Using reinforcement learning for games

Vijaykrishna (170050090)
Pranay Reddy (170070022)
Tushar Gautam (170050026)

Tasks

Tools used:

1. Pytorch
2. Tensorflow
3. Environments from OpenAI

- Experimental analysis of SGD with experience replay on Cartpole
- Experimental analysis of Deep Q Learning on Cartpole
- Using Deep Q learning for Breakout (same ideas but its a more sophisticated game)

Task 1 - SGD with experience replay

Given experience consisting of $\langle state, value \rangle$ pairs

$$\mathcal{D} = \{\langle s_1, v_1^\pi \rangle, \langle s_2, v_2^\pi \rangle, \ldots, \langle s_T, v_T^\pi \rangle\}$$

Repeat:

1. Sample state, value from experience

$$\langle s, v^\pi \rangle \sim \mathcal{D}$$

2. Apply stochastic gradient descent update

$$\Delta \mathbf{w} = \alpha(v^\pi - \hat{v}(s, \mathbf{w}))\nabla_{\mathbf{w}} \hat{v}(s, \mathbf{w})$$

Converges to least squares solution

$$\mathbf{w}^\pi = \underset{\mathbf{w}}{\operatorname{argmin}} \; LS(\mathbf{w})$$

Some explanation for task 1

Credits to David Silver slides for the previous slide

1. In the classic MDP problem one knows the value function for a particular policy and hence the policy iteration step is easy to perform.
2. In this case, however the agent needs to figure out the value function for a given policy by experiencing a few episodes with that policy.
3. With a finite experience, it then estimates the value function at all points using a neural net.

Rewards, states and actions for task 1, 2

Credits: Open AI environment and gym. Tables on the right from their github documentation

Observation

Type: Box(4)

Num	Observation	Min	Max
0	Cart Position	-2.4	2.4
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -41.8°	~ 41.8°
3	Pole Velocity At Tip	-Inf	Inf

Reward

Reward is 1 for every step taken

Actions

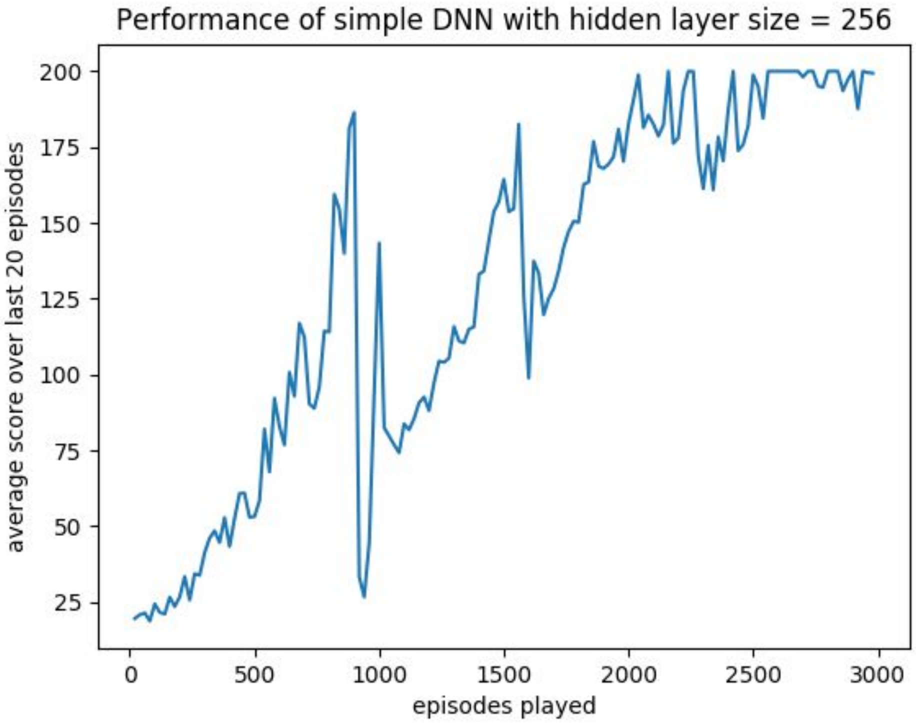
Type: Discrete(2)

