Albert-Ludwigs Universität Freiburg
Institut für Informatik
Deep Learning Lab
Agnihotri, Shashank
Arce y de la Borbolla, José

Deep Learning Lab
Excercise Sheet 4

8.01.2019

Abstract

The purpose of this report is to present the reasoning behind the design choices for solving the OpenAI Gym Cart Pole and Car Racing Environments using a deep reinforcement learning agent, by implementing the algorithms of Deep-Q-Network (DQN), Double-Q to estimate the Quality Function, and ϵ-Greedy with decay and annealing for the exploration vs exploitation tradeoff.

1 Environment Definition

In order to be able to solve the problems at hand, we first need to understand how the environment works, and how we can interact with it.

1.1 CartPole

This first environment replicates the classical control problem of the inverted pendulum, where an agent needs to move a cart along a single axis (left/right) to maintain a pendulum attached to it in an upright position. The environment gives a reward of $+1$ for each action taken, and terminates every time the pendulum tips more than 12^{o} in either direction, the cart leaves the visible area $(-2.4, 2.4)$ or the episode lasts 200 time steps.

1.2 CarRacing

The environment in which the second problem takes place consists of a random generated track circuit with corners of varying difficulty.

The objective is to drive a racing car around the track, visiting every section of the track in the least possible time to maximize the value of the reward. The observations of the environment include an eagle view of the portion of the track the car is currently at, the accumulated reward value and a couple of useful indicators such as the overall speed or the position of the steering wheel. The reward function is calculated as follows: for every passing step, 0.1 is subtracted from the reward, for every newly visited tile $1000/N$ is added to the score, where N is the total number of tiles in the track. The episode ends when all the track tiles have been visited, or after a certain amount of simulation steps have taken place, in our particular case, after 1000 steps.

2 Agents

2.1 Models

For all of the experiments in the cartpole environment, a simple MLP network was used, with 20 hidden units in each of the two layers.

On the other hand, for the more complex carracing environment, the first agent used a convolutional network with 3 convolutional layers and 3 large hidden, fully connected layers. In order to speed up the training time for the experiments with ϵ-Decay and Double-Q learning, the network's complexity was dropped to just 2 convolutional layers and 1 fully connected one.

2.2 DQN and Double-Q Learning

The Double-Q Learning Algorithm was achieved by letting the Q network select the best action for each given state, and then use those predicted indices to estimate the value of the quality function using the TargetQ network, therefore making the learning more robust against noise thanks to a decoupling of the predictions. This in contrast to the original DQN algorithm, in which both the action and the Q function for that given action at a certain state are predicted by the TargetQ network by maximizing and, much likely, overshooting the Q–Function.

Given that the difference in code was minimal, the Double-Q Learning algorithm was implemented in the same class as that of DQN, using the boolean flag `double-q` to select between them.

3 Training

During the training sessions, a couple of different techniques, covered in this section, were implemented with the purpose of improving the learning rate and the final results.

3.1 Buffer Pre-filling

Given that at the start of the training, the buffer is empty, if trained on that data, the agent will strongly learn the same first erroneous first couple of states. In fact, depending on the value of the Batch Size Parameter, it can learn the same state 64, 100 or more times while the buffer is still filling up, state by state. For that reason, before the actual training starts, it was decided to fill up the buffer with states in which complete random actions were taken, up to a point where a certain multiple of the Batch Size has been filled with states. E.g., if the batch size were configured to be 64, then it would probably be a good idea to have at least 5 times that many samples in the experience buffer before training, i.e., $5 \times 64 = 320$ samples. It is also not very recommended to fill the buffer too much, as the rewards for such states would most likely very low, and if there are too many of such states in the buffer, depending on the maximum size of the buffer, it may take a very long time before they get replaced by states with better reward values.

3.2 Exploration vs Exploitation

In order to better learn the subtle nuisances of each environment, a tradeoff between exploration and exploitation must be made, such that the agent tries different policies that may lead to increased rewards, instead of just burning a certain policy into the model's parameters which could lead to smaller rewards in the long run.

For that purpose, given that we used an ϵ-Greedy algorithm, a schedule class was programmed such that the may agent explore more at the first training episodes, while at the same time settling with a more deterministic approach after a certain number of training iterations.

