References

Anschel, O., Baram, N., and Shimkin, N. Averaged-dqn: Variance reduction and stabilization for deep reinforcement learning. arXiv:1611.01929, 2017.

Bellemare, M., Dabney, W., and Munos, R. A distributional perspective on reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML-17)*, pp. 449458, 2017.

Bellemare, M. G., Naddaf, Y., Veness, J., and Bowling, M. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Int. Res.*, 47(1):253279, May 2013. ISSN 1076-9757.

Castro, P. S., Moitra, S., Gelada, C., Kumar, S., and Bellemare, M. G. Dopamine: A research framework for deep reinforcement learning. arXiv:1812.06110 [cs.LG], 2018.

Conti, E., Madhavan, V., Such, F. P., Lehman, J., Stanley, K. O., and Clune, J. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. arXiv:1712.06560, 2018.

Ernst, D., Geurts, P., and Wehenkel, L. Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6:503–556, 2005.

Faußer, S. and Schwenker, F. Neural network ensembles in reinforcement learning. *Neural Processing Letters*, pp. 5569, 2015.

Gordon, G. *Approximation Solutions to Markov Decision Problems*. PhD thesis, Carnegie Mellon University, 1999.

Gordon, G. J. Stable function approximation in dynamic programming. In *Proceedings of the Twelfth International Conference on Machine Learning (ICML-95)*, pp. 261–268, Lake Tahoe, 1995.

Guadarrama, S., Korattikara, A., Oscar Ramirez, P. C., Holly, E., Fishman, S., Wang, K., Gonina, E., Wu, N., Harris, C., Vanhoucke, V., and Brevdo, E. TF-Agents: A library for reinforcement learning in tensorflow, 2018. URL https://github.com/tensorflow/agents.

Hasselt, H. v., Guez, A., and Silver, D. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 2094–2100. AAAI Press, 2016.

Hessel, M., Modayil, J., van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. Rainbow: Combining improvements in deep reinforcement learning. arXiv:1710.02298, 2017.

Kapturowski, S., Ostrovski, G., Quan, J., Munos, R., and Dabney, W. Recurrent experience replay in distributed reinforcement learning. In *8th International Conference on Learning Representations*, Addis Ababa, Ethiopia, 2020.

Khadka, S. and Tumer, K. Evolution-guided policy gradient in reinforcement learning. In *Advances in Neural Information Processing Systems 31 (NeurIPS-18)*, pp. 11961208, Montreal, 2018.

Lu, T., Schuurmans, D., and Boutilier, C. Non-delusional Q-learning and value iteration. In *Advances in Neural Information Processing Systems 31 (NeurIPS-18)*, pp. 99719981, Montreal, 2018.

Maei, H., Szepesvári, C., Bhatnagar, S., and Sutton, R. Toward off-policy learning control wtih function approximation. In *International Conference on Machine Learning*, pp. 719726, Haifa, Israel, 2010.

Melo, F. and Ribeiro, M. I. Q-learning with linear function approximation. In *Proceedings of the International Conference on Computational Learning Theory (COLT)*, pp. 308–322, 2007.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., Bellemare, M., Graves, A., Riedmiller, M., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Science*, 518:529–533, 2015.

Munos, R., Stepleton, T., Harutyunyan, A., and Bellemare, M. Safe and efficient off-policy reinforcement learning. In *Advances in Neural Information Processing Systems 29 (NIPS-16)*, pp. 10541062, Barcelona, 2016.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. Deep exploration via bootstrapped dqn. *Advances in Neural Information Processing Systems 29 (NIPS-16)*, pp. 40334041, 2016.

Pohlen, T., Piot, B., Hester, T., Azar, M. G., Horgan, D., Budden, D., Barth-Maron, G., van Hasselt, H., Quan, J., Vecerík, M., Hessel, M., Munos, R., and Pietquin, O. Observe and look further: Achieving consistent performance on atari. 2018. arXiv:1805.1159.

Riedmiller, M. Neural fitted q iteration—first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the 16th European Conference on Machine Learning*, pp. 317–328, Porto, Portugal, 2005.

Schrittwieser, J., Antonoglou, I., Hubert, T., Simonyan, K., Sifre, L., Schmitt, S., Guez, A., Lockhart, E., Hassabis, D., Graepel, T., Lillicrap, T., and Silver, D. Mastering

atari, go, chess and shogi by planning with a learned model. arXiv:1911.08265 [cs.LG], 2019.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 2018.

Szepesvári, C. and Smart, W. Interpolation-based Q-learning. In *Proceedings of the International Conference on Machine Learning (ICML-04)*, pp. 100–107, 2004.

van Hasselt, H. Double q-learning. In *Advances in Neural Information Processing Systems 23 (NIPS-10)*, pp. 2613–2621, Vancouver, BC, 2010.

Vapnik, V. N. *Statistical Learning Theory*. Wiley-Interscience, September 1998.

Wang, Z., Schaul, T., Hessel, M., van Hasselt, H., Lanctot, M., and de Freitas, N. Dueling network architectures for deep reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML-16)*, pp. 19952003, 2016.

Watkins, C. J. C. H. *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge, UK, May 1989.

Watkins, C. J. C. H. and Dayan, P. Q-learning. *Machine Learning*, 8:279–292, 1992.

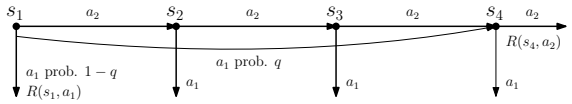

Figure 7: A simple MDP (Lu et al., 2018).

