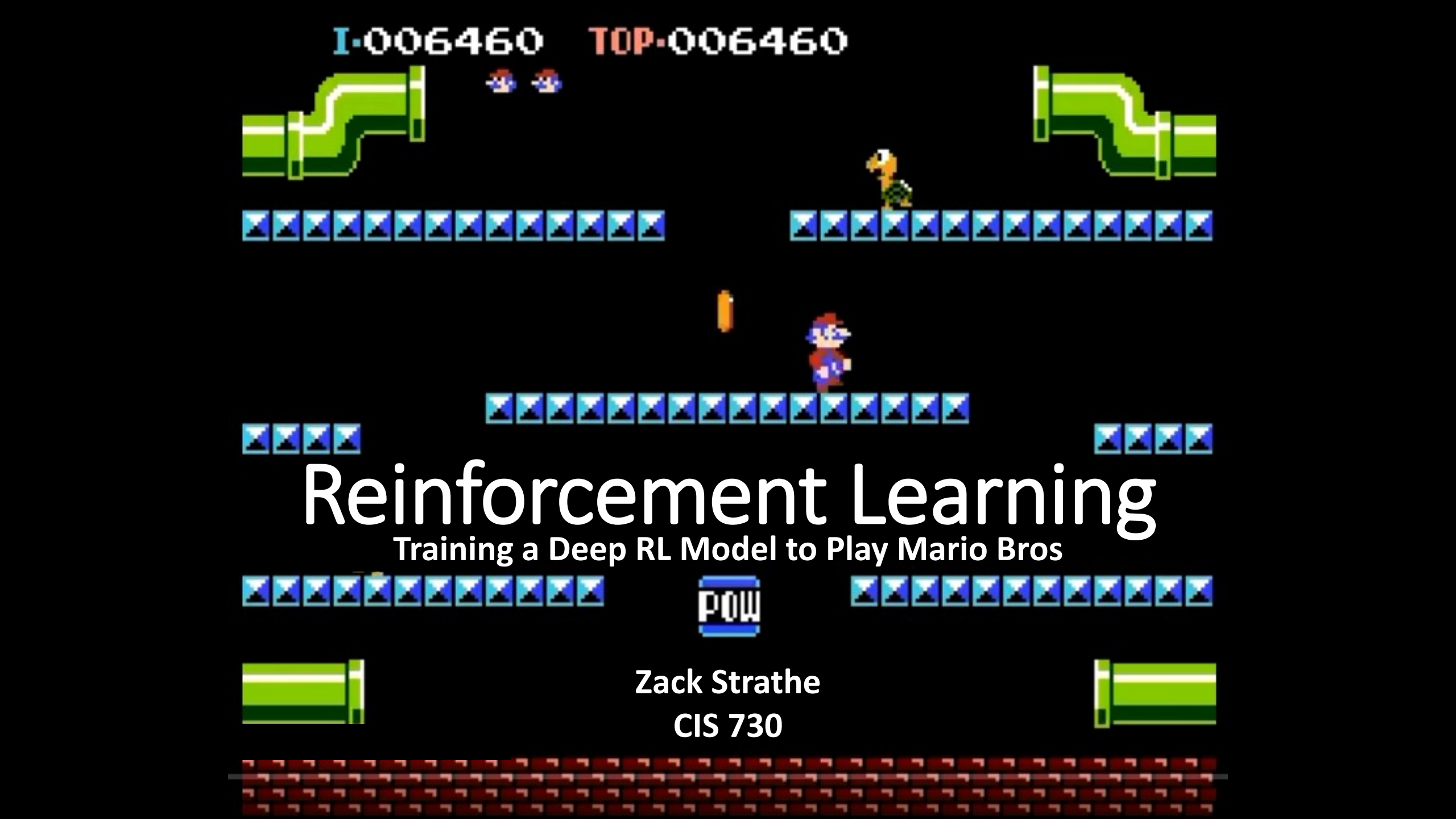


Reinforcement Learning

Training a Deep RL Model to Play Mario Bros

Zack Strathe
CIS 730

Introduction

- With reinforcement learning (RL), models can be trained to learn tasks using only a reward function
  - Such as real-world examples of
    - self-driving cars learning to drive safer
    - software-agents learning to maximize efficiency of an electrical grid
- To explore RL, I implemented a simplified example to train a game-playing agent to play the NES game Mario Bros
  - Goal: to complete the first level

