university of groningen / faculty of science and engineering

On the presence of Winning Tickets in Reinforcement Learning

Bachelor's Project Thesis

Tom Kosters, s3228592, t.h.b.kosters@student.rug.nl,
Supervisor: Dr. Matthia Sabatelli

Abstract: The lottery ticket hypothesis suggests that randomly-initialized neural networks contain smaller sub-networks that can be trained to perform at least as well as the full network. This project demonstrates that such winning tickets can be found in shallow neural networks used for function approximation in simple reinforcement learning tasks, and that they are found across environments, algorithms and pruning strategies.

1 Introduction

The lottery ticket hypothesis (LTH) states that randomly initialized artificial neural networks contain smaller sub-networks that can be trained from scratch to match the performance of the full network after training for the same number of iterations (Frankle and Carbin, 2019).

These tickets are obtained by iteratively pruning and re-training neural networks (Frankle and Carbin, 2019). Different iterative pruning methods have been shown to uncover different winning tickets for the same supervised learning task (Paganini and Forde, 2020).

Most research on the LTH has been done in the context of supervised learning, but recent work has also found tickets in Natural Language Processing and Reinforcement Learning tasks (Vischer et al. 2021, Yu et al. 2020). The neural networks used are often heavily overparameterized, and tickets have mostly been shown for a single algorithm and a single pruning strategy per environment.

This project aims to learn whether winning tickets generally occur in neural networks used for function approximation in reinforcement learning. To this end, three different algorithms trained on two simple control problems are iteratively pruned using three different pruning strategies to see whether such tickets can be found in small single-hidden-layer neural networks approximating state-value and state-action value functions.

2 Background

2.1 Reinforcement Learning

Reinforcement learning (RL) agents learn how to interact with their environment so as to maximize a numerical reward signal over time. The agent and environment interact at each of a series of discrete time steps as the agent is presented with a state $S_t \in \mathcal{S}$ and selects an action $A_t \in \mathcal{A}$, after which it receives a numerical reward R_{t+1} and transitions to state S_{t+1}. Actions are selected according to a policy π, which maps states to probabilities of selecting actions. $\pi(s|a)$ is the probability that $A_t = a$ if $S_t = s$. The goal is to learn an optimal policy π^* that maximizes future rewards (Sutton and Barto, 2018).

The expected return when starting in state s under policy π is given by its *state-value function* v_π:

$$v_\pi(s) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, \pi\right], \qquad (2.1)$$

where γ is the discount factor in $(0, 1]$ that scales the relative preference for earlier rewards over later ones.

The expected return for taking action a in state s under policy π is given by its *state-action value function* q_π:

$$q_\pi(s, a) = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a, \pi\right]. \qquad (2.2)$$

An optimal policy π^* maximizes the state-value and state-action value functions for all $s \in \mathcal{S}$ and all $a \in \mathcal{A}$, leading to optimal state-value function v_* and optimal state-action value function q_*.

Temporal difference (TD) learning is a popular class of RL methods that approximate a value function by bootstrapping from a working estimate of that function (Sutton, 1988). The simplest such method, *TD(0)*, makes the following update to its estimated state-value function V:

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big], \tag{2.3}$$

where α is the learning rate. Updating its estimated utility of a state using the reward received as well as its estimated utility of the state at the next time step allows for learning to start very quickly as useful updates can be made at every time step.

The RL methods used in this project are all based on the popular TD learning algorithm *Q-learning* (Watkins, 1989, Watkins and Dayan, 1992) which updates its estimated state-action value function Q as follows:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)\Big]. \tag{2.4}$$

Q-learning is an *off-policy* algorithm, as it directly approximates q_* by updating Q using the maximum estimated return at each step, regardless of whether the current policy would obtain that return.

2.2 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) uses artificial neural networks to represent value functions in order to deal with state-action spaces that are too large for tabular RL methods such as those mentioned above. The *Deep Q-Network* (DQN) algorithm (Mnih et al., 2015) is a variant of Q-learning that uses a neural network paramaterized by θ to approximate the optimal state-action value function q_*. This addition of function approximation adds further instability to Q-learning, and DQN introduced two methods to improve training stability:

Experience replay de-correlates the experiences used for training by storing experiences $\langle S_t, A_t, R_{t+1}, S_{t+1}\rangle$ in a replay memory buffer D, from which mini-batches are then sampled uniformly at random during training.

A *target network* is a second neural network, parameterized by θ^-, that is used to compute the TD-targets used for training the online network θ. The target network parameters θ^- are kept stable between training iterations of θ, and are updated by copying the θ parameters every C time steps.

