Playing Pacman with DQN and Categorical DQN

Ronak Mehta
Manmeet Singh

CMPE 297: Reinforcement Learning

GitHub repository link: https://github.com/manmeet3/pacman-dqn-variants

Introduction

Bellman Equation $$Q(x, a) = \mathbb{E}\, R(x, a) + \gamma\, \mathbb{E}\, Q(X', A')$$

Q-Learning: Try to map the expected long-term value for the state, action pairs from each state.

Deep Q-Learning: Use a neural network to approximate the Q-value function. The state is given as the input and the Q-value of all possible actions is generated as the output.

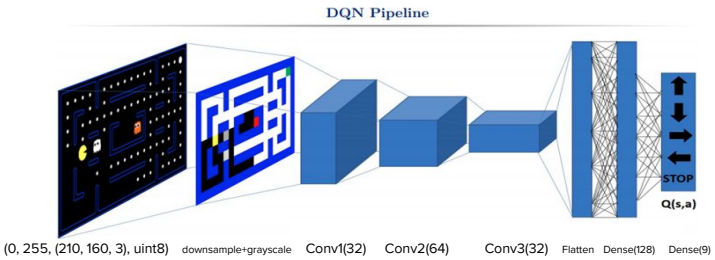
Try to learn the softmax output (value) of each state and action pair

Pacman Environment

- Used MsPacmanNoFrameskip-v4 environment from OpenAI Gym.
- Goal for the agent is to eat all the pellets in the maze while achieving maximum score
- Avoid ghosts
- 4 big pellets in each corner
  - Give extra points
  - Cause ghosts to flee
    - Bonus points for catching fleeing ghosts

Each pellet gives 10 rewards.
Each life costs -100 rewards.
Big Pellets adds 40 rewards.
The game lasts until all lives are lost or pacman eats all pellets.

DQN



Discount factor: 0.99

Learning Rate: 0.001

Epsilon: 1.0

Epsilon Decay: 0.999

Batch Size: 64

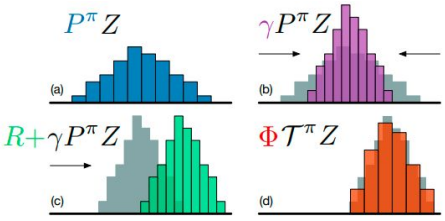
- DQN solves the problem of infinite number of observable states
- Start the game with initial states
- Select exploit or explore
- Single feedforward pass to compute Q-values for all actions from current state
- Make IID samples with mini batches using Replay Buffer which stores state, action, reward, and next state
- Update Q value using Bellman Equation
- Train the network with more episodes

Source: http://cs229.stanford.edu/proj2017/final-posters/5144893.pdf

The gym observation environment is downsampled before convolution operations. The output of convolution operations is flattened into two dense layers which estimate the Q-value associated with each action. We take argmax of these Q-values.

Categorical DQN Extension

$$Z(x,a) \overset{D}{=} R(x,a) + \gamma Z(X', A')$$



Hyper parameters used in our run

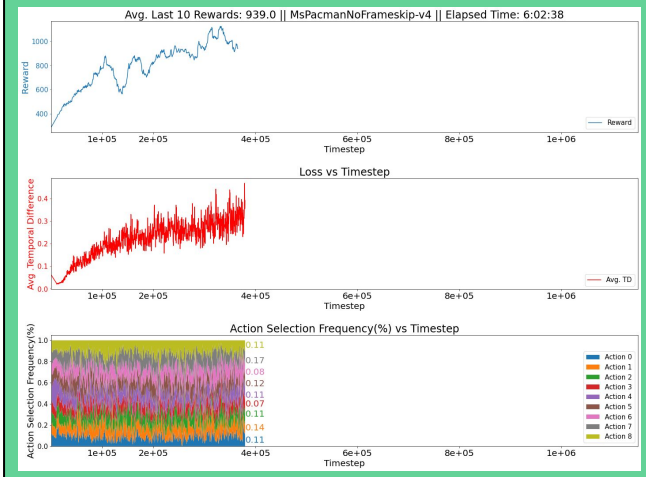
Epsilon: 1.0
Decay: 0.19 % with each action
Gamma: 0.99
Replay Buffer Size: 50k (s, a) pairs
Categorical distribution intervals: 51

