Deep Learning Lab Exercise 4: Reinforcement Learning

Ingmar Baetge, Alex Rose

I. CartPole

We begin the classic control task of CartPole. Instead of imitation learning where the agent analyses expert play to learn the relationship between states and ideal actions (as in Exercise 3), here we use Q-Learning, where the agent samples from a palette of actions in each state, and gradually learns to parameterise a function $Q(State, Action) = Value$. In the CartPole instance our state is a 4-dimensional vector; our function is a simple neural network of two fully-connected 20-unit hidden layers with ReLU activation; and we have two possible actions: *Left* and *Right*.

We also implemented the bonus features:

- **Epsilon Annealing**: in which our probability ϵ of selecting a random (non-optimal) action is reduced from its starting value by a factor of 3.33×10^{-4} after each episode. This gradually shifts the agent's priorities from exploration to exploitation, as time goes on.
- **Boltzmann Exploration**, where instead of selecting randomly with probability ϵ and optimally with probability $1 - \epsilon$, the agent samples an action according to the SoftMax of its action predictions (with optional SoftMax temperature parameter). This allows the agent to shift from exploration to exploitation only when it becomes more certain about its predicted action-values.
- **Double-Q learning**, where actions are selected using the *current* network, but still evaluated using the *target* network. This decorrelates the noise in the two networks, theoretically avoiding overestimation bias in the Q function.

Training was over 1,600 episodes with Adam as our optimiser, learning rate 10^{-4}, batch size 64, buffer capacity of $100,000$, target network update parameter $\tau = 0.01$, and $\epsilon = 0.1$ initially. We see that Boltzmann Exploration creates a high performance agent much more quickly than Basic or Double Q Learning, presumably because in the early episodes when the agent has little idea of the best move, Boltzmann encourages it to explore *much* more than a fixed epsilon parameter would allow. Epsilon Annealing, on the other hand,

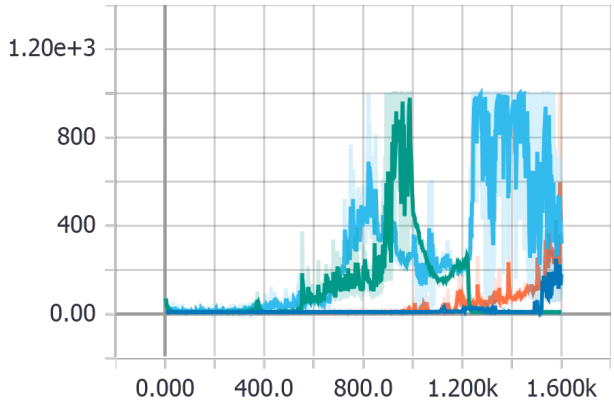

Fig. 1. Reward over training epochs for Cartpole models: Default; Epsilon Annealing; Boltzmann Exploration; Double Q

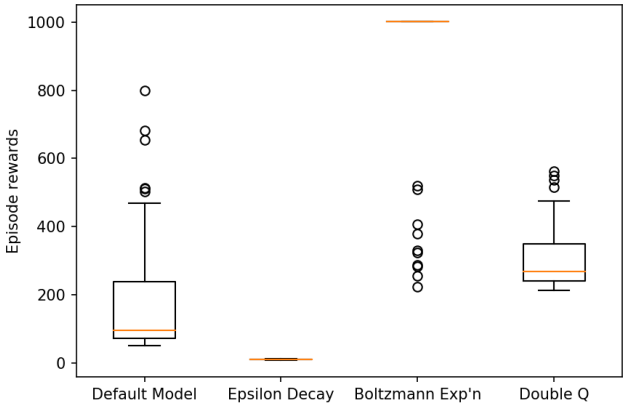

Fig. 2. Test results for Cartpole models

starts out promisingly, but has a disastrous spell later and forgets everything useful it has learned (Figure 1). We then tested each agent over 100 episodes, and found that the Boltzmann agent has learned to play perfectly with an average score over 1000, and the Double Q agent, though less impressive, has still solved the problem, with an average score over 200 (Figure 2). We should note that these results are very high-variance; in previous runs, we saw agents with identical configurations having very different training curves. Therefore, they should not be taken (for example) as definitive proof that Boltzmann exploration is the best method to solve this problem.

