Weekly Report #1

Simon Hugot

BiCS

simon.hugot.001@student.uni.lu

During week 1 and 2, I mostly focused on understanding and exploring AWS Deepracer. This meant create my first model in the Deepracer framework. My goal was also to understand how reinforcement learning works and how it is used inside AWS deepracer.

The AWS Deepracer Framework :

The models can be virtual or physical, they are composed exactly the same, with the same components and functionalities. The simulation process is composed of first a training time with stop conditions, an evaluation in real conditions to evaluate to performances of the car and then cloning the current model and make changes in the hyperparameters or in the reward function.

The AWS Deepracer is composed of multiple services in AWS. There is AWS Redis which is where the models are trained using the data created in RoboMaker, AWS Sagemaker then uses the trained data from Redis to improve the model which is then stored in AWS S3.

Reinforcement Learning :

In reinforcement the agents are continuously improving. The agent is improving its actions by trial and error unlike supervised learning where it compares unknown data with the training data. An agent is the entity that is creating certain actions based on its environment, here the car. The actions can be discrete (turning right,)or continuous(fluctuating speeds). After each successful action the agent enters a new state. Partial state is when it doesn't capture its position with respect to the entire track. At each state the agent earns a reward. Since the robot "wants" earn these rewards after some time it will learn which actions to make in order to maximise long term rewards.

State transitions :

Each action moves the agent from a first state to another generating a reward. This is a task. It can be an episode (limit time, objective to complete, ...) or discrete (no real ending, continuous action).

The goal of reinforcement is to maximise the sum of all rewards during the episode. This is done with a discount factor which determines to what extend the future reward contributes to the overall sum of rewards. (discount factor = 0 means the agent wants short term high rewards, it only cares about the current state) (discount factor = 1 means the agent looks at the entire episode to decide which action to maximise the sum of all rewards). This is important in discrete tasks.

The policy (pi) is where the decisions are made. The agent will learn the ideal actions to make to get high rewards which is then stored in the policy
Two ways of creating a policy :
- Stochastically which includes a probability of taking an action given a state.
- Deterministically which a direct mapping between state and action.
The goal of the training is therefore to get the most efficient policy. Deep reinfocement learning is when the policy is a neural network. Here a Convolutional Neural Network which uses a set of images of the robot and generates an action based on them.

The way we evaluate a policy is through a value function. To know how good a value function is, there is two ways. The first one which looks at the value of a state by being in a certain policy. The second one is by looking at the value of being in a state and taking an action, a state-action.
AWS Deepracer uses the Proximal Policy Optimization Algorithm to train its robots, which is very efficient and stable and easy to use. This uses two neural networks: The policy network(action network) which decides what actions to take given images as input and a value network(critique network) which estimates the final sum of rewards given images as input.

Redis is where the experience data is stored during the training. The episode is divided into segments (steps). For each step the state, action, reward and new state are cashed in Redis as experience. Amazon Sage Maker then uses this experience to improve the model then stored in S3

Hyper parameters

Hyper parameters are external to the training, that means that they have to be defined when setting up the model and cannot be changed during the training. As opposed to normal parameters that change only during training.

There are 8 different hyper parameters. :

- Batch size:
  This determines how much data should the model use before updating the model, here the images.
- Number of epochs:
  This determines how many the times the training model will pass through the batch set. Increase the number of epochs until the validation accuracy is minimized.
- Exploration :
  This is the balance between exploration(gathers more data beyond what is already known) and exploitation(is the use of the current data to make decisions). It randomizes a bit the actions to discover new data. It helps to find a higher reward path.
  (Categorical exploration ?
  Epsilon Greedy ?)

- Learning rate :
  This controls the speed at which the algorithm learns. The algorithm is trying to minimize the difference between the predicted output and the actual output. Its learning how to map the outputs with the inputs by changing the weights in the neural networks. The learning rate is the time between the updates of these weights
- Entropy:
  Controls the degree of randomness of an action. It helps to create new action, and therefore help find the highest possible path. This is to avoid being stuck in a local maxima and using the same set of actions over and over again.
- Discount factor :
  This parameter is to decide how much the future reward will count in the overall sum. The bigger the discount value, the further it will look for rewards. This parameter defines a time sense to the car.
- Loss type:
  It is used to update the networks weights. It makes incrementional changes to the car's behaviour from very random to very strategic. If there is too big of change from random to orginzed the training will be unstable and the agent won't learn. (Huber loss and Mean squarred error loss??)
- Episodes:
  Episodes define the number of tasks the model runs through in training.

The reward function

The reward function is the code that gives the car a specific reward or penalty depending on its actions. It only depends on our input. This function relies on multiple parameters like :
- **all_wheels_on_track** which detects if all the wheels of the car are on the track.
- x,y the coordonates of the car
- distance_from_center which retruns the value of the distance between the ar and the middle of the track.
...
Its objective is to maximize the rewards on long term. To achieve this it is then necessary to first think short term to tell the car what actions to encourage in the future and which ones to discourage.
This reward function function is written in python.