A APPENDIX

As mentioned in the main body, the appendix contains additional materials and supporting information for the following aspects: rational activation functions improving plasticity (4), comparison of rational and rigid networks with different sizes on supervised learning experiments (A.1), results on replacing residual blocks with rational activation functions (A.2), every final and maximal scores obtained by the reinforcement learning agents used in our experiments (A.3), the evolutions of these scores (A.4), the different environment types with illustrations of their changes (A.5), graphs of the learned rational activation functions (A.6) and technical details for reproducibility (A.8).

Rational functions improve plasticity

To prove that rational can help with plasiticy, we tested them in continual learning settings (with more abrupt distribution shifts). We included Concatenated RELU and rational functions to an existing implementation of continual AI[3], in which 4 layers (2 convolutional ones and to fully connected ones) networks are trained on MNIST. The network then continues training on PERM.1, a variation of the dataset, for which a fixed random permutation is applied to every image. Another permutation is used for PERM. 2, used after the training on PERM1. As shown in Fig. 6, networks with rationals are both better at modelling the new data (higher accuracies on the currently trained data), but are also able to retain more information about the data previously trained on. Networks with Continual ReLU (Shang et al., 2016) better retain information on Task 1, while performing on par with ReLU ones for the 2 other tasks.

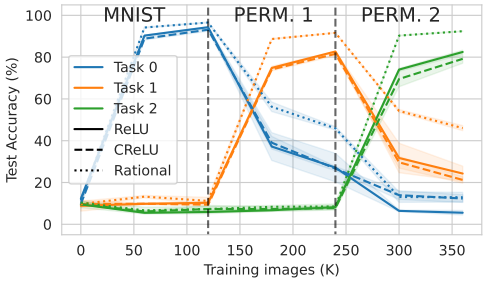


Figure 6: **Rational function improve plasticity on the permutted MNIST experiment.** Rational networks obtain better accuracies on each currently and previously trained datasets.

A.1 RATIONAL EFFICIENT PLASTICITY CAN REPLACE LAYER'S WEIGHT PLASTICITY

We here show that networks with rational activations not only outperform Leaky ReLU ones with the same amount of parameters, but also to outperform deeper and more heavily parametrised neural networks (indicated by the colours). For example, a rational activated VGG4 not only performs better than a rigid Leaky ReLU VGG4 at 1.37M parameters, but even performs similarly to the 4.71M parameters rigid VGG6. Activation's plasticity allowing to reduce the number of layers weights is also shown by the experiments summarized in Tab. 2 in the next section, where blocks from a pretrained ResNet are replaced by a rational function, and the resulting networks are able to recover and surpass their accuracies.

Architecture		VGG4		VGG6		VGG8	
Activation function		LReLU	Rational	LReLU	Rational	LReLU	Rational
CIFAR 10	Training Acc@1	83.0±.3	87.1±.6	86.9±.2	89.2±.2	90.1±.1	92.4±.2
	Testing Acc@1	80.0±1.	84.3±.5	83.1±.6	85.4±.6	85.0±1.	86.9±.3
CIFAR 100	Training Acc@1	64.6±.8	70.4±.9	70.7±.6	86.0±.9	87.7±.2	87.8±.1
	Testing Acc@1	56.5±.9	58.9±.6	59.0±.5	59.9±.9	60.0±.9	59.9±.4
# Network parameters		1.37M		4.71M		9.27M	

Table 1: Shallow rational networks perform as deeper Leaky ReLU ones. VGG networks training and testing top-1 accuracies with different numbers of layers are evaluated on CIFAR10 and CIFAR100. Rational VGG4 has similar performances as VGG6 network, with 3.5 times less parameters, and Rational VGG6 outperforms VGG8, with two times less parameters. Shaded colour pairs included for emphasis.

[3]https://github.com/ContinualAI/colab/blob/master/notebooks/permuted_and_split_mnist.ipynb

A.2 RESIDUAL BLOCK LEARN OF DEEP RESNET LEARN ACTIVATION FUNCTION-LIKE BEHAVIOUR.

We present in this section lesioning experiments, where a residual block is lesioned from a pretrained Residual Network, and the surrounding blocks are fine-tuned (with a learning rate of 0.001) for 15 epochs. These lesioning experiments were first conducted by Veit et al. (2016). We also perform rational lesioning, where we replace a block by an (identity initialised)[4] rational activation function (instead of removing the block), and train the activation function along with the surrounding blocks. The used rational functions have the same order as in every other experiment ($(m, n) = (5, 4)$), that satisfies the rational residual property derive in the paper). We report recovery percentages, computed following:

$$\text{recovery} = 100 \times \frac{\text{finetuned} - \text{surgered}}{\text{original} - \text{surgered}}. \tag{1}$$

We also provide the amount of dropped parameters of each lesioning.

Table 2: Rational functions improve lesioning. The recovery percentages for finetuned networks after lesioning (Veit et al., 2016) of a ResNet layer's (L) block (B) are shown. Residual blocks were lesioned, *i.e.* replaced with the identity (Base) or a rational from a pretrained ResNet101 (44M parameters). Then, the surrounding blocks (and implanted rational activation function) are retrained for 15 epochs. Larger percentages are better, best results are in **bold**.