DQN adapts the Q-learning update rule seen in 2.4 to a differentiable loss function that can be minimized using stochastic gradient descent to train the neural network parameterized by θ that represents the state-action value function (the Q-Network):

$$L(\theta) = \mathbb{E}_{\langle S_t, A_t, R_{t+1}, S_{t+1}\rangle \sim U(D)}\Big[\big(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a; \theta^-) - Q(S_t, A_t; \theta)\big)^2\Big]. \tag{2.5}$$

2.3 Lottery Ticket Hypothesis

The *Lottery Ticket Hypothesis* states that *a randomly-initialized, dense neural network contains a subnetwork that is initialized such that - when trained it isolation - it can match the test accuracy of the original network after training for at most the same number of iterations* (Frankle and Carbin, 2019).

Such a sparse subnetwork is called a *winning ticket*, and it consists of a dense feed-forward neural network $f(x; \theta)$, its parameters at initialization θ_0, and a mask over its parameters $m \in \{0, 1\}^{|\theta|}$ indicating which of them are zeroed. A full ticket can be specified as $f(x; m \odot \theta_0)$.

Tickets are found through *iterative pruning*:

1. Randomly initialize a neural network $f(x; \theta_0)$.
2. Train for j iterations to obtain parameters θ_j.
3. Prune $p\%$ of parameters in θ_j to obtain mask m.
4. Set neural network parameters $\theta \leftarrow m \odot \theta_0$.
5. Train for j iterations to obtain parameters θ'.
6. Prune $p\%$ of remaining parameters in θ', obtaining mask m'.

7. Set $m \leftarrow m'$.

Steps 4-7 can be repeated to find increasingly sparse tickets. Tickets $f(x; m \odot \theta_0)$ that match or exceed the performance of the original network $f(x; \theta_0)$ are winning tickets.

Recent work has shown that multiple winning tickets can exist within a neural network, and that different pruning techniques can uncover different winning tickets (Paganini and Forde, 2020).

3 Method

The goal of this project is to determine whether winning tickets are generally present in neural networks approximating value functions for deep RL, and to do so with limited time and computational resources. To this end, experiments consisted of determining whether winning tickets could be found in any or all combinations of three different deep RL algorithms trained on two simple RL problems using three different iterative pruning strategies: a total of 18 experiments repeated 20 times each.

3.1 Environments

The RL problems used are *cartpole* (Barto et al., 1983) and *acrobot* (Sutton, 1996). The versions used in this project are those made available through the OpenAI Gym toolkit (Brockman et al., 2016).

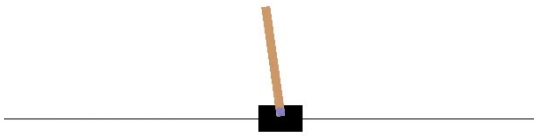

Figure 3.1: CartPole-v0

The `CartPole-v0` environment is a slightly simplified version of the pole-balancing problem described by Barto et al. (1983) that does not take cart friction into account and runs for a maximum of 200 time-steps. The goal is to keep the pole upright (within 15 degrees of vertical) for as long as possible without moving the cart too far from its central starting point.

The observation space consists of four floating point numbers: cart position, cart velocity, pole angle, and pole angular velocity. At each time step the two possible actions are pushing the cart leftwards or pushing the cart rightwards. The reward is +1 for each time step, including the terminal state.

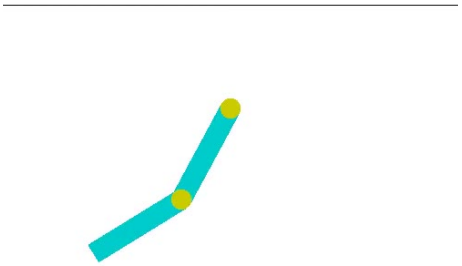

Figure 3.2: Acrobot-v1

The `Acrobot-v1` environment uses the implementation from Geramifard et al. (2015) of the *acrobot* as described by Sutton (1996). The *acrobot* consists of two joints and two links (see 3.2); the goal is to swing the tip of the outer link above the line in as little time as possible by varying the torque on the joint connecting the two links.

The observation space consists of six floating point numbers: the cosine, sine, and angular velocity of both the angle of the inner link Θ_1 (where an angle of 0 means pointing straight down) and the relative angle of the outer link to the inner link Θ_2 (where a relative angle of 0 means they form a straight line). At each time step there are 3 possible actions to modify the torque on the connecting joint: apply positive torque, apply negative torque, or do not apply torque. An episode lasts for a maximum of 500 time steps, or until the goal is achieved; the reward is -1 for each time step that the goal is not reached, and 0 when the goal is reached.

3.2 RL algorithms

DQN

The first algorithm used is DQN, which is described in Section 2.2. The Q-networks used for both the *cartpole* and *acrobot* problems are fully connected feed-forward neural networks with a single hidden layer of 128 nodes; their input- and output-layer sizes correspond to the observation space and action space of their respective environments, which are described in Section 3.1.

