university of groningen / faculty of science and engineering

On the existence of Early-Bird tickets in Reinforcement Learning and how to find them

Bachelor's Project Thesis

Alexandru Dimofte, s4019199, a.dimofte@student.rug.nl,
Supervisor: Dr. Matthia Sabatelli

Abstract: The Winning Lottery ticket hypothesis has shown that a neural network can contain sparse sub-networks that perform similarly or better in comparison. This finding was extended by the Early-Bird ticket hypothesis, which revealed that winning lottery tickets can be found early within the first few initial training steps. This study demonstrates the existence of Early-Bird tickets in neural networks used in reinforcement learning and introduces EB Exploit - an algorithm which can reliably find EB tickets in simple RL problems, as well as increasing efficiency in the training process of such agents.

1 Introduction

The Lottery tickets hypothesis (Frankle & Carbin, 2018) has revealed that dense neural networks can contain sparser sub-networks which perform just as well, or better than the original over-parameterized network. Winning lottery tickets, as they are known, have been found consistently throughout other types of problems besides supervised learning, such as for instance natural language processing (T. Chen et al., 2020; Yu et al., 2019), self-supervised learning (T. Chen et al., 2021), as well as reinforcement learning (Vischer et al., 2021). Unfortunately, the process by which winning ticket are found, known as iterative magnitude pruning, proves to be tedious and computationally expensive, with the most inefficient training iteration being the very first one (You et al., 2019). To address the efficiency issue, You et al. have shown that winning lottery tickets can emerge within the first few epochs of a regular training process in a computer vision task. These sparse sub-networks were named Early-Bird tickets and they were used in the making of an efficient training procedure known as EB Train (You et al., 2019). Inspired by the success of this solution, X. Chen et al. (2020) created EarlyBERT - an efficient natural language processing algorithm which uses the same concepts as Early-Bird tickets.

In this study, Early-Bird tickets are explored further, this time in the context of simple Deep Reinforcement Learning (DRL) tasks. EB tickets rely on the assumption that the training process of a neural network begins first by learning which weights are most important, followed after by optimising these important weights (Achille et al., 2018; You et al., 2019). In contrast to supervised learning, this assumption has not yet been verified in reinforcement learning, which poses some distinct challenges when attempting to find EB tickets. An additional issue is the inherent change in the quality of states that the agent has access to as it learns. For example, in the early stages of learning, the agent may fail consistently and would therefore only have access to states more similar to the initial state. After some training time-steps, the agent may learn to utilize these actions to move within the environment, thus gaining access to states more complicated or more different than the initial state.

These changes in data mean that the learning process of a neural network used within deep reinforcement learning can often be more unstable than in supervised learning, and, as this paper will discuss, it seems that some weights may be more important during the early stages of learning, in the exploration phase, while being less important during the later exploitation phase. For all these reasons, the EB Train procedure cannot be directly applied to reinforcement learning and it is unclear whether EB tickets could be found at all.

To address these issues, this study was split in four parts which built onto each-other. First, the importance of the weights throughout training was visualized using the same methods as You et al. (2019) in subsection 3.3. This is followed with an analysis of the performance of possible EB tickets found at different stages throughout learning (subsections 3.4 and 3.5). Finally, EB Exploit is introduced in subsection 3.6, which is an algorithm that can find EB tickets in deep reinforcement learning, as well as train such agents more efficiently. All the notions necessary to understand this study are explained in the Background section (section 2).

The contributions of this study are the following:

- Showing that neural networks used in deep reinforcement learning learn which weights are most important in the first few episodes of the exploitation period;
- Demonstrating that EB Tickets exist in DRL and they are more efficient if finetuned for the exploitation period rather than retrained;
- A tool which can be used to find EB tickets and train DRL agents efficiently.

2 Background

2.1 Reinforcement Learning

Reinforcement learning is a family of problems in which an agent learns to interact with an environment. The interaction between the agent and the environment is reduced to a set of actions $\mathcal{A}$ that the agent can choose from, a set of states $\mathcal{S}$ and a set of rewards $\mathcal{R}$ given to the agent, for each action it takes. At each time-step t, the agent observes a state $s_t \in \mathcal{S}$ and selects an action $a_t \in \mathcal{A}$ for which the agent receives a reward $r_{t+1} \in \mathcal{R}$ and the environment transitions to a new state $s_{t+1} \in \mathcal{S}$ (Sutton & Barto, 2018).

An action is selected by the agent according to a policy π, which is a function that indicates the utility of each action $a \in \mathcal{A}$ based on the observed state $s \in \mathcal{S}$. This is denoted by $\pi(s)$. The goal is to learn an optimal policy π^* (Equation 2.1) that maximizes the expected return, which is the sum of rewards received for the duration of one episode.

$$\pi^* = \max_{\pi \in \Pi} \mathbb{E}\left[\sum_{t=0}^{t=T-1} r_{t+1}(s_{t+1} \mid s_t,\ \pi(s_t)) \right] \tag{2.1}$$

where Π is the set of all possible policies, $r_{t+1}(s_{t+1} \mid s_t,\ \pi(s_t))$ or (simply r_{t+1}) is the reward, received by transitioning from state s_t to state s_{t+1} via the action selected by the policy for the respective state $\pi(s_t)$ and T is the last state, also referred to as terminal state. The terminal state is either a state where the agent fails, or the last state allowed by the environment (e.g. winning state). Once a terminal state is reached, the environment is restarted and another episode commences (Sutton & Barto, 2018).

In this study, a policy uses a state-action value function $Q(s, a)$ (Equation 2.2) to describe the utility of an action a taken at some state s in terms of the expected sum of discounted future rewards.

$$Q_\pi(s, a) = \mathbb{E}\left[\sum_{t=t_0}^{t=T-1} \gamma^t r_{t+1} \,\middle|\, s_{t_0} = s,\ a_{t_0} = a;\ \pi \right] \tag{2.2}$$

where $\gamma \in (0, 1]$ is a discount factor which scales future rewards such that earlier rewards have a higher contribution. An optimal policy π^* is therefore, one which maximizes the state-action value function defined in Equation 2.2 (Sutton & Barto, 2018).

To reach an optimal policy π^*, a popular algorithm is that of Q-Learning (Dayan & Watkins, 1992), which learns a state-action value function by bootstrapping from previously made estimations of the same function (Equation 2.3).

$$\begin{aligned} Q(s_t, a_t) \leftarrow{} & Q(s_t, a_t) + \\ & + \alpha\big[r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a) - Q(s_t, a_t)\big] \end{aligned} \tag{2.3}$$

where α is the learning rate.

