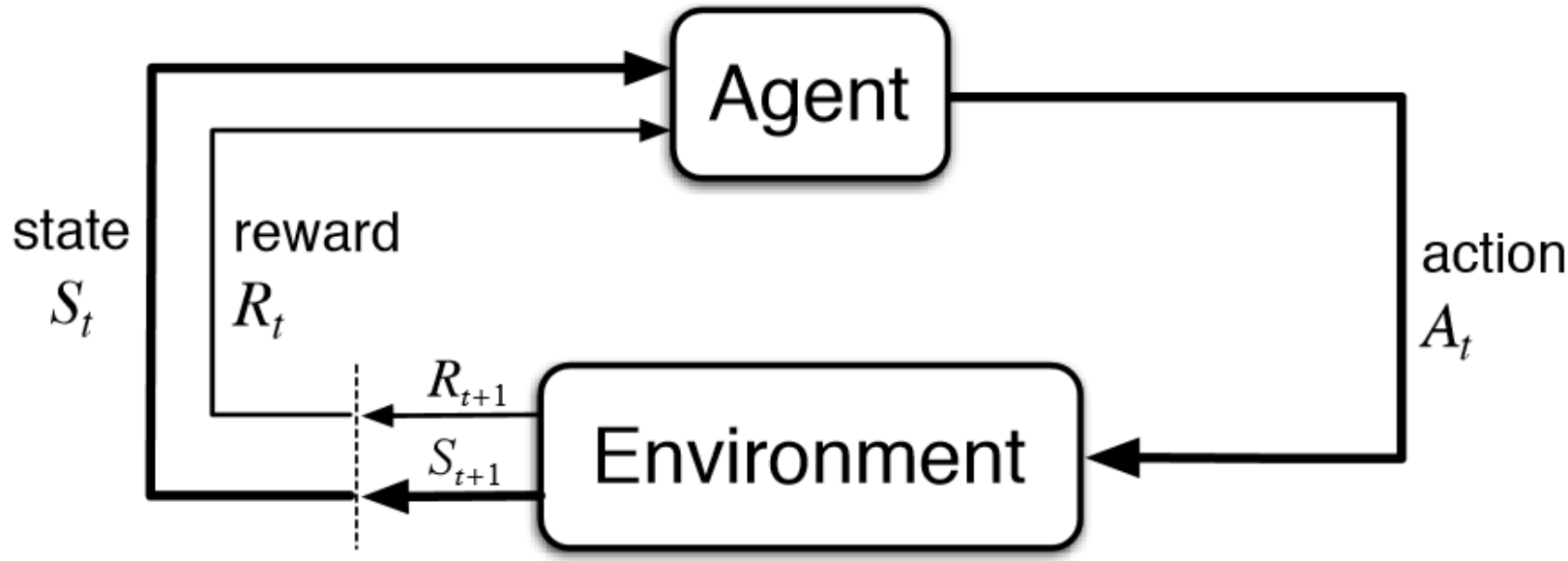
An Introductory Tutorial on Implementing DRL Algorithms with DQN and TensorFlow

Tim Tse

May 18, 2018

Recap: The RL Loop



A Simplified View of the Implementation Steps for RL Algorithms

1. The environment (taken care of by `OpenAI Gym`)
2. The agent
3. A `while` loop that simulates the interaction between the agent and environment

A Simplified View of the Implementation Steps for RL Algorithms

1. The environment (taken care of by `OpenAI Gym`)
2. The agent
3. A `while` loop that simulates the interaction between the agent and environment

Implementing the DQN Agent

- We wish to learn state-action value function $Q(s_t, a_t)$ for all s_t, a_t.

Implementing the DQN Agent

- We wish to learn state-action value function $Q(s_t, a_t)$ for all s_t, a_t.
- Recall the recursive relationship, $Q(s_t, a_t) = r_t + \gamma \max_{a'} Q(s_{t+1}, a')$.

Implementing the DQN Agent

- We wish to learn state-action value function $Q(s_t, a_t)$ for all s_t, a_t.
- Recall the recursive relationship, $Q(s_t, a_t) = r_t + \gamma \max_{a'} Q(s_{t+1}, a')$.
- Using this relation, define MSE loss function

$$L(w) = \frac{1}{N} \sum_{i=1}^{N} (\underbrace{r_t^i + \gamma \max_{a'} Q_{\bar{w}}(s_{t+1}^i, a')}_{\text{target}} - \underbrace{Q_w(s_t^i, a_t^i)}_{\text{current estimate}})^2$$

where $\{(s_t^1, a_t^1, r_t^1, s_{t+1}^1), \cdots, (s_t^N, a_t^N, r_t^N, s_{t+1}^N)\}$ are the training tuples and $\gamma \in [0, 1]$ is the discount factor.