The hyperparameters used were only set and adjusted to yield acceptable performance after training the full DQN model. Acceptable performance here means reaching a mean cumulative reward greater than 195 over 100 testing episodes after training for at most 1000 episodes of *cartpole*, and a mean cumulative reward greater than -100 over 100 testing episodes after training for at most 500 episodes of *acrobot*. The hyperparameters found satisfactory for DQN were used for all three algorithms because the relative performance of the algorithms is not important for this project. The target network parameters are updated at the end of every training episode. A full specification of the hyperparameters can be found in Appendix A.

Double DQN

Double DQN (van Hasselt et al., 2016) adapts DQN to reduce overestimation of state-action values using insights from Double Q-learning (van Hasselt, 2010).

The use of the max operator in Q-learning (see 2.4) and DQN (see 2.5) can lead to overestimation because it leads to the same values being used when selecting and evaluating an action. Double Q-learning deals with this by learning two independent Q-functions that are used symmetrically and alternately so that one function's updates use the other function's state-action value estimates.

Double DQN instead uses the target network θ^- that is already part of DQN, using its estimated value when evaluating the policy selected by the online network θ:

$$L(\theta) = \mathbb{E}_{\langle S_t, A_t, R_{t+1}, S_{t+1}\rangle \sim U(D)}\Bigg[\Big(R_{t+1} + \gamma Q(S_{t+1}, \arg\max_{a \in \mathcal{A}} Q(S_{t+1}, a; \theta); \theta^-) - Q(S_t, A_t; \theta)\Big)^2\Bigg].$$

The target network used in Double DQN is not fully independent from the online network, because it does not alter its own weights through learning. Instead, its weights are updated by copying the weights of the online network every C time steps, just like in DQN.

Deep Quality-Value Learning

Deep Quality-Value Learning (DQV) (Sabatelli et al., 2020) is based on QV(λ) (Wiering, 2005). Like QV(λ) it approximates both the state-value (V) function and the state-action value (Q) function, using state-value estimates to update the Q-function, but in DQV these functions are approximated by (separate) neural networks.

The neural network paramaterized by Φ approximates the V-function and uses the following loss function:

$$L(\Phi) = \mathbb{E}_{\langle S_t, A_t, R_{t+1}, S_{t+1}\rangle \sim U(D)}\Bigg[\Big(R_{t+1} + \gamma V(S_{t+1}; \Phi^-) - V(S_t; \Phi)\Big)^2\Bigg],$$

and the neural network parameterized by θ uses the value function estimates to approximate the Q-function:

$$L(\theta) = \mathbb{E}_{\langle S_t, A_t, R_{t+1}, S_{t+1}\rangle \sim U(D)}\Bigg[\Big(R_{t+1} + \gamma V(S_{t+1}; \Phi^-) - Q(S_t, A_t; \theta)\Big)^2\Bigg].$$

Both neural networks use experience replay for training, and the V-function network uses a target network Φ^-.

3.3 Finding tickets

In each experiment three types of pruned subnetworks are considered, here called 'winning

ticket' (WT), 'mask-only' (MO) and 'random ticket' (RT). The 'winning ticket' uses a mask obtained through iterative magnitude pruning and the set of initialization values the full network started with (as described in 2.3), whereas a 'mask-only' network uses the same mask but not the initialization values (it instead re-initializes to random values before re-training). A 'random ticket' also re-initializes before re-training, but uses a randomly pruned mask with the same sparsity as the winning ticket and mask-only network. The random ticket is used as a baseline to evaluate the impact of the structure (mask) and weights (winning ticket) found through iterative magnitude pruning.

In the specific case of DQV, both networks are pruned and a winning ticket thus consists of a pair of masks and initialization weight vectors. These subnetworks will be referred to in the singular, just like the winning ticket, mask-only subnetwork and random ticket of the other two algorithms.

All networks are pruned and re-trained for 20 iterations, with 20% of remaining weights (or as close to 20% as possible) being pruned at each iteration. Biases are set to 0 after (re-)initialization or resetting of network parameters, and are not pruned. The pruning strategies used are as described by Paganini and Forde (2020) and implemented in PyTorch. Each method can be employed to prune based on weight magnitude (WT & MO) or to prune weights indiscriminately (RT).

Global unstructured pruning considers *all remaining weights* in the neural network regardless of which layer they belong to. The magnitude-based version prunes the 20% that individually have the lowest absolute values. The random version of global unstructured pruning prunes 20% of remaining weights indiscriminately.

Local unstructured pruning prunes 20% of remaining *weights per layer*, so that all layers end up with similar proportional sparsity. The magnitude-based version prunes the weights with the lowest absolute values, and the random version prunes 20% of a layer's remaining weights indiscriminately.