2.2 Deep Reinforcement Learning

Deep reinforcement learning methods make use of artificial neural networks in order to estimate utility functions. A popular DRL algorithm, and the one used for this study is that of DQN (Deep Q-Network) (Mnih et al., 2013), which expands on

Q-learning by using a neural network with parameters θ to approximate the state-action value function $Q(s, a)$. In order to update the parameters of the neural network, the utility update function (Equation 2.3) is turned into the following semi-differentiable loss function (Equation 2.4):

$$L(\theta) = \mathbb{E}_{\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle \sim U(D)} \Big[\big(r_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(s_{t+1}, a; \theta^-) - - Q(s_t, a_t; \theta)\big)^2 \Big] \tag{2.4}$$

This function requires a few more notions to understand. By introducing an approximator neural network into Q-learning, the training process becomes unstable, which led to two additional modifications (Mnih et al., 2013).

The first modification was the use of an experience replay buffer denoted by D. This buffer is a fixed-size queue of trajectories of the form $\langle s_t, a_t, r_{t+1}, s_{t+1}\rangle$ collected by the agent throughout the training episodes. The neural network is not, however, trained with all the trajectories in the order they were collected, since that would likely result in learning a few fixated steps. Instead, to decorrelate the trajectories, a batch of an arbitrary size is sampled at random from the replay buffer D, which is then used to train the network (Mnih et al., 2013).

The second addition was the use of a second neural network with parameters θ^-. At first, the target network, as it is called, is initialized as a copy of the online network with parameters θ (e.g. $\theta^- \leftarrow \theta_0$) and it is updated again after an arbitrary number of episodes (e.g. $\theta^- \leftarrow \theta_i$ at every ith episode), during which, θ is updated as usual via the state-action value loss (Equation 2.4) (Mnih et al., 2013).

2.3 Pruning of neural networks

A neural network approximating a function $f(x)$ with parameters θ is said to be pruned when a percentage P of its weights are removed. This is done by drawing a mask over the parameters $m \in \{0, 1\}^{|\theta|}$, and applying it to the parameters such that a 0 assigned to some weight w_{ij} renders the weight obsolete (pruned), while a 1 allows the weight to be utilised as usual (unpruned). Removing individual weights in this manner is known as *unstructured pruning*, but pruning can also be done by removing entire neuron units, utilising channels and other methods, which fall in the category of *structured pruning* (Blalock et al., 2020). Once a mask m is applied to the parameters of a neural network, they are denoted with $m \odot \theta$. Therefore, if a neural network is denoted with $f(x; \theta)$, then a sparse sub-network found within its architecture (a pruned neural network) is denoted with $f(x; m \odot \theta)$.

Additionally, in order to effectively prune a neural network, one must also choose which weights should be pruned. For this, a metric is used to designate which weights are least important, such that when pruning, the top $P\%$ least useful weights can be removed (Blalock et al., 2020). There are many metrics that can achieve this, but *L1 norm* has been found to achieve the best results. while pruning random weights is usually used as a baseline for comparison (Blalock et al., 2020; Frankle & Carbin, 2018; Vischer et al., 2021).

Pruning can also be done globally or locally. Both methods compare weights with each-other and aim to remove the $P\%$ least useful weights. The difference lies in which weights are being compared. Global pruning refers to a comparison between all the parameters in θ, while local pruning only compares parameters contained in a single layer, usually used to remove the $P\%$ least useful weights layer by layer (Blalock et al., 2020).

Finally, there are many pruning-training combinations that one can consider. For example, one could schedule the amount of weights to be pruned in different ways, like pruning all the desired $P\%$ weights at once, dividing the percentage into smaller pruning steps distributed throughout the training process or by varying the pruning percentage P throughout the training according to some function (Blalock et al., 2020). Likewise, one can choose how and when training may occur relative to when pruning happens. Methods here are focused on how to continue the training process, or more specifically, which parameters to use after pruning has occurred, for which variations include the following (Blalock et al., 2020):

- pruning at some epoch i and continuing training using parameters $m \odot \theta_i$ with no additional change, which will hence-forth be referred to as *fine-tuning*;
- rewinding the parameters to a previous state

$m \odot \theta_{i-n}$ referred to as *rewinding*;

- rewinding the parameters to the initial state $m \odot \theta_0$, referred to as *reinitializing*

2.4 Winning Lottery Tickets

The winning lottery tickets hypothesis states that any randomly-initialised feedforward neural network contains at least one sub-network which can match or outperform the testing performance of the original, unpruned network, while utilising less parameters and converging in at most the same number of training steps (Frankle & Carbin, 2018). The sub-networks, known as *winning lottery tickets* or simply, *winning tickets*, are found via the iterative magnitude pruning algorithm, which is outlined below in Algorithm 2.1.

Algorithm 2.1 Iterative Magnitude Pruning

```
1: Randomly initialize neural network $f(x;\theta_0)$
2: Train for $i$ iterations, resulting in parameters $\theta_i$
3: Prune $P\%$ of parameters, by drawing mask $m$
4: Reinitialize parameters to $\theta \leftarrow m \odot \theta_0$
5: Train for $i$ iterations, resulting in parameters $m \odot \theta'$
6: Prune $P\%$ of parameters, by drawing mask $m'$

7: Set $m \leftarrow m'$
8: Repeat steps 4 to 7 for $j$ times.
```

At step 2 and 5, the network is trained such that after i training steps, the network converges to a local minimum. Step 8 indicates that the process can be repeated as many times as it is needed to reach a desired sparsity, hence the "iterative" part of the algorithm's name.

2.5 Early-Bird Tickets

An Early-Bird ticket is a winning lottery ticket or sparse neural network akin to a winning lottery ticket, found within the first few training iterations, or epochs, of the learning process. The hypothesis states that for any neural network $f(x;\theta)$ which converges to a minimum validation loss f_{loss} in i training steps, with test accuracy f_{acc}, there is at least one sparse sub-network $f(x; m \odot \theta)$ which converges to a minimum validation loss f'_{loss} in t training steps with test accuracy f'_{acc} such that $t \ll i$ (e.g. early stopping) and $f'_{acc} \approx f_{acc}$ (or higher). In other words, the sparse sub-network converges much faster than the unpruned network, while maintaining a similar or better performance (You et al., 2019).

This hypothesis is inspired by empirical results which have revealed that neural networks learn important connections between neurons (weights) within the first few training steps. Having learned the most useful weights, their importance relative to each-other hardly changes throughout the remaining duration of the learning process (Achille et al., 2018), which means that a network's $P\%$ least important weights can be pruned within the first few epochs with a mask which would be similar to the mask taken at the end of the training (i.e. a winning ticket) (You et al., 2019).

