Deep Q-Learning w/ Breakout

Group 8: Project Presentation

Jovan Ko
Samuel Huang
Joanna Doan
Smriti Davey
Lisa Verma

Presentation Outline

What is reinforcement learning?

What is our project based on?

How we implemented it

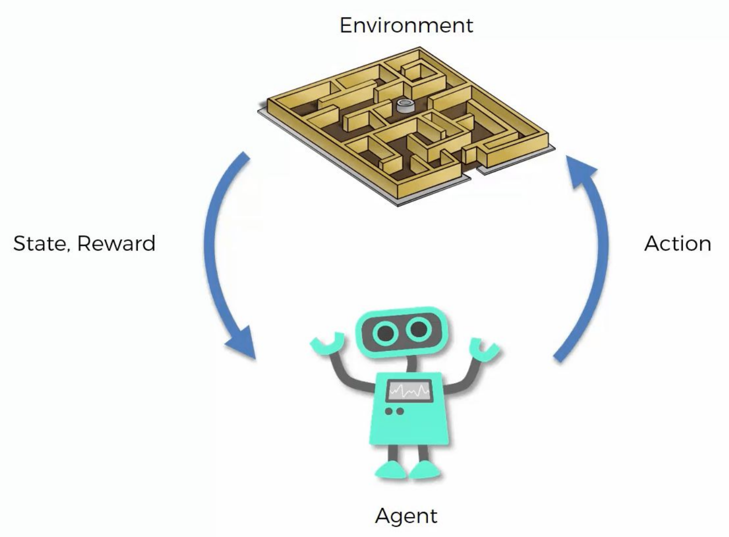
The final outcome

What we learnt from this project and the issues we faced

Introduction

- Video games are tests of AI capabilities
- Our project: training an AI model to play Breakout, an Atari game
  - Used Deep Q-Learning in an OpenAI Gym environment
- Breakout: maneuvering a paddle to hit a ball against a wall of bricks to eliminate them
  - AI model would need spatial awareness, planning, strategy, and precision to do so

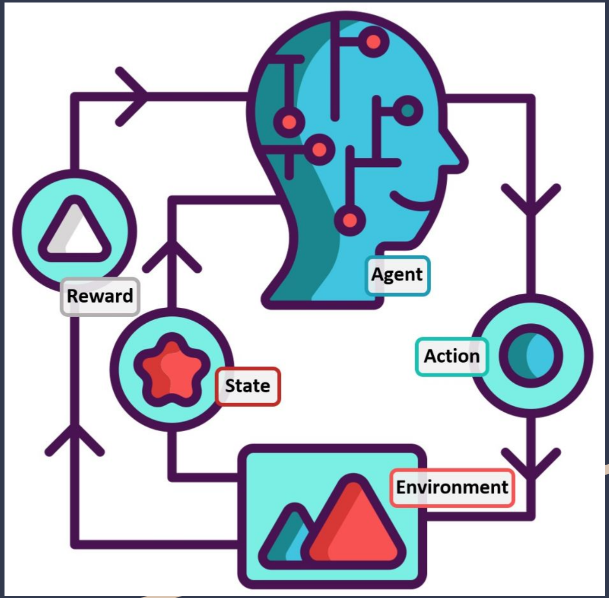
Background



Reinforcement Learning

- Reinforcement learning consists of agents and environment
  - How agents learn from trial and error
- Rewarding or punishing behavior makes it more likely to be repeated or forgoed
- Examples of Applications:
  - To teach computers to control robots
  - To create breakthrough AIs for sophisticated strategy games

