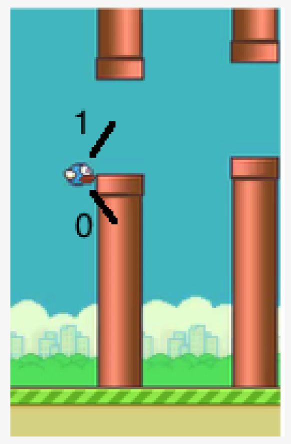
Game Playing with Deep Reinforcement Learning using OpenAi Gym

Introduction

- Historically, designing game players requires domain-specific knowledge of the particular game to be integrated into the model for the game playing program.
- We explore the use of reinforcement learning and neural networks, in order to architect a model which can be trained on more than one game.
  - Typical Q-Learning learns a score for every single possible state-action pair. With images as inputs, this is unfeasible.
  - While training, a network can easily get stuck in a local optima with the wrong hyper-parameters.
- Our goal is to improve on existing networks, such as DeepMind's model designed to learn multiple Atari games [1].
- Creating models which can generalize across multiple environments explores the fundamental goal of Artificial Intelligence.

Problem Statement

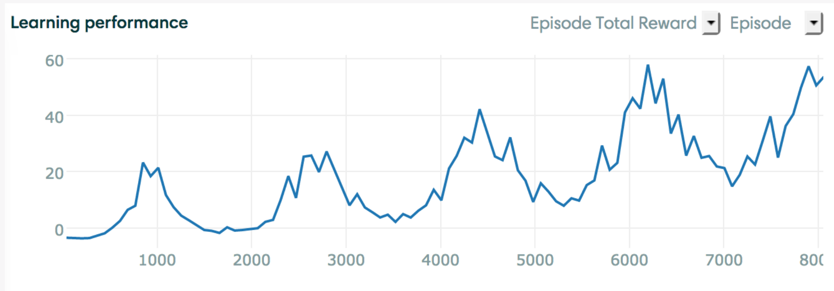
- We begin by training a model to play Flappy Bird.
- The environment will come from the OpenAi Gym interface [2].
- At each time step, we will receive a matrix representing the pixel values of the current frame, as well a reward. We must then return an action to be performed by our agent in the environment.



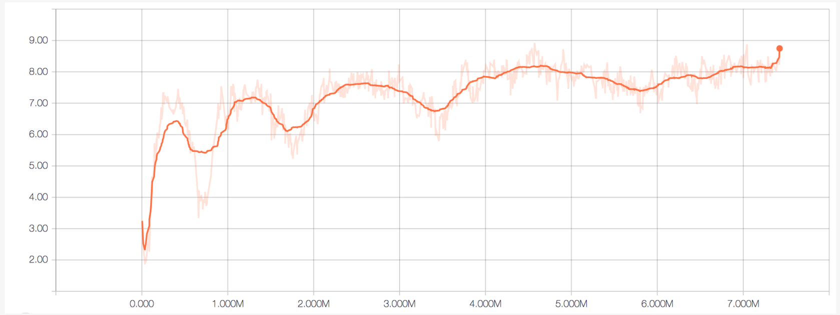
- We evaluate the ability of the model by best average score over 100 episodes as reported by the OpenAi Gym interface.
- We then retrain the same model on a different game, Pixel Copter, to demonstrate the ability to generalize.