To further analyse this hypothesis, You et al. drew masks from each epoch throughout the training process and computed the inverse normalized hamming distances (Equation 2.6) between all the masks with each-other ($h^{-1} \in [0, 1]$, where $h^{-1} = 1$ means the two masks are the same and $h^{-1} = 0$ means they are entirely different).

$$h = \frac{1}{n} \sum_{i}^{n} |m_{1,i} - m_{2,i}| \tag{2.5}$$

$$h^{-1} = 1 - h \tag{2.6}$$

where n is the total number of weights in the network and $m_{1,i} \in \{0,\ 1\}$ represents whether the i-th weight should be pruned according to mask m_1.

The result was a matrix of distances, with each number representing the inverse normalized hamming distance between a mask taken at some respective epoch n and another mask taken at another epoch m. This study used the same method to analyse the existence of EB tickets in reinforcement learning (see subsection 3.3).

Finally, the authors introduce EB Train, a method of training neural networks more efficiently via the finding of EB tickets. It differs from a usual training procedure mainly by the fact that after each epoch e, a mask is drawn and compared (via hamming distance) with the mask drew from the previous epoch $e - 1$. If their hamming distance is lower than a parameter ϵ (e.g. $\epsilon = 0.1$), then the network is pruned, reinitialised to its original parameters θ_0 and retrained to convergence.

3 Methods and Results

Four different experiments were employed, each of them building onto the previous ones.

This section first describes the environments used throughout the experiments (subsection 3.1), followed by an explanation of the training setup that all experiments shared (subsection 3.2), and finally, the individual experiments along with justifications for each choice made (subsections 3.3, 3.4, 3.5, 3.6).

3.1 Environments

The environments used were two classic control problems known as *Cartpole* (Barto et al., 1983) and *Acrobot* (Sutton, 1995). The specific versions of these environments that were used in this study were provided by the OpenAI Gym toolkit (Brockman et al., 2016).

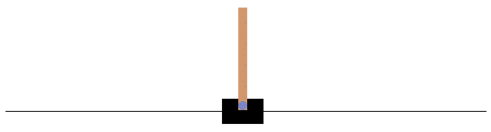


Figure 3.1: `CartPole-v0` state

The first environment, `CartPole-v0` is a pole balancing problem first introduced by Barto et al., which consists of a pole placed vertically on a cart that can move left or right (see Figure 3.1). The aim is to learn a set of movements for the cart which balance the pole vertically (with an allowed deviation from a vertical position of 15 degrees) for as long as possible without moving the cart too far left or right from the central starting position. `CartPole-v0` is a simplified version of the original cartpole problem which does not take into account cart friction and runs for a total of 200 time-steps. At each time-step, the agent has access to an observation (state) of four floating point numbers: cart position, cart velocity, pole angle and pole angular velocity. Using this information, the agent must decide whether to move the cart towards the left or right side. For each time-step in which the pole is within the allowed inclination interval, the agent receives a reward of +1. As soon as the agent fails, either by moving too far off the screen or by letting the pole fall further than 15 degrees, the reward is 0 and the environment is restarted with a fully vertical pole.

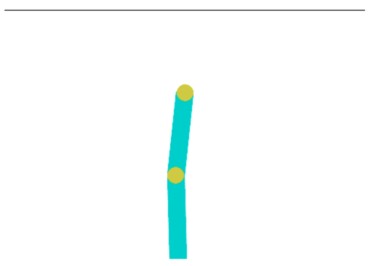


Figure 3.2: `Acrobot-v1` state

The second environment, `Acrobot-v1` uses the implementation of Geramifard et al. (2015) which consists of two links connected via a joint to form a chain. The first link has an additional joint at the first end, which is fixated near the center of the window (see Figure 3.2). This joint is actuated such that the agent can apply positive torque (+1), negative torque (−1) or no torque (0). The goal is to learn how to apply a set of torques on the actuated joint such that the end of the chain reaches the top bar in as little time-steps as possible.

To this end, the agent has access to an observational space consisting of 6 float number: the cosine, sine and velocity of the first link's angle, such that an angle of 0 means pointing straight downwards; as well as the cosine, sine and velocity of the second link's angle relative to the first, such that an angle of 0 means the chain forms a straight line.

An episode in the `Acrobot-v1` environment lasts a total of 500 time-steps, during which the agent receives a null reward ($r = 0$) for a transition to a winning state, and a negative reward ($r = -1$) otherwise.

3.2 Training setup

All experiments used the DQN algorithm (see subsection 2.2) training first on `CartPole-v0` and then on `Acrobot-v1` (see subsection 3.1). For each environment, the hyper-parameters of DQN were tweaked in order to reach acceptable performance. Acceptable performance in this case means reaching a total accumulated reward of 200 in `CartPole-v0` and -100 or higher in `Acrobot-v1`. A detailed view of the specific DQN hyperparameters used can be seen in appendix B.

The architecture of the Q-Networks used in DQN consists entirely of fully-connected layers. All hidden layers used 128 neurons, while the input layers used as many neurons as the observation space needed (e.g. for an observation space of 6 float numbers, 6 neurons were used). Likewise, the output layers had as many neurons as there are actions allowed, 2 neurons for cartpole and 3 neurons for acrobot. All layers, except the output layer, used *ReLu* (Agarap, 2018) as their activation function. In the cartpole problem, two hidden layers were used, and only one in acrobot. This was done in order to achieve acceptable performance as fast as possible in both environments, and also serves to compare different architectures. For an overview of the implementation details regarding the neural networks architecture, please consult appendix A.

Unless explicitly stated otherwise, early-stopping was also employed to stop training once the algorithm converged in order to cut unnecessary or damaging additional training time. The training process was stopped after 10 consecutive episodes in which the agent reached acceptable performance.

Finally, all the experiments involved pruning and drawing masks. The following paragraphs describe how these were done throughout the experiments according to the criteria outlined in subsection 2.3, along with a brief justification for each choice made. These choices applied for drawing masks both with and without pruning.

Pruning structure: in both supervised and reinforcement learning, structured pruning has often been found to lead to worse results, due to removing too many weights at once (Frankle & Carbin, 2018; Vischer et al., 2021). For this reason, unstructured pruning is used throughout the experiments.

Weight importance metric: as discussed in subsection 2.3, L1 norm has been found to achieve the best results, and therefore it was the metric used in this study.