Background



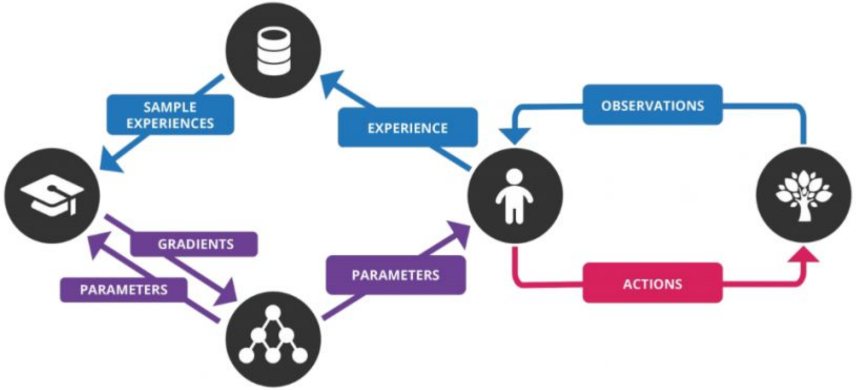
Q-learning

- A reinforcement learning algorithm that learns to map from states to actions to maximize reward
    - Uses a Q-table with expected value of taking each action in each state
    - Table is updated using the Bellman Equation
- Bellman equation is a recursive equation:
    - $Q(s, a) = r + \gamma * \max(Q(s', a'))$
- Simple to implement and can be used to learn complex policies
- Q-learning uses Markov Decision Processes (MDPs) to model environment that the agent is learning in

Background

Deep Q-Learning

- In Deepmind’s 2013 paper: “Playing Atari with Deep Reinforcement Learning”
  - Combining Q-Learning and Convolutional Neural Network
- Deepmind used this to train a model to play some video games from Atari 2600
  - Was able to match and outperform some human players
- This project uses same concept - Deep Q-Learning to train the Atari game ‘Breakout’



Methodology

- Environment: Gymnasium from Faruma (originally OpenAI’s Gym)
  - API has:
    - step() method for learning models to iterate through
    - render() method to render a playthrough or show training steps
- Training Model: StableBaseline3’s DQN Class
  - Allows for easy customization of hyperparameters and training methods
  - Uses a CNN with:
    - input layer: 84x84x4 frame image data
    - 1st hidden layer: 16 8x8 filters w/ stride 4, and rectifier nonlinearity
    - 2nd hidden layer: 32 4x4 filters w/ stride 2 and rectifier nonlinearity
    - 3rd hidden layer: 256 fully-connected rectifier units
- Training Logs and Hyperparameter Tuning:
  - Originally manual tuning and log compiling to get performance plots
  - Finally discovered Weights and Biases API with SB3 integration