Gameplay Information

- Killing enemies is a two-step process
  - Hit from below to stun enemies, then run into them to finish
  - Timed delay before they return to normal state (~9 seconds)
  - Increases difficulty of RL model to find successful state-action pairs
- Reward farming is possible
  - The game rewards 10 points each time an enemy is stunned by hitting it from below
  - 10 points rewarded each time means model may become “stuck” repeating this behavior

Methodology

- Platform for Model
  - Open AI Gym Retro
    - Python library that provides a simple interface for building reinforcement learning models with retro games
    - Utilizes mappings of game variables in memory to utilize for a reward function
    - Mario Bros already integrated, with a reward function based on the game score
- Cloud Computing
  - Due to processing requirements of RL, I found it necessary to utilize cloud computing
    - Google Cloud (with a GPU for faster training)

Reinforcement Learning Model

- Proximal Policy Optimization
  - Model-free reinforcement learning algorithm developed by OpenAI
  - A policy gradient method, uses a clipping factor to ensure that policy does not change too much with each update
  - Ideal for ease of implementation and tuning of parameters
  - Utilizing the ‘PPO2’ algorithm from OpenAI’s Baselines module and a Convolutional Neural Network (CNN)
    - With a CNN, this is a Deep RL model

Experiment Design

- Based on a human player, if model achieves a score of 2,430 points during training, should be able to complete first level
- Models that reached this score were saved and recorded to verify whether the goal was met

Source: video of human player

Model improvement methods (1/4)

- **Modifying the reward function**
  - Base on number of enemies remaining instead of default (game score)
  - Modified data.json and scenario.json files within Gym subdirectory for game

data.json

```
{
  "info": {
    "level": {
      "address": 65,
      "type": "|d1"
    },
    "lives": {
      "address": 72,
      "type": "|u1"
    },
    "score": {
      "address": 148,
      "type": ">d4"
    },
    "enemies": {
      "address": 68,
      "type": "|d1"
    }
  }
}
```