TABLE I

RACECAR Q-NETWORK ARCHITECTURE

Layer	Parameters
Convolution 1 / ReLU	Filters: 16, Size: 7
Max Pooling	Size: 2, Stride: 2
Convolution 2 / ReLU	Filters: 32, Size: 5
Max Pooling	Size: 2, Stride: 2
Convolution 3 / ReLU	Filters: 48, Size: 3
Max Pooling	Size: 2, Stride: 2
FC layer 1 / ReLU	512 units
FC layer 2 / ReLU	128 units
Output	5 units

II. CAR RACING

We used the same Q learning algorithm, code, and hyperparameters as in CartPole, with modifications to suit the Car Racing environment:

- Images preprocessed to greyscale
- A deep ConvNet, as described in Table I.
- Continuous actions discretised simply to *Nothing, Left, Right, Accelerate, Brake*
- A replay buffer of size 500,000.
- Frameskip of 2 during training (i.e. every third frame is seen and acted upon, and actions persist over the next two frames). At test time, no frameskip is used.
- Training over 1,000 episodes

For agent testing, we used two methods:

1) Actions selected using the maximum prediction from the Q network, leading to discrete actions.
2) Actions combined using proportions in the Softmax of the last layer, leading to continuous (and noticeably more "noisy") actions.

All seeds (Numpy, TensorFlow and Gym environment) were set to 0 to allow reproducable tests.

Figure 5 shows our agents doing reasonably well already, with many average scores over 800 (though some agents again "forget" what they've learned during training, and must re-learn it). Notably, Boltzmann exploration does much worse here than in CartPole. One reason is that in Cartpole, for every state there is an unambiguous best action, and once this is found, Boltzmann exploration will quickly allow the agent to exploit it consistently. Whereas in the racecar environment, the reward delta between the best action and other actions can often be quite small, and the agent will be encouraged to explore sub-optimal actions for much longer.

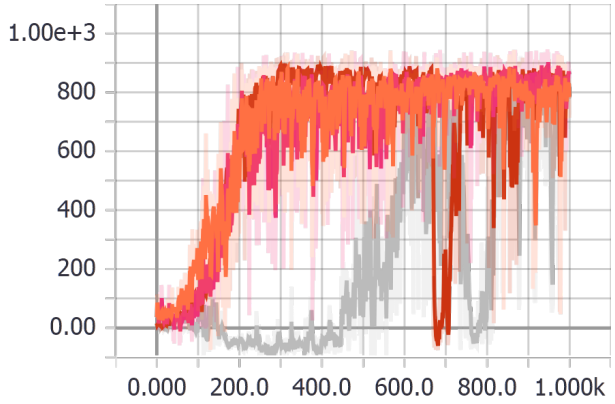

Fig. 3. Racecar training curves for Basic Model, Epsilon Annealing, Boltzmann Exploration, Double Q learning

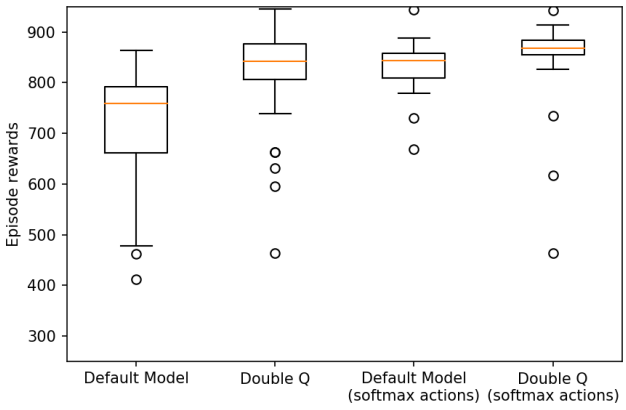

Fig. 4. Racecar test results for Basic Model and Double Q learning

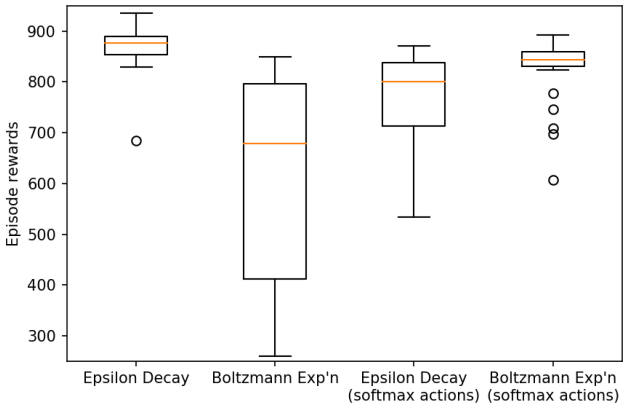

Fig. 5. Racecar test results for Epsilon Annealing and Boltzmann Exploration

We also notice that these agents are far better at recovery than in the imitation learning exercise, since here they have ample opportunity to learn how to behave in off-road situations (whereas in the imitation learning dataset, there were typically very few examples of the expert's car leaving the road, which left the agent largely unguided when something went wrong).

