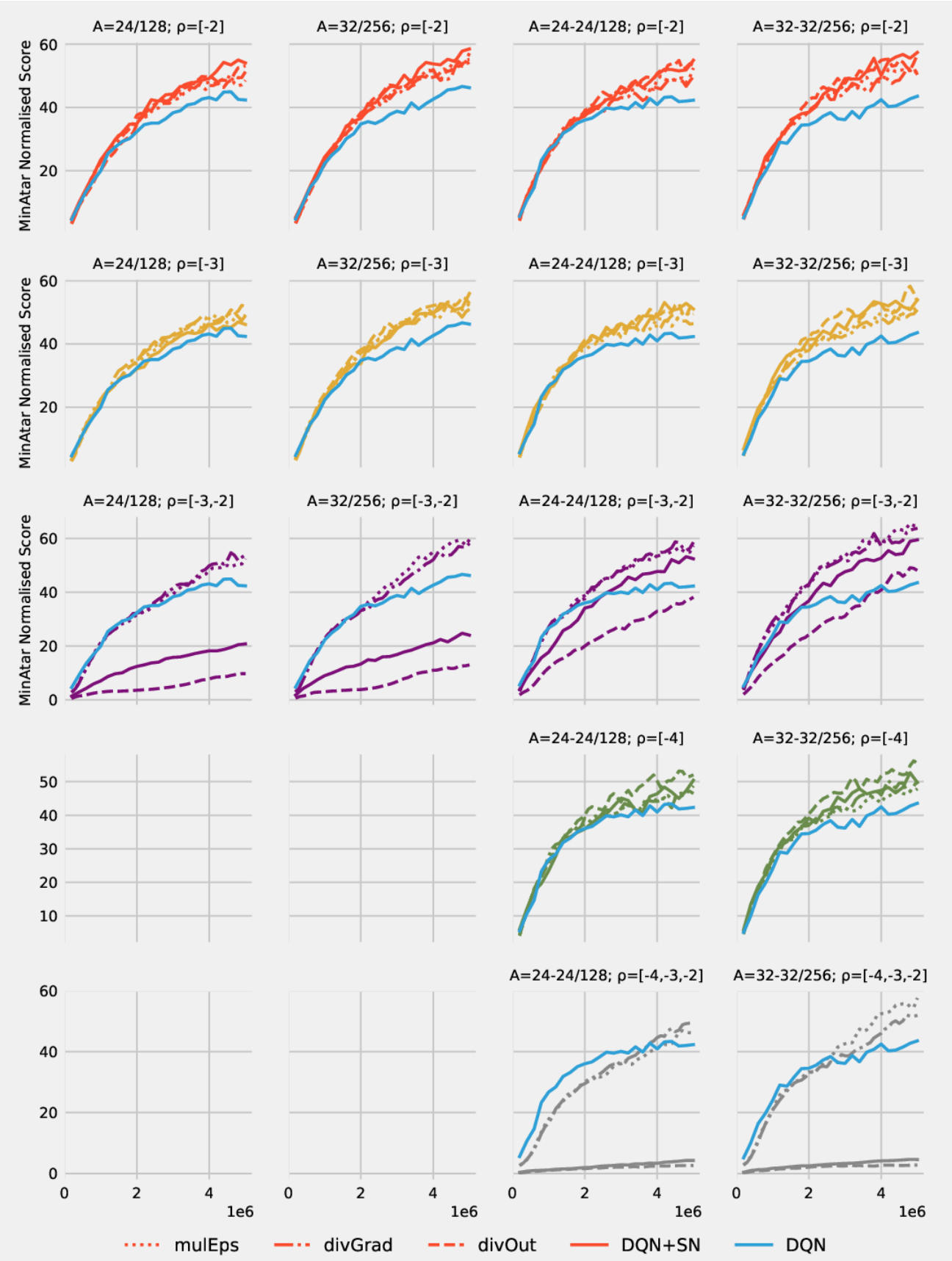

Figure 18: MinAtar Normalised Scored for the four architectures in Table 6 and various subsets of layers whose spectral radii are used for SN or spectral schedulers. Notice that DIVOUT behaves similarly to SN (even when they fail to train), while MULEPS and DIVGRAD converge even when all hidden layers are normalised.

C. Atari experiments

C.1. Evaluation protocols on Atari

In our work we mostly compare our Arcade Learning Environment (ALE) results with the RAINBOW agent, therefore we adopt the evaluation protocol from (Hessel et al., 2018). Every 250K training steps in the environment we suspend the learning and evaluate the agent on 125K steps (or 500K frames). All the agents we train on ALE follow this validation protocol, the only difference being the validation epsilon value: $\epsilon = 0.001$ for C51 and DQN-Adam which we directly compare to RAINBOW and uses the same value and $\epsilon = 0.05$ for DQN-RMSProp which follows the exact same hyper-parameters from (Mnih et al., 2015).