3.2.1 Decay Rate Schedules

As of the writing of this document, the Schedule class offers three different decay rates: constant, linear and exponential. They are defined as follows. In all of the formulas, the following symbols are used:

t is the length of the input interval during which the function will vary from $[0, t] \mapsto [y_0, y_1]$. In our case it defines the number of training steps or episodes in which the value of ϵ will decay.

y_0 is the function value at the start of the input interval

y_1 is the function value at the end of the interval defined by t

All of the formulas are defined in a piecewise manner, such that for $x < 0$, the output will be y_0 and for $x > t$, the output will be y_1.

$$f(x) = \begin{cases} y_0 & x < 0 \\ f_2(x) & 0 \leq x \leq t \\ y_1 & x > t \end{cases}$$

Linear Decay:

$$f_1(x) = y_0 - \frac{y_0 - y_1}{t} x$$

Exponential Decay:

$$f_1(x) = (y_0 - k \cdot y_1)\, e^{\alpha x} + (k \cdot y_1)$$

where:

k is a constant factor that helps in making the exponential actually intersect the value of y_1, instead of just approaching it asymptotically (0.98 for descending schedules, and 1.02 for ascending ones).

α is the multiplicative constant of the exponent. It is calculated as follows:

$$\alpha = \frac{1}{t} \log\left(\frac{(1-k)\, y_1}{y_0 + k \cdot y_1}\right)$$

Stepping The class allows the output of the function to be continuous or stepped. For the later case, a number of steps must be specified for the interval $[0, t]$, for which the function stores the values of the selected continuous function at the start of each subinterval. The function then returns the stored value which corresponds to the index defined by the integer division between the input and the length of the intervals.

Annealing was done by bounding a cosine function between the desired decay function and a constant function valued at y_1. This way, the output value oscillates between the desired final value y_1 and the decayed value of the initial output $f(x)$. In order to achieve this, the amplitude (a) and offset (b) of the cosine function need to depend on the value of x.

$$a = \frac{f_1(x) - y_1}{2}$$

$$b = \frac{f_1(x) + y_1}{2}$$

Using these two values, the annealing function is defined as follows:

$$f_2(x) = a \cdot \cos\left(\frac{(2c+1)\,\pi}{t}x\right) + b$$

where c represents the number of annealing cycles (or peaks of the cosine function) to perform within the $[0, t]$ interval.

Figure 1 shows the plots for sample decay rates with $y_0 = 0.9$, $y_1 = 0.5$, $t = 100$ and $c = 5$, obtained with the help of the `schedule_test.py` script.

3.3 Early Stopping

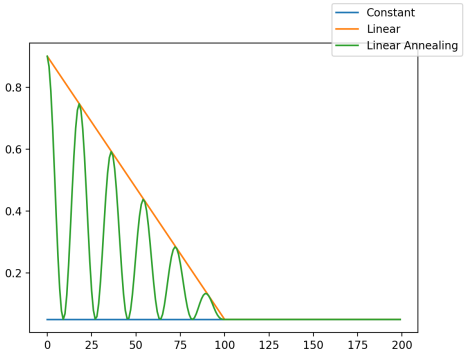
After a couple of training sessions, both in the CartPole and the CarRacing environments, we quickly realized that the agent can overtrain in such a way that it drops all of it's learned performance as if it were training from scratch again. For that reason, Early stopping was implemented by checking if the derivative of the training reward became negative after a certain period of stagnation. That way, if the agent starts performing worse than previous iterations in which it couldn't improve more, then the model would be saved before it started to decline, and the training would stop. If, on the contrary, the agent started to perform better after a period of stagnation, then the early stop countdown would reset back to zero. This was achieved by the implementation of the EarlyStop Class in the code.

4 Results

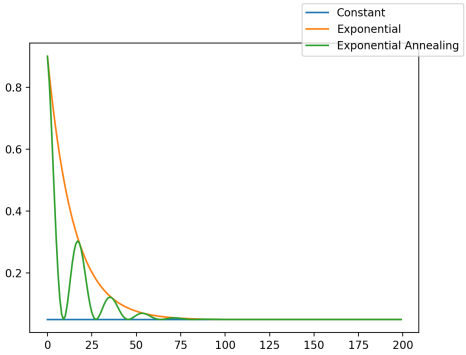
Initially, the agents were trained without much experimentation to check that the code and design choices actually worked. In this first tests, the results obtained were extremely good, but the agents had a tendency to overtrain, and started loosing rewards after having reached a local maximum. Thankfully, a backup of the carracing agent was made (/models/carracing/early stop/carracing_3c3l_s400_rw800) at the point where it was at the very peak performance delivering an astonishing result (reward>800 at 400 training episodes), as can be appreciated in the following table.