Observing our agents' driving in test runs, we saw that their characteristic mistake was to enter turns too fast, cut across the corner (missing some reward tiles), and rejoin the road. They learn to do this manoeuvre quite well, and indeed the Q learning algorithm may be encouraging it: doing so allows the agents to avoid slowing down for the corner, and so they pick up rewards as rapidly as possible over a medium time horizon. But by the end of the episode they leave some tiles uncollected, and the clock runs down, diminishing their final reward.

We also see that the 'Softmax' agents tend to drive much more cautiously, and that there is more "noise" in their actions, e.g. rapid light steering on straight stretches of road, just as we would expect. However, they do seem less likely to enter corners too fast than the 'Hardmax' agents. Overall, Softmax action selection does not seem to be systematically better nor worse.

III. Adding History

We hypothesise that the agents may be making this corner-cutting mistake because they simply don't have a clear idea of how fast they're going from the single-frame input we've used so far. Therefore, we next implemented image history in order for the agent to better judge its speed, in two different ways:

1) Adding the frame immediately prior to the current one to our state buffer, so that the input to our ConvNet is a $(96, 96, \mathbf{2})$ image
2) Subtracting the prior frame from the current one to produce a difference image, and processing this difference image using a separate, smaller ConvNet (which is merged with the first one in the fully connected layers)

Results are shown in Figures 6 and 7. The model trained with an additional difference frame as input exhibited better acceleration and braking behaviour, and it also showed a reduced tendency to cut corners. It is therefore reasonable to assume that giving the model access to speed information helps the agent learn this improved behaviour.

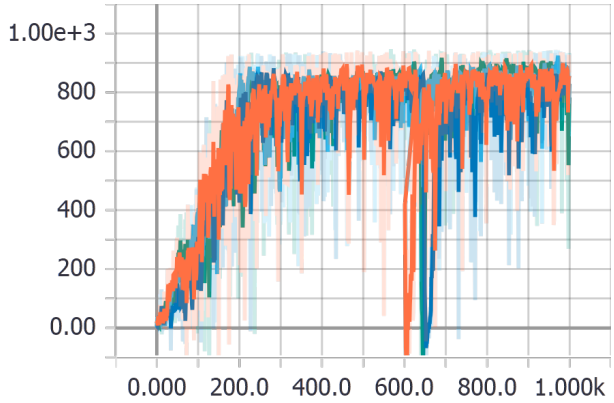

Fig. 6. Racecar training curves for History Frame, Difference Frame, Diff. Frame with Penalty, DF+P with large network

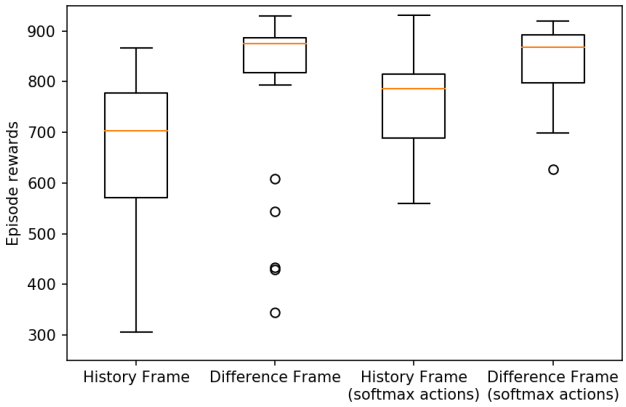

Fig. 7. Racecar test results for History Frame and Diff. Frame

IV. Reward Shaping

We observed that the agent would cut corners quite often - it seemed to learn that there is more benefit in keeping a higher speed, as it will lead to more short term reward, conflicting with the long term goal of collecting every tile of the road, which only becomes relevant towards the end of each episode.

We decided to introduce some reward shaping into the environment - we added two virtual sensors by reading two pixels in front of the car on every step. If these sensors both detect the grass color, a negative reward of -1 is added to the total reward, effectively punishing the agent quite strongly for going off road. By adding this simple reward shaping, the agent exhibited less corner cutting, as this behaviour is now penalized. See figure 8 for the location of the sensor pixels.