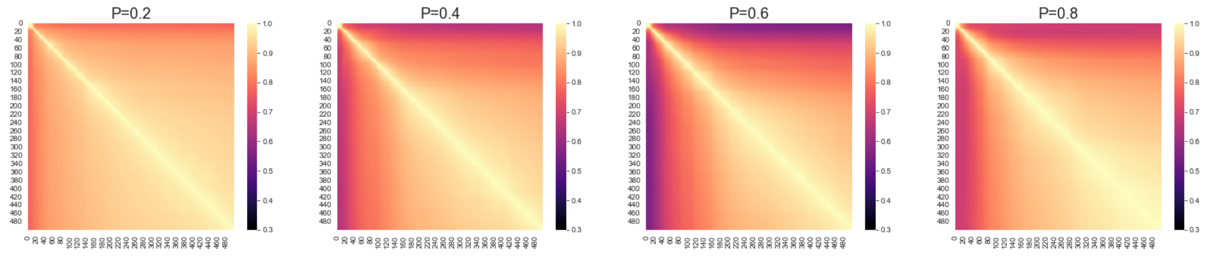
Local vs Global pruning: local pruning has been found to be too aggressive, which led to worse results overall when compared to global pruning (Frankle & Carbin, 2018; Vischer et al., 2021). Therefore global pruning was used in this study.

3.3 Experiment 1 Mask Similarity

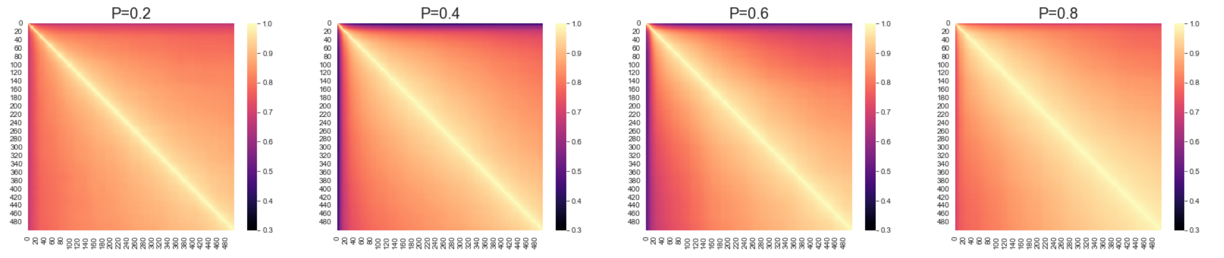
As explained in subsection 2.5, EB Tickets rely on the assumption that a neural network learns which weights are more important than others early in the training (You et al., 2019). This has been empirically shown by Achille et al. in the context of a supervised learning problem, however it is not obvious whether the hypothesis holds for neural networks used in reinforcement learning.

This assumption does not need to be directly verified though, since it would be enough to verify whether masks drawn at different episodes throughout the training are similar to masks drawn later in the training. This would offer a way to visualize whether the importance of weights relative to eachother changes significantly throughout the training process.

To achieve this, an agent was trained for 500 episodes without early-stopping. Every 2 episodes, a mask m_e was drawn from the network's parameters at the respective episode θ_e, such that a set percentage of weights P was deemed pruned by the mask. Following each mask draw, the network continued being trained using the unpruned parameters θ_e. Once training had finished, the inverse normalized hamming distance H between all the masks was calculated, which resulted in a distance matrix displaying similarities between masks drawn at different episodes. This experiment was repeated 10 times across different pruning percentages, $P \in \{0.2,\ 0.4,\ 0.6,\ 0.8\}$. The outcome was a total of 40 mask distance matrices for each environment (80 in total) which were aggregated by taking the average per environment, per pruning percentage P used across the 10 repetitions of the experiment.



(a) Cartpole



(b) Acrobot

Figure 3.3: Experiment 1 results - Distance matrices averaged over 10 experiments for each environment, and each pruning percentage $\mathbf{P} \in \{\mathbf{0.2},\ \mathbf{0.4},\ \mathbf{0.6},\ \mathbf{0.8}\}$.

Mask similarity results

All 8 distance matrices resulting from the aggregating process described above can be visualized in Figure 3.3, displaying one mask distance matrix for each environment and each pruning percentage P as a heatmap. Each value in this heatmap represents the inverse normalized hamming distance between two masks taken at two respective episodes.

The heatmaps reveal bright square formations, darkening quickly at the upper and left-most edges. The bright square formations represent periods in learning when the top $(1 - P) * 100\%$ of weights do not change significantly. It can be seen that especially in the cartpole problem (Figure 3.3a), the longer the training progresses, the smaller the difference between masks becomes.

Interestingly, in both environments, the third heatmap ($P = 0.6$) shows larger differences between masks taken at later episodes and those taken earlier. However, the fourth heatmap ($P = 0.8$) shows much smaller differences and it is more comparable to the first two heatmaps ($P = 0.2$ and $P = 0.4$). This suggests that the largest changes in relative importance that weights undergo throughout their training lie within the range of $[20\%,\ 40\%]$ most important weights, this range constituting of a flux of different weights changing continuously until the network converges. To better explain this, it could be thought of within the metaphor of brain plasticity (Kolb & Whishaw, 1998). It seems that the most plastic part of a neural network may be what constitutes the most important 20% to 40% of weights within the first few episodes. This plasticity is also most pronounced within the first early stages of learning, quickly slowing down as the network learns.

What's curious is that the masks taken approximately within episodes 10 to 90 during the cartpole task seem to be similar to each-other, but different compared to masks drawn later in the training. This is indicated in Figure 3.3a by the bright square formations formed around these initial episodes, which are faint but present nevertheless. The effect is most visible in the plots with $P \in \{0.6,\ 0.8\}$. Furthermore, these episodes coincide exactly with the initial exploration phase, during which ϵ in the ϵ-greedy policy is gradually decaying. These suggest the existence of a small subset of weights (about $20\% - 40\%$ most important) which pose a crucial role in the development of the exploration phase, but are not as useful during exploitation. Furthermore, the network learns this hierarchy of weight importance early in the exploration period. The reason why this is not visible in the acrobot heatmaps (Figure 3.3b) may be due to the fact that the episodes are much longer initially than in cartpole, and since ϵ decays per step, the exploration period lasts around 5-10 episodes, which is not enough to visualize the mask distances taken across every two episodes.

Together, these results show that neural networks used in DRL learn which weights are most important within the first few episodes in the exploitation phase, with their relative importance no longer changing significantly throughout the remaining exploitation episodes. Hints can also be seen that this process may happen in the exploration phase as well, however the current results do not offer a decisive answer for this.

3.4 Experiment 2 Retraining EB Tickets

The results of experiment 1 (subsection 3.3) laid the foundation for testing the existence of EB Tickets in DRL in terms of performance. Since masks drawn in the beginning of the training seem to be similar to masks drawn later, this experiment continued the search for EB Tickets by verifying the overall performance and training length of retrained sparse sub-networks drawn at different episodes throughout the training. These were then compared to unpruned networks and winning tickets in order to verify whether EB tickets were found.

For a sparse sub-network to be deemed an EB ticket it must fulfil the following conditions (see subsection 2.5):

1. Similar performance as an unpruned network or higher;
2. Train for a similar amount of episodes as an unpruned network or less (e.g. via early-stopping);
3. Similar performance and training length to a winning lottery ticket;
4. Be pruned at an earlier episode than a winning lottery ticket.