- Distribution of Z is characterized by the interaction of three random variables: R, (X', A') and its random return Z(X', A')
- Basically instead of always picking the previous best action, you sample nearby
- How the distribution is created:
  1. State distribution created under derived policy pi
  2. Discounting shrinks the distribution
  3. Reward shifts the distribution towards appropriate action state
  4. Projection step and action chosen based on categorical algorithm

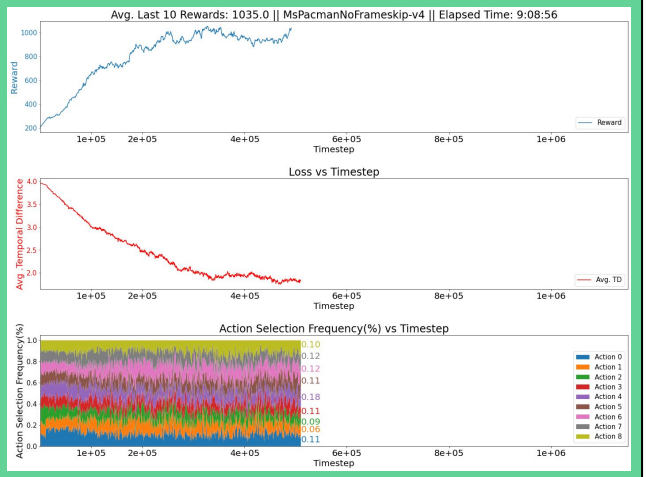
Source: A Distributional Perspective on Reinforcement Learning, Bellemare, et al.

Results

DQN



C51



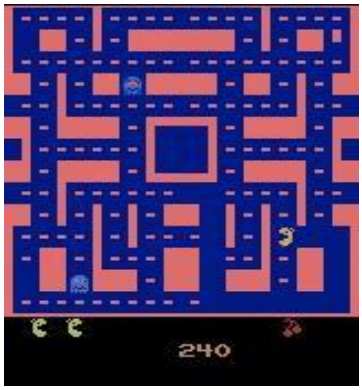
C51 training executes smoother as an outcome of not always just picking the best previous actions

Todo: Update with same number of time steps
Talking points: Talk about Temporal Difference comparison (https://lilianweng.github.io/lil-log/assets/images/TD_MC_DP_backups.png)

Best Episodes

DQN

C51



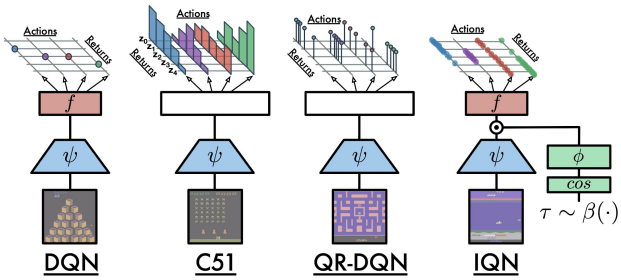
Avg 100 episode performance

A3C	5738.30 ± 171.99 (from OpenAI Leaderboards)
DQN	~940
C51	~1020

*Our estimates are rough based on charts presented on results slide

Todo: add the videos. Figure out how to re-load the saved model weights and play the environment

Alternative Considerations



Source: github/marload/DistRL-TensorFlow2

- Additional hyper parameter tuning
- Add a Noisy Network
  - Train exploitation vs. exploration parameter as part of the NN
- Policy based methods
  - A2C

IQR Additional Information:
https://datascience.stackexchange.com/questions/40874/how-does-implicit-quantile-regression-network-iqn-differ-from-qr-dqn

Conclusion

Key Takeaways:

- DQN may initially perform better by choosing previous best actions, but over time C51 outperforms it.

Challenges:

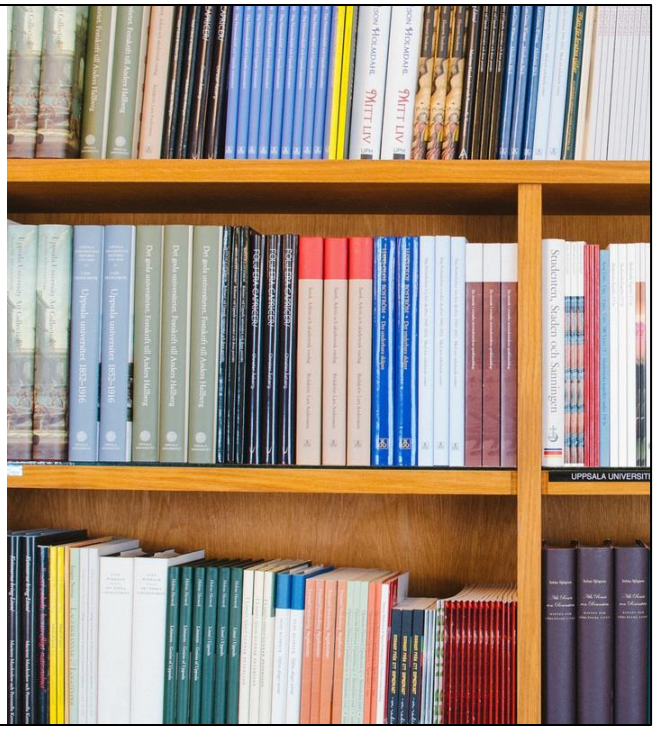
- GPU intensive
- Long training time due to long episodes

Future Improvements:

- Run for more iterations (side by side)
- Try different gamma and epsilon values

References

- https://github.com/marload/DistRL-TensorFlow2
- http://cs229.stanford.edu/proj2017/final-posters/5144893.pdf
- Volodymyr Mnih et al. Playing atari with deep reinforcement learning.
- Volodymyr Mnih et al. Human-level control through deep reinforcement learning
- Bellemare et al. A Distributional Perspective on Reinforcement Learning



BACK UP

We also considered using RAM version of Pacman

- In the MsPacman-ram-v0 environment, the observation is the RAM of the Atari machine, consisting of 128 bytes
- The goal is to find a simple way to ‘represent’ state of the game, then pass that to the fully connected layers (Reinforcement Learning algorithms)
- If the model is trained from the pixels, convolutional neural network of several layers is required. Interestingly, the final output of the convnet would be 1D array of features. The output is then passed to a fully connected layer and that outputs the correct 'action' based on the features the convnet recognized in those image(es)
- So, what OpenAI has done by giving the RAM version is to avoid the task of learning a 'representation' of the game, and learn a 'policy' or what to do based on the state of the game. Hence, instead of convent, fully connected layers could be used to learn a policy and reduce the computation time

Source:
https://stackoverflow.com/questions/45207569/how-to-interpret-the-observations-of-ram-environments-in-openai-gym

Categorical Algorithm

1. Takes a transition S_t, A_t, R_t, S_{t+1}, γ_t;
2. Computes Q value by summing weighted probability for each “bin” in a distribution
3. Choose optimal action (a*) from value function, given S_t;
4. accumulate the probabilities of the aligned atoms of the target distribution $r_{t+\gamma Z(x_{t+1}, a*)}$
5. Compute projection of Tz_j onto the support $\{z_j\}$ **next slide.**
6. Clip non-aligned atoms between Vmin Vmax

Algorithm 1 Categorical Algorithm

input A transition $x_t, a_t, r_t, x_{t+1}, \gamma_t \in [0, 1]$
  $Q(x_{t+1}, a) := \sum_i z_i p_i(x_{t+1}, a)$
  $a^* \leftarrow \arg\max_a Q(x_{t+1}, a)$
  $m_i = 0, \quad i \in 0, \ldots, N-1$
  for $j \in 0, \ldots, N-1$ **do**
    # Compute the projection of $\hat{\mathcal{T}} z_j$ onto the support $\{z_i\}$
    $\hat{\mathcal{T}} z_j \leftarrow [r_t + \gamma_t z_j]_{V_{\text{MIN}}}^{V_{\text{MAX}}}$
    $b_j \leftarrow (\hat{\mathcal{T}} z_j - V_{\text{MIN}})/\Delta z$ # $b_j \in [0, N-1]$
    $l \leftarrow \lfloor b_j \rfloor, u \leftarrow \lceil b_j \rceil$
    # Distribute probability of $\hat{\mathcal{T}} z_j$
    $m_l \leftarrow m_l + p_j(x_{t+1}, a^*)(u - b_j)$
    $m_u \leftarrow m_u + p_j(x_{t+1}, a^*)(b_j - l)$
  end for
output $-\sum_i m_i \log p_i(x_t, a_t)$ # Cross-entropy loss

Source: https://www.youtube.com/watch?v=lWYb7RSxtI0