Default

scenario.json

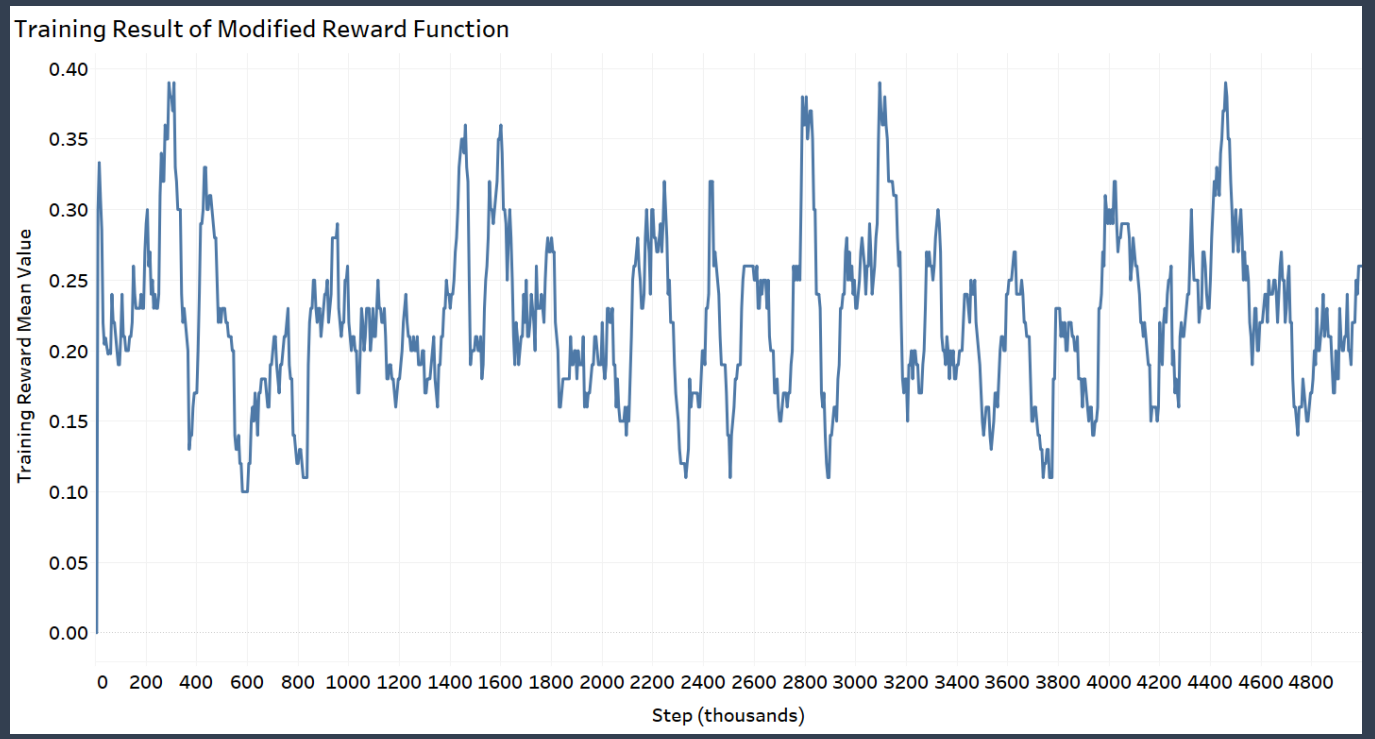
```
{
  "crop": [
    0,
    25,
    0,
    184
  ],
  "done": {
    "variables": {
      "lives": {
        "op": "equal",
        "reference": 0
      }
    }
  },
  "reward": {
    "variables": {
      "score": {
        "reward": 1
      }
    }
  }
}
```

Modified

scenario2.json

```
{
  "crop": [
    0,
    25,
    0,
    184
  ],
  "done": {
    "variables": {
      "lives": {
        "op": "equal",
        "reference": 0
      }
    }
  },
  "reward": {
    "variables": {
      "enemies": {
        "op": "negative",
        "reward": 1
      }
    }
  }
}
```