Env	Training		Testing	
	No. of Episodes	Training Time	Reward (over 15 episodes) Mean	σ
CartPole	**651**	00:15:11	947.666	171.628
CarRacing	10k	9d 04:58:54	**874.872**	**41.302**

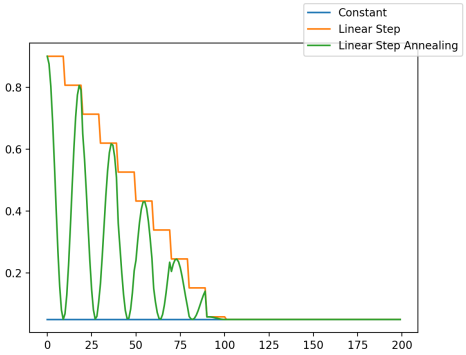
Table 1: Initial Results.



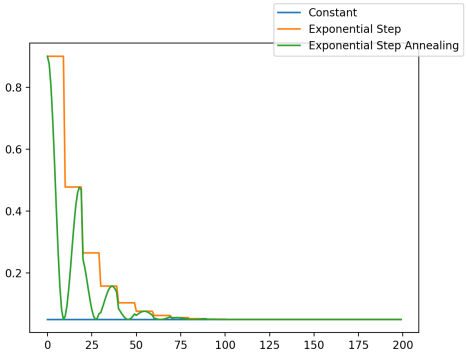
(a) Linear Decay (with and without Annealing)



(b) Exponential Decay (with and without Annealing)



(c) Linear Step Decay (with and without Annealing)



(d) Exponential Step Decay (with and without Annealing)

Figure 1: Supported Decay Functions.

The initial experiment was left to train for a very long time, in the hopes of it converging to a perfect score every episode, but during this session, its performance started dropping and never recovered. For that purpose, the experimentation using ϵ-Decay, Double-Q Learning and, especially, early stopping was implemented to try to minimize this effects.

In an effort to make the experiments more comparable, both the model's architecture and the hyperparameters shown in the following table were kept constant during all training sessions:

HyperParameter	CartPole	CarRacing
Initial Epsilon (ϵ_0)	0.90	0.75
Final Epsilon (ϵ_1)	0.05	0.05
Decay Episodes (t)	400	150
Annealing Cycles (c)	10	10
Batch Size	100	100
Learning Rate (η)	1×10^{-4}	1×10^{-4}
TargetQ Soft-Update Weight (τ)	0.01	0.01
Early Stop Patience	20	25
Skip Frames	–	3
History Length	–	5

Table 2: Common Hyperparameters

Table 3 summarizes the results obtained by each agent during training and testing. Only the values for the last training and testing sessions are shown.

Env	ϵ-Greedy Decay		DQN				Double Q-Learning			
			Training		Testing		Training		Testing	
	Epsilon Schedule	Cosine Annealing	No. of Episodes	Training Time	Reward (over 15 episodes) Mean	σ	No. of Episodes	Training Time	Reward (over 15 episodes) Mean	σ
CartPole	Constant	×	950	00:06:46	**1002.0**	**0.0**	1437	00:26:48	255.666	6.639
	Linear	×	1367	00:08:44	**1002.0**	**0.0**	865	00:05:54	485.866	482.941
	Exponential	×	1951	00:22:58	935.933	247.198	1450	00:19:43	**1002.0**	**0.0**
	Linear	✓	3073	00:29:11	737.733	438.236	1930	00:08:06	625.933	460.742
	Exponential	✓	1344	00:12:29	805.466	318.421	1402	00:13:51	668.133	394.270
CarRacing	Constant	×	299	05:42:46	334.396	150.460	281	5:36:26	424.248	145.881
	Linear	×	309	05:39:57	495.891	132.130	303	05:49:14	552.772	319.903
	Exponential	×	302	05:52:01	-82.639	8.742	310	06:38:11	442.345	196.175
	Linear	✓	308	05:36:11	495.555	278.910	300	05:43:28	**652.773**	254.303
	Exponential	✓	260	04:43:41	442.027	156.047	297	05:59:33	383.484	89.433

Table 3: Experimentation Results.

