CSE474/574: Introduction to Machine Learning (Fall 2018)
Project 4: Tom and Jerry in Reinforcement learning

Overview

The project combines reinforcement learning and deep learning. Your task is to teach the agent to navigate in the grid-world environment. We have modeled Tom and Jerry cartoon, where Tom, a cat, is chasing Jerry, a mouse. In our modeled game the task for Tom (an agent) is to find the shortest path to Jerry (a goal), given that the initial positions of Tom and Jerry are deterministic. To solve the problem, we would apply deep reinforcement learning algorithm - DQN (Deep Q-Network), that was one of the first breakthrough successes in applying deep learning to reinforcement learning.

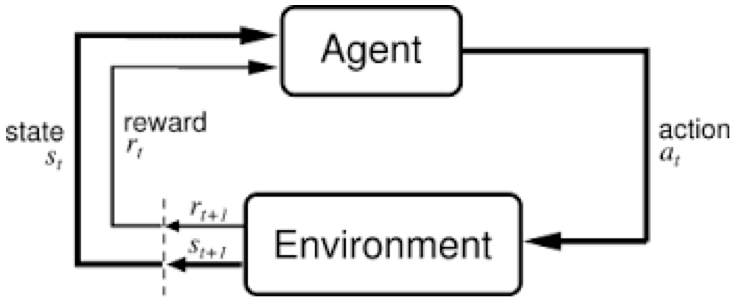
Reinforcement Learning
Reinforcement Learning is the science of making optimal decisions using experiences.

- Agent: An agent takes actions; for example, "Tom" navigating a video game. The algorithm is the agent.
- Action (A): A is the set of all possible moves the agent can make. It should be noted that agents choose among a list of possible actions. In video games, the list might include running right or left, jumping high or low, crouching or standing still.
- Discount factor: The discount factor is multiplied by future rewards as discovered by the agent in order to dampen these reward's effect on the agent's choice of action. It is designed to make future rewards worth less than immediate rewards; i.e. it enforces a kind of short-term hedonism in the agent. Often expressed with the lower-case Greek letter gamma: γ. If γ is .8, and there's a reward of 10 points after 3 time steps, the present value of that reward is $0.8^3 \times 10$. A discount factor of 1 would make future rewards worth just as much as immediate rewards.
- Environment: The world through which the agent moves. The environment takes the agent's current state and action as input, and returns as output the agent's reward and its next state
- State (S): A state is a concrete and immediate situation in which the agent finds itself; i.e. a specific place and moment, an instantaneous configuration that puts the agent in relation to other significant things such as tools, obstacles, enemies or prizes. It can the current situation returned by the environment, or any future situation.
- Reward (R): A reward is the feedback by which we measure the success or failure of an agent's actions. For example, in the given game, when Tom reaches close to Jerry, he wins points. From any given state, an agent sends output in the form of actions to the environment, and the environment returns the agent's new state (which resulted from acting on the previous state) as well as rewards, if there are any. Rewards can be immediate or delayed. They effectively evaluate the agent's action.
- Policy (π): The policy is the strategy that the agent employs to determine the next action based on the current state. It maps states to actions, the actions that promise the highest reward.
- Value (V): The expected long-term return with discount, as opposed to the short-term reward R. $V\pi(s)$ is defined as the expected long-term return of the current state under policy π. We discount rewards, or lower their estimated value, the further into the future they occur. See discount factor. And remember Keynes: "In the long run, we are all dead." That's why you discount future rewards.
- Q-value or action-value (Q): Q-value is similar to Value, except that it takes an extra parameter, the current action a. $Q\pi(s, a)$ refers to the long-term return of the current state, taking action a under policy π. Q maps state-action pairs to rewards.

Arooshi Avasthy arooshia@buffalo.edu

- Trajectory: A sequence of states and actions that influence those states. From the Latin "to throw across." The life of an agent is but a ball tossed high and arching through space-time.

So environments are functions that transform an action taken in the current state into the next state and a reward; agents are functions that transform the new state and reward into the next action. We can know the agent's function, but we cannot know the function of the environment. It is a black box where we only see the inputs and outputs. It's like most people's relationship with technology: we know what it does, but we don't know how it works. Reinforcement learning represents an agent's attempt to approximate the environment's function, such that we can send actions into the black-box environment that maximize the rewards it spits out.