Each instance of this experiment began by training a DQN agent without early stopping such that the Q-Network with parameters θ could be pruned at some episode e_{prune} chosen arbitrarily. Following the pruning, the Q-Network was reinitialized to its initial parameters θ_0 and retrained using $m \odot \theta_0$ with early-stopping for a maximum of 500 episodes. Finally, once training finished, the agent was tested on 100 episodes.

All the data necessary to find EB tickets was collected from 10 repetitions of this experiment for each environment, over different pruning percentages $P \in \{0.2,\ 0.4,\ 0.6,\ 0.8\}$ across 12 different episodes at which to prune e_{prune}, focused mostly in the early episodes ($e_{prune} \in \{5, 20, 30, 40, 50, 75, 100, 130, 180, 250, 400, 500\}$.

The result was 120 training-retraining-testing "sessions" for each of the 2 environments and for each pruning percentage P (a total number of $120 * 2 * 4 = 960$ sessions).

In order to perform the comparisons with unpruned networks necessary to verify the 1st and 2nd criteria, 10 individual training iterations were performed with early-stopping for each environment using an unpruned Q-Network.

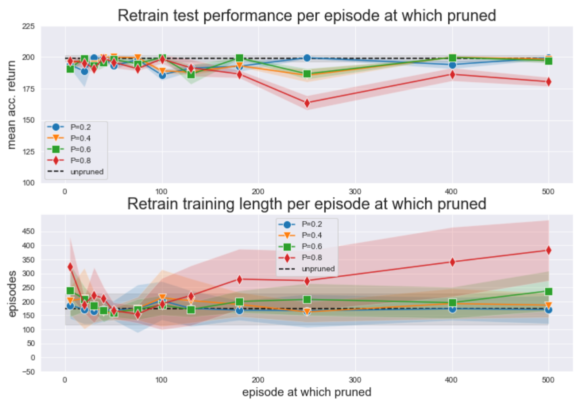
In addition, 10 winning lottery tickets were found for each pruning percentage P via magnitude pruning at the end of the training process. This is equivalent to performing iterative magnitude pruning with only 1 iteration. If EB tickets are present at the beginning of the training process, they should be comparable in terms of performance and retrain length to the one-time magnitude pruned winning tickets, which would satisfy the 3rd criteria.

Regarding the 4th and final criteria, winning lottery tickets are always pruned once a full network reaches early-stopping, which therefore means all EB tickets must be pruned before that.

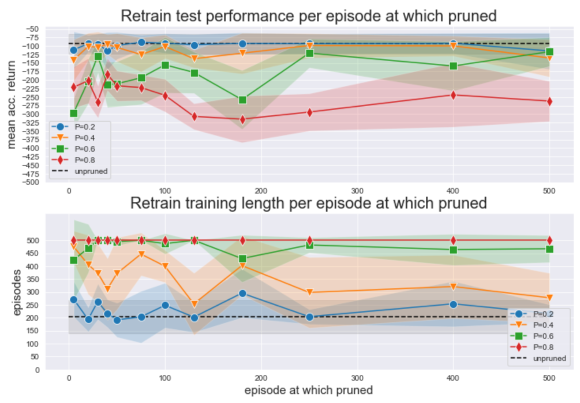
Retrain results

All figures resulted from this experiment contain two graphs. The top graphs represent the testing performance of retrained networks pruned at some episode represented along the x axis. The bottom graphs represent the amount of episodes it took during retraining to reach early-stopping.

A comparison of performance and retrain length between different pruning percentages can be seen in Figure 3.4, which displays the results for the cartpole environment in Figure 3.4a and those for acrobot in Figure 3.4b. Both figures compare the results of the pruned networks to the results of the unpruned ones.



(a) Cartpole retrain results



(b) Acrobot retrain results

Figure 3.4: Experiment 2 results - retrain test performance and retrain length per episode at which pruned.

In Figure 3.4a, it can be seen that during the cartpole task, neural networks pruned approximately within the episodes 25 to 100 achieve both a similar test performance compared to an unpruned neural network (top graph), while also training in a similar amount of time (bottom graph), something which holds for all pruning percentages P attempted. For networks pruned in episodes outside this interval, the effect still holds, however not for networks which were pruned with $P = 0.6$ or $P = 0.8$.

For Acrobot (Figure 3.4b), it is only the networks pruned with $P = 0.2$ that achieve both similar test performance and similar retraining length. As expected, this holds for most networks pruned at episodes in the exploitation phase, but not as much for those pruned during exploration (e.g. 5th episode), since they under-performed both in test performance and training length. As for other pruning percentages, the effect seems to degrade proportionally to the amount of weights pruned (i.e. larger P, weaker effect). For instance, networks pruned with $P = 0.8$ resulted in overall poor performance during testing, while also never achieving early-stopping.

Figures 3.5 (cartpole) and 3.6 (acrobot) compare the retrain results of networks pruned at different episodes (in yellow), to the results of the winning tickets (black dashed lines). In cartpole, across all pruning percentages except $P = 0.8$, the results follow closely those of the winning tickets, both in terms of test performance, as well as retrain length. This trend also continues for networks pruned with $P = 0.8$ but only those pruned within episodes 40 and 100. The same cannot be said for the acrobot results, since they only match the winning tickets for the networks pruned with $P = 0.2$.

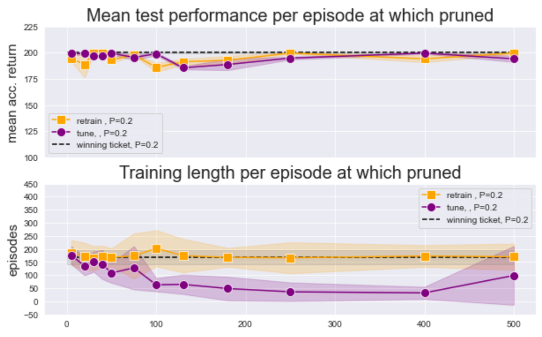
Nevertheless, these results fit all 4 criteria outlined above, and thus confirm the existance of EB tickets across all pruning percentages in the cartpole environment, and for $P = 0.2$ in acrobot.

3.5 Experiment 3 Finetuning EB Tickets

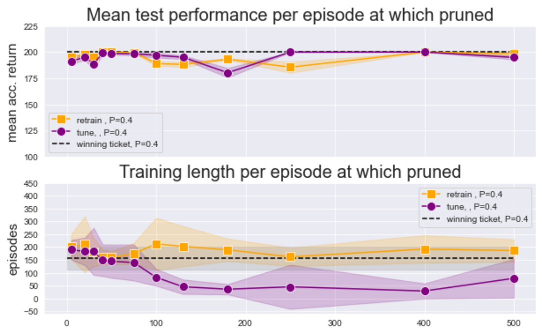
This experiment draws inspiration from the results of experiment 1 and 2. More specifically, as discussed in the results of subsection 3.3, the importance of the weights relative to each-other appears to be different between the initial exploration phase and the exploitation phase. With this assumption, reinitializing a network to parameters $m \odot \theta_0$, with a mask taken at some episode during exploitation, could destabilize the relationship between the exploration and exploitation phases during retraining.