Num	Action
0	Push cart to the left
1	Push cart to the right

Architectures for Task 1 & Task 2

- We trained a single layer hidden network with the following constraints:
  - Input: State of the MDP
  - Hidden Layers: A single hidden layer with 256 nodes.
  - Output: Q vector for all actions

Task 1 - Results



Task 2 - Deep Q Learning

- Represent value function by deep Q-network with weights w

$$Q(s, a, w) \approx Q^{\pi}(s, a)$$

- Define objective function by mean-squared error in Q-values

$$\mathcal{L}(w) = \mathbb{E}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{\text{target}} - Q(s, a, w)\right)^2\right]$$

- Leading to the following Q-learning gradient

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right)\frac{\partial Q(s, a, w)}{\partial w}\right]$$

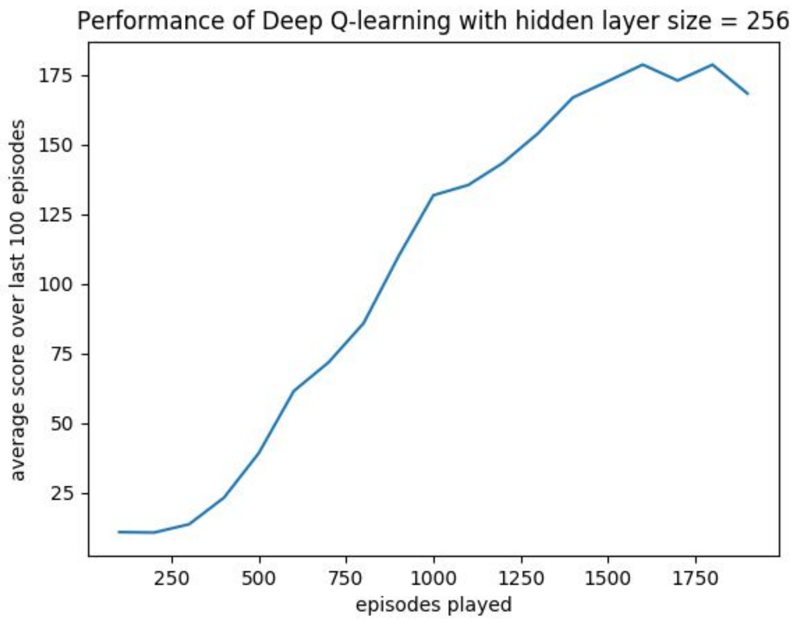
- Optimise objective end-to-end by SGD, using $\frac{\partial L(w)}{\partial w}$

Some explanation for task 2

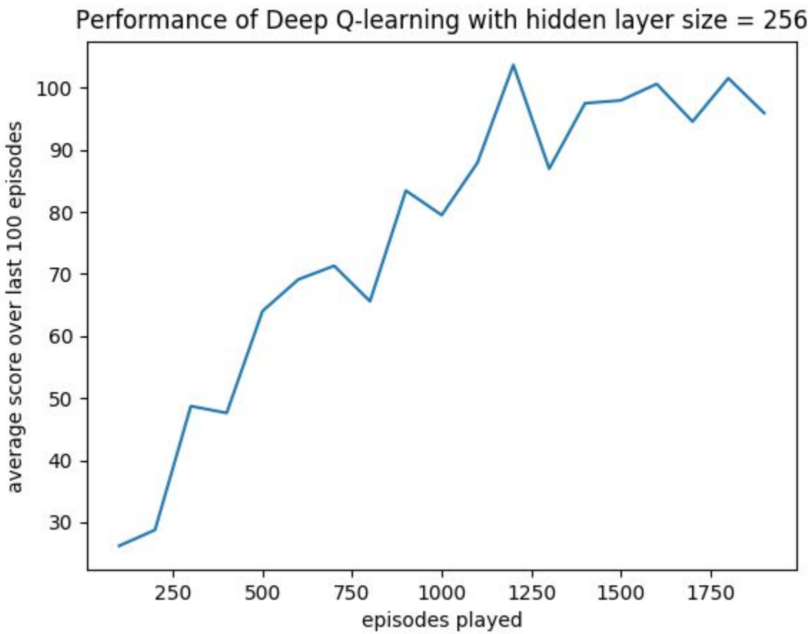
Credits to David Silver slides for the previous slide