In the feedback loop above, the subscripts denote the time steps t and t+1, each of which refer to different states: the state at moment t, and the state at moment t+1. Unlike other forms of machine learning – such as supervised and unsupervised learning – reinforcement learning can only be thought about sequentially in terms of state-action pairs that occur one after the other.

Reinforcement learning judges' actions by the results they produce. It is goal oriented, and its aim is to learn sequences of actions that will lead an agent to achieve its goal, or maximize its objective function.

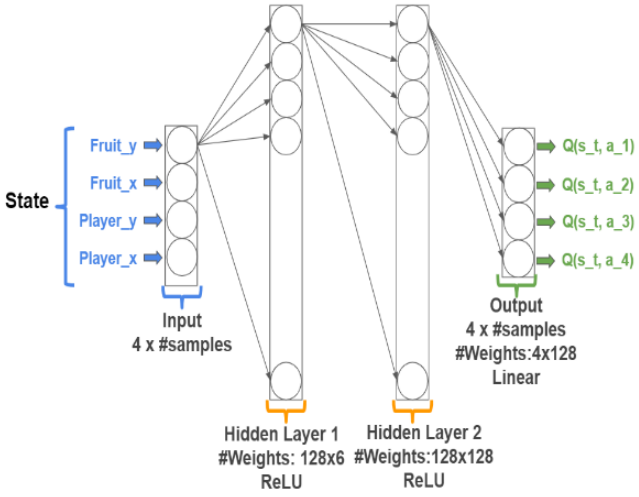
Breaking it down, the process of Reinforcement Learning involves these simple steps:

- Observation of the environment
- Deciding how to act using some strategy
- Acting accordingly
- Receiving a reward or penalty
- Learning from the experiences and refining our strategy
- Iterate until an optimal strategy is found

Tasks implemented

Build a 3-layer neural network using Keras library

Neural Network structure for our task¶



Our DQN takes a stack of six-tuple as an input. It is passed through two hidden networks, and output a vector of Q-values for each action possible in the given state.

Implement Q-function

- The central idea in Q-Learning is to recognize or learn the optimal action in every state visited by the system (also called the optimal policy) via trial and error.
- The trial and error mechanism can be implemented within the real-world system (commonly seen in robotics) or within a simulator (commonly seen in management science / industrial engineering).
- The agent chooses an action, obtains feedback for that action, and uses the feedback to update its database.
- In its database, the agent keeps a so-called Q-factor for every state-action pair. When the feedback for selecting an action in a state is positive, the associated Q-factor's value is increased, while if the feedback is negative, the value is decreased.
- The feedback consists of the immediate revenue or reward plus the value of the next state.

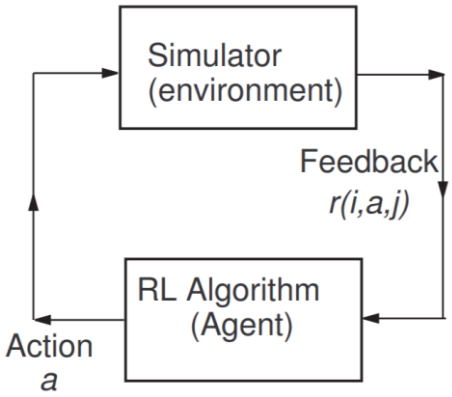


Figure 1: Trial and error mechanism of RL. The action selected by the RL agent is fed into the simulator. The simulator simulates the action, and the resultant feedback obtained is fed back

into the knowledge-base (Q-factors) of the agent. The agent uses the RL algorithm to update its knowledge-base, becomes smarter in the process, and then selects a better action.

- The immediate reward is denoted by r(i, a, j), where i is the current state, a the action chosen in the current state, and j the next state.
- The value of any state is given by the maximum Q-factor in that state. Thus, if there are two actions in each state, the value of a state is the maximum of the two Q-factors for that state.
- In mathematical terms:

$$\text{feedback} = r(i, a, j) + \lambda \max_b Q(j, b),$$

where λ is the discount factor, which discounts the values of future states. Usually, $\lambda = 1/(1 + R)$ where R is the rate of discounting.