A major difference between the RAINBOW protocol and the *null op starts* protocol used in earlier works is that in previous works the agent is evaluated for 30 or 100 (Van Hasselt et al., 2016) episodes and is allowed to play up to $18,000$ frames (5 minutes of emulator time) or by the end of the episode, whatever came first, whereas we always evaluate for up to $500,000$ frames.

Episodes are limited at 108K steps, the agent receives a game over signal when losing a life as in previous works and we use the *null op starts* to induce stochasticity in ALE games both at training and evaluation time (Van Hasselt et al., 2016).

C.2. DQN-Adam

Next, we wanted to showcase SN on an algorithm with a simpler objective such as DQN. However our initial experiments on MinAtar suggested that SN has a greater impact on Adam than on the RMSProp optimiser used in DQN (Mnih et al., 2015). Since we also wanted to be able to compare our results with those of RAINBOW we use similar hyper-parameters to those in (Hessel et al., 2018). We list the full details in Table 7 especially since these hyper-parameters differ considerably from the the original DQN agent.

HYPER-PARAMETER	VALUE
discount γ	0.99
update frequency	4
target update frequency	8000
starting ϵ	1.0
final ϵ	0.01
ϵ steps	250000
ϵ schedule	linear
warmup steps	20000
replay size	1M
batch size	32
history length	4
cost function	SMOOTHL1LOSS
optimiser	ADAM
learning rate η	0.00025
damping term ϵ	0.0003125
β_1, β_2	(0.9, 0.999)
validation steps	125000
validation ϵ	0.001

Table 7: DQN-Adam hyper-parameters.

AGENT	MEAN	MEDIAN
DQN*	357.36	102.94
DQN SN[-2]	375.37	105.19
DQN (Wang et al., 2016)	216.84	78.37
DQN-ADAM*	358.45	119.45
DQN-ADAM SN[-2]	**719.95**	**178.18**

Table 8: Mean and median Human Normalised Score on 54 Atari games with random starts evaluation. References indicate the sources for the scores for each algorithm. We mark our own implementations of the baseline with $*$. Our agents are evaluated with the protocol in (Hessel et al., 2018). Note that the scores we report for our own implementation of DQN baseline are different from those reported in (Mnih et al., 2015) because the evaluation protocol has changed.

C.3. DQN - RMSprop

We also applied SN to a DQN agent optimised with RMSprop as in the original (Mnih et al., 2015). In conjunction with RMSprop the impact of SN seems minimal, far from the impressive improvement observed for the DQN-Adam agent. We leave for future work explaining the interaction between normalisation and RMSProp. See Table 8 for comparing the Human Normalised Score of DQN-RMSprop to other agents, and Fig. 22 for individual plots per game.

C.4. Effective rank

Authors of (Kumar et al., 2020) is making the case that for TD-learning with function approximation trained with SGD the neural network is being implicitly under-parametrised early in training. Empirically they show this by looking at the *effective rank* of the feature matrix $\mathbf{\Phi}$ which they approximate with the number of first k singular values of $\mathbf{\Phi}$ that capture 99% variance of all the singular values: $\sum_i^k \sigma_i(\mathbf{\Phi}) \ / \ \sum_j^d \sigma_j(\mathbf{\Phi}) \geq 0.99$. In this case the feature matrix $\mathbf{\Phi}$ is the input to the last linear layer in our neural network.

We perform their experiment, this time with a C51 agent with and without normalised layers looking to better understand the regularisation effects of normalisation. Figure 21 shows an evolution of the effective rank for the baseline agent that is consistent with the report of (Kumar et al., 2020). Interestingly, the baseline agent is consistently the one making use of fewer and fewer dimensions in the feature