Local structured pruning prunes 20% of a *layer's remaining nodes*, zeroing all weights in their weight vectors (i.e. all weights connected to that node in the layer being pruned). The magnitude-based version used here prunes the nodes with the lowest L_2-norm, the square root of the sum of its squared weights. The reason for using the L_2-norm rather than the L_1-norm (the sum of the weights' absolute values) is that the sum of squares disproportionately favours weights with higher magnitude, so that nodes with one or two very strong connections are more likely to be retained than nodes with several average-magnitude connections that may be less important. The random version of this pruning strategy prunes 20% of remaining nodes indiscriminately.
The networks used in this project are shallow and have few input nodes [4-6] and output nodes [1-3], so both the first (input-to-hidden) and second (hidden-to-output) layers prune the hidden nodes (of which there are 128 at the start) and no full input- or output-nodes are pruned.

3.4 Evaluation

After each iteration of pruning and re-training the tickets were evaluated by taking the mean cumulative reward over 100 testing episodes of *cartpole* or *acrobot*. After 20 runs of 20 iterations the mean cumulative scores and standard deviations for each type of ticket were plotted and compared.

The relative performance of different types of pruned sub-network (WT, MO and RT) at different levels of sparsity can be compared visually in graphs. Additionally, a threshold is used to highlight when performance deteriorates significantly compared to the unpruned model; this threshold is full model performance minus 10 for *cartpole* and full model performance minus 25 for *acrobot*, and is indicated by a grey dotted line in the graphs (Section 4, Appendix B).

4 Results

Graphs of all results can be found in Appendix B; results tend to vary across environments and pruning strategies more than between algorithms.

4.1 Relative performance

Tables 4.1 and 4.2 show at what stage of pruning performance falls below the threshold (see section 3.4) for the winning ticket (WT), mask-only (MO) network and random ticket (RT) in each of the experiments.

Table 4.1: Percentage of weights remaining when ticket drops below threshold (Cartpole).

Experiment	WT	MO	RT
DQN - GU	10.7%	13.4%	26.2%
DQN - LU	8.6%	6.9%	21.0%
DQN - LS	8.6%	5.5%	21.0%
DDQN - GU	10.7%	8.6%	21.0%
DDQN - LU	6.9%	6.9%	21.0%
DDQN - LS	8.6%	8.6%	21.0%
DQV - GU	8.6%	8.6%	21.0%
DQV - LU	10.7%	8.6%	21.0%
DQV - LS	5.5%	6.9%	21.0%

Table 4.2: Percentage of weights remaining when ticket drops below threshold (Acrobot).

Experiment	WT	MO	RT
DQN - GU	13.4%	32.8%	26.2%
DQN - LU	10.7%	26.2%	26.2%
DQN - LS	16.8%	26.2%	41.0%
DDQN - GU	16.8%	26.2%	26.2%
DDQN - LU	21.0%	26.2%	26.2%
DDQN - LS	16.8%	10.7%	41.0%
DQV - GU	21.0%	41.0%	26.2%
DQV - LU	21.0%	26.2%	32.8%
DQV - LS	10.7%	16.8%	41.0%

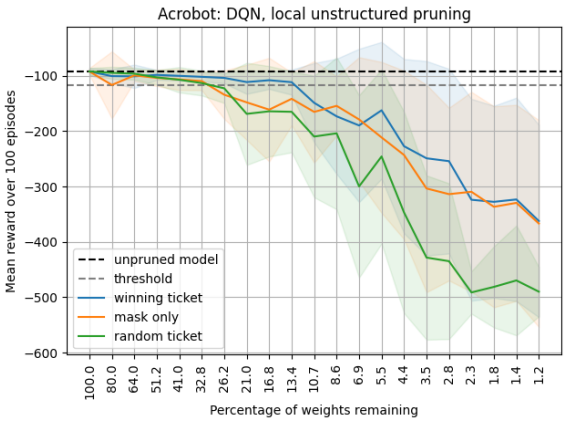

Figure 4.1: Acrobot, DQN, LU pruning

The winning ticket outperformed the random ticket in all 18 experiments, always reaching a lower percentage of weights remaining before falling below the threshold. Figure 4.1 shows the winning ticket dropping below the threshold at 10.7% of

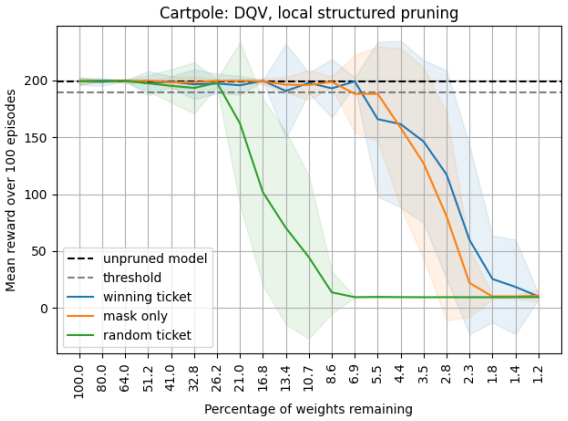

Figure 4.2: Cartpole, DQV, LS pruning

weights remaining, while the random ticket and mask-only network both drop below the threshold at 26.2% remaining. Figure 4.2 shows WT and MO performing similarly.