Recovery (%)	Lesioning	L2B3	L3B19	L3B22	L4B2
Training	Original (Veit et al., 2016)	100.9	90.5	100	58.9
	Rational (ours)	**101.1**	**104**	**120**	**91.1**
Testing	Original (Veit et al., 2016)	**93.1**	97.1	81.6	81.7
	Rational (ours)	90.5	**97.6**	**91.5**	**85.3**
% dropped params		0.63	2.51	2.51	10.0

As the goal is to show that flexible rational functions can achieve similar modelling capacities to the residual blocks, we did not apply regularisation methods and mainly focused on training accuracies. We can clearly observe that rational activation functions lead to performance improvements that even surpass the original model, or are able to maintain performances when the amount of dropped parameters rises.

A.3 COMPLETE SCORES TABLE FOR DEEP REINFORCEMENT LEARNING

Through this work, we showed the performance superiority of reinforcement learning agents that embed additional plasticity provided by learnable rational activation functions. We used human normalised scores (*cf.* Eq. 2) for readability. For completeness, we provide in this section the final raw scores of every trained agent. As many papers provide the maximum obtained score among every epoch and every agent, even if we consider it to be an inaccurate and noisy indicator of the performances, for which random actions can still be taken (because of ϵ-greedy strategy also being used in evaluation). A fairer indicator to compare methods is the mean score. We thus also provide final mean scores (of agents retrained among 5 seeded reruns) with standard deviation. We start off by providing the human scores used for normalisation (provided by van Hasselt et al., in Table 5), then provide final mean and maximum obtained raw scores of every agent.

[4] all weights are initially set to 0 but a_1 (and b_0), both set to 1.

Algorithm	DQN			DDQN	DQN with Plasticity		
Activation	LReLU	SiLU	d+SiLU	LReLU	PELU	rational	joint-rational
Asterix	1.85±1.2	0.52±0.6	2.14±1.4	48.9±17.7	25.8±3.7	**242**±23.5	168±32.6●
Battlezone	11.4±7.0	21.2±15.0	11.3±6.7	68.2±34.8	46.6±19.5	70.1±2.1●	**77.4**±8.7
Breakout	558±166	93.9±57.6	11.7±14.0	286±122	788±79.2	1134±130●	**1210**±36.0
Enduro	16.3±21.3	37.0±17.7	0.37±0.5	47.7±18.1	24.5±42.6	**141**±15.0	129±14.7●
Jamesbond	8.62±6.4	6.08±3.7	5.28±4.4	10.7±11.1	74.2±51.5	308±48.5●	**312**±59.5
Kangaroo	11.8±12.5	128±95.6●	13.9±18.5	17.2±14.5	57.7±14.6	107±43.1	**193**±86.8
Pong	101±5.5	96.1±12.0	104±3.3	91.3±30.8	106.4±2.2	107.0±2.4●	**107.3**±2.7
Qbert	55.4±17.1	14.2±17.0	2.74±0.2	74.0±21.7	101±6.6	**120**±2.8	117±4.9●
Seaquest	0.57±0.4	3.67±4.1	0.18±0.2	2.17±0.9	9.21±2.5	16.3±0.5●	**18.4**±3.3
Skiing	-90.7±37.9	-111±-0.7	-85.5±43.4	-86.9±46.6	-111±-.7	**-59.5**±60.7	-60.2±56.1●
Space Inv.	33.9±4.3	33.1±11.9	32.4±12.4	31.0±1.0	50.1±3.3●	42.3±3.1	**95.1**±17.7
Tennis	8.94±17.3	26.3±53.3	78.5±64.3	32.1±51.6	106±53.3	257.8±2.8●	**258.3**±5.2
Timepilot	14.9±14.3	19.3±31.0	18.3±38.1	6.61±7.5	124±26.1	**341**±105	253±11.0●
Tutankham	0.03±2.8	58.2±48.6	2.89±4.0	24.4±-0.4	91.6±29.3	130±10.7●	**134**±29.3
Videopinball	440±123	55.8±61.9	-4.03±32.5	626±241	299±168	**1616**±1026	906±539●
# Wins	0/15	0/15	0/15	0/15	0/15	6/15	9/15
# Super-Human	3/15	1/15	1/15	2/15	6/15	**11/15**	**11/15**

Table 3: Neural plasticity leads to vast performance improvements. Normalised mean scores and standard deviations (in percentage, *cf.* Appendix A.8 for the equation) of rigid baselines (*i.e.* DQN and DDQN with Leaky ReLU, DQN with SiLU and SiLU + dSiLU), as well as DQN with plasticity: using PELU, rational (full) and joint-rational (regularised), are reported over five experimental random seeded repetitions (larger mean values are better). The best results are highlighted in **bold** and runner-ups denoted with • markers. The last rows summarise the number of times best mean scores were obtained by each agent and the number of super-human performances.

Final mean and maximum obtained scores of Rainbow agents:

Evaluation	**Final Mean Scores**			Max. Obtained Scores		
Plasticity	rigid	full	regularised	rigid	full	regularised
Breakout	52	279	**303**	383	**569**	**569**
Enduro	844	**1473**	1470	1388	**1973**	1964
Kangaroo	40	**2157**	2139	**6300**	6000	4800
Q*bert	149	**11931**	11551	16125	**23550**	**23550**
Seaquest	82	247	**282**	920	**1280**	**1280**
Space Inv.	595	**1263**	1157	2070	**3395**	2875
Time Pilot	3926	5386	**6411**	12700	**15900**	**15900**

Table 4: Final mean and maximum obtained scores obtained by rigid Rainbow agents (*i.e.* using Leaky ReLU), as well as Rainbow with full (*i.e.* using rational activation functions) and regularised (*i.e.* using joint-rational ones) plasticity (only 1 run because of computational cost, larger values are better).