Stable Baselines 3

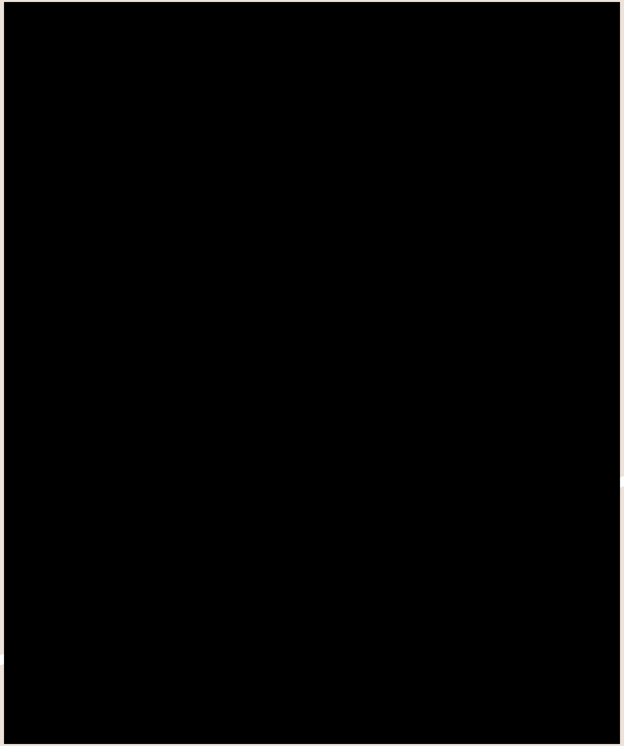
Model Result

After hours of hours of training…

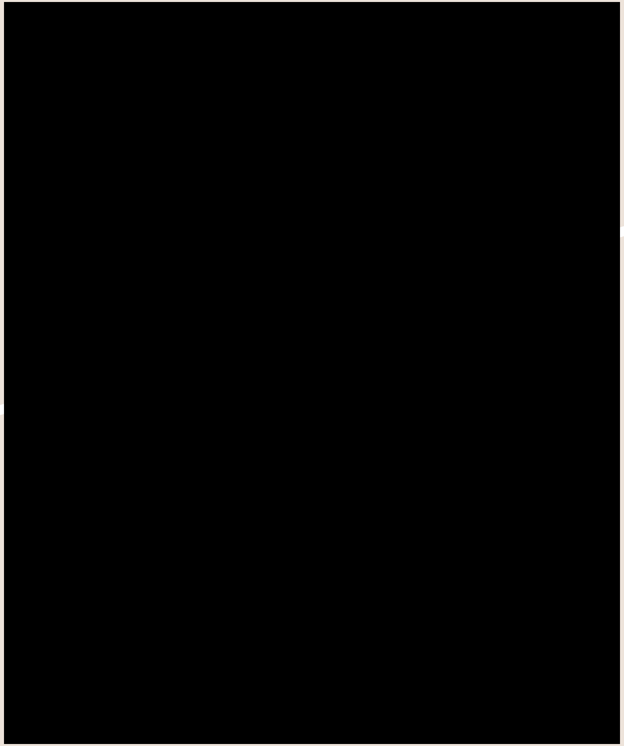
Training History

- ▶ Ver.00 (56 mins) - Default settings with decent result `ep_rew_mean = 20+`
- ▶ Ver.01 (56 mins)
- ▶ Ver.02 (27 mins)
- ▶ Ver.03 (7 hrs)
- ▶ Ver.04 (3 hrs)
- ▶ Ver.05 (20 mins) - Start training on Cuda (GPU) and up x2 training speed
- ▶ Ver.06 (4.5 hrs) - Actual good result `ep_rew_mean = 300+`
- ▶ Ver.07 (7.5 hrs) - Still a good result but still can't finish the game `ep_rew_mean = 350`
- ▶ Ver.08 (8 hrs) - worse than ver.06 `ep_rew_mean = 300`
- ▶ Ver.09 (2 hrs) - not enough training time `ep_rew_mean = 150`
- ▶ Ver.10 (4 hrs) - not enough training time? `ep_rew_mean = 320`
- ▶ Ver.11 (9.5 hrs) - good result `ep_rew_mean = 390`