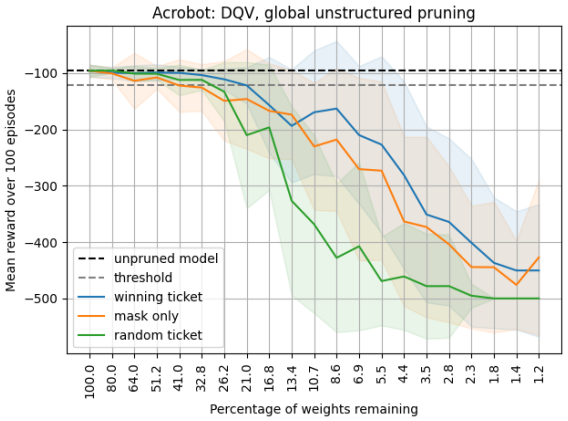

Figure 4.3: Acrobot, DQV, GU pruning

The mask-only network outperformed the random ticket in 13 experiments, and matched or outperformed the winning ticket in 8 of those. In 3 other experiments, the mask-only network matched the random ticket's performance instead, and in the 2 remaining experiments it dropped below the threshold quicker than the random ticket. All experiments in which the mask-only network performed similarly or worse than the random ticket were in the *acrobot* environment. Figure 4.3 shows the mask-only network falling below the threshold

before the random ticket, but decaying in performance less rapidly afterwards.

4.2 Resulting input layer masks

Tables 4.3 and 4.4 show the mean percentage of weights remaining for each input node in the iteratively pruned mask of the Q-network of the DQN algorithm after the 10th pruning iteration of local unstructured pruning (when 10.7% of total weights remain in each layer).

Table 4.3: Percentage of weights remaining for each observation parameter (cartpole).

Observation	RT	WT	Difference
Cart position	10.5%	8.1%	−22%
Cart velocity	10.1%	7.6%	−25%
Pole angle	11.1%	14.0%	+25%
Pole ang. vel.	11.3%	13.3%	+18%

Table 4.4: Percentage of weights remaining for each observation parameter (Acrobot).

Observation	RT	WT	Difference
$cos(\Theta_1)$	10.6%	15.6%	+47%
$sin(\Theta_1)$	11.1%	14.8%	+33%
$cos(\Theta_2)$	10.6%	14.0%	+31%
$sin(\Theta_2)$	10.9%	10.0%	−8%
$\omega(\Theta_1)$	10.8%	6.8%	−37%
$\omega(\Theta_2)$	10.0%	2.8%	−71%

ω: angular velocity

5 Discussion

Winning tickets seem to occur across all combinations of environments, algorithms and pruning strategies. The random ticket does not match the winning ticket's performance in any of the experiments. Some cases are a lot more convincing than others, however, and performance can be especially close when unstructured iterative pruning strategies are used while re-training on the *acrobot* problem.

Neural networks using only the mask of the winning tickets and not the weights reach similar levels of sparsity as the full winning ticket before dropping off in performance on the *cartpole* problem. In the *acrobot* environment they often drop below the threshold quicker, but then maintain a similar rate of performance degradation as the winning ticket. This is largely in line with recent findings by Vischer et al. (2021) demonstrating that the mask explains most of the winning ticket effect for MLP-based RL agents such as the ones used in this project.

The relatively aggressive pruning of some input nodes compared to others (see Tables 4.3 and 4.4) suggests that the corresponding observations are more or less important in solving the problem. Vischer et al. (2021) have used the same kind of differences in input layer pruning to form 'minimal task representations'; interestingly, they found a similar pattern for *cartpole* - preference given to maintaining connections related to pole angle and angular velocity - but a very different pattern for *acrobot*: they found that the input nodes for angular velocity were best maintained, whereas in this project those were the ones most heavily pruned.
Part of the reason for this difference could be the much smaller networks and lower amount of training steps used here compared to that paper.

Further research could vary the neural network architecture within the same RL algorithm to see whether this has an impact on the presence of winning tickets. Another interesting line of inquiry would be to see whether winning tickets and mask-only subnetworks for DQV can be de-coupled, e.g. seeing how switching out the state-value network mask and keeping the state-action value mask would affect performance.

References

Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-13(5):834–846, 1983. ISSN 2168-2909.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym. *arXiv:1606.01540 [cs]*, 2016.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neu-

ral networks. *International Conference on Learning Representations*, 2019.