A. Delusional Bias Example

We describe a simple MDP, taken directly from (Lu et al., 2018), to show concretely how delusional bias causes problems for Q-learning with function approximation. The MDP in Fig. 7 illustrates the phenomenon: Lu et al. (2018) use a linear approximator over a specific set of features in this MDP to show that:

(a) No $\pi \in G(\Theta)$ can express the optimal (unconstrained) policy (which requires taking a_2 at each state);

(b) The optimal *feasible* policy in $G(\Theta)$ takes a_1 at s_1 and a_2 at s_4 (achieving a value of 0.5).

(c) Online Q-learning (Eq. 1) with data generated using an ε-greedy behavior policy must converge to a fixed point (under a range of rewards and discounts) corresponding to a "compromise" admissible policy which takes a_1 at both s_1 and s_4 (value of 0.3).

Q-learning fails to find a reasonable fixed-point because of delusion. Consider the backups at (s_2, a_2) and (s_3, a_2). Suppose $\hat{\theta}$ assigns a "high" value to (s_3, a_2), so that $Q_{\hat{\theta}}(s_3, a_2) > Q_{\hat{\theta}}(s_3, a_1)$ as required by π_{θ^*}. They show that any such $\hat{\theta}$ also accords a "high" value to (s_2, a_2). But $Q_{\hat{\theta}}(s_2, a_2) > Q_{\hat{\theta}}(s_2, a_1)$ is inconsistent the first requirement. As such, any update that makes the Q-value of (s_2, a_2) higher *undercuts the justification* for it to be higher (i.e., makes the "max" value of its successor state (s_3, a_2) lower). This occurs not due to approximation error, but the inability of Q-learning to find the value of the optimal *representable* policy.

A.1. CONQUR on the Simple MDP

We ran CONQUR on the MDP in Fig. 7 to verify its effectiveness in removing delusional bias in this simplified setting. We instantiate CONQUR with eight nodes, with ten independent runs, each with a different random seed. In all ten runs CONQUR converges to the optimal expressible (delusion-free) greedy policy: it selects action 1 at states 1 and 2, and action 2 at states 3 and 4. This clearly improves over Q-learning, which finds a sub-optimal policy (due to Q-updates of infeasible action combinations) (Lu et al., 2018). This example shows the effectiveness of CONQUR in removing delusional bias.

B. Consistency Penalization Experiments

Both DQN and DDQN uses a delayed version of the Q-network $Q_{\theta^-}(s', a')$ for label generation, but in a different way. In DQN, $Q_{\theta^-}(s', a')$ is used for both value estimate and action assignment $\sigma_{\text{DQN}}(s') = \text{argmax}_{a'} Q_{\theta_k}(s', a')$, whereas in DDQN, $Q_{\theta^-}(s', a')$ is used only for value estimate and the action assignment is computed from the current network $\sigma_{\text{DDQN}}(s') = \text{argmax}_{a'} Q_{\theta_k}(s', a')$.

With respect to delusional bias, action assignment of DQN is consistent for all batches after the latest network weight transfer, as $\sigma_{\text{DQN}}(s')$ is computed from the same $Q_{\theta^-}(s', a')$ network. DDQN, on the other hand, could have very inconsistent assignments, since the action is computed from the current network that is being updated at every step.

B.1. Training Methodology and Hyperparameters

We implement consistency penalty on top of the DQN and DDQN algorithm by modifying the open-source TF-Agents library (Guadarrama et al., 2018). In particular, we modify existing `DqnAgent` and `DdqnAgent` by adding a consistency penalty term to the original TD loss.

We use TF-Agents implementation of DQN training on Atari with the default hyperparameters, which are mostly the same as that used in the original DQN paper (Mnih et al., 2015). For conveniece to the reader, some important hyperparameters are listed in Table 2. The reward is clipped between $[-1, 1]$ following the original DQN.

B.2. Evaluation Methodology

We empirically evaluate our modified DQN and DDQN agents trained with consistency penalty on 15 Atari games. Evaluation is run using the training and evaluation framework for Atari provided in TF-Agents without any modifications.

B.3. Detailed Results

Fig. 8 shows the effects of varying λ on both DQN and DDQN. Table 3 summarizes the best penalties for each game and their corresponding scores. Fig. 9 shows the training curves of the best penalization constants. Finally, Fig. 10 shows the training curves for a fixed penalization of $\lambda = 0.5$. The datapoints in each plot of the aforementioned figures are obtained by averaging over window size of 30 steps, and within each window, we take the largest policy value (and over ≈2–5 multiple runs). This is done to reduce visual clutter.

Hyper-parameter	Value
Mini-batch size	32
Replay buffer capacity	1 million transitions
Discount factor γ	0.99
Optimizer	RMSProp
Learning rate	0.00025
Convolution channel	$32, 64, 64$
Convolution filter size	$(8 \times 8), (4 \times 4), (3 \times 3)$
Convolution stride	4, 2, 1
Fully-connected hidden units	512
Train exploration $\varepsilon_{\text{train}}$	0.01
Eval exploration $\varepsilon_{\text{eval}}$	0.001

Table 2: Hyperparameters for training DQN and DDQN with consistency penalty on Atari.

