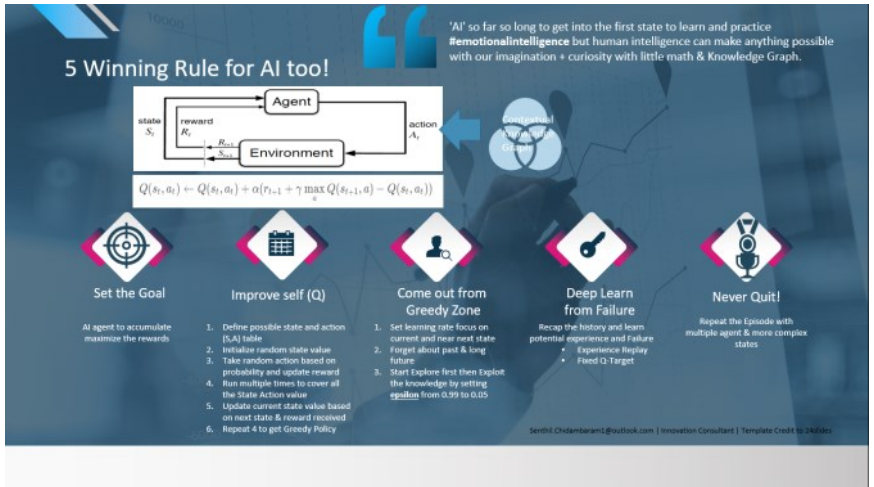


AI Navigator - Report

06 June 2020 13:11

S.No	Topic	Description
1.	Project Goal	Train a AI agent to collect min +13 rewards as a score within 100 consecutive episodes
2.	Solution Approach	<p>1. Set the Goal for the AI - Agent</p> <ol style="list-style-type: none"> 1. Initialize Agent , Network Model and integrate with environment 2. Reset the Environment and initialize 'Q table ' which help Agent to select Optimum Policy based on 'action-value' from the Q table 3. Start with zero /random step but improve self - $Q(S,A)$ find the 'action-value' for each state and do iterative steps to find the maximum 4. Form a Greedy policy - Best Action to take to get maximize rewards 5. Explore and Exploit the environment using discounted learning , Epsilon start from .99 to 0.05 6. Store Memories and learn from failure or potential actions using Agent.Step 7. Using Fixed target and Replay memories and take function approximation using Deep Qnetwork algorithm <p>in Simple , Here is the 5 Winning rules for AI too</p>  <p>The infographic '5 Winning Rule for AI too!' illustrates the reinforcement learning process. At the top, it shows the interaction between an Agent and an Environment. The Agent takes an action A_t based on the current state S_t, which results in a reward R_t and a next state S_{t+1}. The Q-value is updated using the formula: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha (R_t + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t))$. Below this, five rules are listed with icons:</p> <ol style="list-style-type: none"> Set the Goal: AI agent to accumulate maximize the rewards. Improve self (Q): <ol style="list-style-type: none"> 1. Define possible state and action (S,A) table 2. Initialize random state value 3. Take random action based on probability and update reward 4. Run multiple times to cover all the State Action value 5. Update current state value based on next state & reward received 6. Repeat 4 to get Greedy Policy Come out from Greedy Zone: <ol style="list-style-type: none"> 1. Set learning rate focus on current and near next state 2. Forget about past & long future 3. Start Explore first then Exploit the knowledge by setting epsilon from 0.99 to 0.05 Deep Learn from Failure: Recap the history and learn potential experience and Failure. <ul style="list-style-type: none"> • Experience Replay • Fixed Q Target Never Quit! Repeat the Episode with multiple agent & more complex states. <p>Source: Senthil Chidambaram@outlook.com Innovation Consultant Template Credit to 3d4dies</p>
3.1	Core Module - Qdeepnetwork (nn_model.py)	<p>Build a Convolutional Neural Network Model with 3 layers once for input possible 37 state vectors then 64 hidden layers and 4 output layers to choose the best probable actions for the given environment state</p> <ol style="list-style-type: none"> 1. Initialize and set the properties for Feedforward Convolutional Neural Network with relu as an activation function
3.2	Agent (agent.py)	<p>Define a Ai agent with 2 important properties like</p> <ul style="list-style-type: none"> • 'Qnetwork table ' to store 'action-value' matrix for each state and corresponding actions taken • self-memory to store experiences <p>Define methods</p> <ul style="list-style-type: none"> • Act • Step • Learn

		<div data-bbox="592 94 1513 745"> <h3>Algorithm: Deep Q-Learning</h3> <ul style="list-style-type: none"> Initialize replay memory D with capacity N Initialize action-value function \hat{q} with random weights \mathbf{w} Initialize target action-value weights $\mathbf{w}^- \leftarrow \mathbf{w}$ for the episode $e \leftarrow 1$ to M: <ul style="list-style-type: none"> Initial input frame x_1 Prepare initial state: $S \leftarrow \phi(\langle x_1 \rangle)$ for time step $t \leftarrow 1$ to T: <div data-bbox="632 510 734 539">SAMPLE</div> <div data-bbox="767 439 1481 745"> <p>Choose action A from state S using policy $\pi \leftarrow \epsilon\text{-Greedy}(\hat{q}(S, A, \mathbf{w}))$</p> <p>Take action A, observe reward R, and next input frame x_{t+1}</p> <p>Prepare next state: $S' \leftarrow \phi(\langle x_{t-2}, x_{t-1}, x_t, x_{t+1} \rangle)$</p> <p>Store experience tuple (S, A, R, S') in replay memory D</p> <p>$S \leftarrow S'$</p> <div data-bbox="651 663 730 692">LEARN</div> <p>Obtain random minibatch of tuples (s_j, a_j, r_j, s_{j+1}) from D</p> <p>Set target $y_j = r_j + \gamma \max_a \hat{q}(s_{j+1}, a, \mathbf{w}^-)$</p> <p>Update: $\Delta \mathbf{w} = \alpha (y_j - \hat{q}(s_j, a_j, \mathbf{w})) \nabla_{\mathbf{w}} \hat{q}(s_j, a_j, \mathbf{w})$</p> <p>Every C steps, reset: $\mathbf{w}^- \leftarrow \mathbf{w}$</p> </div> </div>
3.3	Navigator.ipynb	<p>Python notebook contains code to train as well as test the trained agent score</p> <ol style="list-style-type: none"> 1. Import Unity Environment and load the banana brain 2. Initialize AI Agent and take random action to see the score 3. Define DQN and test_run function which actually make the agent interact through Python API and train and save weights and re run 4. Set 10000 Episodes and started trained and at 800th episodes average score crossed 17 and network weights been saved after some trail and error approach by changing the epsilon value and seed value mostly 5. Once get the avg score >17 , loaded the weightage file and test the agent performance by setting the 'train_mode= False' while re initiating the environment 6. Tested and confirm Avg scores is 13+ within 50 consecutive episodes and results been captured 7. History shows number of trial runs and training parameters
T		