UF Herbert Wertheim College of Engineering
UNIVERSITY *of* FLORIDA

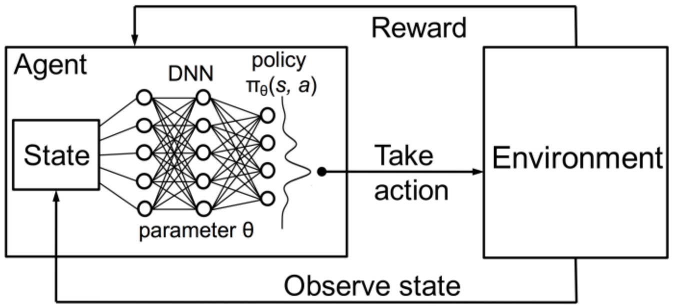
POWERING THE NEW ENGINEER TO TRANSFORM THE FUTURE

Deep Reinforcement Learning with Double Q-learning

Google Deepmind – Dec 2015 – Hado van Hasselt, Arthur Guez, David Silver

Presented by: T1 - James Bocinsky and Lorenzo Tabares

Reinforcement Learning



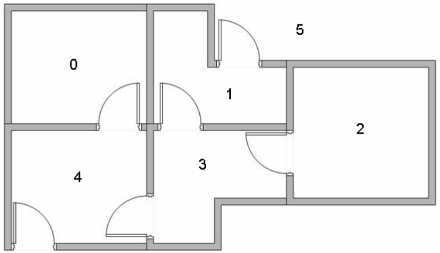
- Agent – game player
- State – current game environment
- Actions – action performed by agent at a given state
- Reward – defined by some aspect of the game to encourage
- Reinforcement learning uses its actions and the corresponding reward it received to properly learn a policy (state/action relationship)

Background (Q Learning)

- Used to solve sequential decision problems
  - Learn estimates for the optimal value of each action
- The Optimal Policy
  - Maximize reward in current state and for future states
- Optimal Action Values
  - Estimated with Q-Learning through observations

Intuition Example

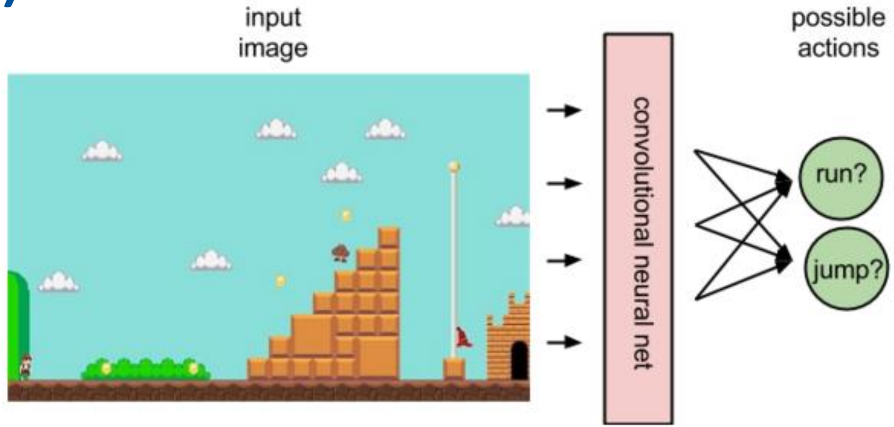
Goal: exit building to state 5



Action

$$
\begin{array}{c c}
\text{State} & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
R= \begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} &
\begin{bmatrix}
-1 & -1 & -1 & -1 & 0 & -1 \\
-1 & -1 & -1 & 0 & -1 & 100 \\
-1 & -1 & -1 & 0 & -1 & -1 \\
-1 & 0 & 0 & -1 & 0 & -1 \\
0 & -1 & -1 & 0 & -1 & 100 \\
-1 & 0 & -1 & -1 & 0 & 100
\end{bmatrix}
\end{array}
$$

Deep Q-Learning (DQN)



- Q Learning
  - Table consists of:
    - Row vectors as states
    - Column vectors as actions
  - Updated using Bellman equation:

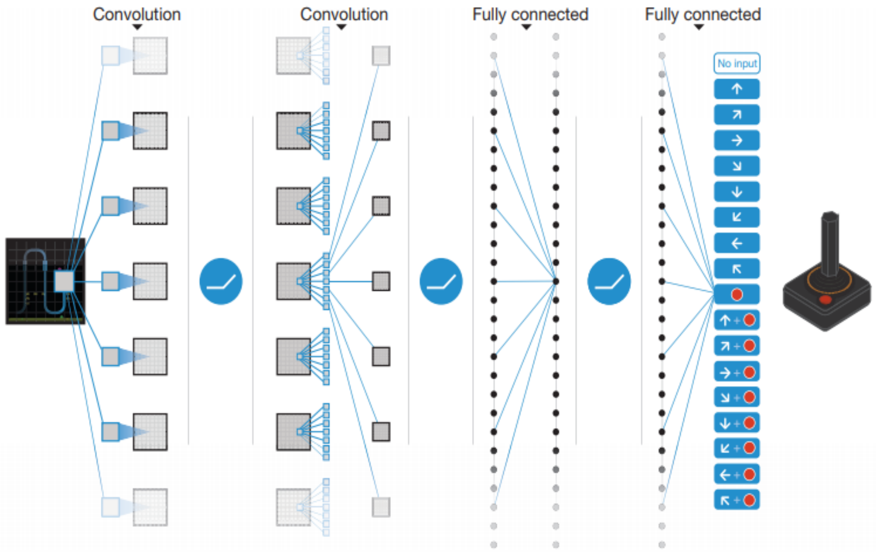
$$Q(s,a) = r + \gamma(\max(Q(s',a'))$$

  - Maximizes immediate reward and future discounted rewards.

- Convolutional Neural Network implementation
  - Determines the optimal policy through approximation.
  - Minimizes the loss function to approximate the optimal value function

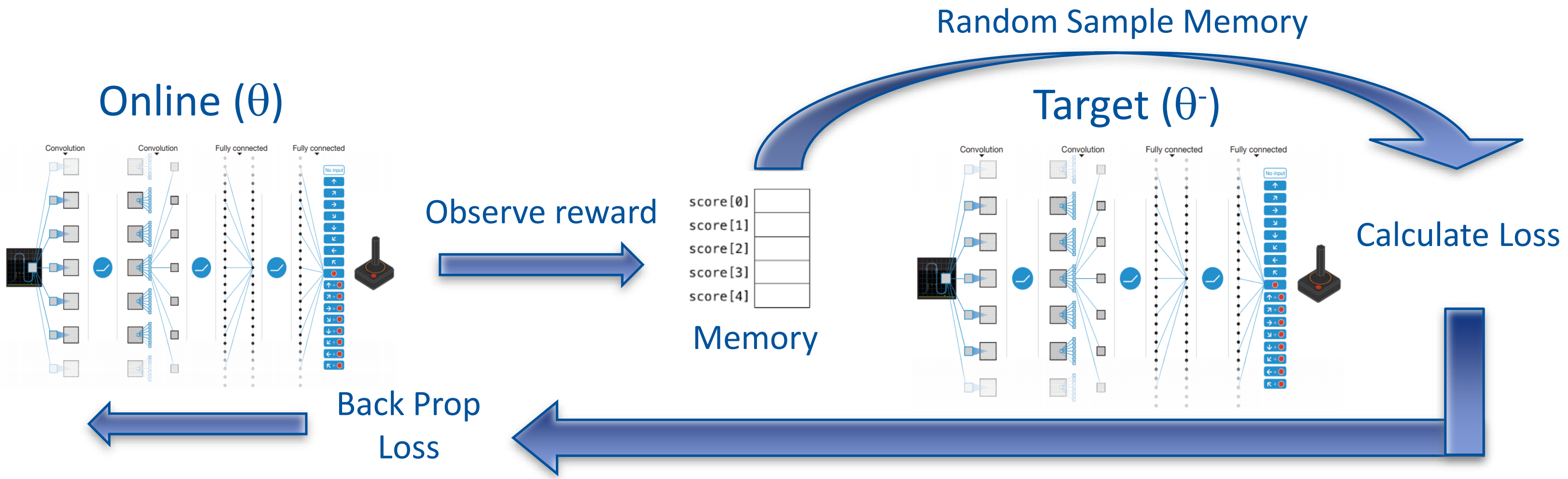
Paper's Architecture

- DQN – 3 convolution layers followed by 1 fully connected
- ~ 1.5M parameters
- Input – Last 4 frames of Atari game
  - Provides DQN with a sense of movement in the game
- Output – game controls



