AWS

Thursday, 8 August 2019 10:00 AM

Reinforcement learning models interact with an environment, actions are rewarded or punished based on consequences from previous iterations.

In RL models, several components work together to help the model learn.

1. Agent
2. Action
3. Environment
4. Rewards
5. State

In RL, there is an agent that takes an action based on its environment. Actions cause the environment to change to a new state, which may or may not yield points for the model.

Agent: the entity exhibiting certain behaviour (actions) based on its environment.

Actions: what the agent chooses to do at certain places in the environment, such as turning, going straight, going backwards, etc. Actions can be discrete or continuous.

States: has to do with where in the environment the agent resides(at a specific location) or with what is going on in the environment(for a robotic vacuum, perhaps that its current location is also clean). By taking actions, the agent moves from one state to a new state. States can be partial or absolute.

What is the difference between a partial state and an absolute state?
Absolute means the agent can see the whole environment, partial means it can only see a part of the environment.

Discount factor: determines the extent to which future rewards should contribute to the overall sum of expected rewards.

Policy: this determines what action the agent takes given a particular state. Policies are split between stochastic and deterministic policies.

Policy functions are often denoted by the symbol π

Stochastic - determines a probability for choosing a given action in a particular state(e.g. an 80% chance to go straight, 20% chance to turn left

Deterministic - directly maps actions to states.

Value functions indicate which actions you should take to maximise rewards over the long-term(the expected rewards when starting from some given state). These are often represented with the capital letter V

PPO: Proximal Policy Optimization

Policy network(actor network): decides which action to take given an input image
Value network(critic network): estimates the cumulative result given the input image.

Tuning your model

Optimization: the highly iterative process used in order to find the best policies to follow in the environment.

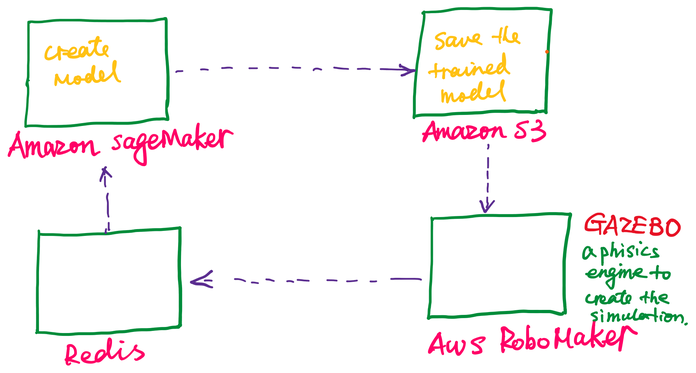
sagemaker: used to train the RL models.
provides the compute power for the machine learning algorithms for DeepRacer.

Robomaker: used to create the virtual space (enviroment) used for training.
provides the simulation engine for DeepRacer.

S3: the storage bucket used to hold the model files.

Redis: holds experience data from training.

episodes → steps segments → experiences { state, action, reward, new state }



GAZEBO simulates the laws of physics for each component of AWS DeepRacer car, the chassis, the wheels, the camera, their dimensions, Properties like mass, camera's field of view. collision, friction. acceleration.

parameters: internal to the model and can be learned from training.

hyperparamet are external to the model and set by you before training.

ML engineers job is finding the right balance of these hyperparameters.

RL models learns by rewards.

The Reward Function

all_wheels_on_track, is_left_of_center, is_reversed, Boolean
x, y, distance_from_center, heading, progress, steps, speed, steering_angle, track_width, waypoints. List
closest_waypoints

Hyperparameters.

Learning rate (0.001) $10^{-5} \sim 10^{-3}$ (NO)
Model Size
Batch Size (32), 64, 128, 256, 512 (YES)
Number epochs (3) ~ 10 (NO)
Exploration (categorical Parameters) EpsilonGreedy (NO)
Entropy
Discount factor (0.999) 0 ~ 1 (NO)
Loss Typ (Huber Loss) Mean square error loss
Episodes: number of episodes. How many time car crashed or cross the border.

Batch_size: this determines how many images, randomly sampled from the most recent episode, are used for model training before updating the model.

Epochs: Determines how many times you will loop through the batched data before updating the training weights. A larger number of epochs is likely needed if your model is still improving but training otherwise ceased.

Learning rate: Controls the speed at which your algorithm learns (if enlarges or shrinks the weight update after each epoch.
Larger learning rate can allow for faster training.
Smaller learning rates reach a more optimal solution.

Exploitation vs. Exploration - Exploitation is using the existing knowledge to choose actions, while exploration means it will gather additional information by trying out new actions in a given state (that may or may not result in improved rewards)

•	Categorical exploration - has a discrete action space
•	Epsilon exploration - has a continuous action space

Epsilon greedy - will decrease the exploration value over time, so that it explores a lot at first, but later will perform actions more and more based on experience (while still allowing for some random exploration).

Entropy - This controls the degree of randomness the agent takes in a given situation. The higher the value, the more it will randomly explore.

If the model appears to be trying the same actions over and over, we need:

1. Expand its exp ion
2. Increase entro
3. Use epsilon greedy

- Discount factor - helps decide how important future rewards are in training. With a larger value, it looks further into the future for expected rewards.
- Loss type/functions - used to update the network weights during training. A good algorithm (in combination with the learning rate) helps make a smooth transition from random actions to strategic actions. The online platform for AWS DeepRacer has two loss functions: Huber loss and Mean squared error loss.
- Episodes - This is how many tasks the agent will take part in during training, with weight updates occurring after this number of episodes.

Huber Loss / Mean square error loss
small updates: they behave similarly
larger updates: Huber Loss takes smaller increments compared to Mean Square Error Loss.
Convergence problems: Use the uber loss type.