	DQN	λ_{best}	DQN(λ_{best})	DDQN	λ'_{best}	DDQN(λ'_{best})
Assault	2546.56	1.5	**3451.07**	2770.26	1	2985.74
Atlantis	995460.00	0.5	**1003600.00**	940080.00	1.5	999680.00
BattleZone	**67500.00**	2	55257.14	47025.00	2	48947.37
BeamRider	7124.90	0.5	**7216.14**	5926.59	0.5	6784.97
Boxing	86.76	0.5	90.01	82.80	0.5	**91.29**
Breakout	220.00	0.5	219.15	214.25	0.5	**242.73**
Enduro	1206.22	0.5	**1430.38**	1160.44	1	1287.50
Gravitar	475.00	1.5	**685.76**	462.94	1.5	679.33
JourneyEscape	-1020.59	0.25	-696.47	-794.71	1	**-692.35**
MsPacman	**4104.59**	2	4072.12	3859.64	0.5	4008.91
NameThisGame	7230.71	1	9013.48	9618.18	0.5	**10210.00**
Qbert	13270.64	0.5	**14111.11**	13388.92	1	12884.74
Seaquest	5849.80	1	6123.72	**12062.50**	1	7969.77
SpaceInvaders	2389.22	0.5	2707.83	3007.72	0.5	**4080.57**
StarGunner	40393.75	0.5	55931.71	55957.89	0.5	**60035.90**
TimePilot	4205.83	2	7612.50	6654.44	2	**7964.10**
Tutankham	222.76	1	**265.86**	243.20	0.25	247.17
VideoPinball	**569502.19**	0.25	552456.00	509373.50	0.25	562961.50
Zaxxon	5533.33	1	**10520.00**	7786.00	0.5	10333.33

Table 3: Consistency penalty ablation results on best penalty constants for DQN and DDQN, averaged over 5 random seeds.

C. Full CONQUR Experiments

Our results use a frontier queue of size (F) 16 (these are the top scoring leaf nodes which receive gradient updates and rollout evaluations during training). To generate training batches, we select the best node's regressor according to our scoring function, from which we generate training samples (transitions) using ε-greedy. Results are reported in Table 4, and training curves in Fig. 11. We used Bellman error plus consistency penalty as our scoring function. During the training process, we also calibrated the scoring to account for the depth difference between the leaf nodes at the frontier versus the leaf nodes in the candidate pool. We calibrated by taking the mean of the difference between scores of the current nodes in the frontier with their parents. We scaled this difference by multiplying with a constant of 2.5.

In our implementation, we initialized our Q-network with a pre-trained DQN. We start with the expansion phase. During this phase, each parent node splits into ℓ children nodes and the Q-labels are generated using action assignments from the Boltzmann sampling procedure, in order to create high quality and diversified children. We start the dive phase until the number of children generated is at least F. In particular, with $F = 16$ configuration, we performed the expansion phase at the zero-th and first iterations, and then at every tenth iteration starting at iteration 10, then at 20, and so on until ending at iteration 90. All other iterations execute the "dive" phase. For every fifth iteration, Q-labels are generated from action assignments sampled according to the Boltzmann distribution. For all other iterations, Q-labels are generated in the same fashion as the standard

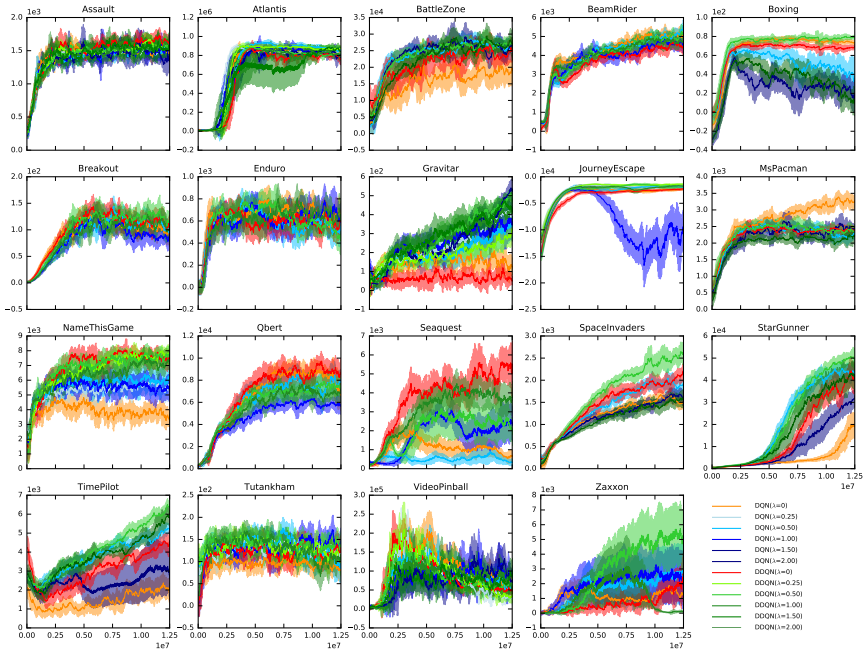

Figure 8: DQN and DDQN training curves for different penalty constant λ. Shaded area shows 95% confidence interval over 5 random seeds. Consistency penalty outperforms on 11 games: BattleZone, Gravitar, NameThisGame, SpaceInvaders, StarGunner, TimePilot, Zaxxon, Tutankham, JourneyEscape, Atlantis and BeamRider.

Q-learning (taking the max Q-value). The generated Q-labels along with the consistency penalty are then converted into gradient updates that applies to one or more generated children nodes.