To fix this, finetuning was considered as an alternative to reinitialization, the idea being that finetuning after pruning within exploitation would simply continue training within the same phase where the importance of the weights does not change significantly, and may thus converge faster.

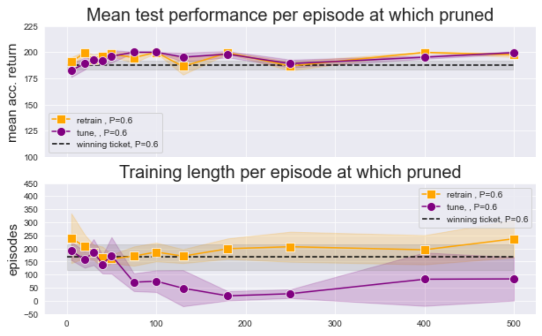
This experiment was designed the same on all aspects as experiment 2, with the only difference



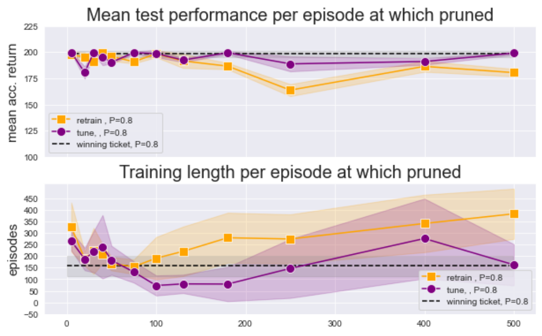
(a) $P = 0.2$



(b) $P = 0.4$

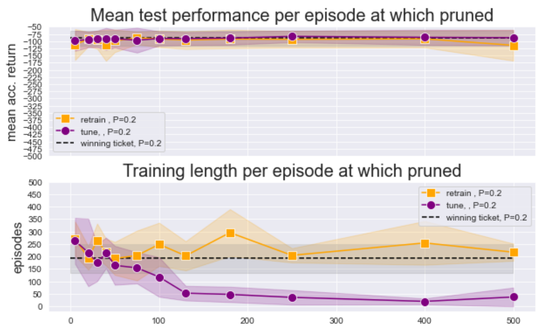


(c) $P = 0.6$

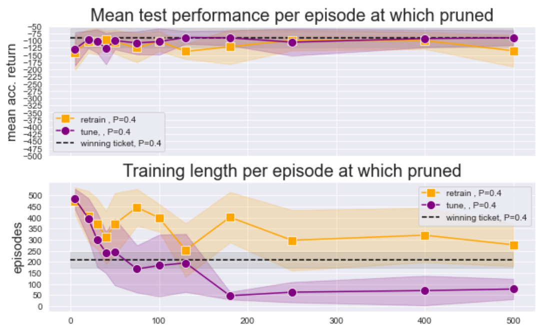


(d) $P = 0.8$

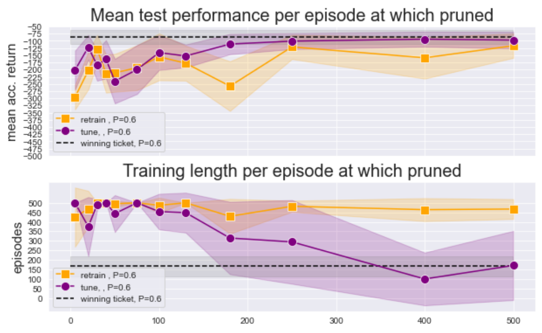
Figure 3.5: Cartpole - comparison of retrained networks (yellw), with finetuned networks (purple) and winning tickets (dash black lines)



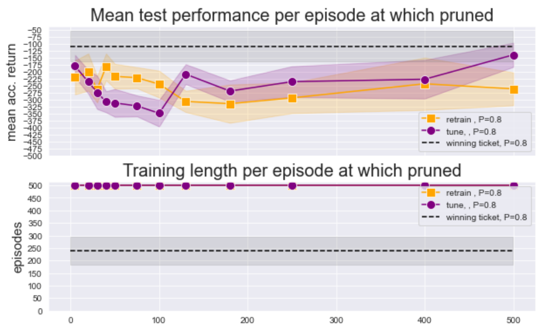
(a) $P = 0.2$



(b) $P = 0.4$



(c) $P = 0.6$



(d) $P = 0.8$

Figure 3.6: Acrobot - comparison of retrained networks (yellow), with finetuned networks (purple) and winning tickets (dash black lines)

being that once the Q-Network was pruned at an episode $n = e_{prune}$, rather than reinitializing to the original parameters θ_0, the agent was finetuned from the latest parameters with the mask applied, $m \odot \theta_n$ until early-stopping for a maximum number of episodes of 500 episodes.

Like before, the results of the finetuned networks were compared first to unpruned networks and then to winning lottery tickets found by retraining. In addition, they were also compared to the retrained networks found in the previous experiment. The expectation was that finetuning from the moment of pruning would require significantly less episodes to converge compared to retraining.

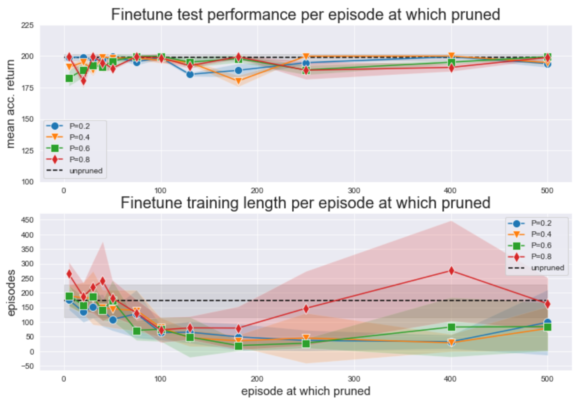
FineTune results

Like in the retrain experiment (subsection 3.4), all figures are composed of two graphs, the top graph showing the average test performance upon finetuning or retraining a network pruned at some episode indicated on the x axis. The bottom graph represents the amount of episodes required to reach early-stopping, from the moment of pruning, as a function of episode at which pruned.

