Project 4

Reinforcement Learning

Illa Nuka Saranya

50248926

05-Dec-2018

Saranya@buffalo.edu

1. PROBLEM DEFINITION

The task is to teach the agent to navigate in the grid-world environment. 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. We apply deep reinforcement learning algorithm - DQN (Deep Q-Network which is one of the first breakthrough successes in applying deep learning to reinforcement learning.

2. Implementation

1. Build a 3-layer neural network using Keras library

In order to make the production and updation of Q-table effective in big space environments, create a neural network that will approximate, given a state, the different Q-values for each action. Our Deep Q Neural Network takes the states as an input(Grid Positions of Tom and jerry). These pass through its network, and output a vector of Q-values for each action possible in the given state. We need to take the biggest Q-value of this vector to find our best action.

The model used here is a sequential model with three layers.

- The model's structure is: LINEAR → RELU → LINEAR → RELU → LINEAR.
- Activation function for the first and second hidden layers is 'relu'
- Activation function for the output layer is 'linear' (that will return real values)
- Input dimensions for the first hidden layer equals to the size of your observation space (state_size)
- Number of hidden nodes is 128 for both hidden layers
- Number of the output should be the same as the size of the action space (action_size)

```
first_dense_layer_nodes  = 128 #number of hidden nodes
 second_dense_layer_nodes = 4 #number of output nodes
 model.add(Dense(first_dense_layer_nodes, input_dim=4))
 model.add(Activation('relu'))
 model.add(Dense(first_dense_layer_nodes,input_dim=128))
 model.add(Activation('relu'))
 model.add(Dense(second_dense_layer_nodes))
 model.add(Activation('linear'))
```

Learning rate schedules seek to adjust the learning rate during training by reducing the learning rate according to a pre-defined schedule. Common learning rate schedules include time-based decay**,** step decay and exponential decay**.**

2. Implement exponential-decay formula for epsilon

- Here we use exponential decay of epsilon value as our agent will randomly select its action at first by a certain percentage, called "exploration rate" or "epsilon".
- This is because at first, it is better for the agent to try all kinds of things before it starts to see the patterns. When it is not deciding the action randomly, the agent will predict the reward value based on the current state and pick the action that will give the highest reward.

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * e^{-\lambda|S|},$$

- If agent has a long time to accrue reward we begin with more exploration, so that future actions can be planned more effectively with knowledge and so the value of epsilon is more in the beginning of the training.
- As time progresses we move towards more exploitation, hence the decay of epsilon value takes place when the agent has only a short amount of time to accrue reward

```
self.epsilon=self.min_epsilon+np.dot((self.max_epsilon-self.min_epsilon),exp(-
np.dot(self.lamb,self.steps)))
```

3. Implement Q-function

Q-Learning is an example of model-free learning algorithm. It does not assume that agent knows anything about the state-transition and reward models. However, the agent will discover the good and bad actions by trial and error. The basic idea of Q-Learning is to approximate the state-action pairs Q-function from the samples of Q(s, a) that we observe during interaction with the environment. This approach is known as **Time-Difference Learning.**

- It shows that if the agent learns the Q function instead of the V* function, it will be able to select optimal actions even when it has no knowledge of functions r(s,a) and δ(s,a)
- It need only consider each action a in its current state a and choose action that maximizes Q(s,a)
- Surprising that one can choose globally optimal action sequences by reacting repeatedly to the local values of Q for the current state
- Without conducting a look ahead search to explicitly consider what state results from what action
- Value of Q for the current state and action summarizes in a single number all information needed to determine discounted cumulative reward
- **calculate the maximum expected future reward, for each action at each state**

The model is trained and improvised to increase the mean reward value and to help in quick convergence (Agent reaching the goal state).

Choose an action *a* in the current states based on the current Q-value estimates. As all vales are zero in the beginning, that's where the exploration/exploitation trade-off takes place.

Greedy Epsilon strategy to update Q table:

- We specify an exploration rate "epsilon," which we set to 1 in the beginning. This is the rate of steps that we'll do randomly. In the beginning, this rate must be at its highest value, because we don't know anything about the values in Q-table. This means we need to do a lot of exploration, by randomly choosing our actions.
- We generate a random number. If this number > epsilon, then we will do "exploitation" (this means we use what we already know to select the best action at each step). Else, we'll do exploration.
- The idea is that we must have a big epsilon at the beginning of the training of the Q-function. Then, reduce it progressively as the agent becomes more confident at estimating Q-values.

The hyper-parameters are changed to obtain the best possible values.

For the below values, lambda and learning rate are kept as 0.00005 and 0.0025 respectively.

All of the below parameters are measured for 10k episodes.

Optimizer	Reward Mean	No of times Tom caught Jerry	First episode at which Tom caught Jerry	Total time elapsed
Adam	4.6971	2741	17	893.64s
RMSprop	4.0676	2728	123	915.34s
Adagrad	-0.3765	28	177	918.78s

We choose Adam as the optimizer as it gives best results.

MinEpsilon	MaxEpsilon	Reward Mean	No of times Tom caught Jerry in 10k episodes	First episode at which Tom caught Jerry	Total time elapsed
0.05	2	**3.9445**	**2417**	61	**970.67s**
0.05	1	4.6971	2741	17	893.64s
0.01	0.5	**3.247**	3326	78	**976.2s**
0.01	0.1	1.8626	4491	339	963.3s

It is observed that though the reduction of the max_epsilon to lower values improves the number of times the model convergence as it gets trained, but the mean reward and the first time of convergence are at poor values.