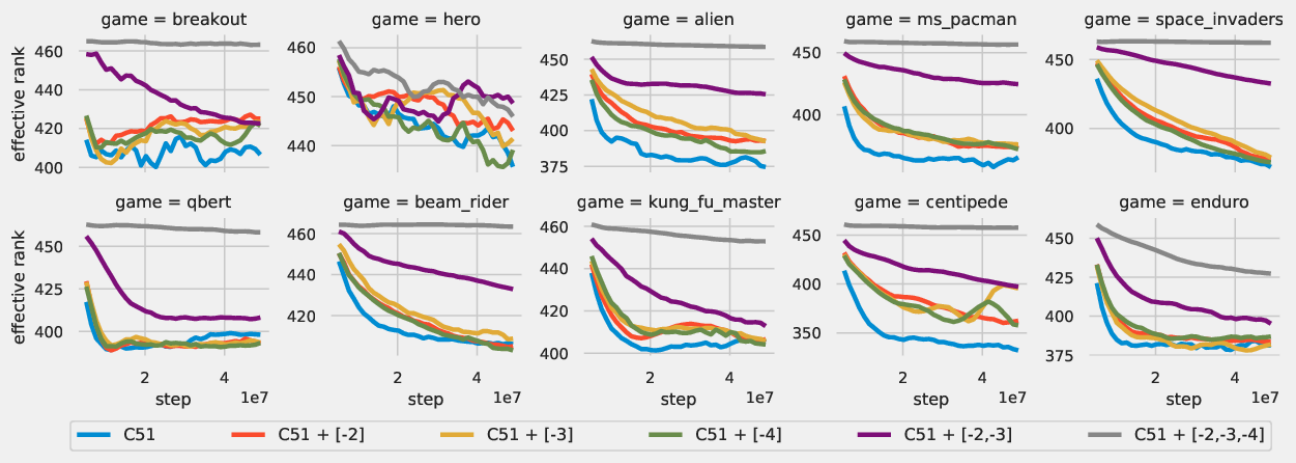

Figure 21: **Spectral Normalisation preserves the Effective Rank of the features.** Evolution of the effective rank of the features before the last linear layer of an C51 agent trained on 10 Atari games.

space as training progresses while the normalised agents preserve the rank. We further corroborate this finding with that of (Miyato et al., 2018) which is arguing that one of the possible disadvantages of Weight Normalisation (WN) as opposed to SN is that it prematurely producing sparse representations. Our experiments shows further shows that SN helps with preserving the rank of the features early on in training even when compared to an un-normalised agent.