C.1. Training Methodology and Hyperparameters

Each iteration consists of 10k transitions sampled from the environment. Our entire training process has 100 iterations which consumes 1M transitions or 4M frames. We used RMSProp as the optimizer with a learning rate of 2.5×10^{-6}. One training iteration has 2.5k gradient updates and we used a batch size of 32. We replace the target network with the online network every fifth iteration and reward is clipped between $[-1, 1]$. We use a discount value of $\gamma = 0.99$ and ε-greedy with $\varepsilon = 0.01$ for exploration. Details of hyper-parameter settings can be found in Table 5, 6.

C.2. Evaluation Methodology

We empirically evaluate our algorithms on 59 Atari games (Bellemare et al., 2013), and followed the evaluation procedure as in Hasselt et al. (2016). We evaluate our agents on every 10-th iteration (and also the initial and first iteration) by suspending our training process. We evaluate on 500k frames, and we cap the length of the episodes for 108k frames. We used ε-greedy as the evaluation policy with $\varepsilon = 0.001$. We evaluated our algorithm under the *no-op starts* regime—in this setting, we insert a random number of "do-nothing" (or *no-op*) actions (up to 30) at the beginning of each episode.

C.3. Detailed Results

Fig. 11 shows training curves of CONQUR with 16 nodes under different penalization strengths $\lambda \in \{1, 10\}$. While each game has its own optimal λ, in general, we found that $\lambda = 10$ gave the best performance for most games. Each plotted step of each training curve (including the baseline) shows the best performing node's policy value as evaluated with full rollouts. Table 4 shows the summary of the highest policy values achieved for all 59 games for CONQUR and the baseline under 16 nodes. Both the baseline and CONQUR improve overall, but CONQUR's advantage over the baseline is amplified. These results all use a splitting factor of $c = 4$. (We show results with 8 nodes and a splitting factor of 2 below.)

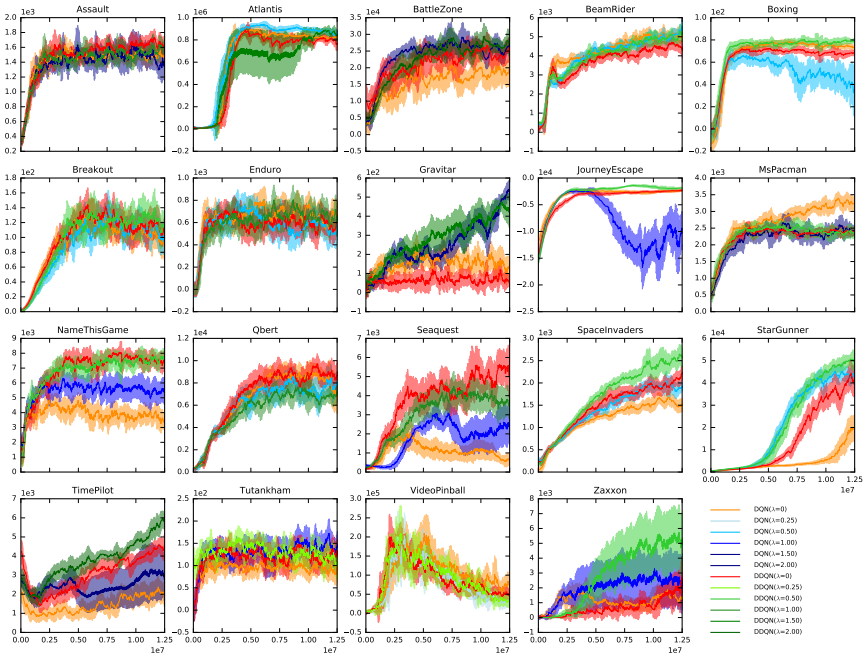

Figure 9: DQN and DDQN training curves for the respective best λ and baseline. Shaded area shows 95% confidence interval over 5 random seeds.

C.4. Additional Results: CONQUR with 8 Nodes

As an additional study of CONQUR, we present results of the running our method using 8 nodes (rather than the 16 used above), and compare it to a multi-DQN baseline that also uses 8 "nodes" (i.e., 8 separate DQN runs). We use a splitting factor $c = 2$ for CONQUR. Table 7 shows the average scores for each game using CONQUR and the baseline with 8 nodes. Unsurprisingly, CONQUR with 8 nodes does not perform as well as CONQUR with 16 nodes; but as in the 16-node case, CONQUR outperforms the baseline when each uses 8 nodes. More importantly, the average improvement of 24.5% for CONQUR with 16 nodes over the corresponding baseline *exceeds* the 19.6% improvement of CONQUR in the 8-node case. This is a strong indication that increasing the number of nodes increases the performance gap *relative* to the corresponding multi-DQN baseline; this implies that a good search heuristic is critical to effectively navigate the search space (as compared to randomly selected nodes) with a greater number of candidate hypotheses.[10]

C.5. Additional Results: CONQUR Training of Entire Q-Network from Scratch

We present results of running our method using 2 nodes with full network training from scratch, and compare it to a multi-DQN baseline that also uses 2 "nodes" (i.e., 2 separate DQN runs). We use a configuration of $c = 1, m = 4, F = 2, \ell = 1$ for CONQUR. Table 8 shows the average scores for each game using CONQUR and the baseline with 2 nodes, 9 of 11 games show at least 32% gain over the baseline, the other 2 are about the same as baseline.

[10] Average score improvements exclude games where the baseline score is zero.