Implementing the DQN Agent

- We wish to learn state-action value function $Q(s_t, a_t)$ for all s_t, a_t.
- Recall the recursive relationship, $Q(s_t, a_t) = r_t + \gamma \max_{a'} Q(s_{t+1}, a')$.
- Using this relation, define MSE loss function

$$L(w) = \frac{1}{N} \sum_{i=1}^{N} (\underbrace{r_t^i + \gamma \max_{a'} Q_{\bar{w}}(s_{t+1}^i, a')}_{\text{target}} - \underbrace{Q_w(s_t^i, a_t^i)}_{\text{current estimate}})^2$$

  where $\{(s_t^1, a_t^1, r_t^1, s_{t+1}^1), \cdots, (s_t^N, a_t^N, r_t^N, s_{t+1}^N)\}$ are the training tuples and $\gamma \in [0, 1]$ is the discount factor.
- Parameterize $Q(\cdot, \cdot)$ using a function approximator with weights w.

Implementing the DQN Agent

- We wish to learn state-action value function $Q(s_t, a_t)$ for all s_t, a_t.
- Recall the recursive relationship, $Q(s_t, a_t) = r_t + \gamma \max_{a'} Q(s_{t+1}, a')$.
- Using this relation, define MSE loss function

$$L(w) = \frac{1}{N} \sum_{i=1}^{N} (\underbrace{r_t^i + \gamma \max_{a'} Q_{\bar{w}}(s_{t+1}^i, a')}_{\text{target}} - \underbrace{Q_w(s_t^i, a_t^i)}_{\text{current estimate}})^2$$

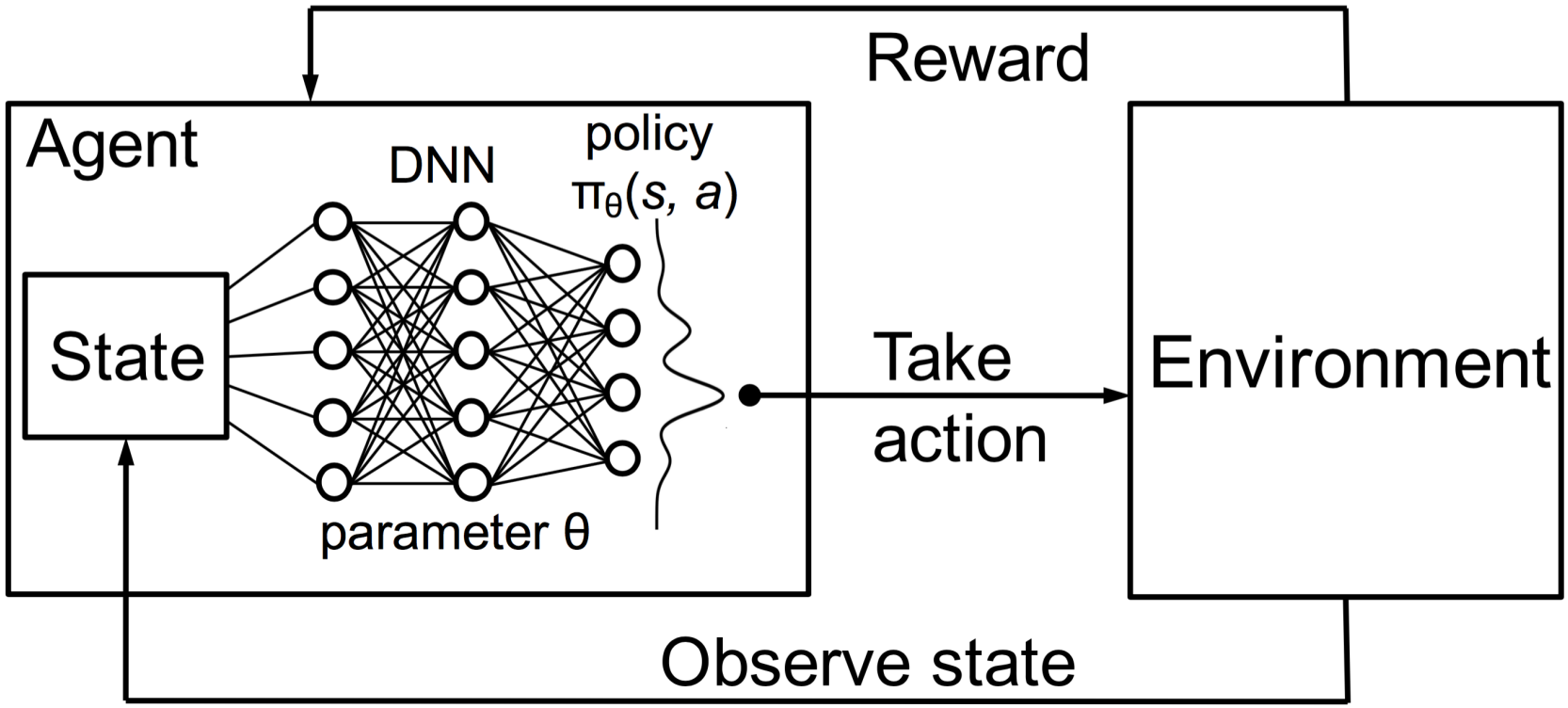
  where $\{(s_t^1, a_t^1, r_t^1, s_{t+1}^1), \cdots, (s_t^N, a_t^N, r_t^N, s_{t+1}^N)\}$ are the training tuples and $\gamma \in [0, 1]$ is the discount factor.
- Parameterize $Q(\cdot, \cdot)$ using a function approximator with weights w.
- With "deep" RL our function approximator is an artificial neural network (so w denotes the weights of our ANN).