High Level Algorithm

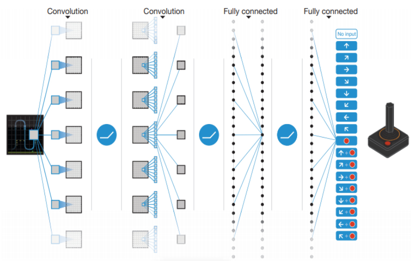
- Two networks of the same architecture are used
- "Online" network – Used to form memory of previous actions and rewards given
- "Target" network – Uses memory to calculate loss

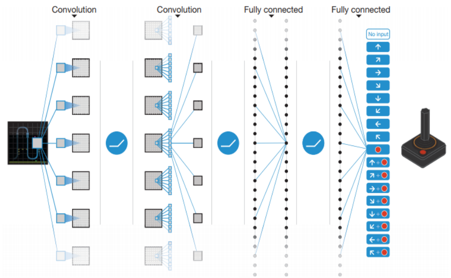


Updating weights

- Memory is formed
- Loss calculated from target network parameters
- Weights copied every tau steps
- Target network is essentially the Online network at a snapshot in time

Online (θ)



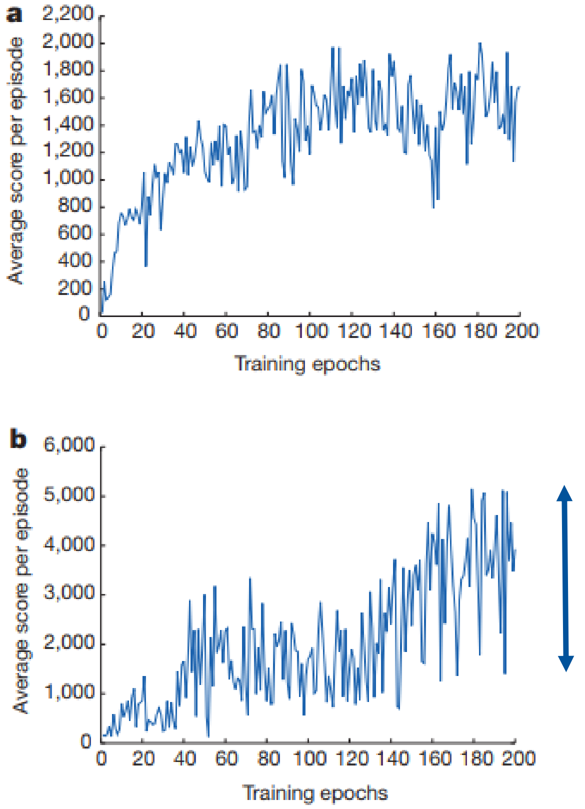


Target (θ^-)

Copy Weights

Double DQN Motivation

- Policy – the state action relationships
- DQN – is unstable due to policy reward overestimation
- Overestimation – when the policy's reward estimation is overly large
- This does not typically affect the ability to learn a policy as long as values are uniformly higher than relative action preferences, but this is not always the case



Formal Equations

- Q value function

$$Q_\pi(s,a) \equiv \mathbb{E}\left[R_1 + \gamma R_2 + \ldots \mid S_0 = s, A_0 = a, \pi\right]$$

- Parameter update equation (should remind you of typical SGD)

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(Y_t^{\mathrm{Q}} - Q(S_t, A_t; \boldsymbol{\theta}_t))\nabla_{\boldsymbol{\theta}_t} Q(S_t, A_t; \boldsymbol{\theta}_t)$$

- Target Output

$$Y_t^{\mathrm{Q}} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \boldsymbol{\theta}_t)$$