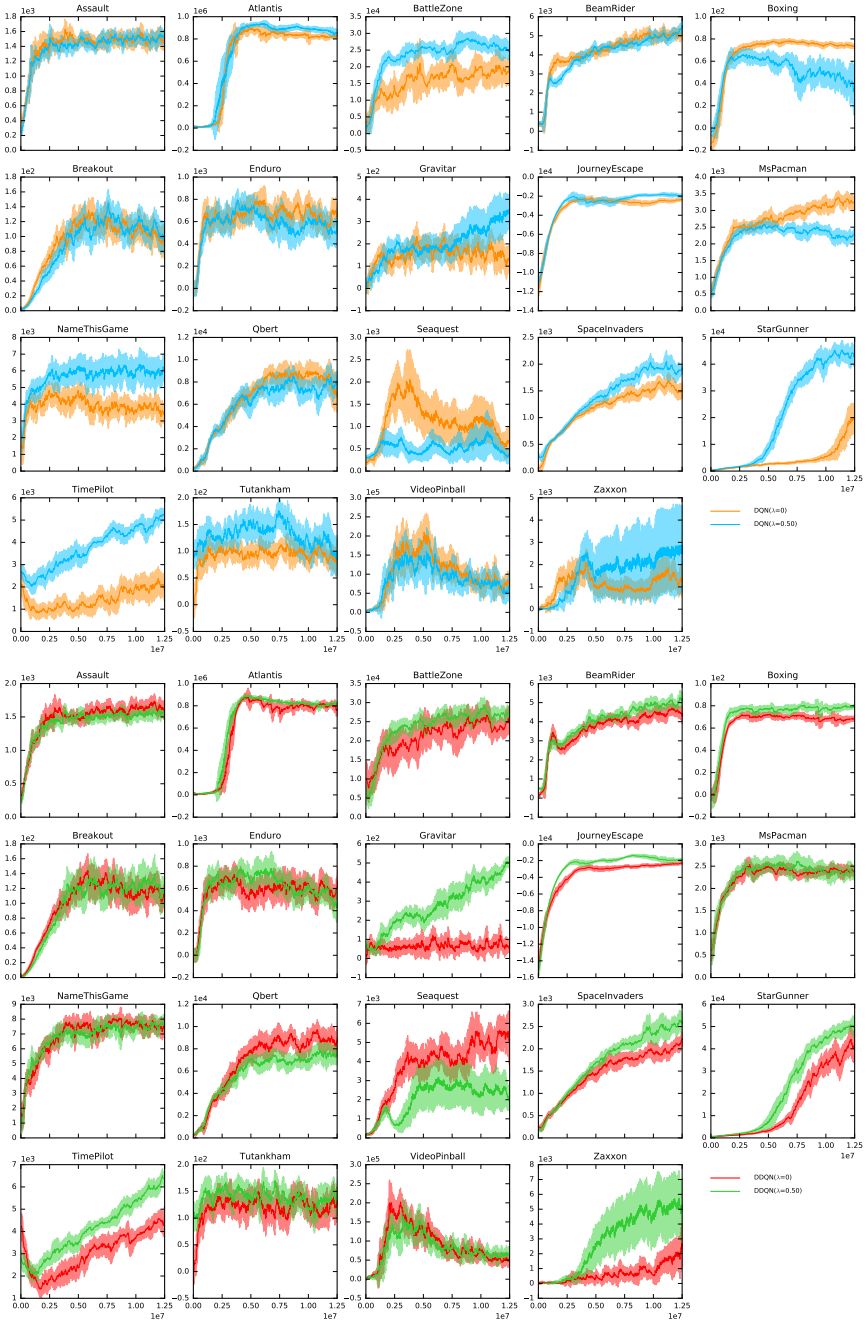

Figure 10: DQN and DDQN training curves for $\lambda = 0.5$ and the baseline. Shaded area shows 95% confidence interval over 5 random seeds.