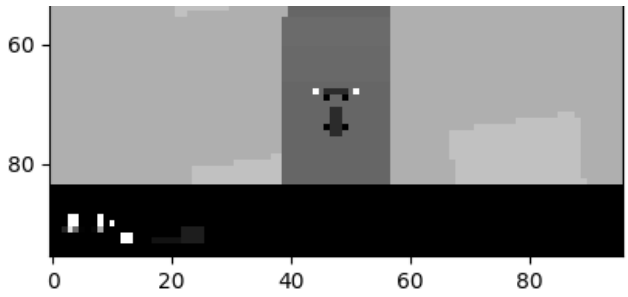

Fig. 8. Sensor positions for reward shaping: Detecting grass in the two white locations (pixels $(44, 68)$ and $(51, 68)$) was penalized with a negative reward of -1. As the car position is almost static, the sensor locations were hardcoded.

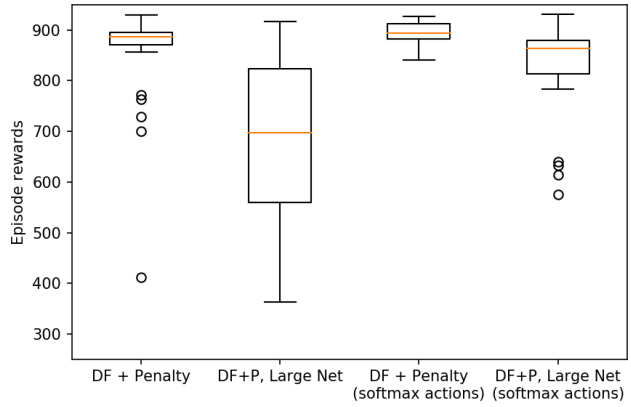

Fig. 9. Racecar test results for Diff. Frame + Penalty, DF + P. with large network

Of course, by implementing reward shaping we have slightly modified the original problem, and encoded some expert guidance or knowledge into the reward function. So in a sense, the Q-learning algorithm is no longer as 'pure'.

In any case, the question remains: why was reward-shaping necessary? That is, why did Q-learning reward short term gains from corner cutting at the cost of a larger long-term performance hit? The obvious answer is the discount factor of 0.99 that we used: for example, a missed reward 500 timesteps into the future would be discounted by a factor of $0.99^{500} \approx \frac{1}{152}$. We did also experiment with agents that discounted less or not at all, but these showed worse performance.

A video recording of this Difference Frame + Penalty (DF + P) agent's performance over one episode (the first that we tested) is available in our repository [2].

TABLE II

FINAL RACECAR TESTING RESULTS

Captions	Avg Test Score	Avg Score (Softmax)
Default Model	701.3	833.1
Epsilon Decay	871.0	771.0
Boltzmann Exp'n	612.6	828.2
Double Q	811.5	845.6
History Frame	650.0	754.8
Difference Frame	809.5	838.3
DF + Penalty	855.0	895.5
DF+P, Large Net	669.2	827.3
DF+P, No Skip	627.3	229.5

V. IMPROVING ON THE DIFFERENCE FRAME + PENALTY AGENT

In many episodes, the DF + P agent performs a perfect run, driving reasonably fast and picking up every reward tile. Sometimes it still misses a few tiles, especially on sharp corners. We tried improving this with one final model, where we doubled the number of filters in every convolution layer, and increased the number of units in fully connected layers by 50%. We did this with the hypothesis that the agent may have been unable to learn complex enough representations with the incumbent model to guide its actions, and in the belief that the larger model should be protected from over-fitting by the continuous stream of new training data supplied to the gradient descent algorithm. Unfortunately, as can be seen in Figure 9, the larger model was not as successful.

CONCLUSION

Our final results are summarised in Table II. The best-performing agent was the Difference Frame model with an off-track Penalty (using Softmax action interpretation), and almost as good was the Epsilon-Annealed model (using standard action interpretation), in which no reward shaping was required.

ACKNOWLEDGEMENTS

We would like to thank Guilherme Miotto and Baohe Zhang for all the fruitful discussions, the tutors of this course for their valuable input, and the taxpayers of Germany for the electricity we have consumed.

REFERENCES

[1] AI gym car racing: https://gym.openai.com/envs/CarRacing-v0/

[2] Exercise code: https://github.com/rosea-tf/dl-lab-2018/tree/submit/exercise4_R_NR