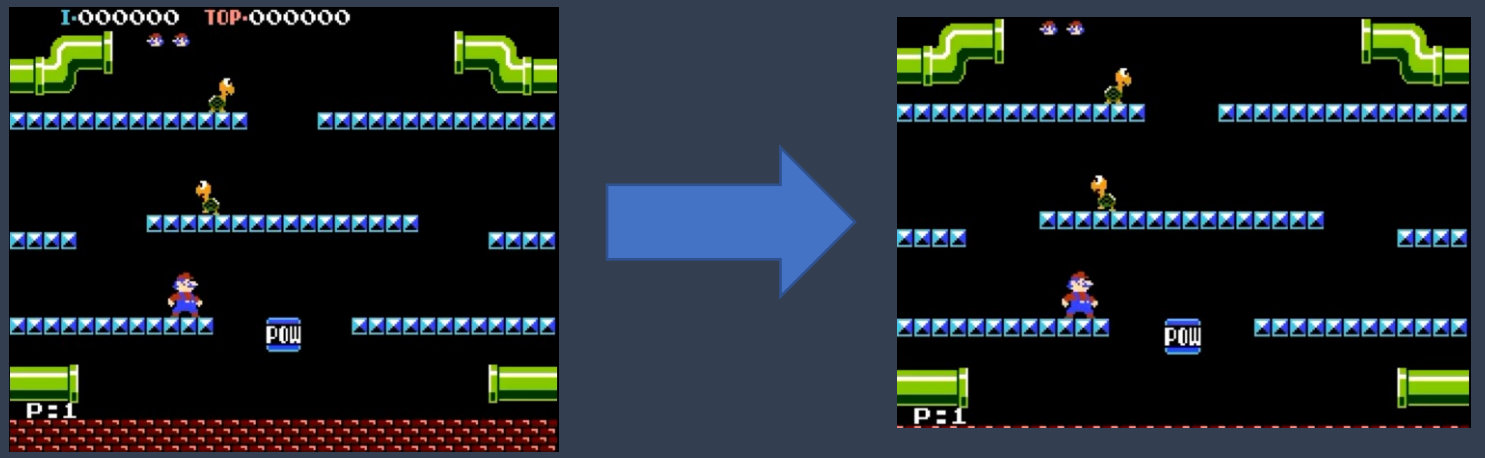
Results: modifying the reward function



- Plot of mean reward from model trained to 5 million steps shows that new reward function is ineffective
- Likely fails because rewards are too sparse:
  - Does not allow the model to generalize state and action pairs, since most actions are accompanied by zero change to reward

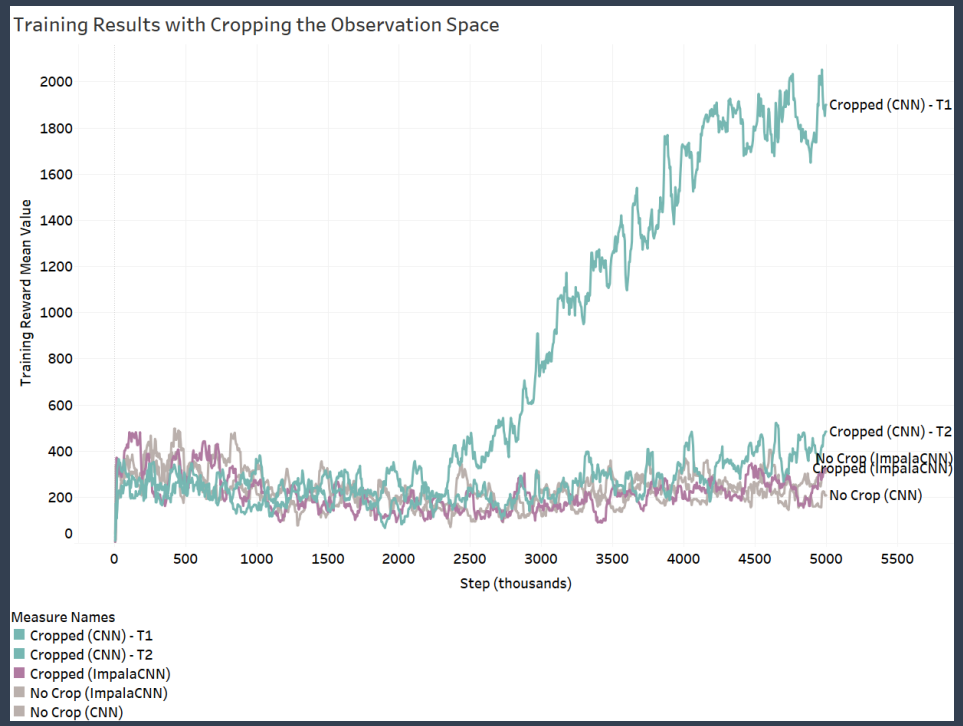
Model improvement methods (2/4)

- Modifying the Observation Space
  - Test cropping out unnecessary parts of the screen image
    - Eliminate “noise” such as the score display