The core of the Q-Learning algorithm uses the following updating equation:

$$Q(i, a) \leftarrow [1 - \alpha]Q(i, a) + \alpha [\, r(i, a, j) + \lambda \max_b Q(j, b) \,]$$

where α is the learning rate (or step size).

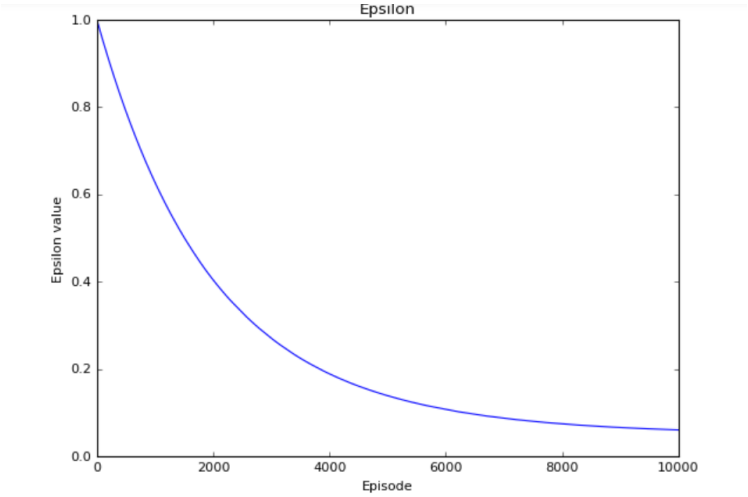
Implement exponential-decay formula for epsilon

Epsilon-greedy method is used for exploration during training. This means that when an action is selected in training, it is either chosen as the action with the highest q-value, or a random action. Choosing between these two is random and based on the value of epsilon, and epsilon is annealed during training such that initially, lots of random actions are taken (exploration), but as training progresses, lots of actions with the maximum q-values are taken (exploitation). After enough random exploration of actions, the Q-values tend to converge serving the agent as an action-value function which it can exploit to pick the most optimal action from a given state. There's a tradeoff between exploration (choosing a random action) and exploitation (choosing actions based on already learned Q-values). Epsilon is introduced to prevent the action from always taking the same route, and overfitting. Instead of just selecting the best learned Q-value action, we'll sometimes favor exploring the action space further. Lower epsilon value results in episodes with more penalties (on average) which is obvious because we are exploring and making random decisions.

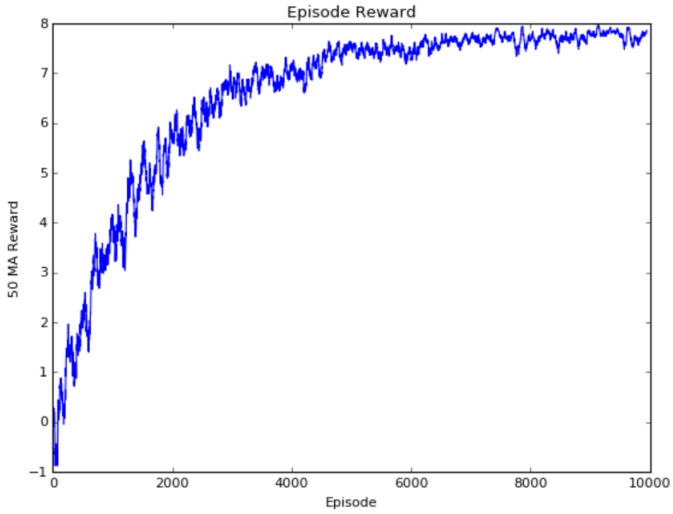
Exponential-decay formula for epsilon:
$\epsilon = \epsilon min + (\epsilon max - \epsilon min) * e^{-\lambda |S|}$,
$\epsilon = \epsilon\text{min} + (\epsilon\text{max} - \epsilon\text{min}) * e^{-\lambda |S|}$,

where $\epsilon min, \epsilon max \in [0,1]$ $\epsilon\text{min}, \epsilon\text{max} \in [0,1]$ \ λ - hyperparameter for epsilon \ |S| - total number of steps

Experiments: Tuning Hyperparameters Epsilon, Episodes, Rewards



This strategy is based on the -greedy strategy, however the value decays over time. In practice this means that the strategy starts out with a high , and thus a high exploration rate. Over time this grows ever smaller until it fades, optimally as the policy has converged, so that an optimal policy can be executed without having to take further (possibly sub optimal) exploratory actions. This strategy functions the same as normal -greedy, however the epsilon value changes as a function of time, according to γ t with γ as a function of time.



