Reinforcement Project 1

Afroze Baqapuri and Alexander Solsmed

December 2013

1 Analysis of standard car

In the following subsections, we describe our experimental setup, some details about our implementation and our test results for the standard car. After implementing the neural network as described in the project instructions and the SARSA algorithm with a greedy epsilon choice algorithm ($\epsilon = 0.1$). For collecting data on performance, we ran 10 cars sequentially and then took the mean of their results. Each car was initialised with neural network weights reset to zero. Each trial had a time limit of 1000 time steps before aborting.

1.1 Learning curve

The data was saved into two matrices, one for the cumulative reward gained over the whole trial (regardless of whether the finish line was reached or not) and one for time taken to finish (or NaN if goal was not reached in the maximum number of time steps). The time to complete a race in each trial (latency) decreases rapidly and starts to plateau. After about 400 trials the plateau value is reached and we get convergence of the system.

When the average latency values were calculated, any NaN entries were discarded first. The figure 1 (left) shows the latency plot for the default epsilon value of 0.1 averaged over 10 cars. The dashed line represents the actual latency per trial smoothed by a gaussian of window size 31 and $\sigma = 9$. The solid line represents a curve fit on the raw date according to the equation: $y = a + b/x$ where $x = \sqrt[7]{trial/num\_trials}$.

The figure clearly shows the latency per trial decreasing rapidly first then slowly converging around 400-500 trials. After convergence the average latency is less than 100 time steps to reach the finish line. The plot is a clear indicator that algorithm is learning well for the car.

1.2 Integrated reward

In parallel with the recording of time to track completion, the cumulative reward for each trial was recoded and plotted over the number of trials. In the beginning, the car doesn't know what to do and will crash into the walls often,

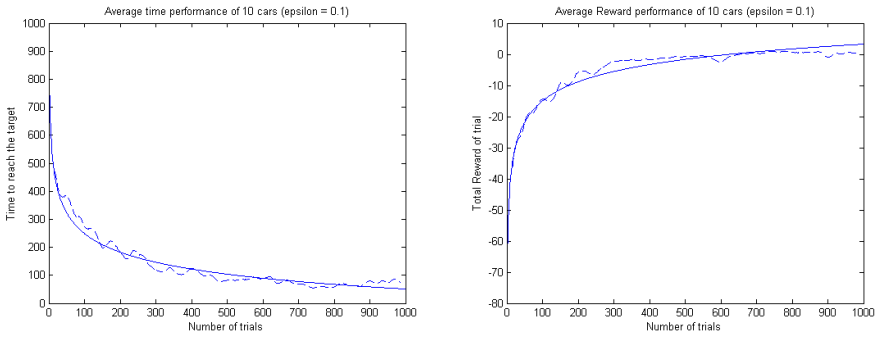

Figure 1: Average performance per trial ($\epsilon = 0.1$): Time Latency (left), Integrated Reward (right)

getting negative rewards. Thus, for the early trials, the cumulative reward at the time of crossing the finish line is largely negative, in the negative hundreds.

As the car builds up knowledge in its neural network, actions are taken which are less likely to lead to negative rewards and indeed after the convergence is reached, the cumulative reward is close to zero and even positive later on, as the learning is perfected. This behaviour is illustated in Figure 1 (right) which plots the total reward accumulated per trial averaged over 10 cars, with default ϵ of 0.1. Like before, the dashed line is smoothed version of raw data and solid line is result of curve fitting using function $y = a + b/x$ where $x = \sqrt[7]{trial/num\_trials}$.

The shape of the integrated reward curve is very similar to the latency curve, both are like the mirror image of each other along the horizontal axis. This is expected behavior because a bad trial (which takes a large number of time steps) is usually going to have a lot of crashes (since a lot of time steps mean the car is either moving very slowly or moving around a lot of wrong paths and usually crashing) and hence a highly negative cumulative reward. On the other hand a car which quickly reaches the minimum time would likely have found a good path and thus have experienced less number of crashes, and also a crash results in a time penalty. Hence the two curves are very closely linked.

1.3 Exploration-exploitation

It is desiarable to let the car explore different strategies, to get a broad view of what the different options are and to avoid local minima for optimal sequences of actions. Regardless of how an action was chosen by the policy (random or greedily), any action taken will contribute to the learning by the system. We play around with the epsilon in our system to get a fair comparison of exploitation vs exploration, to select the most optimum choice. The results are plotted in 2.

The left figure shows that the time latency is, on average, constantly high for a large static ϵ (for example 0.9). This is because there is a high chance of

random action being chosen which means that the benefits of learned experiences will be largely ignored while picking the action. As the epsilon is decreased (but still held constant/static) the benefits of the learning shine through more and more, until about ϵ 0.1. A very small ϵ doesn't grant the system enough needed exploration (within the 1000 trials) to build any understanding that is descriptive enough to exploit the environment, and the system is highly likely to be trapped in a local minima. We tried experimenting with $\epsilon = 0.01$ but that resulted in very poor performance with car almost never reaching the finish line. The result with accumulated reward is also expected as reasonable low static ϵ (0.1) converges to near-postive total reward value, while a very large ϵ value (0.9) stays bad in terms of total reward. We tried with other ϵ value between 0.1 and 0.9 which follow the same pattern, but for the sake of brevity we do not plot them here.