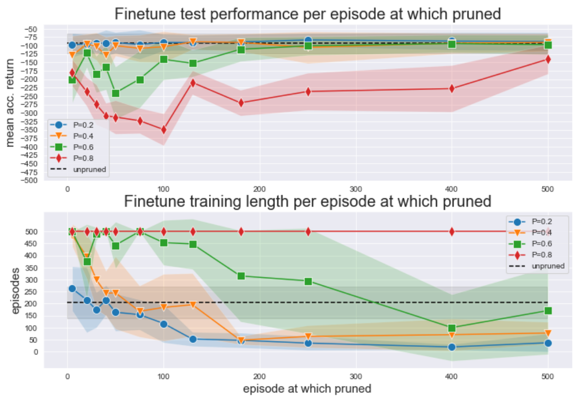
Figure 3.7 shows a comparison across the different pruning percentages P with the unpruned networks (3.7a for cartpole and 3.7b for acrobot). In cartpole, the test performance is maintained consistently close to an unpruned network, while requiring much less training time overall. In acrobot, the test performance is similar to the unpruned networks across all episodes for $P \in \{0.2,\ 0.4\}$, while also achieving similar or faster training time compared to the unpruned networks. Like it is the case with retraining (Figure 3.4b), the effect weakens with each increase in pruning percentage P.

Figure 3.5 and Figure 3.6 show a comparison between finetuned networks, retrained networks and winning lottery tickets, for the cartpole environment and, respectively, for acrobot. Here, a noticeable improvement in training length when finetuning can be seen across all results except for those of acrobot with $P = 0.8$ (Figure 3.6d), which never achieve early-stopping.

For cartpole, nearly all finetuned sparse neural networks pruned before the 200th episode require much less training steps than the unpruned networks, winning lottery tickets as well as the retrained EB tickets. All this while also achieving similar test performance to all three (or higher in Figure 3.5c). There is, however, more volatility when pruning within the first 50 episodes, which can be explained by the fact that masks drawn at those episodes are different than the masks drawn during exploitation, as shown by the results of experiment 1 (subsection 3.3). Nevertheless, EB tickets seem to be widely present in this environment across all pruning percentages.



(a) Cartpole finetuning results



(b) Acrobot finetuning results

Figure 3.7: Experiment 3 results - finetuning test performance and finetune length per episode at which pruned.

As for the acrobot results, the largest difference lies in the training lengths. Finetuning allows a network pruned with $P = 0.2$ or $P = 0.4$ to train in a similar amount of time or faster compared to both an unpruned network (Figure 3.7b) and a winning lottery ticket (subfigures 3.6a and 3.6b). This means that pruning early with $P = 0.2$ and $P = 0.4$

and finetuning can lead to an EB ticket, but pruning more aggressively can not.

3.6 Experiment 4 EB Exploit algorithm

Using the knowledge gained from the experiments above, this paper introduces *EB Exploit*, which is a simplified and modified version of EB Train, adapted to DRL. This algorithm is described below in algorithm 3.1, and it can be used to find and train EB tickets in DRL.

Algorithm 3.1 EB Exploit

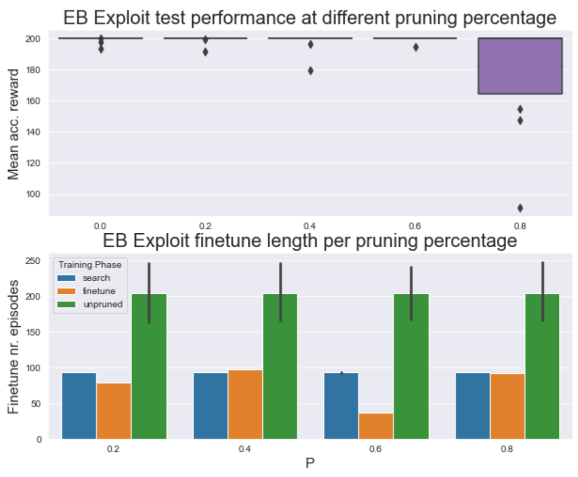
```
1: Randomly initialize neural network $f(x;\theta)$ used in a DRL agent;
2: Train normally until the end of the exploration period (e.g. $\epsilon == \epsilon_{min}$);
3: While training, draw masks at a window of $n$ episodes;
4: Compute the normalized hamming distance $h$ between each two consecutive masks (Equation 2.5);
5: If $h < \xi$ where $\xi \in (0,1)$, prune using the latest mask;
6: Finetune network with parameters $f(x; m \odot \theta_0)$ until early-stopping.
```

For the purpose of testing the algorithm's efficacy, as well as giving an example, EB Exploit with $\xi = 0.1$ was ran 10 times for each pruning percentage $P \in \{0.2,\ 0.4,\ 0.6,\ 0.8\}$ in each environments, resulting in a total of 80 runs. Like in the previous experiment, the resulting sparse networks were tested in terms of test performance and training length after finetuning.

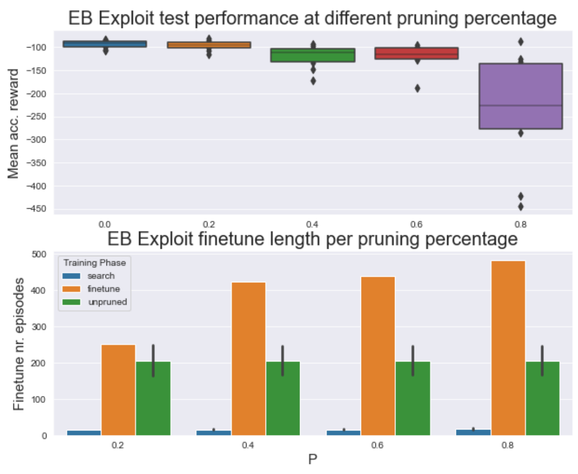
EB Exploit results

Figure 3.8 contains the results of this experiment, Figure 3.8a for cartpole results and Figure 3.8b for acrobot. Both sub-figures contain two graphs. The top graph displays the average test performance of an agent trained with EB Exploit at different pruning percentages. $P = 0.0$ indicates an unpruned network and acts as the comparison baseline. The bottom graph shows the average training length of different phases. The 'search' phase consists of all episodes within steps 1, 2, 3 and 4 in algorithm 3.1. The 'finetune' phase consists of the training length from step 5 and 6. For comparison, the training length of a unpruned network is also displayed.

For cartpole (Figure 3.8a), the test performance is in most cases acceptable, while also training in much less time compared to an unpruned network, even when 60% of the network was pruned. Pruning with $P = 0.8$ however, leads most often to lower performance.



(a) Cartpole EB Exploit results



(b) Acrobot EB Exploit results

Figure 3.8: Experiment 4 results - EB Exploit test performance and length per episode at which pruned.