Alborz Geramifard, Christoph Dann, Robert H. Klein, William Dabney, and Jonathan P. How. RLPy: A Value-Function-Based Reinforcement Learning Framework for Education and Research. *Journal of Machine Learning Research*, 16(46):1573–1578, 2015.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Michela Paganini and Jessica Zosa Forde. Bespoke vs. Prêt-à-Porter Lottery Tickets: Exploiting Mask Similarity for Trainable Sub-Network Finding. *arXiv preprint arXiv:2007.04091*, 2020.

Matthia Sabatelli, Gilles Louppe, Pierre Geurts, and Marco A. Wiering. The Deep Quality-Value Family of Deep Reinforcement Learning Algorithms. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2020.

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

Richard S Sutton. Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding. *NeurIPS Proceedings*, 1996.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2018.

Hado van Hasselt. Double Q-learning. *Advances in Neural Information Processing Systems*, 23: 2613–2621, 2010.

Hado van Hasselt, Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

Marc Aurel Vischer, Robert Tjarko Lange, and Henning Sprekeler. On Lottery Tickets and Minimal Task Representations in Deep Reinforcement Learning. *arXiv preprint arXiv:2105.01648*, 2021.

C. J. C. H. Watkins. *Learning from Delayed Rewards*. PhD thesis, University of Cambridge, 1989.

C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3–4):279–292, 1992.

Marco A. Wiering. QV(λ)-learning: A New On-policy Reinforcement Learning Algorithm. *Proceedings of the 7th European Workshop on Reinforcement Learning*, pages 17–18, 2005.

Haonan Yu, Sergey Edunov, Yuandong Tian, and Ari S. Morcos. Playing the lottery with rewards and multiple languages: lottery tickets in RL and NLP. *arXiv preprint arXiv:1906.02768*, 2020.

A Appendix - Implementation Details and Hyperparameters

Table A.1: ANN details.

	Cartpole	**Acrobot**
Activation function	hyperbolic tangent	ReLU
Input nodes	4	6
Hidden nodes	128	128
Output nodes (Q-net)	2	3
Output nodes (V-net)	1	1

Table A.2: Hyperparameters.

	Cartpole	Acrobot
Action selection	epsilon-greedy	epsilon-greedy
Discount factor	0.99	0.99
Learning rate	0.0025	0.001
Optimizer	Adam	Adam
Epsilon (start)	1	1
Epsilon decay factor	0.95 / episode	0.999 / step
Epsilon (minimum)	0.01	0.01
Replay batch size	32	64
Replay memory size	1000	2000
Replay start	200 transitions	1000 transitions
Target network update	every episode	every episode
Training episodes[1]	max. 1000	max. 500
Early stopping threshold	200	-100

[1] Early stopping was used to save time and computational resources. Early stopping occurs after 10 consecutive training episodes where a cumulative score equal to or greater than the threshold mentioned in the table above.