Final mean scores of all agents:

Algorithm	Random	DQN			DDQN	DQN with Plasticity		
Network type	-	LReLU	SiLU	d+SiLU	LReLU	PELU	full	regularised
Asterix	67.9±2.2	206±90	107±45	228±108	3723±1324	1998±275	**18109±1755**	12621±2436
Battlezone	788±38	4464±2291	7612±4877	4429±2183	22775±11265	15807±6320	23403±701	**25749±2837**
Breakout	0.14±01	155±46	26.2±16	3.4±3.89	79.4±33.8	219±22	315±36	**336±10**
Enduro	0±0	121±158	274±131	2.77±3.41	353±134	181±315	**1043±111**	957±109
Jamesbond	6.39±0.41	37.6±23.6	28.4±13.8	25.5±16.2	45.2±40.7	275±187	1122±176	**1137±216**
Kangaroo	14.2±0.9	335±342	3500±2607	393±504	484±395	1586±398	2940±1175	**5266±2365**
Pong	-20.2±0	15.9±2	14.1±4.3	16.9±1.2	12.4±11	17.8±0.8	18±0.9	**18.1±1**
Q*bert	40.6±2.8	6715±2058	1754±2048	371±28	8954±2616	12143±795	**14436±336**	14080±593
Seaquest	20.1±0.4	250±162	1504±1677	94.6±87.2	898±353	3740±991	6603±200	**7461±1321**
Skiing	-16104±92	-27365±4794	-29890±4	-26725±5485	-26892±5881	-29912±10	**-23487±7624**	-23582±7058
Space Inv.	51.6±1.1	531±62	520±169	509±176	490±15	759±48	650±45	**1395±251**
Tennis	-23.9±0.0	-22.4±3.0	-19.4±9.2	-10.4±11.1	-18.4±8.9	-5.6±9.2	20.5±0.5	**20.6±0.9**
TimePilot	688±30	1428±739	1644±1566	1594±1918	1016±401	6818±1323	**17632±5242**	13261±576
Tutankham	3.51±0.54	3.55±4.3	81.9±66	7.41±5.96	36.4±0	127±40	179±15	**184±40**
VideoPinb.	6795±461	45683±11383	11730±5941	6439±3336	62151±21791	42051±15356	**149712±91219**	86942±48143

Table 5: Final mean raw scores (with std. dev.) of rigid baselines (*i.e.* DQN and DDQN with Leaky ReLU, DQN with SiLU and SiLU + dSiLU), as well as DQN with full plasticity (*i.e.* using rational activation functions) and regularised plasticity (*i.e.* using joint-rational ones) on Atari 2600 games, averaged over 5 seeded reruns (larger mean values are better).

Maximum obtained scores:

Algorithm	Random	DQN			DDQN	DQN with Plasticity		
Network type	-	LReLU	SiLU	d+SiLU	LReLU	PELU	full	regularised
Asterix	71	9250	3400	3800	20150	9300	84950	49700
Battlezone	843	88000	81000	70000	97000	68000	78000	94000
Breakout	0	427	370	344	411	430	864	864
Enduro	0	1243	928	1041	1067	1699	1946	1927
Jamesbond	6	5600	5750	700	7500	6150	9250	13300
Kangaroo	15	14800	15600	10200	13000	12400	16200	16800
Pong	-20	21	21	21	21	21	21	21
Q*bert	45	19425	11700	5625	19200	18900	24325	25075
Seaquest	20	7440	8300	740	15830	14860	9100	26990
Skiing	-15997	-5987	-6505	-6267	-5359	-5495	-5368	-5612
Space Inv.	53	2435	2205	2460	2290	2030	2490	3790
Tennis	-23	8	1	-1	4	-1	24	36
Time Pilot	730	11900	15500	12500	12200	16300	72000	28000
Tutankham	4	249	267	267	274	397	334	309
VideoPinb.	7599	998535	950250	338512	991669	322655	997952	998324

Table 6: Maximum obtained scores (with std. dev.) of rigid baselines (*i.e.* DQN and DDQN with Leaky ReLU, DQN with SiLU and SiLU + dSiLU), as well as DQN with full plasticity (*i.e.* using rational activation functions) and regularised plasticity (*i.e.* using joint-rational ones) on Atari 2600 games, averaged over 5 seeded reruns (larger values are better).

Human scores used for normalisation:
Asterix: 7536, Battlezone: 33030, Breakout: 27.9, Enduro: 740.2, Jamesbond: 368.5, Kangaroo: 2739, Pong: 15.5, Q*bert: 12085, Seaquest: 40425.8, Skiing: -3686.6, Space Invaders: 1464.9, Tennis: -6.7, Time Pilot: 5650, Tutankham: 138.3, Video Pinball: 15641.1

A.4 EVOLUTION OF THE SCORES ON EVERY GAME

The main part present some graphs that compares performance evolutions of the Rainbow and DQN agents with plasticity, as well as Rigid DQN, DDQN and Rainbow agents. We here provide the evolution of the scores of every tested DQN and the DDQN agents on the complete game set. DQN agents with higher plasticity are always the best-performing ones. Experiments on several games (e.g. Jamesbond, Seaquest) show that using DDQN does not prevent the performance drop but only delays it.