Implementing the DQN Agent

- We wish to learn state-action value function $Q(s_t, a_t)$ for all s_t, a_t.
- Recall the recursive relationship, $Q(s_t, a_t) = r_t + \gamma \max_{a'} Q(s_{t+1}, a')$.
- Using this relation, define MSE loss function

$$L(w) = \frac{1}{N} \sum_{i=1}^{N} (\underbrace{r_t^i + \gamma \max_{a'} Q_{\bar{w}}(s_{t+1}^i, a')}_{\text{target}} - \underbrace{Q_w(s_t^i, a_t^i)}_{\text{current estimate}})^2$$

  where $\{(s_t^1, a_t^1, r_t^1, s_{t+1}^1), \cdots, (s_t^N, a_t^N, r_t^N, s_{t+1}^N)\}$ are the training tuples and $\gamma \in [0, 1]$ is the discount factor.
- Parameterize $Q(\cdot, \cdot)$ using a function approximator with weights w.
- With "deep" RL our function approximator is an artificial neural network (so w denotes the weights of our ANN).
- For stability, target weights $\bar{w}$ are held constant during training.


Reward
Agent
DNN
policy
$\pi_\theta(s, a)$
State
Take
action
Environment
parameter θ
Observe state

Translating the DQN Agent to Code...

Let's look at how we can do the following in TensorFlow:

1. Declare an ANN that parameterizes $Q(s, a)$.
   - I.e., our example ANN will have structure `state_dim`-256-256-`action_dim`.
2. Specify a loss function to be optimized.

Two Phases of Execution in TensorFlow

1. Building the computational graph.
   - Specifying the structure of your ANN (i.e., which outputs connect to which inputs).
   - Numerical computations **are not** being performed during this phase.
2. Running `tf.Session()`.
   - Numerical computations **are** being performed during this phase.
   - For example,
     - Initial weights are being populated.
     - Tensors are being passed in and outputs are computed (forward pass).
     - Gradients are being computed and back-propagated (backward pass).

Implementation Steps for RL Algorithms

1. The environment (taken care of by `OpenAI Gym`)
2. The agent
3. The logic that ties the agent and environment together

The Interaction Loop Between Agent and Environment

for *e number of epochs* **do**
    Initialize environment and observe initial state `s`;
    while *epoch is not over* **do**
        In state `s`, take action `a` with an exploration policy (i.e., ϵ-greedy) and receive next state `s'` and reward `r` feedback;
        Update exploration policy;
        Cache training tuple (`s`,`a`,`r`,`s'`);
        Update agent;
        `s` $\leftarrow$ `s'`;
    end
end

Algorithm 1: An example of one possible interaction loop between agent and environment.