Proposed Double DQN

- Original Target Evaluation

$$Y_t^{Q} = R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}_t)$$

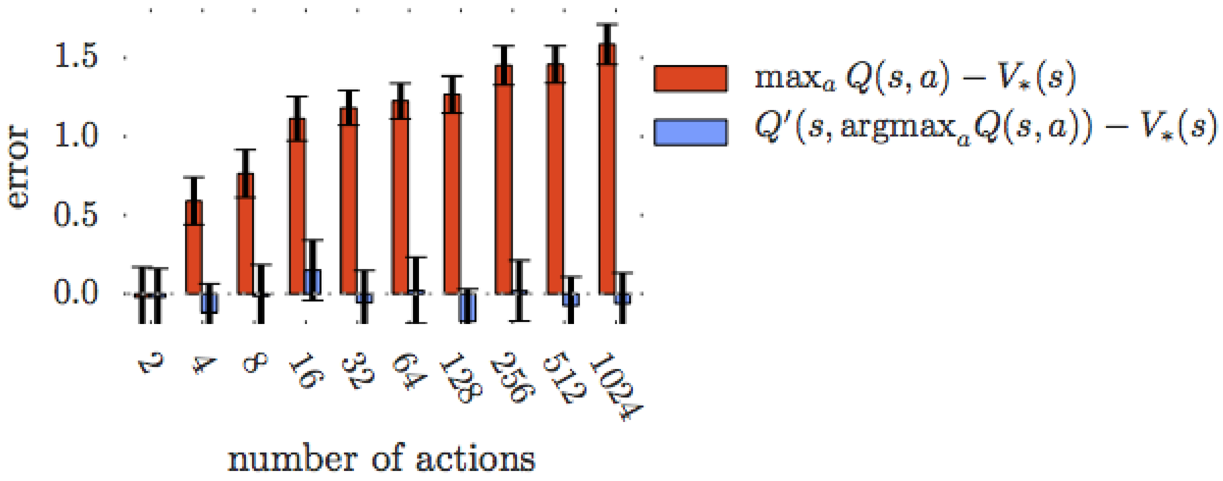
- Double Deep Q-Learning Target Evaluation

$$Y_t^{\text{DoubleQ}} \equiv R_{t+1} + \gamma Q(S_{t+1}, \underset{a}{\operatorname{argmax}}\, Q(S_{t+1}, a; \boldsymbol{\theta}_t); \boldsymbol{\theta}'_t)$$

  - Action is still calculated through online network parameters (θ)
  - However evaluation is done through target network parameters (θ^-)