Model Result



ver.02



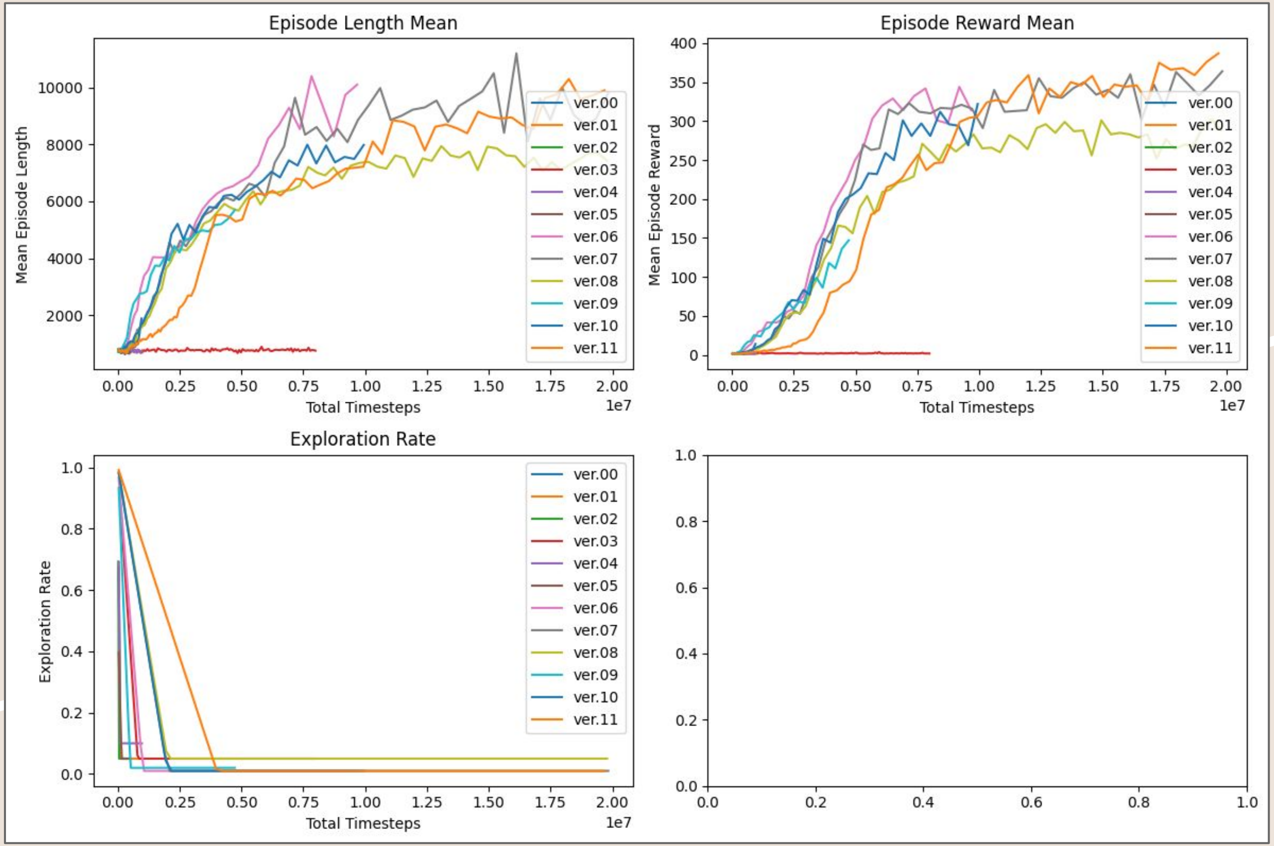
ver.06

Training Plots

Before implementing Weights & Biases Plots

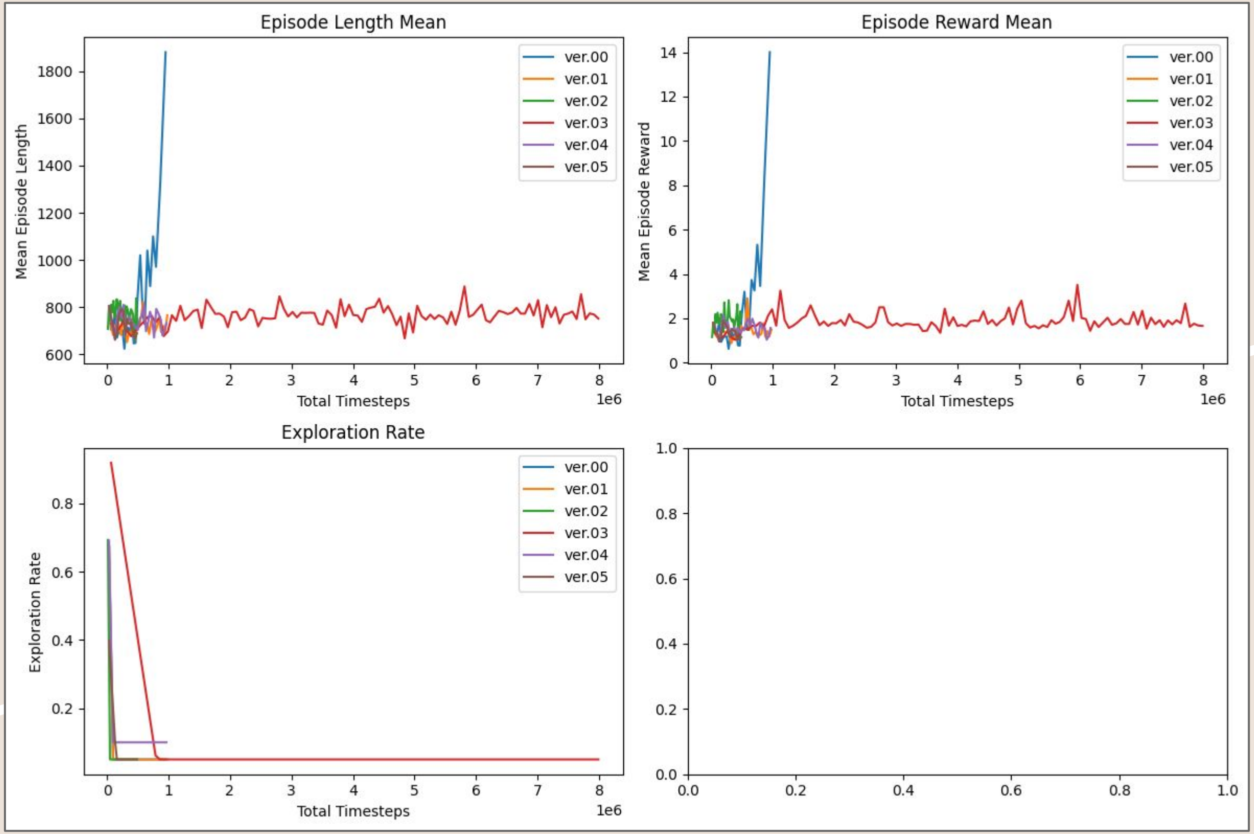
Results

All training data (ver.00 ~ ver.11)



As you can see from the graph, it separate into two main parts: one earlier model and later improved model.

Results Early training model (ver.00 ~ ver.05)