In these figures we are infact plotting the smoothed version of raw data (for better visualization and comparison) but we also observed that increasing the epsilon values considerably increased the stochasticity of values in the corresponding graphs (more shaky and deviating from the mean).

With a dynamically changing epsilon (changing epsilon for each trial, but keepig constant during a trial) we can begin with a very large epsilon to explore the environment and then gradually switch over to using experience in later trials. In our experiments, we let epsilon decrease as a function $\epsilon(trial) = e^{a \cdot \frac{trial}{num\_trials}}$ where a is our parameter which we varied between $a = -1$ and $a = -9$. A steeper descent of epsilon (corresponding to $a = -9$) was better than a slow descent ($a = -1$). This shows that a system which is highly exploratory initially but quickly shifts to highly exploitative shows much better performance (converges faster) as compared to system with slow decay of epsilon. This behavior is also replicated in the integrated reward section.

However we also notice that the final converged value of both dynamic epsilons is the same, for both time latency and total reward per trial. This is because given enough time, even a system where epsilon decays slowly, becomes highly exploitative. The only difference is that it will take more number of trials to extract the full benefits of exploitation, thus reach later convergence. Generally speaking, if we have more complicated track then an epsilon which decays slowly would perform better since it would be good for the car to explore around for decent number of trials before finally inclining more towards exploitation of what its already learned. If the track is relatively simple then final converged time latency for steep and shallow decay would be more or less similar, which is true for our case.

Lastly, if we compare the performance of static vs dynamic epsilon, then on average the dynamic epsilon seems to perform better since it is more robust / less sensitive to the changes in the value of factor a (which controls steepness of decay). On the other hand, a good static value of epsilon is highly dependent on the track used and a very low or very large value selected may ultimately result in bad results.

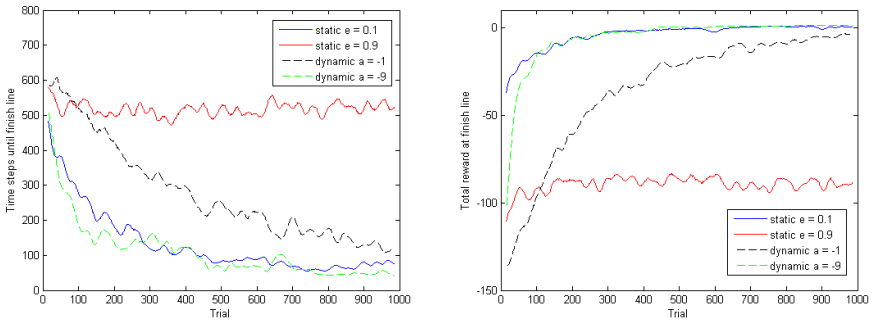

Figure 2: Explorarion-exploitation tradeoff comparison with static and dynamic epsilon: Time Latency (left), Integrated Reward (right)

1.4 Navigation map

Finally we visualized the effect of learning by plotting the optimal action maps of our track during various phases of learning. The optimal actions for six different phases of learning corresponding to $\bar{v} = 0$ are shown in figure 4. Due to size limitations it is a bit hard to visualize the action arrows, but the following color legend can be used to help in visualizing.

Analyzing the figure shows us that in initial phases (especially when trial number = 60) the action map is very simple and at not very optimal. But as we move to later phases, the map starts becoming more complex (various regions with different optimal actions) and more fine tuned to the particular track. This shows that algorithm is doing good job of learning.

2 Optimal car

We give a bonus reward related to the latency, which decreases as a negative exponent as the number of time steps increases. The optimal car also keeps track of its best yet lap (with respect to the number of time steps to completion) and uses that for the race, regardless of the current state of the memory from learning. In this regard, the optimal car affords a greater exploratory emphasis during training, for the sake of finding an even faster way to finish the race than it previously has. Furthermore, the optimal car uses a dynamic epsilon during its training, which builds upon the curve we tested using $a = -9$, but also add a constant chance of random action at 3%. The epsilon is set according to $epsilon = 0.03 + 0.97e^{-9 \cdot \frac{trial}{num_trials}}$.

In the training of the Monaco track, after about 100-200 trials, the optimal car has usually found paths to completion taking about 100 time steps. After about 500 time steps, the optimal car is close to an optimal path (being 5-10 steps longer than optimal) and it spends the remainder of the time trying to perfect this.