Overoptimism Dependent on Action Space

- Error increases with number of actions for DQN, but not for Double DQN



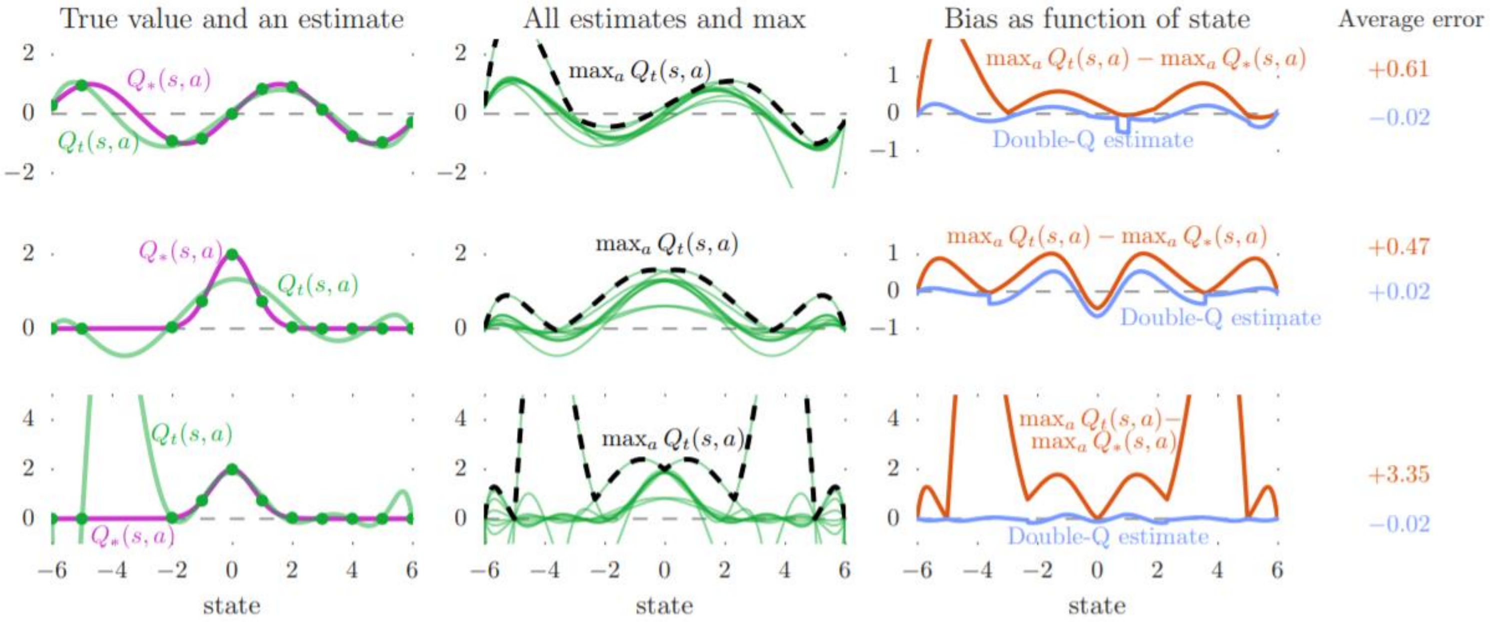
Learning Overestimations

True distribution:

sin(s)