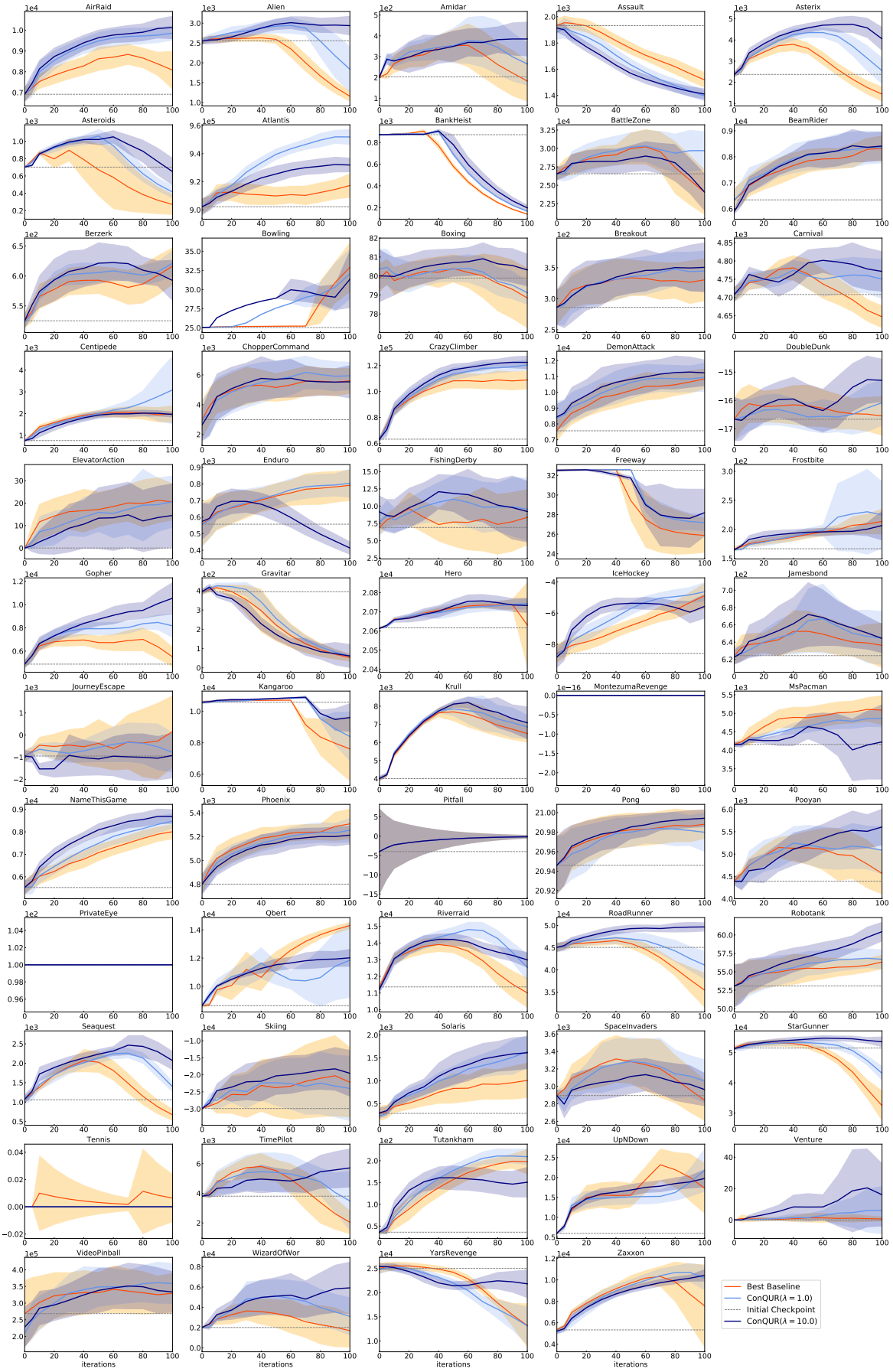

Figure 11: Training curves on 16 nodes with up to 30 no-op starts. Shading shows 95% confidence interval over 5 random seeds.

	CONQUR(λ_{best}) (16 nodes)	Baseline (16 nodes)	Checkpoint
AirRaid	**10613.01**	9656.21	6916.88
Alien	**3253.18**	2713.05	2556.64
Amidar	**555.75**	446.88	203.08
Assault	2007.81	**2019.99**	1932.55
Asterix	**5483.41**	4649.52	2373.44
Asteroids	**1249.13**	1196.56	701.96
Atlantis	**958932.00**	931416.00	902216.00
BankHeist	**1002.59**	965.34	872.91
BattleZone	31860.30	**32571.80**	26559.70
BeamRider	9009.14	**9052.38**	6344.91
Berzerk	**671.95**	664.69	525.06
Bowling	38.36	**39.79**	25.04
Boxing	**81.37**	81.26	80.89
Breakout	**372.31**	359.17	286.83
Carnival	**4889.19**	4860.66	4708.14
Centipede	**4025.57**	2408.23	758.21
ChopperCommand	**7818.22**	6643.07	2991.00
CrazyClimber	**134974.00**	119194.00	63181.14
DemonAttack	**11874.80**	11445.20	7564.14
DoubleDunk	**-14.04**	-15.25	-16.66
ElevatorAction	24.67	**28.67**	0.00
Enduro	**879.84**	835.11	556.97
FishingDerby	**16.28**	13.22	6.92
Freeway	32.65	32.63	32.52
Frostbite	**289.25**	230.29	166.44
Gopher	**11959.20**	9084.00	4879.02
Gravitar	**489.22**	446.64	394.46
Hero	**20827.00**	20765.70	20616.30
IceHockey	**-3.15**	-3.55	-8.59
Jamesbond	**710.78**	681.05	624.36
JourneyEscape	902.22	**1437.06**	-947.18
Kangaroo	**11017.65**	10743.10	10584.20
Krull	**9556.53**	9487.49	3998.90
MontezumaRevenge	0.00	0.00	0.00
MsPacman	5444.31	**5487.12**	4160.50
NameThisGame	**9104.40**	8445.43	5524.73
Phoenix	5325.33	**5430.49**	4801.18
Pitfall	0.00	0.00	-4.00
Pong	21.00	21.00	20.95
Pooyan	**5898.46**	5728.05	4393.09
PrivateEye	100.00	100.00	100.00
Qbert	13812.40	**15189.00**	8625.88
Riverraid	**15895.10**	15370.10	11364.90
RoadRunner	**50820.40**	47481.10	45073.25
Robotank	**62.74**	57.66	53.08
Seaquest	**3046.34**	2691.88	1060.77
Skiing	**-13638.80**	-14908.21	-29897.07
Solaris	**1991.33**	1202.89	285.46
SpaceInvaders	**3556.10**	3520.96	2895.30
StarGunner	**55679.27**	55176.90	51490.60
Tennis	0.00	0.00	0.00
TimePilot	6698.88	**7327.71**	3806.54
Tutankham	**252.51**	220.90	36.07
UpNDown	31258.84	**34455.20**	5956.24
Venture	**37.37**	3.64	0.00
VideoPinball	**423012.59**	383105.41	268476.04
WizardOfWor	**8154.73**	4782.11	2012.24
YarsRevenge	26188.24	**26330.31**	25000.36
Zaxxon	**11723.20**	11589.90	5334.44

Table 4: Summary of scores with ε-greedy ($\varepsilon = 0.001$) evaluation with up to 30 no-op starts. We ran CONQUR with 16 nodes and with $\lambda \in \{1, 10\}$. We report the scores of the best λ_{best} for each game. For most games, λ_{best} is 10. Scores averaged over 5 random seeds.