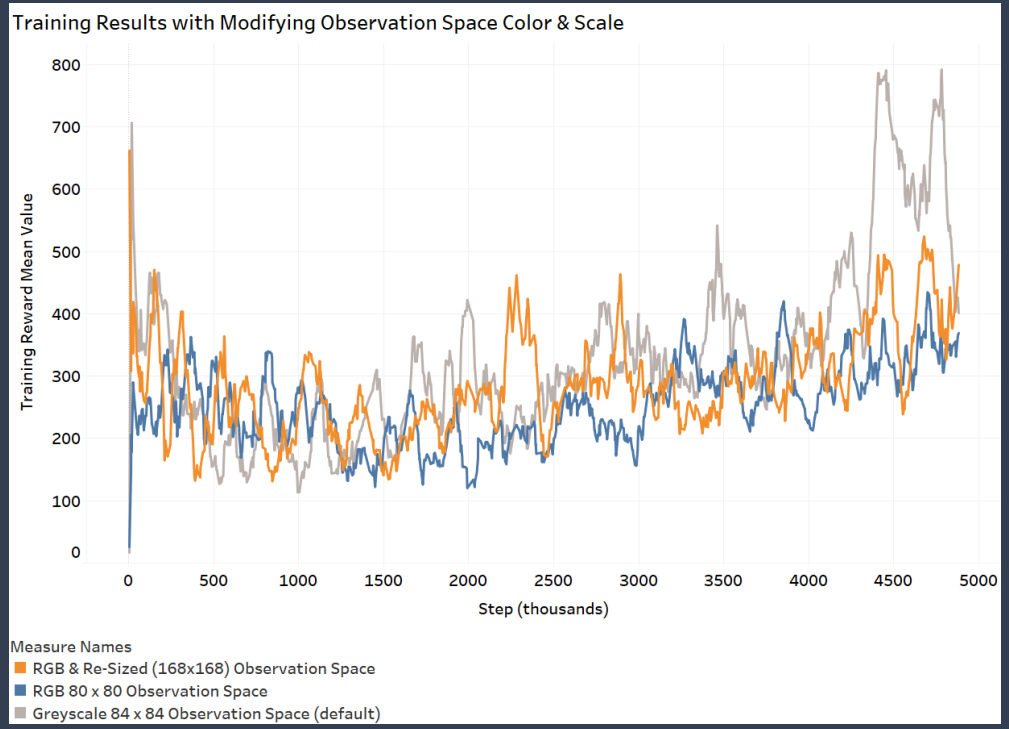
- Test changing default sampling method from 84x84 pixel grayscale images to RGB (colored) images
  - Also test doubling observation size to 168x168 pixels

Results: modifying the observation space (cropping)



- While the first model trained looked very promising, further tests using the same parameters did not improve performance significantly
- However, there does appear to be benefit so further models incorporate cropping

Results: modifying the observation space (RGB images & upscaled)



- There does not appear to be any tangible benefit from utilizing RBG images, or with rescaling, compared to default (84x84 greyscale)
- While not logged, training time was significantly slower as well

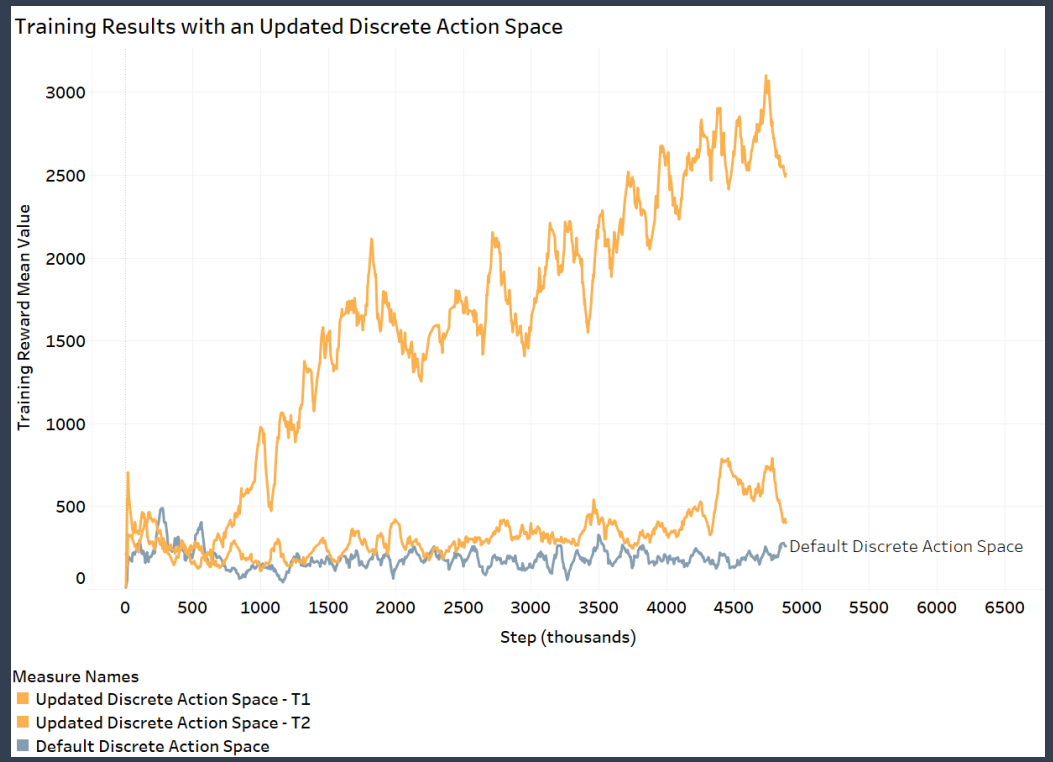
Model improvement methods (3/4)

