Machine Learning Engineer Nanodegree
Capstone Proposal

Mary Troiani
04 Oct 2017

Deep Q Network Reinforcement Learning Proposal

Domain Background

Q-learning is a reinforcement learning technique used to find a policy that gives the optimal action for a given state. This technique works by learning an action-value function that, once properly trained, maps possible actions to the expected value for taking that action, allowing for the selection of the action with the highest expected value. While very simple problems with a small number of actions and states, such as the Train a Smartcab to Drive project, can rely on a table to store the state/action value mappings this quickly becomes unrealistic for problems with a larger number of states and actions.

To adapt to more complex environments Q-learning can be used with function approximation to account for larger numbers of states and actions and even continuous state spaces. As deep neural networks have become more popular research has turned to using deep neural networks to learn the required function. This approach was popularized by DeepMind when they introduced Deep Q Networks in their 2015 paper, *Human-level control through deep reinforcement learning* published in Nature[1]. As this technique is further refined it has promise to dramatically change approaches to robotic control and other temporally-defined domains.

Problem Statement

For this project I propose solving the MountainCar-v0 reinforcement learning problem available on OpenAI gym[2]. MountainCar is a challenge introduced by Andrew Moore in his 1990 PhD thesis and popularized by Barto and Sutton in their classic text *Reinforcement Learning: An Introduction*[3]. The challenge for this problem is to get a car to the top of a large hill from the middle of a valley. The car is underpowered, and unable to climb the hill on it's own, so the only way to get to the top of the hill is to oscillate between the two hills, picking up speed on the left hill to make it over the right hill.

In each episode of MountainCar the car starts at a random position between -0.6 and -0.4 (in the valley) at a velocity of 0. The episode ends when the car has made it to position 0.5 or after 200 timesteps have elapsed. A reward of -1 is given for each timestep prior to termination. MountainCar-v0 is considered solved when an agent can achieve an average reward of -110.0 over 100 consecutive trials.

Datasets and Inputs

In this project there is no input dataset, rather, the data is produced while training by selecting an action and observing the reward for each timestep of the episode. For MountainCar there are three possible actions: push left (0), no push (1), and push right (2). There are also two observables from the state - velocity (ranging from -0.07 and 0.07), and position (ranging from -1.2 and 0.6). In order to train the DQN there must be an input dataset, so this dataset will be created by giving the agent the state variables, allowing the agent to select an action, and then recording the reward and whether or not the episode has terminated. This data will be stored in a buffer and fed into the DQN as a random sample. As new data is created older data points will be removed, allowing the DQN to slowly learn from more recent runs while continuing to learn from older experiences.

Solution Statement

The goal of MountainCar is to get the best possible score over 100 consecutive trials and is considered a successful if that average score is at least -110.0. For this problem using a table to store the state/action data is impossible due to the continuous nature of the environment, so the optimal policy will be determined using a deep neural network to approximate the action-value function. The agent learns by use of the Bellman equation, $Q(s,a) = r + \gamma(\max(Q(s',a')$, by which the Q-value for a given state (s) and action (a) are determined from the current reward (r) plus the maximum future reward for the next state (s', a') discounted by an amount γ. Our DQN will learn to approximate this using a loss function, $loss = \Sigma(Q\text{-target} - Q)^2$ to minimize the difference between the predicted Q-target and Q-value[4].

Benchmark Model

OpenAI gym contains other models which have solved the environment. The benchmark model will be another model that has solved the environment and the number of episodes it required before the solution. Currently the top model on OpenAI gym solved the challenge after 1119 episodes with an average reward of -109.53.

Evaluation Metrics

MountainCar-v0 is considered solved when the agent achieves an average reward of -110.0 over 100 consecutive trials. The agent can further be evaluated by the number of episodes required before achieving this average reward.

Project Design

A basic process for solving MountainCar-v0 using a DQN follows:

1. Initialize the hyperparameters for the DQN and reinforcement learning equation, which will be tweaked to find a model with good performance. These include:
    a. ε, the decaying exploration factor, which allows the agent to gradually transition from randomly chosen actions (exploration) to purposely chosen actions (exploitation)
    b. γ, the future reward discount factor from the Bellman equation
    c. The DQN's batch size, number of layers, and size
    d. The dataset buffer size
    e. The number of episodes to train on
2. Create the DQN, dataset buffer, and set up the MountainCar environment.

3. Train the DQN using an iterative process for each episode:
    a. Choose a random number - if larger than ε choose a random action from the set of actions (push left, no push, push right), else allow the agent to select the action with the highest expected Q-value
    b. Execute the action to transition to the next state. This includes:
        i. Updating the current reward
        ii. Updating the current state
        iii. Updating the buffer to include the latest transition
    c. Randomly sample from the buffer to create a batch for the DQN to train on.
    d. Obtain the Q-target from the DQN and use this to compute the loss and update the network using backpropagation.
    e. Check if the current episode if over (either the car has made it to the goal or the episode has reached 200 timesteps)
4. Test the trained DQN. This requires running 100 episodes of MountainCar in which the trained DQN selects the action given the state of the environment (during testing there is no exploration factor ε). If the average reward during this testing is greater than or equal to -110.0 then the DQN has successfully learned an appropriate action-value function to solve the challenge. Otherwise, try tweaking the hyperparameters an initialize a new training procedure.

References:

1. Human-level control through deep reinforcement learning, Nature: https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf
2. MountainCar-v0, OpenAI gym: https://gym.openai.com/envs/MountainCar-v0/#moore90
3. Reinforcement Learning: An Introduction, Sutton & Barto, MIT Press, 1998
4. Simple Reinforcement Learning with Tensorflow, Arthur Juliani: https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0