Part 2 - Reinforcement Learning

Runar, Lars and Årne

1. How would this problem be expressed in the MDP formalism? What challenges arise in the current setting?

   Previously we had a one-environment problem, which could be represented by the standard MDP formalisation $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma \rangle$. However, this new problem introduces multiple environments. We assume that these different environments might span different state-spaces (e.g. factories of different length) and have different transition functions (e.g. different floors might have different friction coefficients). This needs to be accounted for in the problem's MDP formalism, which is achievable by making the transition function and state-space a function of its environment; $\langle \mathcal{S} = \mathcal{S}(\text{env}), \mathcal{A}, \mathcal{R}, \mathcal{T}(\text{env}), \gamma \rangle$. Here, $\mathcal{A}$ is the set of actions $\{\text{Accelerate left}, \text{Idle}, \text{Accelerate right}\}$, $\mathcal{S}(\text{env})$ is the state-space of a given environment, and $\mathcal{T}(\text{env})$ the transition function of a given environment.

   A challenge which might arise in this current setting is that we need to develop an agent with the ability to learn a general policy which works across several environments, even environments it has never seen. Therefore we need to ask the question of how we could force an agent to learn this general policy without overfitting to any environment it is trained on.

2. Assume your aim is to train an agent to maximize reward on a set of possible environments. How would you change or adapt your training to deal with this scenario?

   Previously we used deep Q-learning to tackle the task on a single environment, which is a good architecture for learning policies on problems with discrete action-spaces and continuous state-spaces. One could assume that training this same architecture over several different uniformly sampled environments would force the agent to learn an optimal policy which performs best averaged over all training environments. However, this assumption might only hold if the reward function is equal over all different environments. If not, the agent could choose the "shortcut" of optimizing a single environment with a generous reward function.

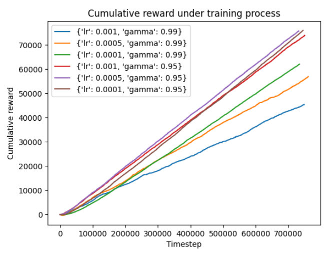
3. Download and instantiate the environments `cookiedisaster-v1`, `cookiedisaster-v2`, `cookiedisaster-v3`, train and run your agent there. Compare and comment on the result across the environment.

   Our training algorithm consists of training the previous architecture for 500 episodes, where each episode runs for a number of timesteps governed by the function $\min(100, f(\mu = 1500, \sigma = 500))$ where f is a Gaussian Distribution. Each episode we randomly sample a new environment and reset it.

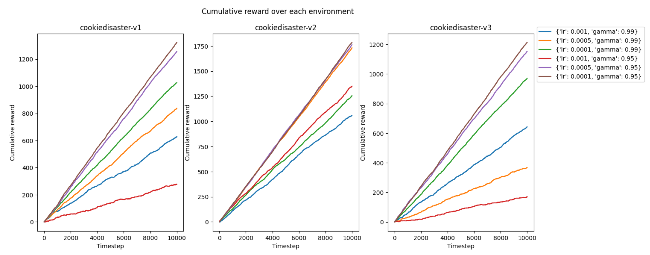
   Hyperparameter tuning was done using grid-search with the following potential hyperparameter values; $\epsilon = 0.2$, learning rate $\alpha \in \{0.001, 0.0005, 0.0001\}$ and $\gamma \in \{0.95, 0.99\}$. Action selection using epsilon-greedy is only done during training, and is omitted otherwise. The cumulative rewards during training, and final performance measure on each environment are plotted under figure 1.

   In general all agents performed relatively well, and all managed to develop a general policy which performs variably well. However, the best performing hyperparameter pair was $\alpha = 0.0001, \epsilon = 0.2, \gamma = 0.95$.

   To gauge generalisation ability we also experimented by training all agents on only the first two environments, using the third as an unseen test environment. Even in this limited scenario our models performed reasonably well on the unseen environment. This information substantiates our hypothesis that simply training an agent over several randomly sampled environments leads to the agent developing a generalised policy.



(a) Cumulative reward during training



(b) Cumulative reward measure after training over each environment

Figure 1: Cumulative reward during training of algorithms of different hyperparameters