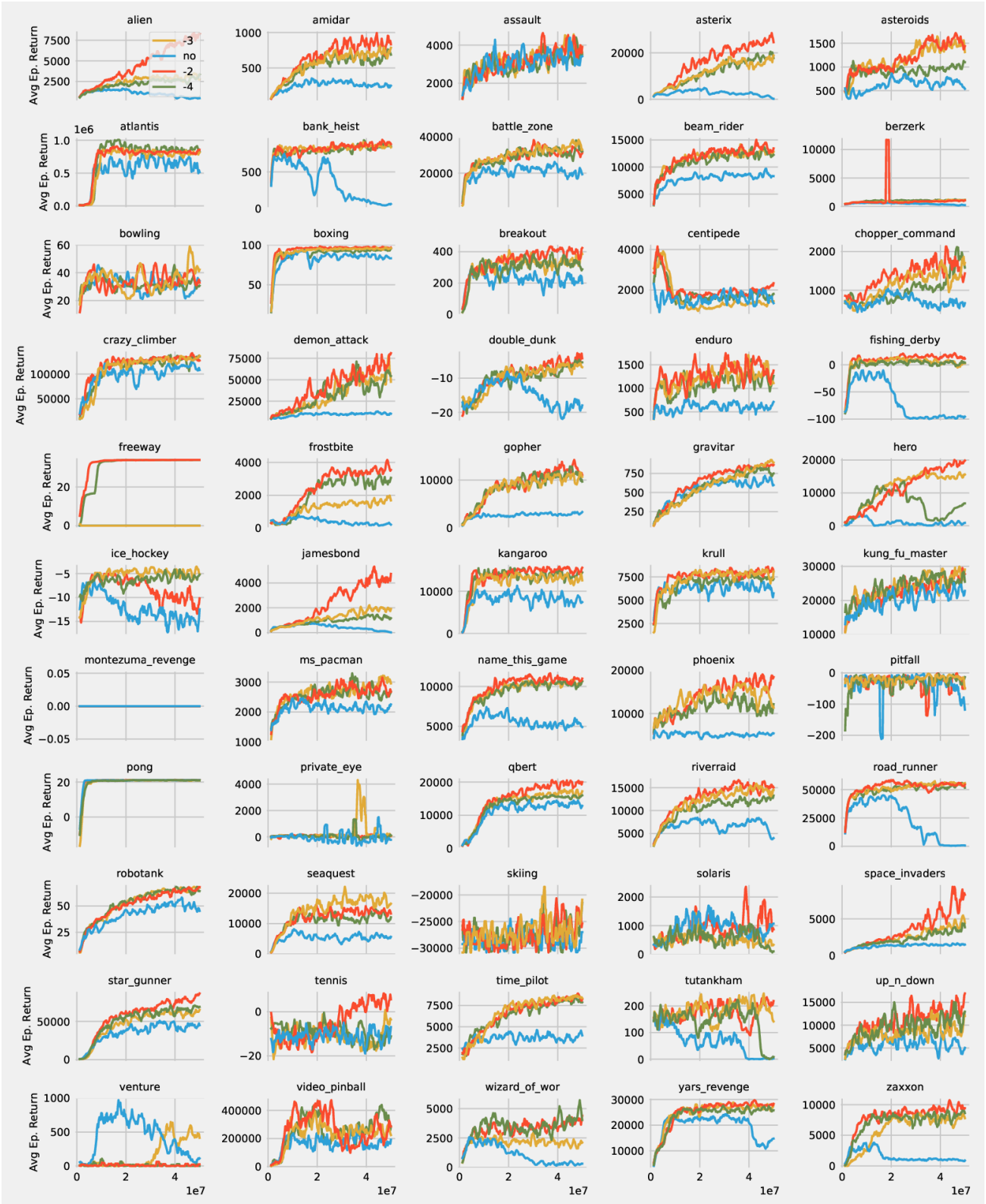

Figure 19: Performance curves of a DQN baseline optimised with Adam with SN applied on three different layers.

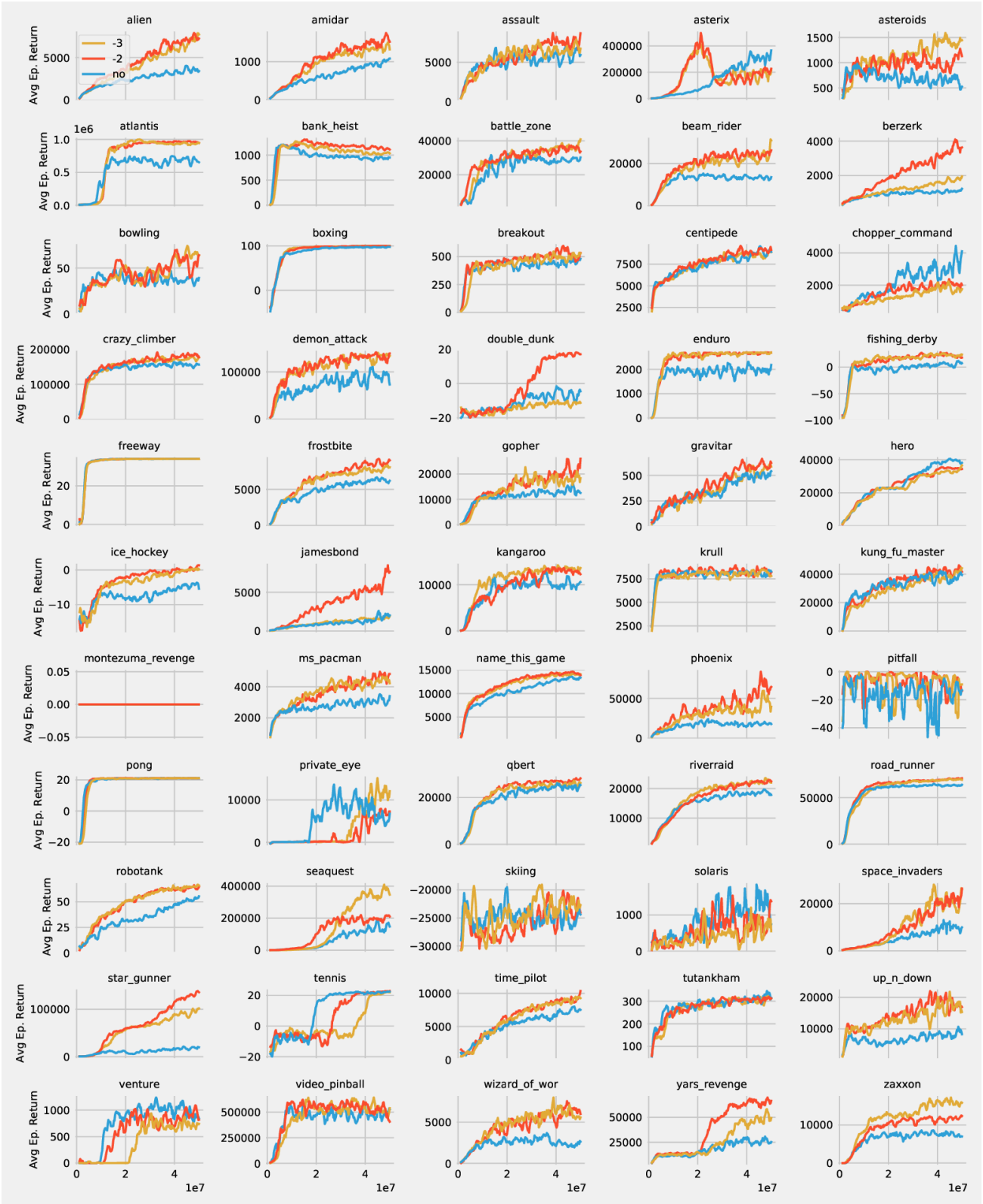

Figure 20: Performance curves of a C51 baseline with SN applied on three different layers.