Experiments and Results

Deep Q-Learning Flappy Bird

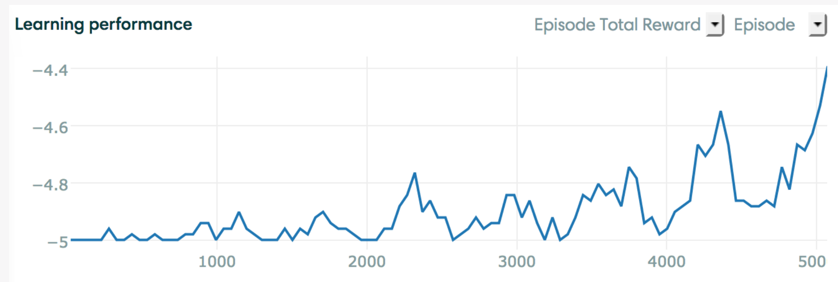


OpenAi Gym submission of score over about 7 million iterations

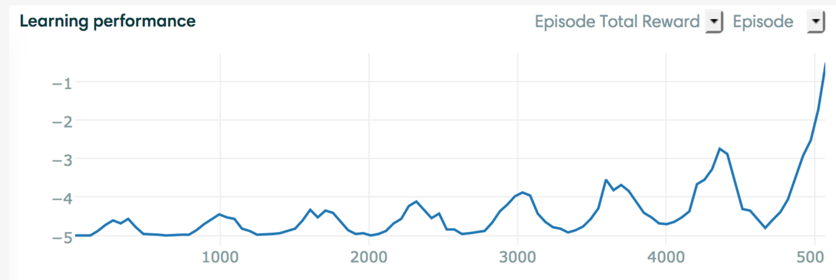


DQN predicted Q-value over the course of training

Double Deep Q-Learning

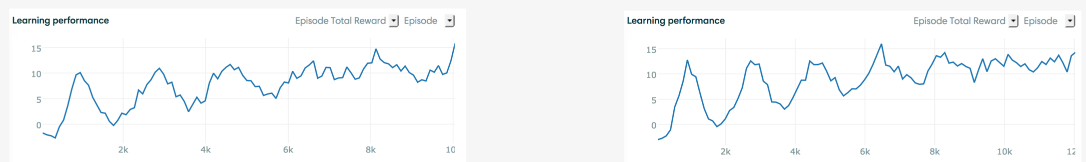


OpenAi Gym submission of first epoch using standard DQN



OpenAi Gym submission of first epoch using Double DQN.

Pixel Copter Results



Preliminary OpenAi Gym score results for Pixel Copter using DQN (left) and DDQN (right)

Conclusions and Future Work

- We were able to receive superhuman results on Flappy Bird using Deep Reinforcement Learning.
  - We reduced training time by utilizing our new sinusoidal epsilon function
  - We can further reduce training time and improve performance by using Double Deep Q-Learning
  - First epoch of Double Deep Q-Learning for Flappy Bird shows a much smoother score curve over episodes, confirming the increased stability hypothesis.
- Our model was able to transfer over to another game, Pixel Copter
  - Preliminary results indicate that we may need to tweak the original model to achieve superhuman results
  - Double Deep Q-Learning does not seem to offer as significant of an improvement compared to Flappy Bird. This may be due to the simplicity of the graphics.
- We can replace the first fully connected layer with an LSTM, which has been shown to add robustness to the model [4].

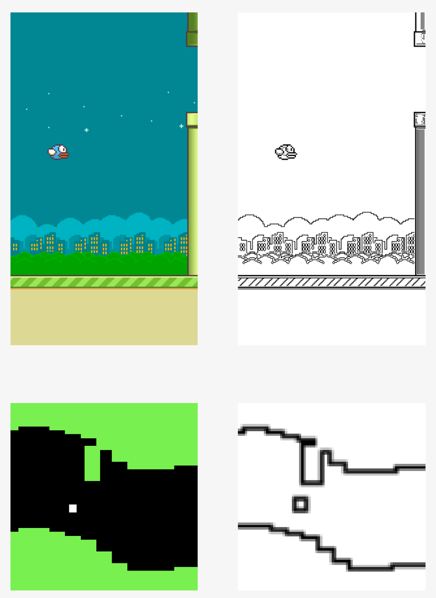
Acknowledgements

1. V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. ntonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013
2. Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, Wojciech Zaremba .(2016). OpenAI Gym. arXiv:1606.1540
3. Van Hasselt, H., Guez, A., and Silver, D. (2015). Deep reinforcement learning with double Q- learning. CoRR, Abs1509.06461.
4. HAUSKNECHT, M. J., AND STONE, P. Deep recurrent q-learning for partially observable mdps. CoRR abs1507.06527 (2015)

Methods and Algorithms

Input Preprocessing

- Convert to greyscale
- Denoise the image using adaptive thresholding
- Normalize values to be between 0 and 1
- Reduce input image to 80x80 pixels
- Stack last 4 frames as input to network