For acrobot (Figure 3.8b), while the test performance stays relatively high for all pruning percentages except $P = 0.8$, the length of training is often higher than that of an unpruned network. With that said, the increased volatility of the results in the acrobot environment were to be expected given the results of the previous experiments. Furthermore the algorithm does find EB tickets for $P = 0.2$. It should also be noted that this experiment used a relatively high ξ value, which is why pruning is done so early in acrobot. Comparing this to Figure 3.7b, where EB tickets were found by pruning at later episodes (e.g. 50 to 100), it seems that lowering ξ may lead to better tickets.

4 Discussion

In a series of three experiments, this study demonstrated the existence of Early Bird tickets in Deep Reinforcement learning. The EB tickets were first retrained and then finetuned, which revealed that finetuning EB tickets is much more efficient both in terms of training length and performance. Comparing them to winning lottery tickets revealed similar or higher performance with faster training time, especially in the case of finetuned EB tickets.

Using this knowledge, EB Exploit was designed and tested in its own experiment, which revealed the fact that DRL agents can indeed be trained efficiently, as long as one of its parameters, ξ, is tuned appropriately.

With the above being said, this study suffers from limitations which stem from the lack of variety in algorithms, environments and neural network architectures. The effect was only tested using DQN on two environments that do not require complex neural networks. Because of that, the EB tickets effect was tested only on MLPs.

Moreover, throughout all experiments, large differences were observed between the results of the two environments. While the environments are very different, part of the difference could also be attributed to the implementation of the Q-Networks. As mentioned above, the network used in cartpole used two hidden layers rather than just one, like in the acrobot problem. Pruning a higher percentage of the weights in cartpole, therefore, would leave the network with more weights than in acrobot.

Future studies could extend this research to other types of DRL algorithms, first within the same temporal-difference learning class of reinforcement learning algorithms, and then to others, such as for instance policy gradient methods. A further extension could be to examine EB tickets in convolutional layers across more environments, as well as other neural network architectures.

Finally, this study also demonstrates that the training process of neural networks used in DRL consists of learning which weights are more important than others within the first few early episodes, followed by no significant changes in this importance hierarchy. In addition, the first experiment hints at two possible effects. First, it seems that learning the importance of weights happens two separate times within DRL: in the exploration phase and in the exploitation phase. Second, as discussed in the results of this experiment, it seems that the weights which fall within the most important [20%, 40%] hierarchy range is changing much more rapidly than within other ranges, with weights entering this range and leaving swiftly. In the respective subsection, the metaphor of brain plasticity was used in mentioning that this range of weights is more plastic than others. Future research could investigate a possible similarity between the change in weight importance across training and the change in brain plasticity present in the human brain as the human ages. Just like human brain plasticity decreases as the human ages (Kramer et al., 2004), weights importance seems to stop changing significantly as the network is trained. Furthermore, critical learning periods in reinforcement learning could be analyzed more directly using similar methods as Achille et al. (2018).

In conclusion, the key takeaways of this study are that EB tickets seem to exist in deep reinforcement learning, along with the main assumption of how neural networks learn weight importance; EB tickets are much more efficient when finetuned rather than when they are retrained and EB Exploit can be used to find EB tickets, as well as to train reinforcement learning agents more efficiently.

References

Achille, A., Rovere, M., & Soatto, S. (2018). Critical learning periods in deep networks. In *International conference on learning representations.*

Agarap, A. F. (2018). Deep learning using rectified linear units (relu). *arXiv preprint* *arXiv:1803.08375*.

Barto, A. G., Sutton, R. S., & Anderson, C. W. (1983). Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE transactions on systems, man, and cybernetics*(5), 834–846.

Blalock, D., Gonzalez Ortiz, J. J., Frankle, J., & Guttag, J. (2020). What is the state of neural network pruning? *Proceedings of machine learning and systems*, *2*, 129–146.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Chen, T., Frankle, J., Chang, S., Liu, S., Zhang, Y., Carbin, M., & Wang, Z. (2021). The lottery tickets hypothesis for supervised and self-supervised pre-training in computer vision models. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 16306–16316).

Chen, T., Frankle, J., Chang, S., Liu, S., Zhang, Y., Wang, Z., & Carbin, M. (2020). The lottery ticket hypothesis for pre-trained bert networks. *Advances in neural information processing systems*, *33*, 15834–15846.

Chen, X., Cheng, Y., Wang, S., Gan, Z., Wang, Z., & Liu, J. (2020). Earlybert: Efficient bert training via early-bird lottery tickets. *arXiv preprint arXiv:2101.00063*.

Dayan, P., & Watkins, C. (1992). Q-learning. *Machine learning*, *8*(3), 279–292.

Frankle, J., & Carbin, M. (2018). The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

Geramifard, A., Dann, C., Klein, R. H., Dabney, W., & How, J. P. (2015). Rlpy: a value-function-based reinforcement learning framework for education and research. *J. Mach. Learn. Res.*, *16*(1), 1573–1578.

Kolb, B., & Whishaw, I. Q. (1998). Brain plasticity and behavior. *Annual review of psychology*, *49*(1), 43–64.

Kramer, A. F., Bherer, L., Colcombe, S. J., Dong, W., & Greenough, W. T. (2004). Environmental influences on cognitive and brain plasticity during aging. *The Journals of Gerontology Series A: Biological Sciences and Medical Sciences*, *59*(9), M940–M957.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Sutton, R. S. (1995). Generalization in reinforcement learning: Successful examples using sparse coarse coding. *Advances in neural information processing systems*, *8*.

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press.

Vischer, M. A., Lange, R. T., & Sprekeler, H. (2021). On lottery tickets and minimal task representations in deep reinforcement learning. *arXiv preprint arXiv:2105.01648*.

You, H., Li, C., Xu, P., Fu, Y., Wang, Y., Chen, X., . . . Lin, Y. (2019). Drawing early-bird tickets: Towards more efficient training of deep networks. *arXiv preprint arXiv:1909.11957*.

Yu, H., Edunov, S., Tian, Y., & Morcos, A. S. (2019). Playing the lottery with rewards and multiple languages: lottery tickets in rl and nlp. *arXiv preprint arXiv:1906.02768*.

A Neural Network details

Parameter	**Cartpole**	**Acrobot**
Input neurons	4	6
Hidden neurons	128	128
Nr. Hidden layers	2	1
Output neurons	2	3
Optimiser	Adam	Adam
Hidden layers activation function	ReLu	ReLu

B DQN Details

Parameter	**Cartpole**	**Acrobot**
Learning Rate	0.0025	0.001
Batch size	32	64
Replay buffer size	1000	2000
Start learning after	200 transitions	1000 transitions
Target network update	every episode	every episode
Max training episodes	500	500
Update frequency	every 4 transitions	every 1 transition
Early-Stopping threshold	200	-100
Discount factor γ	0.99	0.99
ϵ decay factor	0.95	0.999
ϵ start	1.0	1.0
ϵ min	0.01	0.01