4.1 Training Curves

Except for the first experiment on each environment, all of the learning curves were smoothed out by averaging the results with those of previous episodes in order to reduce the amount of noise in the plots and more clearly see the trends instead of trying to find the actual curves amidst colorful pieces of abstract art. For the training reward curves, values for 100 episodes were averaged together, while for the validation data, given that it was evaluated every 10 training episodes, they were averaged with the previous 10 episodes to be comparable to the training curves. The raw curves were also captured and can be seen in the Tensorboard interface.

4.1.1 CartPole

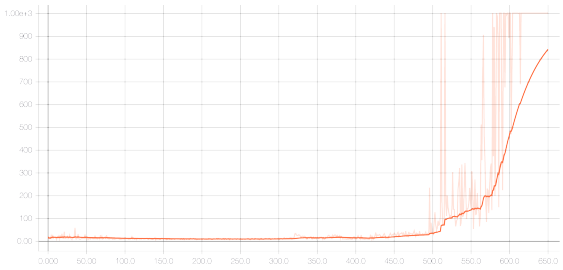
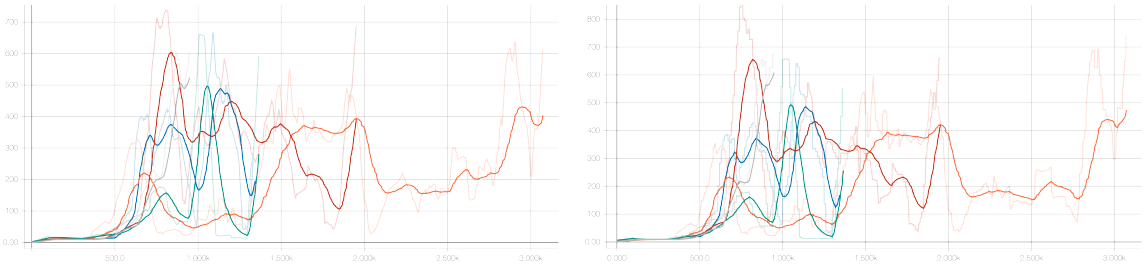
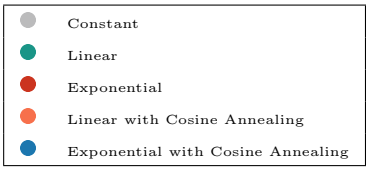


Figure 2: Initial Training Learning Curves (episodes).



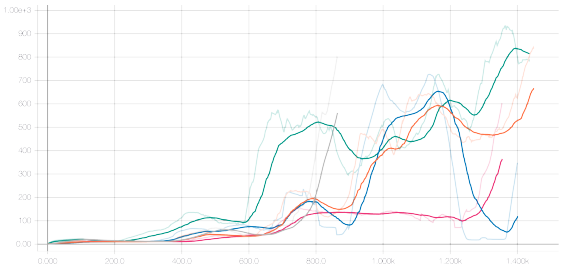
(a) Training Rewards

(b) Validation Rewards

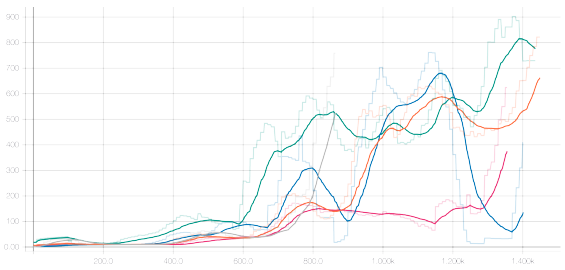


(c) Legend.

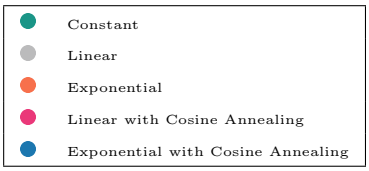
Figure 3: Experimental Learning Curves using ϵ-Decay and DQN.



(a) Training Rewards



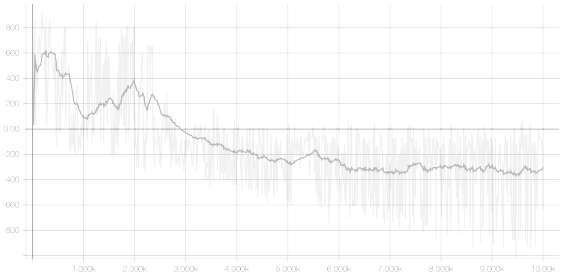
(b) Validation Rewards



(c) Legend.