B Appendix - Iterative Pruning Results

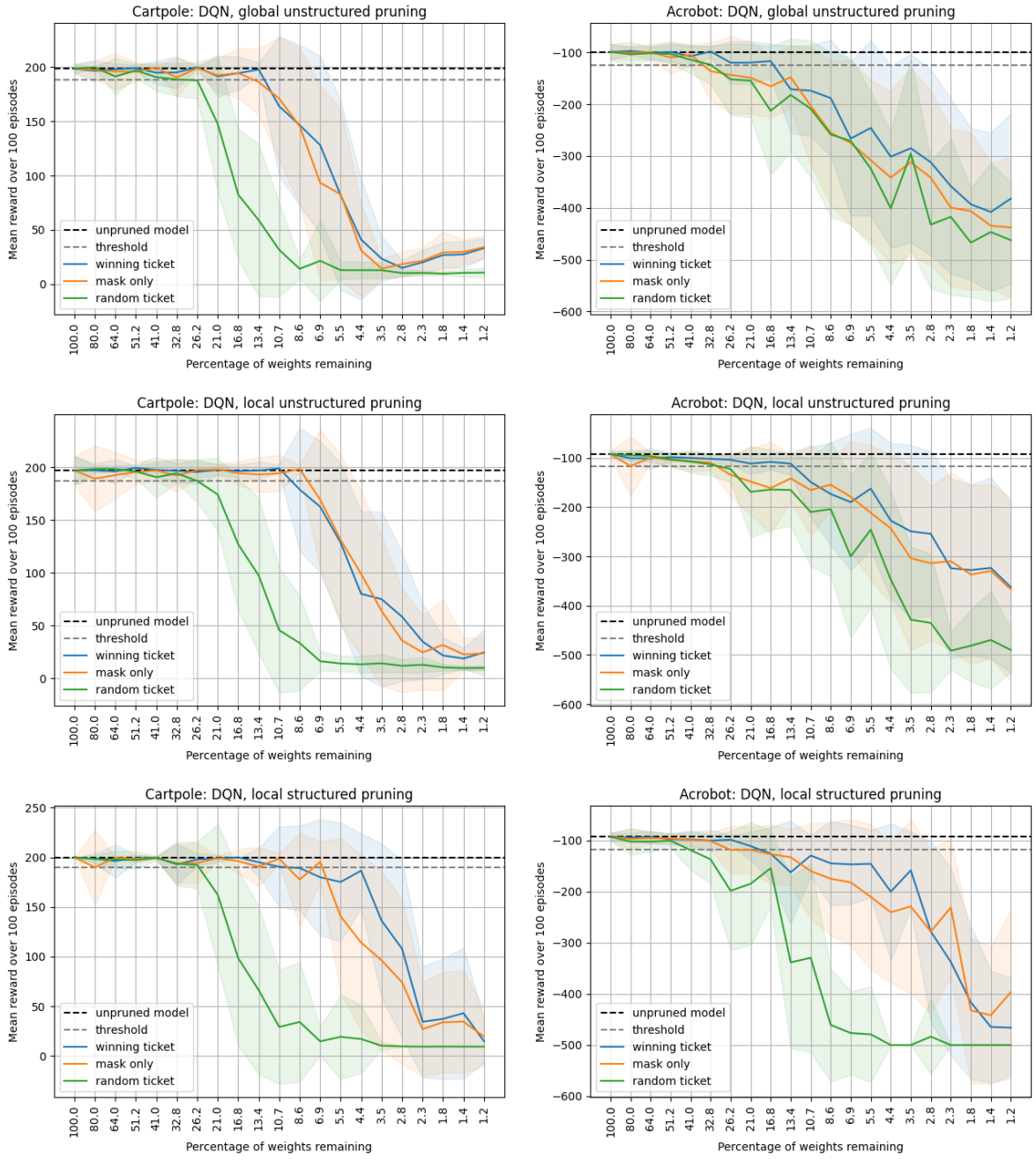

Figure B.1: DQN iterative pruning results

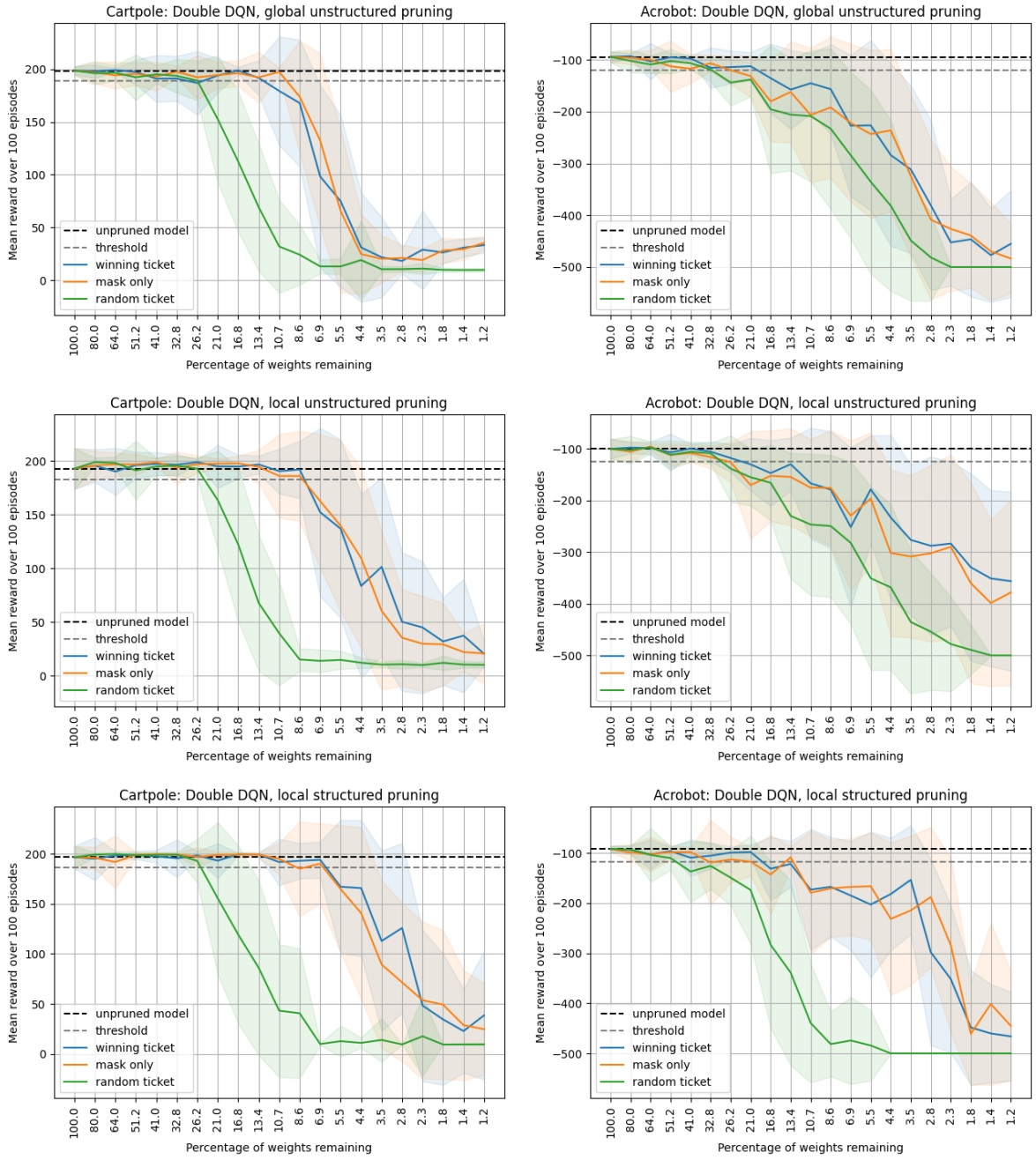