With increase in number of episodes reward points for the agent increases.

Ways to Improve the performance of the applied algorithm:

- *Change from Decaying ϵ-greedy to Softmax*
  Softmax When exploring, *ϵ*-greedy samples an action from a uniform distribution, which means it could pick an unsatisfactory action just as likely as an action which is preferable. In a Softmax policy the distribution between actions is not uniform, but is biased towards promising actions.
  The Softmax strategy calculates the probability of taking an action as follows for the output layer:

$$P(a) = \frac{e^{x_j^T}}{\sum_{k=1}^{K} e^{x_k^T}}$$

  where: x = the net input (connected nodes multiplied by their weight) for output node j
  K = the amount of output nodes
  T = the temperature parameter which can be tweaked to impact exploratory behavior

- *Adding more layers to the Neural Network*

- *Adopting Convolutional Neural Network.*
  Convolutional NNs are better at dealing with multiple kinds of spatial deformations

Writing tasks [max 40 points total]

1. Explain what happens in reinforcement learning if the agent always chooses the action that maximizes the Q-value. Suggest two ways to force the agent to explore. [20 points]

Ans) The agent might be stuck performing non-optimal actions. Unless it has already found the optimal policy, exploring actions which do not have the highest Q-value might allow it to find a better policy.

Two ways to force a Q-learning agent to explore.

- The *ε*-greedy strategy is to select the greedy action (one that maximizes Q[s,a]) all but *ε* of the time and to select a random action *ε* of the time, where $0 \leq \varepsilon \leq 1$.
- An alternative is "optimism in the face of uncertainty": initialize the Q-function to values that encourage exploration. If the Q-values are initialized to high values, the unexplored areas will look good, so that a greedy search will tend to explore.

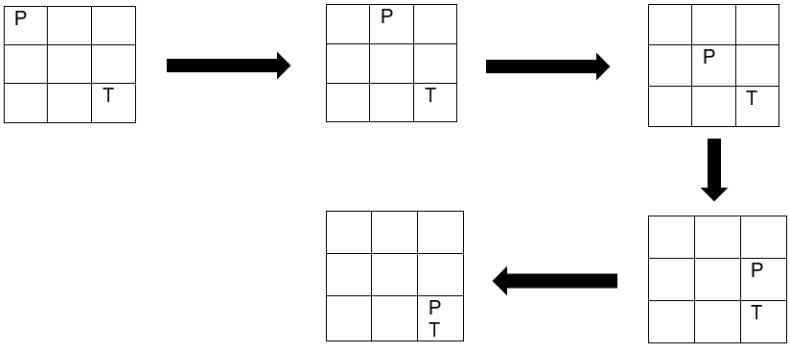
2. Calculate Q-value for the given states and provide all the calculation steps. [20 points]
   Consider a deterministic environment which is a 3x3 grid, where one space of the grid is occupied by the agent (green square) and another is occupied by a goal (yellow square). The agent's action space consists of 4 actions: UP, DOWN, LEFT, and RIGHT. The goal is to have the agent move onto the space that the goal is occupying in as little moves as possible. The episode terminates as soon as the agent reaches the goal.

   Initially, the agent is set to be in the upper-left corner and the goal is in the lower-right corner. The
   agent receives a reward of:
   1 when it moves closer to the goal

-1 when it moves away from the goal
0 when it does not move at all (e.g., tries to move into an edge)

Ans) **Q-Table**

- Initialize the Q-table by all zeros.
- Start exploring actions: For each state, select any one among all possible actions for the current state (S).
- Travel to the next state (S') as a result of that action (a).
- For all possible actions from the state (S') select the one with the highest Q-value.
- Update Q-table values using the equation.
- Set the next state as the current state.
- If goal state is reached, then end and repeat the process.