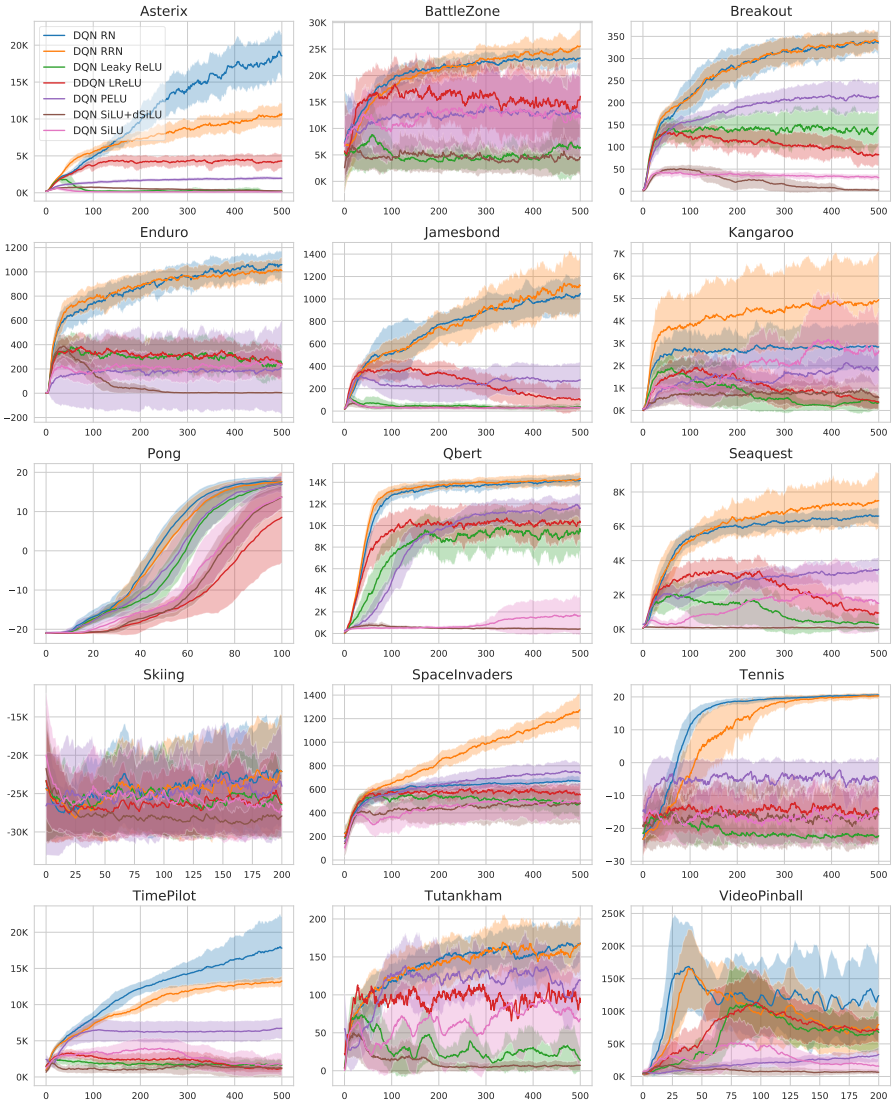


Figure 7: Smoothed (*cf.* Eq. 3) evolutions of the scores on every tested game for DQN agents with full (*i.e.* using rational activation functions) and regularised (*i.e.* using joint-rational ones) plasticity, and original DQN agents using Leaky ReLU, SiLU and SiLU+dSiLU, as well as for DDQN agents with Leaky ReLU.

A.5 ENVIRONMENTS TYPES: STATIONARY, DYNAMICS AND PROGRESSIVE

The used environments have been separated in 3 categories, describing their potential changes through agents learning. This categorisation is here illustrated with frames of the tested games. As one can see: Breakout, Kangaroo, Pong, Skiing, Space Invaders, Tennis, Tutankham and VideoPinball can be categorised as **stationary environment**, as changes are minimal for the agents in these games. Asterix, BattleZone, Q*bert and Enduro present environment changes, that are early reached by the playing agents, and are thus **dynamic environments**. Finally, Jamesbond, Seaquest and Time Pilot correspond to **progressive environments**, as the agents needs to master early changes to access new parts of these environments.