Hyperparameters	Description	Value
Dive levels d to run	We run d levels of diving phase after each expansion phase	9
Boltzmann Iteration	Every module this number of iteration/level, Q-labels are generated from Boltzmann distribution in order to create diversified node.	5
Online network target network swap frequency	Iteration (Frequency) at which the online network parameters swap with the target network	5
Evaluation frequency	Iteration (Frequency) at which we perform rollout operation (testing with the environment).	10
Learning Rate	Learning rate for the optimizer.	2.5×10^{-6}
Optimizer	Optimizer for training the neural network.	RMSprop
Iteration training data transition size	For each iteration, we generate this number of transitions and use it as training data.	10k
Training step frequency	For each iteration, we perform (iteration training data transition size / training step frequency) number of gradient updates.	4
Mini-batch size	Size of the mini batch data used to train the Q-network.	32
$\varepsilon_{\text{train}}$	ε-greedy policy for exploration during training.	0.01
$\varepsilon_{\text{eval}}$	ε-greedy policy for evaluating Q-regressors.	0.001
Training calibration parameter	Calibration to adjust the difference between the nodes from the candidate pool m which didn't selected during both the expansion nor the dive phases. The calibration is performed based on the average difference between the frontier nodes and their parents. We denote this difference as $\triangle$.	$2.5\triangle$
Temperature τ	Temperature parameter for Boltzmann sampling. Adaptively multiplied or divided by a factor of 1.5 or 4 respectively.	1
Discount factor γ	Discount factor during the training process.	0.99

Table 5: Common Hyperparameters for CONQUR training and evaluation.

Hyperparameters	Description	Value
Splitting factor c	Number of children created from a parent node	4
Candidate pool size m	Pool of candidate leaf nodes for selection into the dive or expansion phase	46
Maximum frontier nodes F	Maximum number of child leaf nodes for the dive phase	16
Top nodes to expand ℓ	Select the top ℓ nodes from the candidate pool for the expansion phase.	4

Table 6: Hyperparameters for CONQUR (16 nodes) training and evaluation.

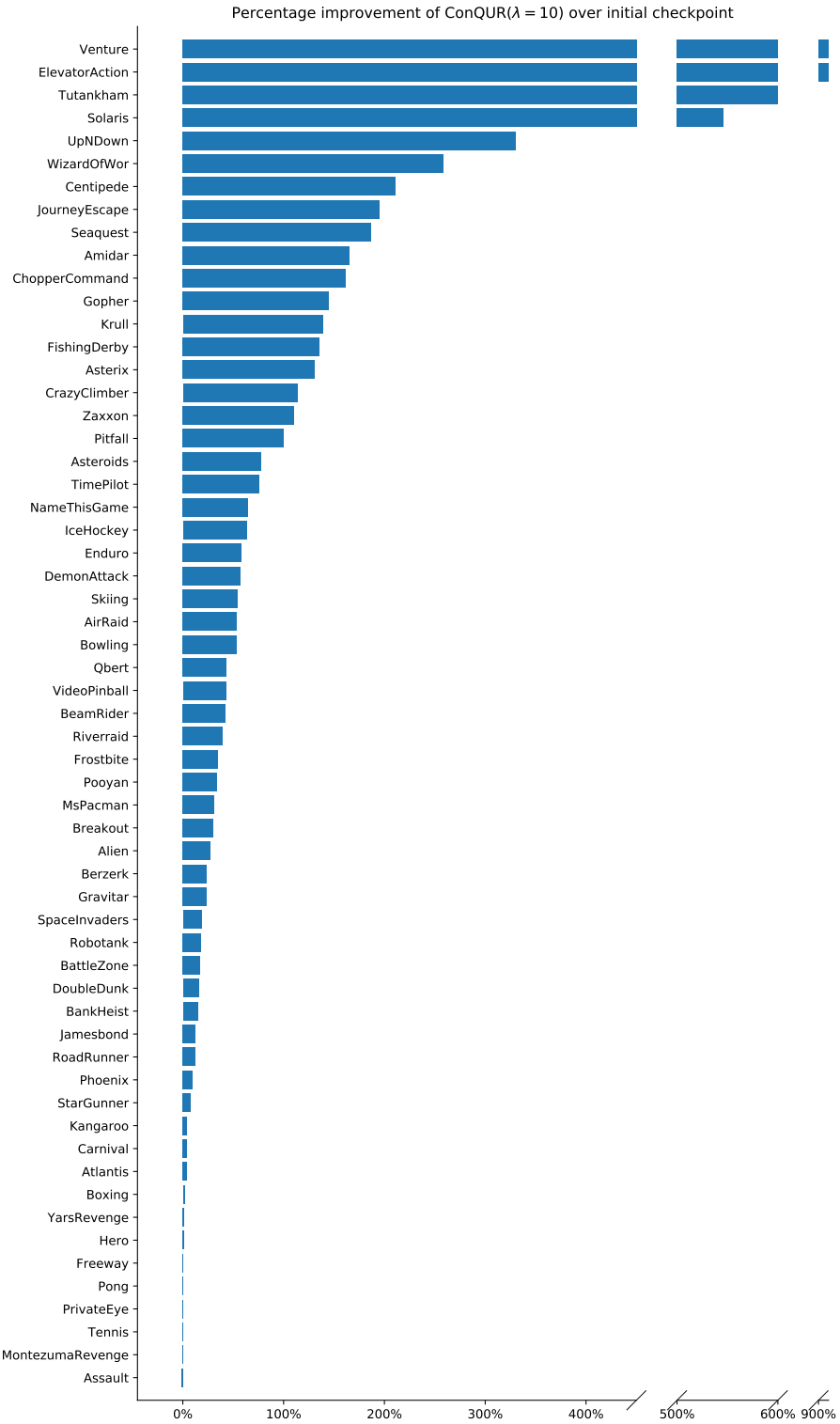

Figure 12: Improvement CONQUR($\lambda = 10$) with 16 nodes achieves over the initial checkpoint Q-network. Score averaged over 5 random seeds.