- Modifying the Action Space
  - Specify valid buttons / actions rather than the default
    - Left, Right, A (jump)
  - Method utilized by top teams in OpenAI Retro Contest to play Sonic the Hedgehog

```
class MarioDiscretizer(Discretizer):
    def __init__(self, env):
        super().__init__(env=env, combos=[
            ['LEFT'],        #0 left
            ['RIGHT'],       #1 right
            ['A']            #2 jump
        ])

def wrap_deepmind_retro(env, scale=False, frame_stack=4):
    """
    Configure environment for retro games, using config similar to DeepMind-style Atari in wrap_deepmind
    """
    env = MarioDiscretizer(env)
    env = WarpFrame(env)
    env = ClipRewardEnv(env)
    if frame_stack > 1:
        env = FrameStack(env, frame_stack)
    if scale:
        env = ScaledFloatFrame(env)
    return env
```

Results: modifying the action space



- Similar to cropping the observation space, while the first model trained looked very promising, another test using the same parameters did not significantly improve performance
- But training performance appears to improve slightly, so further models utilize modified actions

Model improvement methods (4/4)

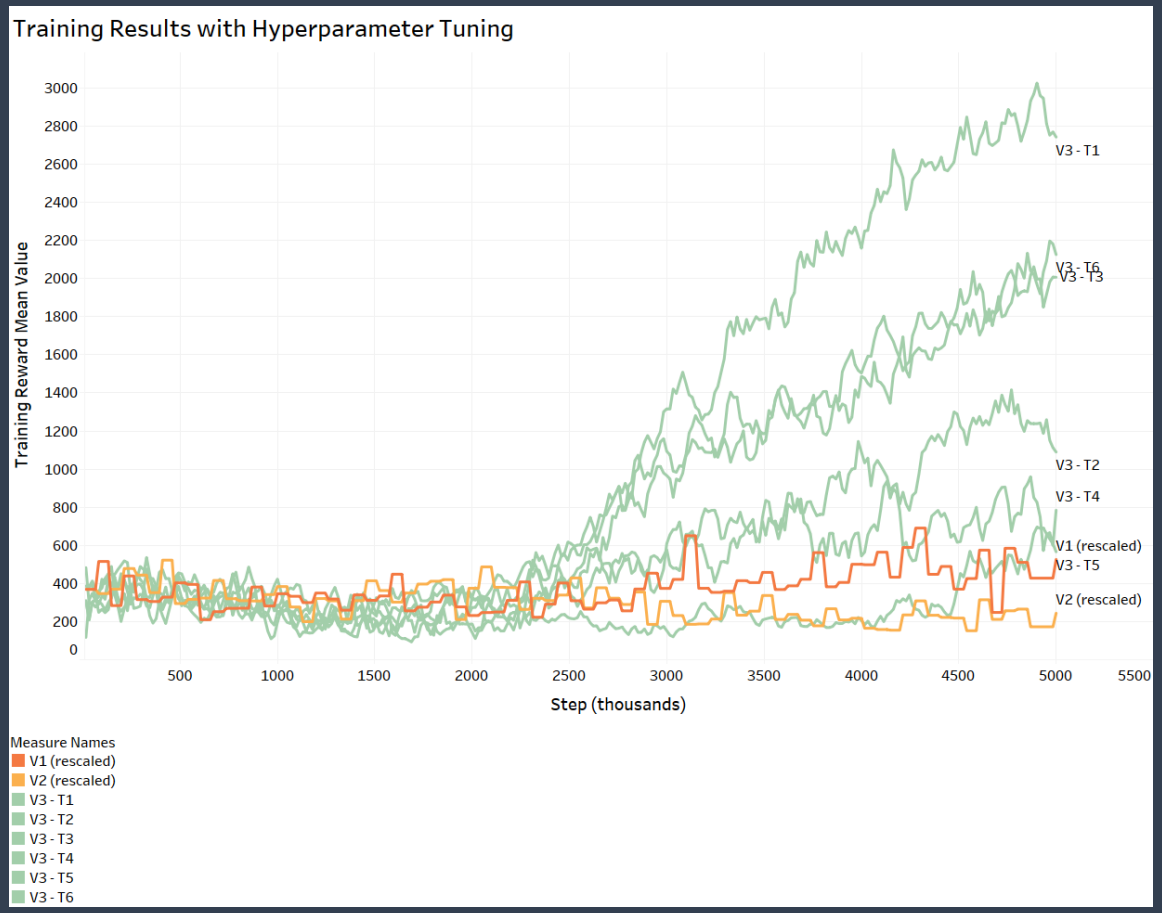
- Hyperparameter Tuning
  - Test algorithm hyperparameters to improve model training performance
  - Based hyperparameters on Team Dhajarama (winning team in OpenAI Retro Contest)

	Baselines PPO2 Default	Team Dhajarama PPO Hyperparameters (bolded if different from default)
policy	n/a	CnnPolicy
nsteps	2048	**8192**
nminibatches	4	**8**
lam	0.95	0.95
gamma	0.99	0.99
noptepochs	4	4
ent_coef	0	**0.001**