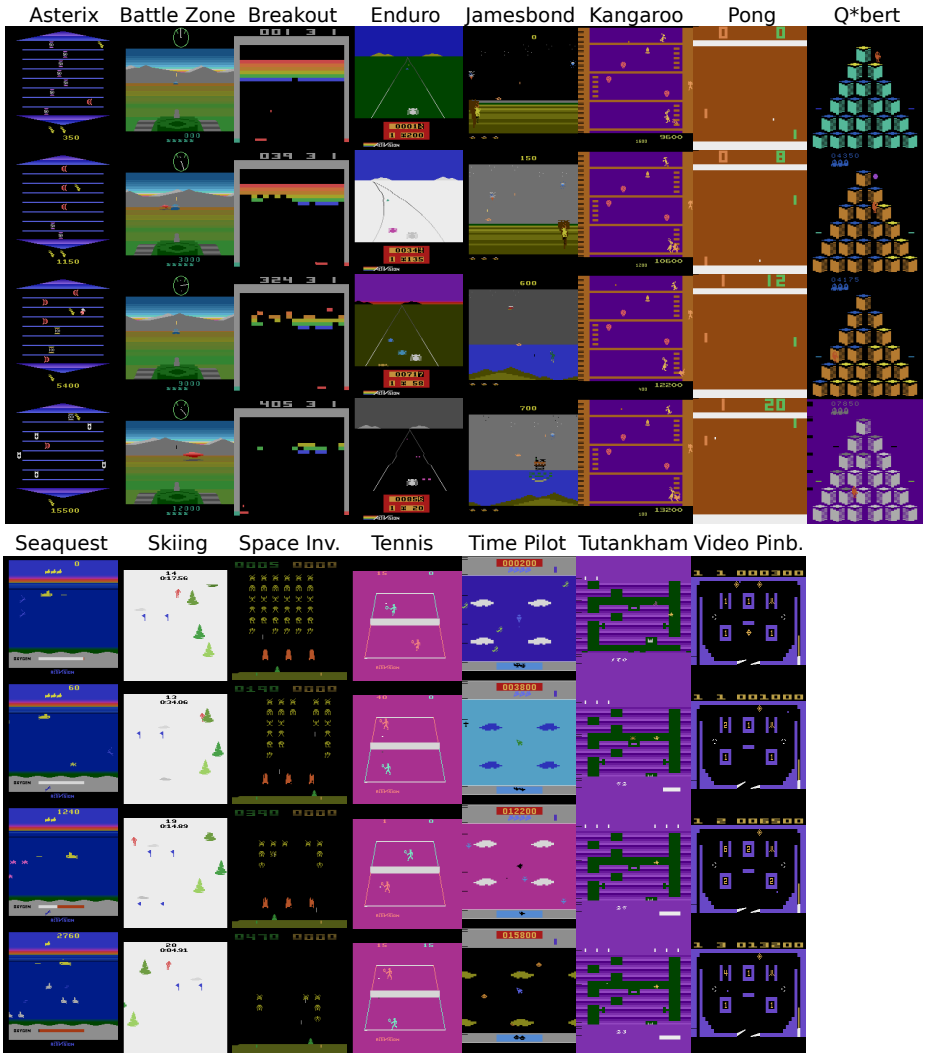
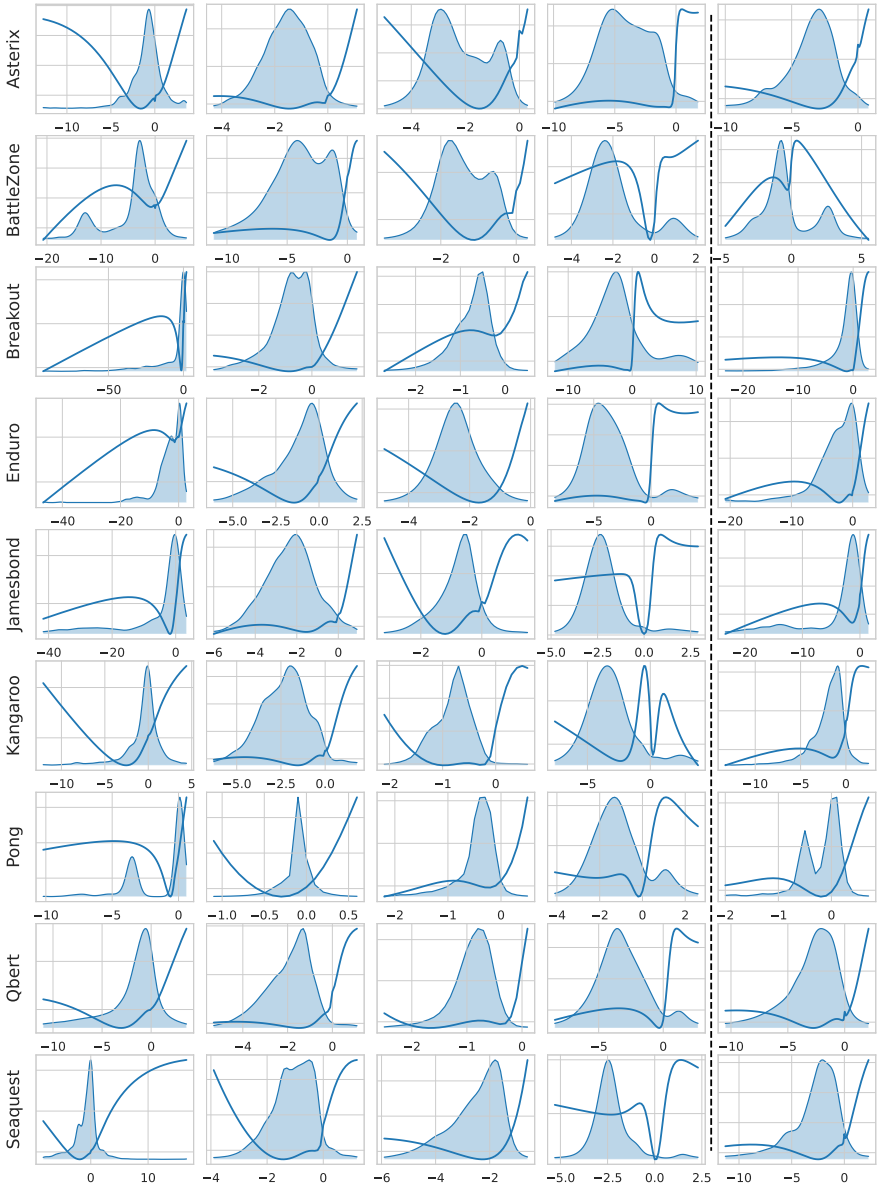


Figure 8: Images extracted from DQN agents with full plasticity playing the set of 15 Atari 2600 games used in this paper. Stationary environments (e.g. Pong, Video Pinball) do not evolve during training, dynamic ones provide different input/output distributions that are early accessible in the game (e.g Q*bert, Enduro) and progressive ones (e.g. Jamesbond, Time Pilot) require the agent to improve for the it to evolve.