State	UP	DOWN	LEFT	RIGHT
0	3.900	3.9403	3.900	3.9403
1	2.049	2.9701	2.900	2.9701
2	1.9403	1.99	1.9403	1.99
3	0.9701	1	0.9701	1.99
4	0	0	0	0

The core of the Q-Learning algorithm uses the following updating equation:

$Q(i, a) \leftarrow [1 - \alpha]Q(i, a) + \alpha [r(i, a, j) + \lambda \max b Q(j, b)]$

where α is the learning rate (or step size).

Calculations:

$Q(S3, Down) = 1 + \lambda * Max(Q(S4,action))$
$= 1 + 0.99 * 0$

= 1

After moving down from S3 state S4 is achieved where Max(Q(S4,0)) is zero.

Q(S3, Right) = 0 + λ* Max(Q(S3,action))

= 1 + 0.99 * 1

= 0.99

Taking right at S3 will result in moving out of grid and therefore the state remains same.
So immediate reward would be zero and Max(Q(S3,action)) =1 because Q(S3,D) =1 and no other move takes us closer to goal, so Max(Q(S3,action)) =1.

Q(S2, Right) = 1 + λ* Max(Q(S3,action))

= 1 + 0.99 * 1

= 1.99

Taking Right from S2 puts us to state S3 and makes us move closer to goal so +1 is the immediate reward.

Q(S1, Down) = 1 + λ* Max(Q(S2,action))

= 1 + 0.99 * 1.99

= 2.9701

Immediate reward is +1 because we are moving closer to the goal.

Q(S1, Up) = 0 + λ* Max(Q(S1,action))

= 0 + 0.99 * 2.9701

= 2.9403

Immediate reward is zero because on moving up we move out of grid so state remains same.

Q(S0, Right) = 1 + λ* Max(Q(S1,action))

= 1 + 0.99 * 2.9701

= 3.9403

Immediate reward is +1 because on moving right we move closer to goal and achieve new state S1 where Max(Q(S1,action)) = 2.9701.

Q(S3, Left) = -1 + λ* Max(Q(S2,action))

= -1 + (0.99 * 1.99)

=0.9701

On taking left we go back to state S2 and move away from goal, so immediate reward is -1.

Q(S1, Left) = -1 + λ* Max(Q(S0,action))

= -1 + (0.99 * 3.9403)
= -1 + 3.9009
=2.90099

On turning left we are moving away from goal so immediate reward is -1, so Max(Q(S0,action)) =3.9403.

Q(S0, Up) = 0 + λ* Max(Q(S0,action))
= 0 + (0.99 * 3.9403)
= 0 + 3.9009
=3.90099
State remains same so immediate reward is zero.

Q(S0, Left) = 0 + λ* Max(Q(S0,action))
= 0 + (0.99 * 3.9403)
= 0 + 3.9009
=3.90099

State remains same so immediate reward is zero.

Q(S3, Up) = -1 + λ* Max(Q(S2,action))
= -1 + (0.99 * 1.99)
= -1 + 1.9701
=0.9701

Here next state after going up will be 2 blocks away from goal and state S2 is also 2 blocks away from goal. So we can assume going up from S3 we move away from goal and new state would be similar to S2.

Q(S0, Down) = 1 + λ* Max(Q(S1,action))
= 1 + (0.99 * 2.9701)
= 1 + 2.9403
= 3.9403
New state will be similar to S1 and since we are moving closer to goal so immediate reward will be 1.

Q(S1, Right) = 1 + λ* Max(Q(S2,action))
= 1 + (0.99 * 1.99)
= 1 + 1.9701
= 2.9701

Q(S2,Up) = -1 + λ* Max(Q(S1,action))
= -1 + (0.99 * 2.9701)
= -1 + 2.9403
= 1.9403

Q(S2,Left) = -1 + λ* Max(Q(S1,action))
= -1 + (0.99 * 2.9701)
= -1 + 2.9403
= 1.9403

Q(S2,Down) = 1 + λ* Max(Q(S3,action))
= 1 + (0.99 * 1)
= -1 + 0.99
= 1.99

References:

[1] https://theses.ubn.ru.nl/bitstream/handle/123456789/5216/Nieuwdorp%2C_T._1.pdf?sequence=1

[2] https://medium.freecodecamp.org/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe

[3]http://www.aispace.org/exercises/exercise11-a-1.shtml

[4] https://skymind.ai/wiki/deep-reinforcement-learning

Arooshi Avasthy arooshia@buffalo.edu