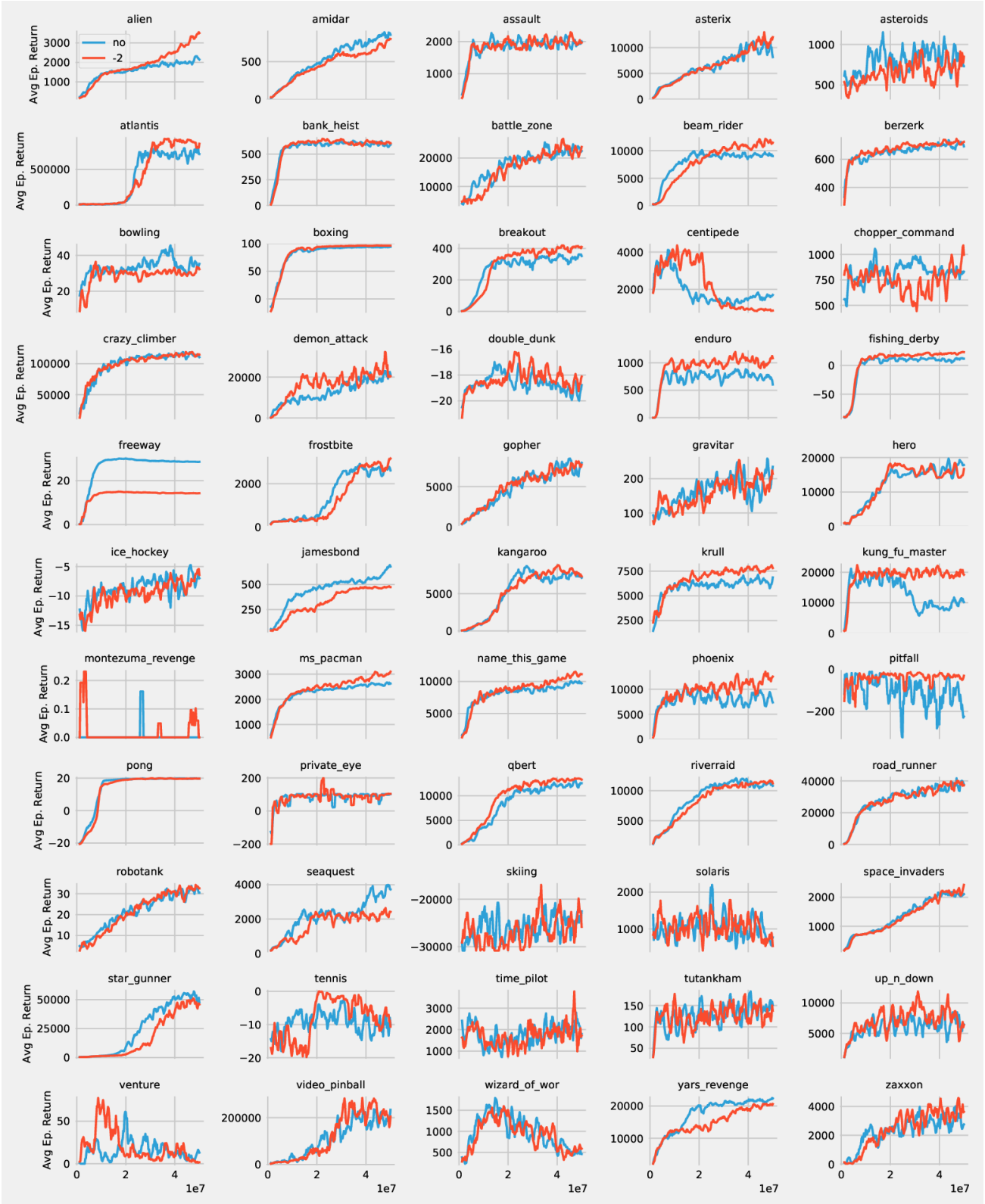

Figure 22: Performance curves of a DQN baseline optimised with RMSprop as in (Mnih et al., 2015) with SN applied on the penultimate layer.