A.6 LEARNED RATIONAL ACTIVATION FUNCTIONS

We have explained in the main text how rational functions of agents used on different games can exhibit different complexities. This section provides the learned parametric rational functions learned by DQN agents with full plasticity (left) and by those with regularised plasticity (right) after convergence for every different tested game of the gym Atari 2600 environment. Kernel Density Estimations (with Gaussian kernels) of input distributions indicates where the functions are most activated. Rational functions from agents trained on simpler games (*e.g.* Enduro, Pong, Q*bert) have simpler profiles (*i.e.* fewer distinct extremas).



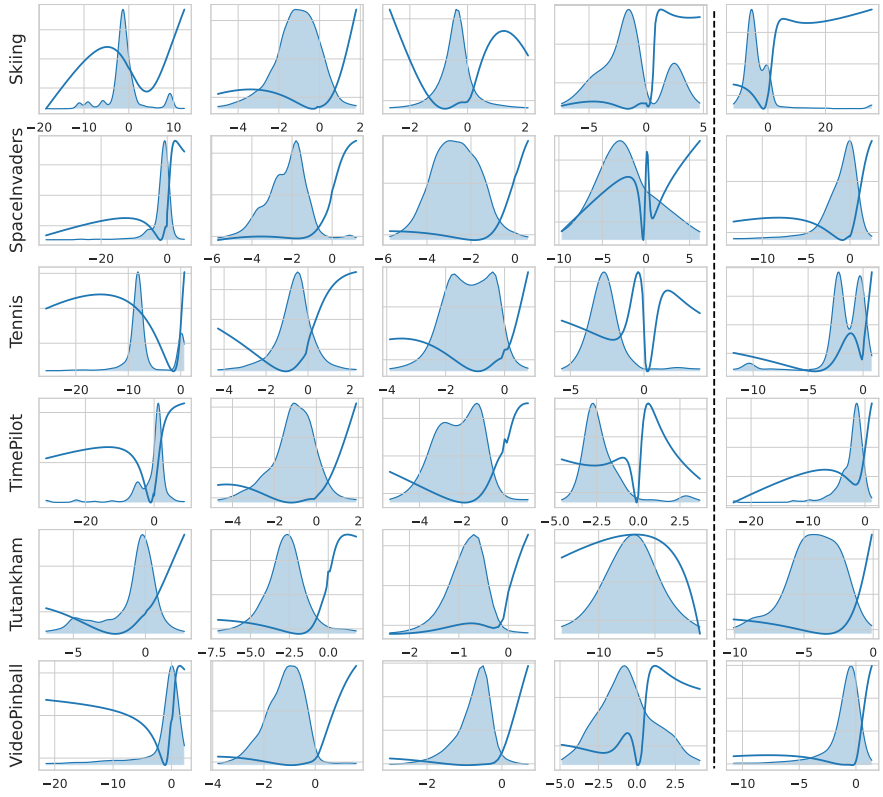


Figure 9: Profiles (dark blue) and input distributions (light blue) of rational functions (left) and joint-rational ones (right) of DQN agents on the different tested games. (Joint-)rational functions from agents of simpler games have simpler profiles (*i.e.* fewer distinct extrema).

A.7 EVOLUTION OF RATIONALS ON THE PERM-MNIST CONTINUAL LEARNING EXPERIMENT

Figure 10 depicts the evolutions of rational functions through the permuted MNIST experiment. One can see that while the function of the first layer remains stable through the successive datasets, the second one flatten at its most activated region (around 0), while the third one increase its slope in this region, leading to higher gradients. This suggests that rational functions can help adapting the gradient scales at each layer. Further investigating this is an interesting line of future work.

