COMP9444 18s2 Assignment 2 Report

Keshi Chen z5142821

Chuguan Tian z5145006

Description:

In this assignment we implemented a Deep Reinforcement Learning algorithm on a classic control task in the OpenAI AI-Gym Environment. Specifically, we will implement Q-Learning using a Neural Network as an approximator for the Q-function.

Basic Implementation

Hyper Parameters:

Discount factor: 0.9
Initial epsilon: 0.6
Final epsilon: 0.1
Epsilon decay steps: 100
Batch size: 128
Replay experience buffer size: 1000

Placeholders:

- `state_in` takes the current state of the environment, which is represented in our case as a sequence of reals.
- `action_in` accepts a one-hot action input. It is used to "mask" the q-values output tensor and return a q-value for that action.
- `target_in` is the Q-value to move the network towards producing. This is one of the components that seperates RL from other forms of machine learning.

Network Graph:

- We defined a DQN network.

- `q_values`: Tensor containing Q-values for all available actions.

```
q_values = tf.matmul(yIn, weightOut) + biasOut
```

- `q_action`: This should be a rank-1 tensor containing 1 element. This value is the q-value for the action set in the action_in placeholder

```
q_action = tf.reduce_sum(tf.multiply(q_values, action_in), reduction_indices=1)
```

- Loss/Optimizer is a function of target_in and q_action.

```
loss = tf.reduce_sum(tf.square(target_in - q_action))
optimizer = tf.train.AdamOptimizer().minimize(loss)
```

Main Loop:

- For each time step, the agent chooses an action by greedy exploration, then receive the latest observation, and update the experience replay. We then train the agent from randomly picked batch of replays and recalculate the Q-value before we move to the next state .

Structure of Network: One layer deep Q learning neural network

Input layer size: number of states

Hidden layer size: 32

Output layer size: number of actions

Activation function: tanh

Output: Q values, Q(action)

Loss function: (Q(target) - Q(action))^2

Optimizer function: AdamOptimizer

Special implementation:

In order to let the agent learn better, we force the reward to be –1 as punishment, whenever the agent lose the game, instead of 1 by the environment's default.

After several times of trying various of different methods, we can get 200 rewards after 100 episodes, and after 200 episodes, the average reward appears to be stable in most of the cases:

episode: 100 epsilon: 0.2174232107162985 Evaluation Average Reward: 200.0
episode: 200 epsilon: 0.07958392686562948 Evaluation Average Reward: 200.0
episode: 300 epsilon: 0.029130291078343817 Evaluation Average Reward: 200.0
episode: 400 epsilon: 0.01066262864537686 Evaluation Average Reward: 200.0
episode: 500 epsilon: 0.0039028669271942975 Evaluation Average Reward: 200.0

Future development:
Extend the network into Double DQN to make the performance more stable