	CONQUR(λ_{best}) (8 nodes)	Baseline (8 nodes)	Checkpoint
AirRaid	**10647.80**	9050.86	6885.72
Alien	**3341.36**	3207.5.05	2556.64
Amidar	**577.45**	573.55	202.74
Assault	1892.02	**1976.80**	1873.05
Asterix	**5026.24**	4935.21	2373.44
Asteroids	**1194.90**	1170.11	704.38
Atlantis	**949012.00**	932668.00	902216.00
BankHeist	909.61	**924.75**	871.91
BattleZone	**32139.90**	30983.10	26558.70
BeamRider	**8613.98**	8109.63	6397.49
Berzerk	**659.64**	634.83	524.76
Bowling	**30.07**	**25.29**	25.04
Boxing	**81.78**	81.48	80.29
Breakout	350.11	**362.98**	286.14
Carnival	**4862.30**	4800.83	4708.23
Centipede	**2747.89**	2608.78	757.51
ChopperCommand	**7188.25**	6737.21	2641.71
CrazyClimber	**131675.00**	122424.00	63181.11
DemonAttack	**11346.20**	10947.90	8022.08
DoubleDunk	**-13.57**	-15.35	-16.66
ElevatorAction	**28.00**	**21.33**	0.00
Enduro	**849.07**	811.58	556.56
FishingDerby	**13.34**	11.56	7.15
Freeway	32.60	32.60	32.52
Frostbite	**296.57**	220.81	165.01
Gopher	**9999.61**	8013.34	4879.13
Gravitar	475.03	**480.64**	394.46
Hero	**20803.60**	20774.80	20598.40
IceHockey	**-3.23**	-4.78	-8.63
Jamesbond	664.98	**669.54**	626.53
JourneyEscape	-462.64	**391.44**	-947.18
Kangaroo	**10974.00**	10733.60	10584.20
Krull	9503.62	**9538.22**	4039.78
MontezumaRevenge	**1.46**	0.00	0.00
MsPacman	5066.17	**5227.84**	4160.50
NameThisGame	**9181.30**	8410.29	5529.50
Phoenix	**5307.46**	5227.84	4801.18
Pitfall	0.00	0.00	-4.00
Pong	21.00	20.99	20.95
Pooyan	**5778.99**	5184.14	4393.09
PrivateEye	100.00	100.00	100.00
Qbert	11953.40	**13965.80**	8625.88
Riverraid	**15614.40**	14812.40	11253.30
RoadRunner	**49864.80**	46302.20	45073.25
Robotank	**61.92**	56.90	53.08
Seaquest	**2647.82**	2560.61	1034.83
Skiing	**-14058.90**	-14079.80	-29896.80
Solaris	**1956.24**	1182.59	291.70
SpaceInvaders	**3436.16**	3292.68	2895.30
StarGunner	**55479.00**	54207.30	51419.60
Tennis	0.00	0.00	0.00
TimePilot	6717.62	**6799.19**	3806.22
Tutankham	**242.03**	229.23	36.00
UpNDown	**22544.60**	23331.20	5956.21
Venture	**15.41**	1.50	0.00
VideoPinball	382110.59	**390540.41**	209620.0
WizardOfWor	**5750.05**	3632.17	2011.77
YarsRevenge	**25631.10**	25396.70	25319.20
Zaxxon	**10751.80**	11156.20	5186.01

Table 7: Summary of scores, averaged over 5 random seeds, with ε-greedy ($\varepsilon = 0.001$) evaluation with up to 30 no-op starts. As a side study, we ran CONQUR with 8 nodes and with $\lambda \in \{1, 10\}$. We report the scores of the best λ_{best} for each game. For most games, λ_{best} is 10. The 8 nodes configuration follows the same as in Table 5, except that $c = 2, m = 38, F = 8, \ell = 2$.

	ConQUR (λ_{best})	Baseline	Initial
AirRaid	**3595.03**	3354.98	323.80
Berzerk	**668.86**	505.90	215.74
Bowling	**49.19**	34.55	0.42
ChopperCommand	1117.17	**1152.53**	555
Freeway	**10.30**	5.82	0.00
Jamesbond	**82.15**	74.36	50.16
Kangaroo	**469.98**	311.07	3.24
PrivateEye	**996.09**	20.57	-86.15
Robotank	**15.56**	11.74	6.36
Solaris	**2345.47**	1684.93	1202.35
VideoPinball	**227390.17**	143394.81	4051.15

Table 8: Summary of scores, averaged over 5 random seeds. As a side study, we ranCONQUR with 2 nodes and training from scratch. We report the score of the best λ_{best} for each game. For most games, λ_{best} is 10. The 2 nodes configuration is identical to that of Table 5, except with $c = 1, m = 4, F = 2, \ell = 1$.