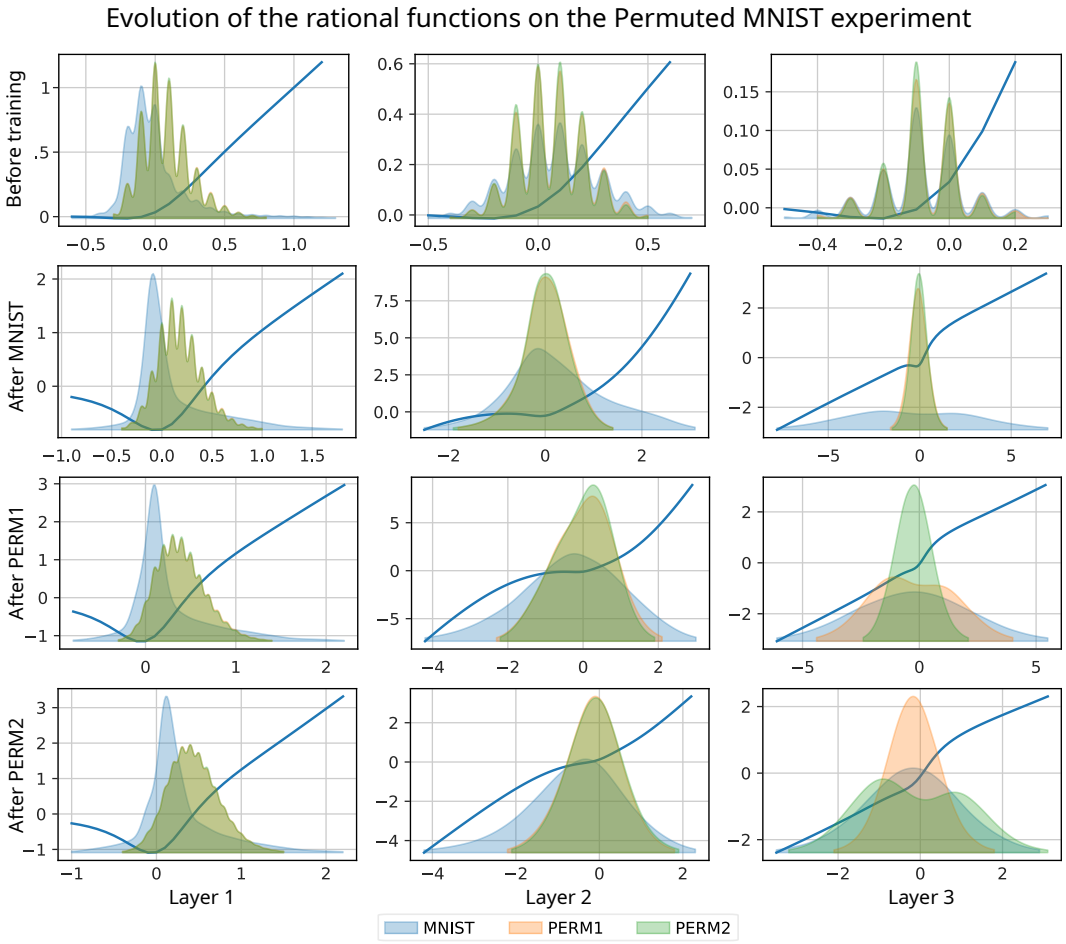


Figure 10: Evolution of the rational activation functions on the permuted MNIST experiment (*cf.* 4). The 3 rational activation functions used for training (and retraining) are adapting to fit the data (depicted in semi transparent).

A.8 TECHNICAL DETAILS TO REPRODUCE THE EXPERIMENTS

We here provide details on our experiments for reproducibility. **We used the seed 0, 1, 2, 3, 4 for every multi-seed experiment.**

SUPERVISED LEARNING EXPERIMENTS

For the lesioning experiment, we used an available[5] pretrained Residual Network. We then remove the corresponding block (and potentially replace it with an identity initialised rational activation function) (surgered). We finetune the new models, allowing for optimisation of the previous and next layers (and potentially the rational function) for 15 epochs with SGD (learning rate of 0.001).

[5]https://download.pytorch.org/models/resnet101-5d3b4d8f.pth

For the classification experiments, we run on CIFAR10 and CIFAR100 (Krizhevsky et al., MIT License), we let every network learn for 60 epochs. We use the code provided by Molina et al. (2020), with only one classification layer in these smaller VGG versions (VGG4, VGG6 and VGG8, against 3 for VGG16 and larger). We use SGD as the optimisation algorithm, with a learning rate of 0.02 and 128 as batch size. The VGG networks contain successive VGG blocks that all consist of n convolutional layers, i input channels and o output channels, stride 3 and padding 1, followed by an activation function, and 1 Max Pooling layer. For each used architecture, the (n, i, o) parameters of the successive blocks are:

- VGG4: $(1, 3, 64) \rightarrow (1, 64, 128) \rightarrow (2, 128, 256)$
- VGG6: $(1, 3, 64) \rightarrow (1, 64, 128) \rightarrow (2, 128, 256) \rightarrow (2, 256, 512)$
- VGG8: $(1, 3, 64) \rightarrow (1, 64, 128) \rightarrow (2, 128, 256) \rightarrow (2, 256, 512) \rightarrow (2, 512, 512)$

The output of these blocks is then passed on to a classifier (linear layer). Only activation functions differ between the Leaky ReLU and the Rational versions.

Reinforcement Learning Experiments

To ease the reproducibility of our the reinforcement learning experiments, we used the *Mushroom RL* library (D'Eramo et al., 2020) on the Arcade Learning Environment (GNU General Public License). We used states consisting of 4 consecutive grey-scaled images, downsampled to 84×84. Computing the gradients for rational functions takes longer than e.g. ReLU. However, we used a CUDA optimized implementation of the rational activation functions that we open source along with this paper. In practice, we did not notice any significant training time difference.

Network Architecture. The input to the network is thus a 84x84x4 tensor containing a rescaled, and gray-scaled, version of the last four frames. The first convolution layer convolves the input with 32 filters of size 8 (stride 4), the second layer has 64 layers of size 4 (stride 2), the final convolution layer has 64 filters of size 3 (stride 1). This is followed by a fully-connected hidden layer of 512 units.
All these layers are separated by the corresponding activation functions (either Leaky ReLU, SiLU, SiLU for convolution layers and dSiLU for linear ones, PELU, rational functions (at each layer) and joint-rational ones (shared accross layers) of order $m = 5$ and $n = 4$, initialised to approximate Leaky ReLU). We used the default PeLU initial hyperparameters (a=1, b=1, c=1) and let the weights optimizer tune them through training, as for rational functions. For CRELU, we took the implementation from ML Compiled [6], and halves the number of filters in the following convolutional layers to keep the same network structure intact, as done by Shang et al. (2016).

Hyper-parameters. We evaluate the agents every 250K steps, for 125K steps. The target network is updated every 10K steps, with a replay buffer memory of initial size 50K, and maximum size 500K, except for Pong, for which all these values are divided by 10. The discount factor γ is set to 0.99 and the learning rate is 0.00025. We do not select the best policy among seeds between epochs. We use the simple ϵ-greedy exploration policy, with the ϵ decreasing linearly from 1 to 0.1 over 1M steps, and an ϵ of 0.05 is used for testing.