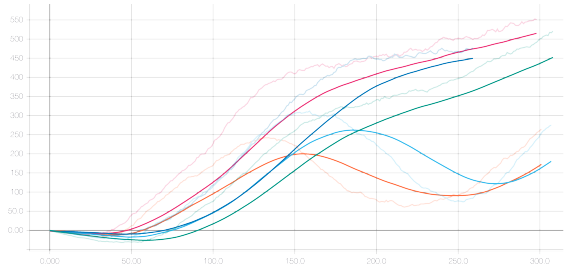
Figure 4: Experimental Learning Curves using ϵ-Decay and Double-Q Learning.

4.1.2 CarRacing

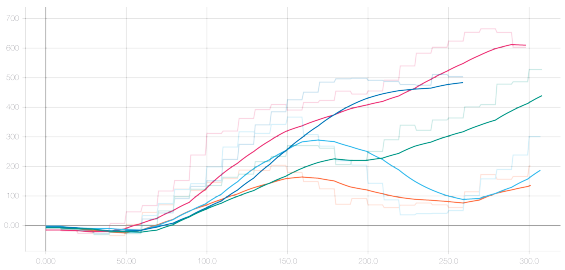


(a) Training Rewards

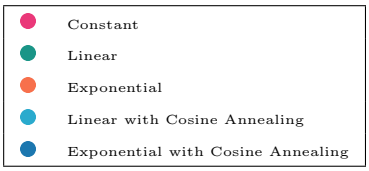
Figure 5: Initial Training Learning Curves (10k episodes).



(a) Training Rewards

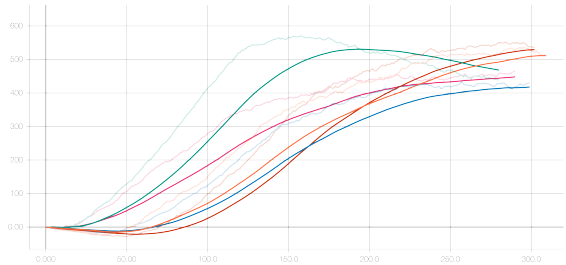


(b) Validation Rewards

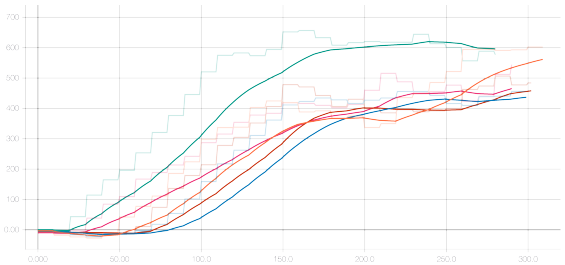


(c) Legend.

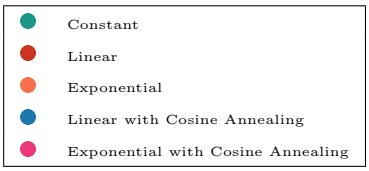
Figure 6: Experimental Learning Curves using ϵ-Decay and DQN.



(a) Training Rewards



(b) Validation Rewards



(c) Legend.

Figure 7: Experimental Learning Curves using ϵ-Decay and Double-Q Learning.

5 Discussion

Due to the highly cumulative non-deterministic nature of the algorithm, mostly because of the random weight initialization, the randomness in choosing whether to explore or to go with predicted actions, and the stochastic nature of exploration itself, no two training sessions behaved similar to each other. By running the same training session with exactly the same hyperparameters, the results might have gone well, or after having reached a certain reward at some

point in time, suddenly drop down to negative rewards. This behavior was experienced in both environments, making it very difficult to observe patterns.

Having said that, even though it is difficult to draw many precise conclusions, during most of the training session it could be observed that an exponential decay rate for ϵ had a tendency for having a performance worse than the other approaches, and that the cosine annealing either not made much of a difference in the final outcomes or it's oscillations actually rippled through the entire training session, making it converge in a less stable way, therefore taking longer to achieve the same result as in the cases were cosine annealing was not used.

Finally, due to the simple nature of the CartPole environment, agents were able to achieve super-human capacity for the task in a relatively short period of time. The same thing can not be said about the CarRacing environment given that it is far more complex. We believe that this was the result of using a rather conservative policy for the early stopping strategy.

Although overall we are happy with the results we also believe that, in retrospective, some more tuning of either the learning rate or the value for the TargetQ update weight (τ) could have been done in order to have a more consistent and less noisy result instead of such jagged learning curves which are probably due to overshoots on the gradient descent, or a wild cat and mouse chase between the Q and TargetQ networks.