Figure B.2: Double DQN iterative pruning results

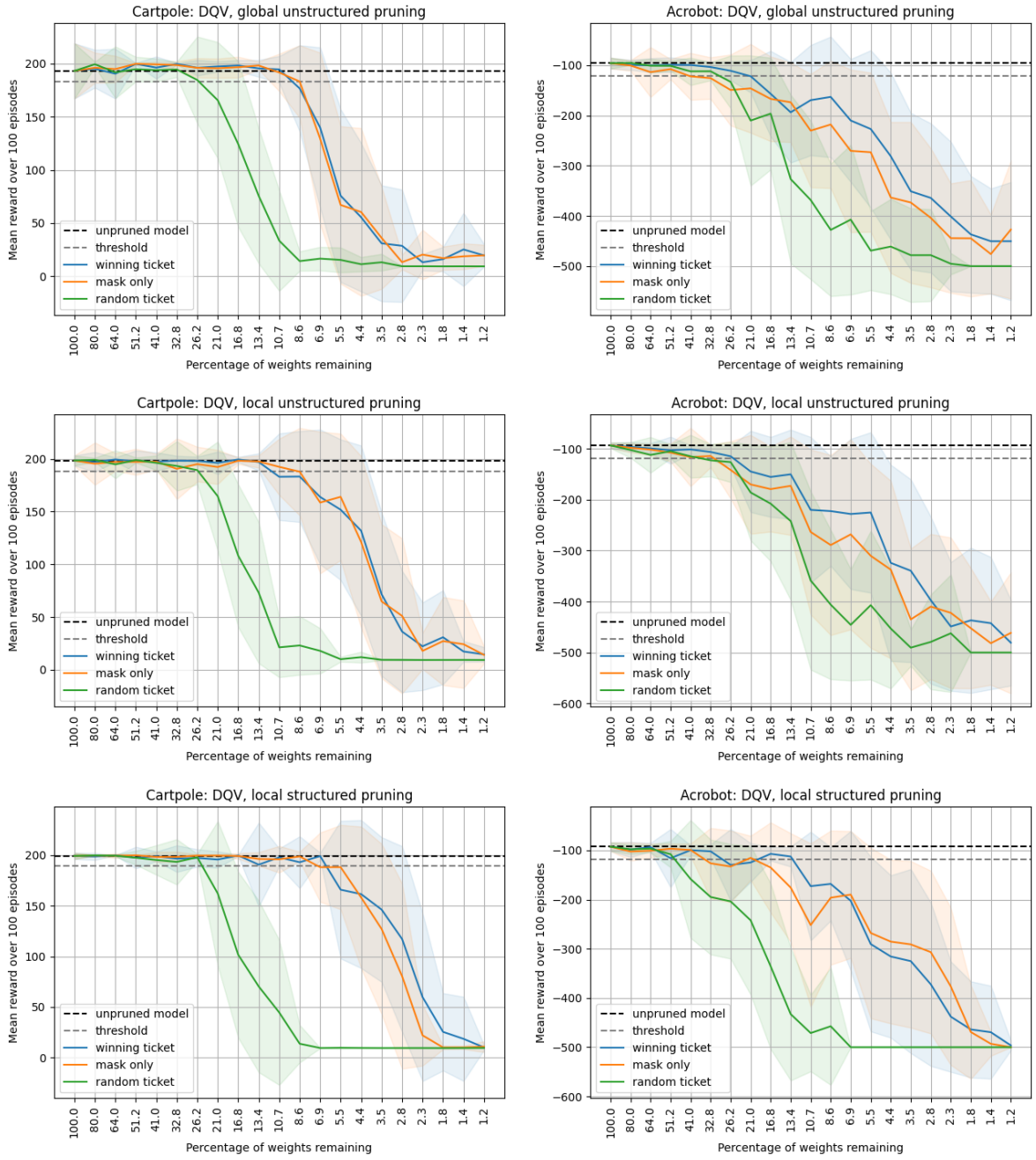

Figure B.3: DQV iterative pruning results