lr	3.00E-04	**2.00E-05**
cliprange	0.2	0.2
total_timesteps	n/a	1.00E+10

	V1	*V2*	*V3*
policy	CnnPolicy	CnnPolicy	CnnPolicy
nsteps	8192	8192	**2048**
nminibatches	8	8	8
lam	0.95	0.95	0.95
gamma	0.99	0.99	0.99
noptepochs	4	4	4
ent_coef	0.001	0.001	0.001
lr	2.00E-05	**3.00E-04**	**3.00E-04**
cliprange	0.2	0.2	0.2
total_timesteps	5.00E+06	5.00E+06	5.00E+06

Source: Team Dhajarama Github Repo

Results: hyperparameter tuning



- V3 version of hyperparameters appears to *more-consistently* generate a model that begins to converge on a successful policy of state-action pairs by 5 million steps

Conclusion

Results of viewing model video playback

	Cropped model (A)	Updated Discrete Action Space (B)	Model using V3 Hyperparameters (C)
Training Steps	5 million	5 million	5 million
Mean training score	1,901	2,508	2,742
Able to complete level 1	**No**	**No**	**No**
With model training continued to 20 million steps:			
Mean training score	6,705	n/a (didn't test)	6,471
Able to complete level 1	**Yes**	**n/a (didn't test)**	**Yes**

- None of the models that initially achieved the baseline score were actually able to complete the first level
- However, two of the models (A & C) had training continued to 20 million steps
  - video playback confirmed that they could complete the level

Future Work

- Due to stochasticity observed with training models using the OpenAI Baselines library:
  - Evaluate alternative implementations of the PPO algorithm

- Potentially implement behavior cloning / imitation learning
  - Pre-learn complex sequences, such as the two-step process of killing enemies in Mario Bros

Video of trained models