The only difference from the evaluation of Mnih et al. (2015) and of van Hasselt et al. (2016) evaluation is the use of the Adam optimiser instead of RMSProp, for every evaluated agent.

Normalisation techniques. To compute human normalised scores, we used the following equation:

$$\text{score}_{\text{normalised}} = 100 \times \frac{\text{score}_{\text{ agent}} - \text{score}_{\text{ random}}}{\text{score}_{\text{ human}} - \text{score}_{\text{ random}}}, \tag{2}$$

For readability, the curves plotted in the Fig. 4 and Fig. 8 are smoothed following:

$$score_t = \alpha \times score_{t-1} + (1 - \alpha) \times scores_t, \tag{3}$$

with $\alpha = 0.9$.

Overestimation computation. We used the following formulae to compute relative overestimation.

$$\text{overestimation} = \frac{\text{Q-value} - R}{R} \tag{4}$$

[6] https://ml-compiled.readthedocs.io/en/latest/activations.html

RL NETWORK ARCHITECTURE

The DQN, DDQN and Rainbow agents networks architecture, rational plasticity (with rational activations functions at each layer) and of the regularized ones (with one joint-rational activation function shared across layers). For the other activation functions, the "Rat." blocks are replaced with Leaky ReLU, CReLU, SiLU, or PELU. For the d+SiLU networks, SiLU is used on the convolutional layers (*i.e.* first two), and dSiLU in the fully connected ones (*i.e.* last two).

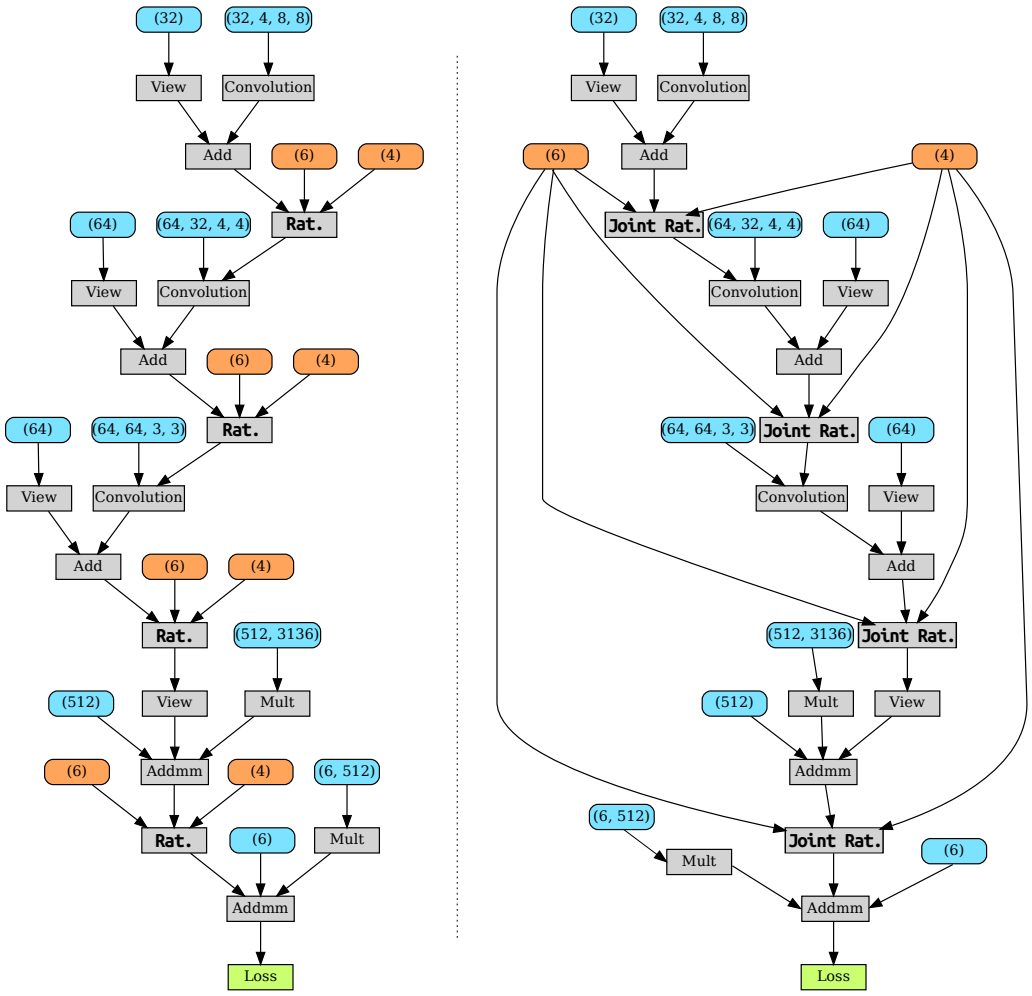


Figure 11: left: The DQN agents' neural network equipped with Rational Activation Functions (Rat.). Any other network with classical activation functions (as Leaky Relu or SiLU) would be similar, with the corresponding activation function instead of the rational one. right: The agents' network using the regularized joint-rational version of the network. The same activation is used across the layers. The parameters of the rational activation (in orange) function are shared. In both graphs, operations are placed in the grey boxes and parameters in the blue ones, (or orange for the rationals' ones).