It is observed that the model when trained using Adam optimizer and the max and min epsilon values of 0.05 and 1 are trained very well.

Writing Tasks:

1. In reinforcement learning, if the agent always chooses the action that maximizes the Q-value, there will be no exploration of new actions because maximum of Q-value can be obtained at every stage even without much exploration because of the already exploited values.

Ways to force the agents to explore.

Avoid forgetting previous experiences

Because there is high correlation between states and actions, the variability of weights can be a problem. In the case of sequential model, the agent tends to overwrite old experiences with the new ones. Hence we use the memory that stores experience tuples while interacting with the environment, and then we sample a small batch of tuple to feed our neural network randomly. Allows the agent to not only learn from recent observations but also from the previous observations

Reducing Correlation between experiences

As every action affects the next state, we get a sequence of experience tuples which can be highly correlated.

When the network is trained in sequential order, we risk our agent being influenced by the effect of this correlation which can be avoided by sampling from the replay buffer at random hence preventing the action values from oscillating or diverging catastrophically.

Replay buffer randomly chooses fixed size of samples in batches and feed them to the network

Negative Scores:

Giving negative scores when the agent does some wrong actions improvises the way in which the agent chooses its next actions.

2. Given is a deterministic environment of 3 x 3 grid and 5 states from S_0 to $S_{4.}$

$$Q(s_t, a_t) = r_t + \gamma * max_a Q(s_{t+1}, a)$$

The Q-Learning algorithm is as follows:

1. We first initialize the Q-Table.
2. Choose an action
3. Perform an action
4. Measure reward
5. Update Q.

We build a Q-table, with *m* cols (m= number of actions), and *n* rows (n = number of states). We initialize the values at 0.

	Up	Down	Right	Left
S_0	0	0	0	0
S_1	0	0	0	0
S_2	0	0	0	0
S_3	0	0	0	0
S_4	0	0	0	0

We first choose the state S_4 and perform an all possible actions.

As S_4 itself is the goal, the episode ends right here and hence there are no next states possible.As we can't perform any action here as the episode terminates, all the values remain zero for the state 4 in Q-Table.

Choose State 3

At state 3, the agent can perform three actions namely left, up and down and hence three states are possible.

$Q(S_{3,}up) = -1+ 0.99* \max_a Q(S_{12}, S_{23}) = -1 + 0.99 *\max(0,0) = -1$

$Q(S_{3,}down) = 1+ 0.99 (\max_a Q(S_{33})) = 1+ 0.99(0) =1$

$Q(S_{3,}left) = -1 + 0.99(\max_a Q(S_{22,}S_{12,}S_{21,}S_{23})) = -1+ 0.99(0,0,0,0) =-1$

	Up	Down	Right	Left
S_0	0	0	0	0
S_1	0	0	0	0
S_2	0	0	0	0
S_3	-1	1	0	-1
S_4	0	0	0	0

$Max_a(S_{3,}a)= Q(S_{23,}down) = 1$

Choose State 2

$Q(S_{2,}up) = -1 + 0.99(\max_a Q(S_{11,} S_{13,} S_{22})) = -1 + 0.99*\max(0,0,0) = -1$

$Q(S_{2,}down) = 1+ 0.99 (\max_a Q(S_{31,} S_{33,} S_{22})) = 1+ 0.99*\max (0,1,0) =1.99$

$Q(S_{2,}left) = -1 + 0.99(\max_a Q(S_{11,}S_{31,}S_{22})) = -1+ 0.99*\max (0,0,0) =-1$

$Q(S_{2,}right) = 1 + 0.99(max_a\, Q(S_{13,}\, S_{33,}\, S_{22})) = -1 + 0.99 * max(0,1,0) = 1.99$

$Max_a(S_{2,}a) = Q(S_{22,}right) = 1.99$

Choose State 1

$Q(S_{1,}down) = 1 + 0.99\,(max_a\, Q(S_{12,}\, S_{21,}\, S_{32,}\, S_{23})) = 1 + 0.99(0,-1,1.99,1) = 2.9701$

$Q(S_{1,}left) = -1 + 0.99(max_a\, Q(S_{12,}S_{21})) = -1 + 0.99(0) = -1$

$Q(S_{1,}right) = 1 + 0.99(max_a\, Q(S_{32,}S_{12})) = 1 + 0.99(0,0) = 1$

$Max_a(S_{1,}a) = Q(S_{12,}right) = 2.9701$

Choose State 0

$Q\,(S_{0,}\, down) = 1 + 0.99\,(max_a\, Q(S_{22,}S_{31,}S_{11})) = 1 + 0.99(2.7901,0,0) = 3.9403$

$Q\,(S_{0,}right) = 1 + 0.99(max_a\, Q(S_{11,}S_{13,}S_{22})) = 1 + 0.99(0,2.7901,1.99) = 3.9403$

	Up	Down	Right	Left
S_0	0	3.9403	1	0
S_1	0	2.9701	1	-1
S_2	-1	1	1.99	-1
S_3	-1	1	0	-1
S_4	0	0	0	0

This table represents the states of the optimal path. If we consider the other states as well,the tabular values gets updated as follows.

	Up	Down	Right	Left
S_0	0	3.9403	3.9403	0
S_1	0	2.9701	2.9701	-1
S_2	-1	1.99	1.99	-1
S_3	0.9701	1	0	0.9701
S_4	0	0	0	0