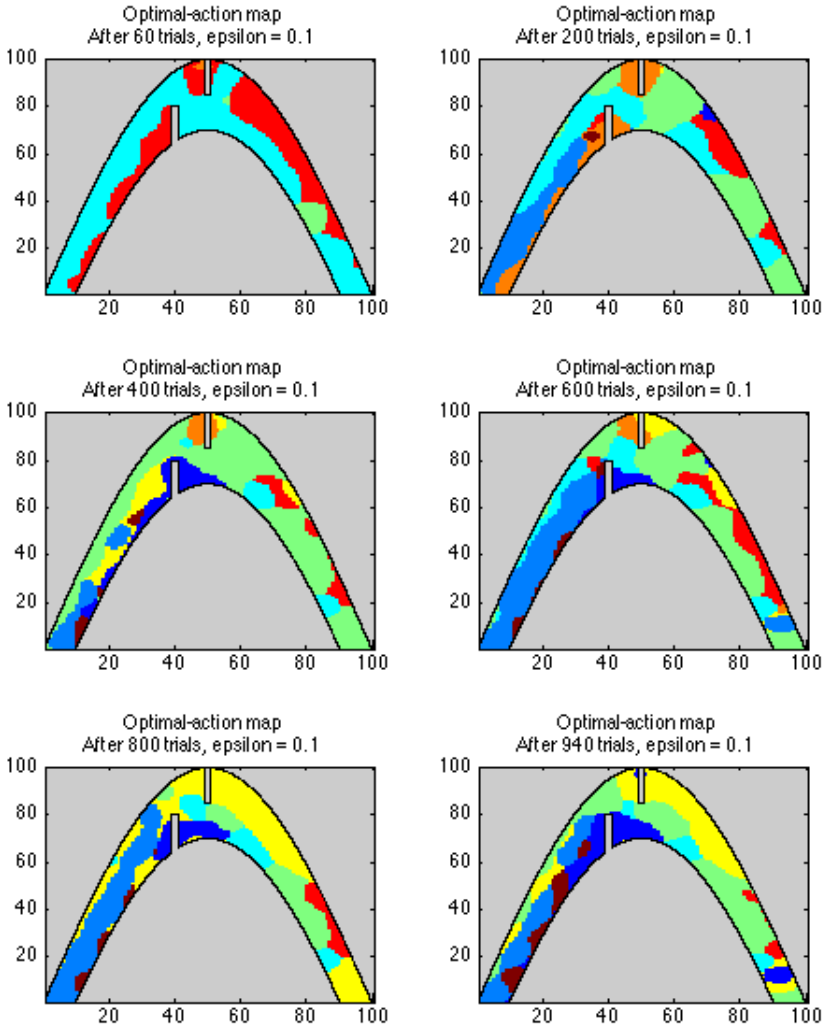

Figure 3: Optimal Action Map of track for different phases of learning.
cyan: EAST, red: WEST, dark blue: NORTH, yellow: SOUTH
blue: N-E, orange: S-W, maroon: N-W, green: S-E.

Our optimal car has a consistently lower latency and has a very high chance of finding a very short path.

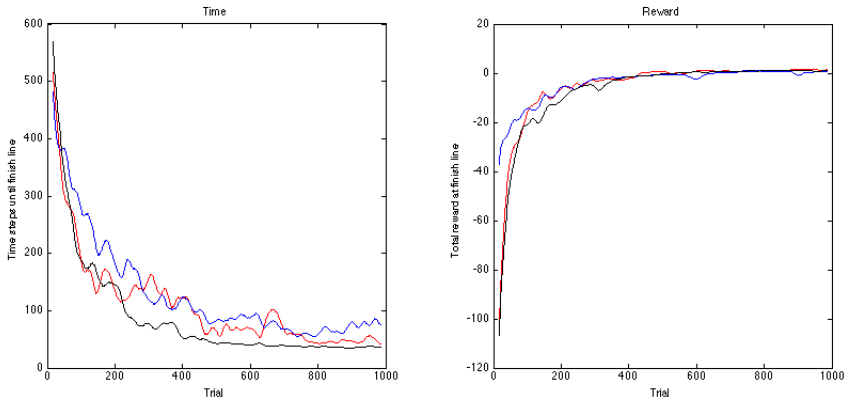

Figure 4: Performance of optimal car compared to best performances of standard car using static and dynamic epsilons.

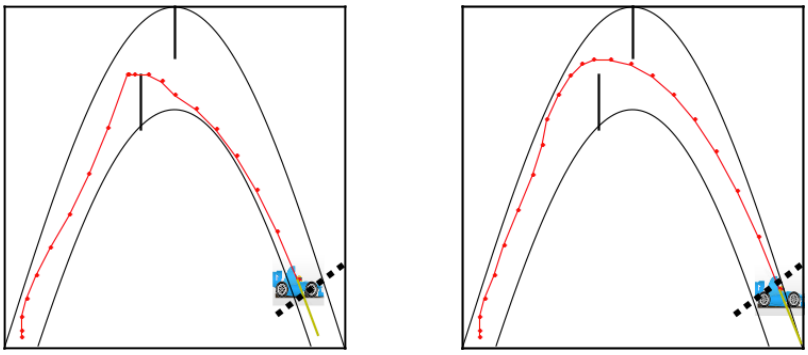

Figure 5: Best solution found by optimal car. Time to finish is 22 time steps (left) Handcrafted path by human. Time to finish is 21 time steps (right)