Deep Q-Learning

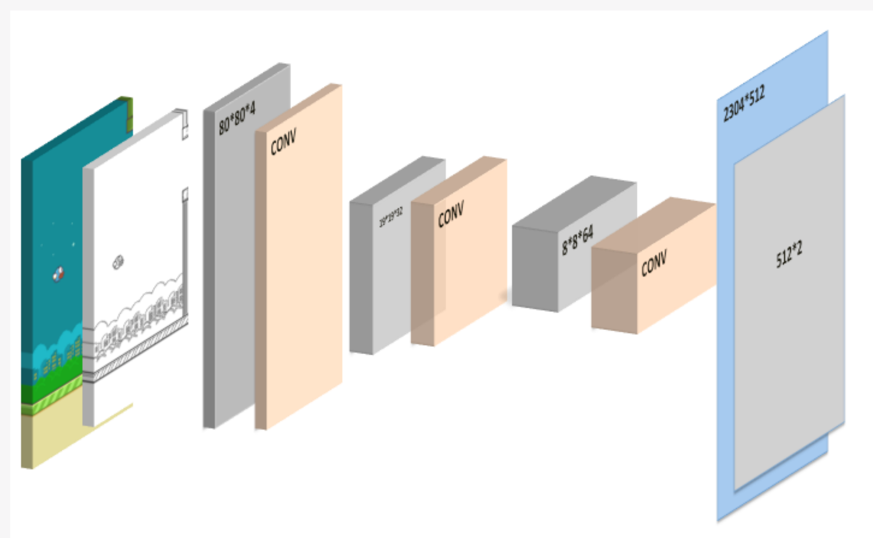
- We use the same update equation from Q-Learning based off the Bellman Equation
  - $Q(s_t, a_t) = Q(s_t, a_t) + \alpha[r_t + \gamma max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$
  - Instead of explicitly computing Q for all state action pairs, we approximate it using a neural network
- Q values are continuous which can be modeled as a regression task and can be optimized with simple squared loss as follows
  - $Loss = 0.5 * [r + max_{a'} Q(s', a') - Q(s, a)]^2$

Double Deep Q-Learning [3]

- In Deep Q-Learning, agent tends to overestimate the Q value
- Double DQN targets to reduce this maximization bias and increase model stability with reduced variance
- Learns two Q functions independently Q_1 and Q_2, both based off original network model.
  - $Q_1(s, a) = r + \gamma Q_2(s', argmax_{a'} Q_1(s', a'))$
  - $Q_2(s, a) = r + \gamma Q_1(s', argmax_{a'} Q_2(s', a'))$

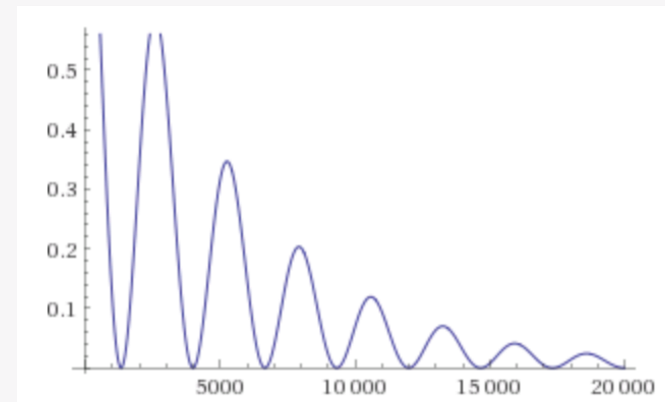
Models

Network Architecture				
Type	Classes / Filters	Filter Size	Stride	Activation
Conv-1	32	8x8	4	ReLU
Conv-2	64	4x4	2	ReLU
Conv-3	64	3x3	1	ReLU
Fully Connected-1	512			ReLU
Fully Connected-2	# of actions			Linear

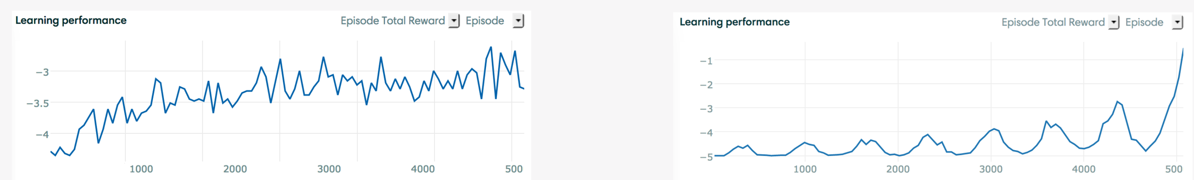


Mini Epochs using ϵ

- We have introduced a new ϵ decaying function: one which exponentially decays over time in a sinusoidal fashion.
- $\epsilon = \epsilon_0 \cdot \epsilon_d^x \cdot \frac{1}{2}(1 + cos(\frac{2\pi xn}{X}))$
  - ϵ_0 is initial epsilon
  - ϵ_d is decay rate
  - n is number of mini epochs
  - X is number of training episodes
  - x is current training episode number



Save the model at every minimum, creating a mini-epoch



Flappy Bird scores when training a first epoch of 5000 episodes. Standard exponentially decaying ϵ function (left) vs our sinusoidal decaying function (right)

Robert Chuchro chuchro3@stanford.edu

Deepak Gupta dgupta9@stanford.edu

Stanford