1. The intentions are same as task 1 except that the update rules are different.
2. The neural network's target now depends on sum of its own output for the new state and the reward
3. Thus, unlike task 1, the neural network can be trained immediately based on reward and state transition.

Task 2 - Results



Gamma = 0.9



Gamma = 0.8

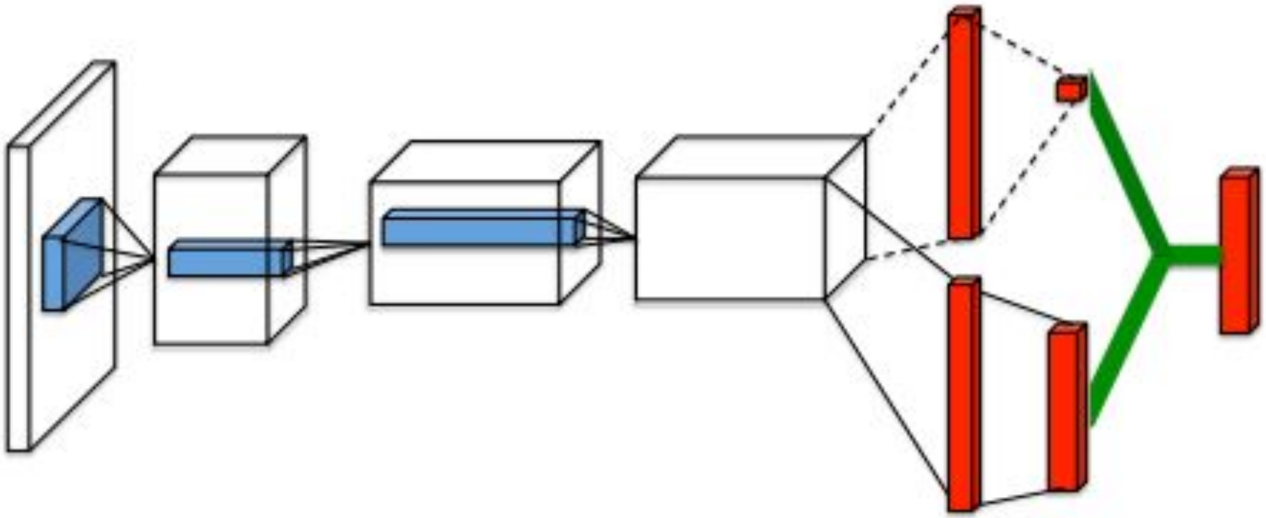
Task 3 - Deep Q Learning for Breakout

- The previous results were obtained on an easier task. In this task we apply the Deep Q Learning method on “Breakout”- an old arcade game (brick-breaker).
- The main difference in this deep q learning module is a better neural network architecture (a Convolutional Neural Network) to leverage local similarity in images.

Task 3 - States, Rewards and Actions

- State- The state is a minimized representation of the RGB window of the game, The window is trimmed and converted into black and white to reduce state dimensionality. Four such frames together make a state.
- Reward- If a brick is successfully broken, a unit reward is awarded to the agent.
- Action- There are four actions: fire, move left, move right and do nothing.

Task 3 - Architecture



Task 3 - Results