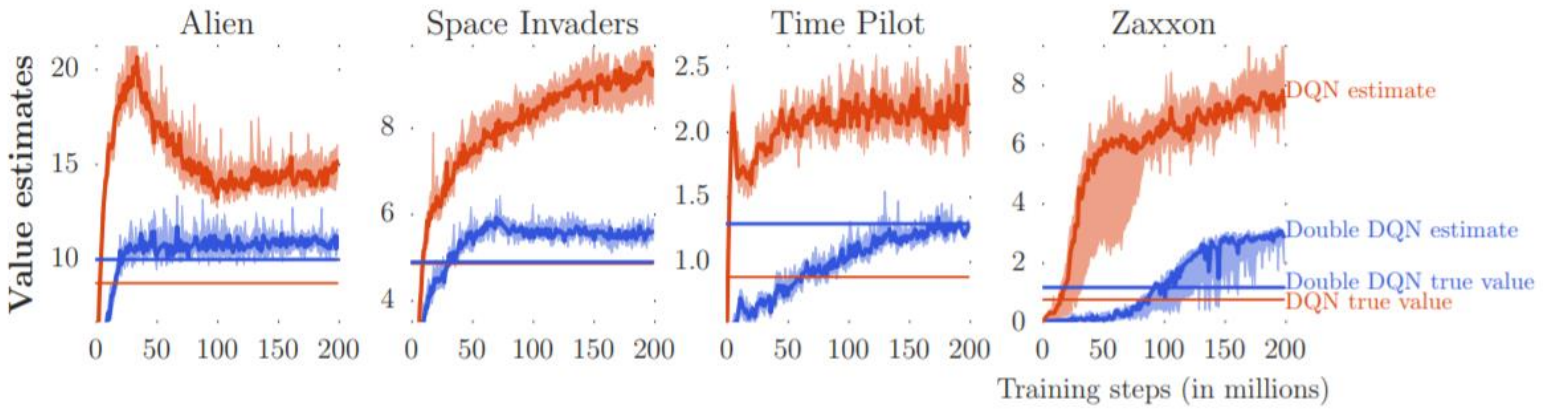
$2\exp(-s^2)$

$2\exp(-s^2)$



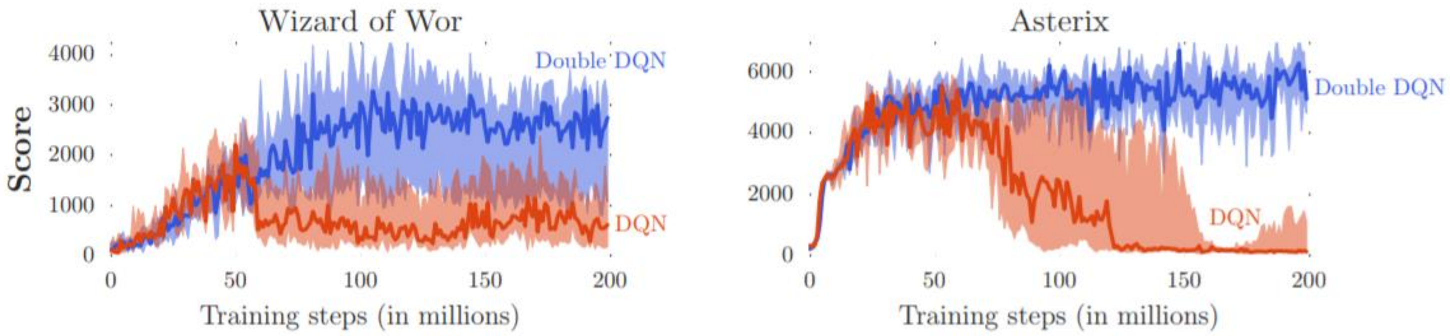
Overestimation Comparison

- Horizontal line represents actual reward returned from each visited state.
- If there was no bias, you should see the estimate meet the horizontal line at the end of training



Overestimation Comparison

- Once overestimation begins you see the score of DQN drop severely.
- Double DQN does not overestimate and is seen to be much more stable over training.



Performance Comparison

- If training is done for a short amount of time, the difference between DQN vs Double DQN is not that much.
- When you do more training you can see that DQN overestimates and decreases in performance, showing that Double DQN is more stable.

5 mins of play

	DQN	Double DQN
Median	93.5%	114.7%
Mean	241.1%	330.3%

30 mins of play

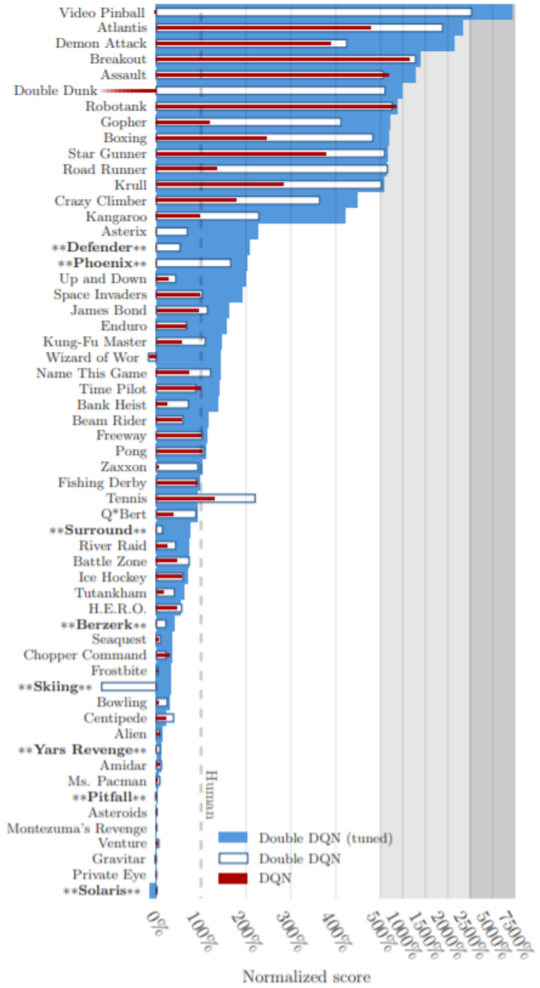
	DQN	Double DQN	Double DQN (tuned)
Median	47.5%	88.4%	116.7%
Mean	122.0%	273.1%	475.2%

Performance Comparison

- Scores normalized relative to human score
- 100% score is at human level

$$\text{score}_{\text{normalized}} = \frac{\text{score}_{\text{agent}} - \text{score}_{\text{random}}}{\text{score}_{\text{human}} - \text{score}_{\text{random}}}$$

- Tuned Double DQN performs better than a human for more than half of the games tested
- Exceeds regular DQN



Positives

- Extensive use of Graphs
  - Helps visualized results
  - Concrete evidence
- Proofs and derivations were provided with all claims and formulas
- The authors provided good intuition for why double DQN would have such strong results with visualizations and equations

Negatives

- No discussion on
  - Potential problems using Double DQN
  - Future improvements on the design
  - Further application
  - Why double DQN affected certain games more drastically than others
  - Details of architecture

Impact

- DQN
  - Proven to be overoptimistic and reduce performance
- Double DQN
  - Mitigates overestimation
  - Achieved record performance on Atari 2600 games
  - Can be used in future DQN projects to reduce training instability and uncertainty
  - Provides more confidence that the solution it formed is reasonable due to stability

Video Demo

Short video demo: https://youtu.be/V1eYniJ0Rnk

Questions?