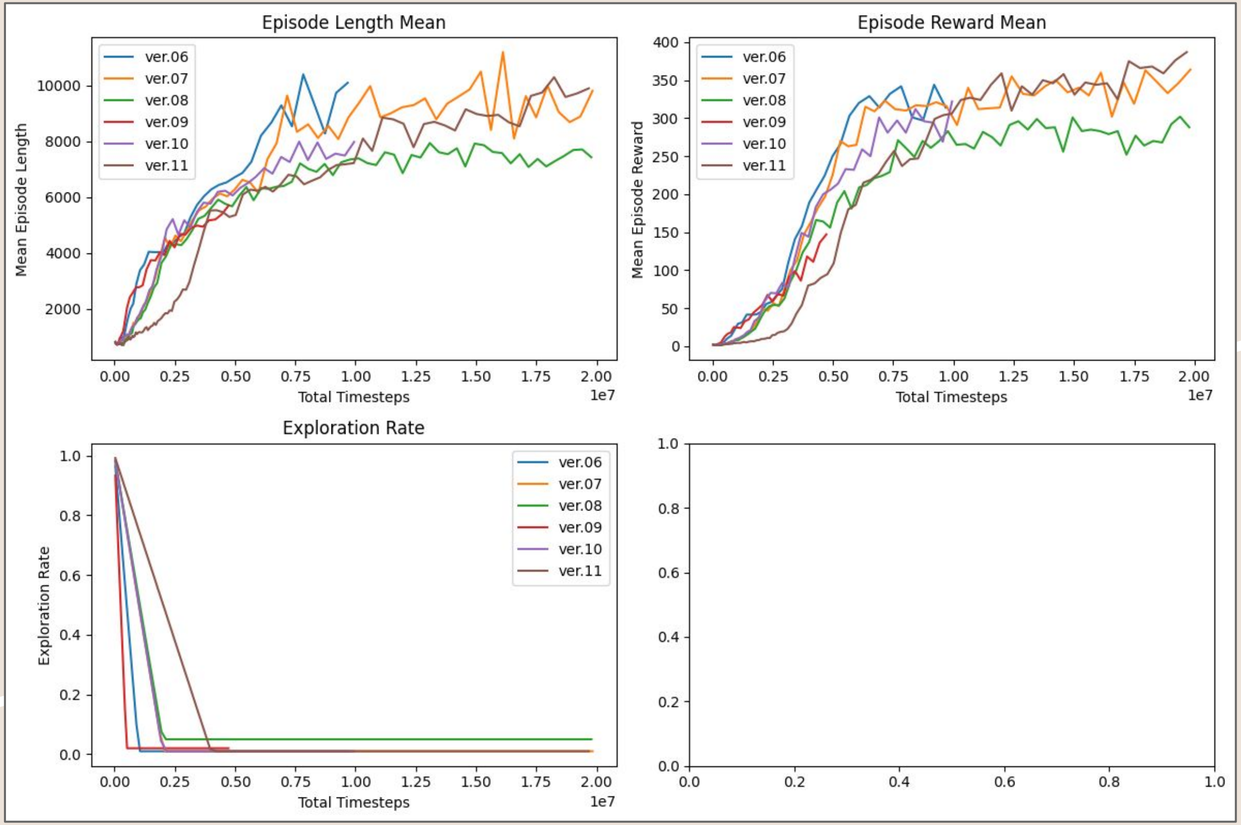


Parameters

```
policy = "CnnPolicy"
learning_rate = 0.0001 -> 0.001
buffer_size = 100
learning_starts = 50000
batch_size = 32
tau = 1.0                    # soft update coefficient
gamma = 0.99                 # discount factor
train_freq = 4
gradient_steps = 1
target_update_interval = 10000
exploration_fraction = 0.1
exploration_initial_eps = 1.0
exploration_final_eps = 0.05
```

```
total_timesteps = 1000000 -> 500000
log_interval = 1000 -> 500

model.learn(total_timesteps=total_timesteps, log_interval=log_interval,
            progress_bar=True)
model.save("dqn_breakout_02")
```

Results Improved training model (ver.06 ~ ver.11)



Parameters

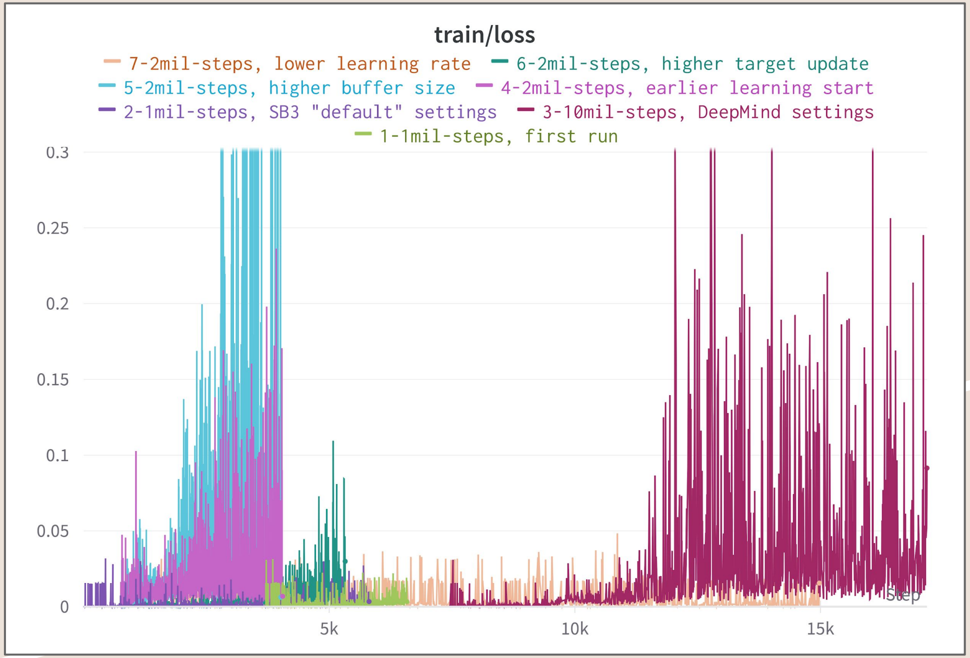
```
policy = "CnnPolicy"
learning_rate = 0.0001
buffer_size = ~~100~~ -> 10000
learning_starts = ~~50000~~ -> 100000
batch_size = 32
tau = 1.0                          # soft update coefficient
gamma = 0.99                       # discount factor
train_freq = 4
gradient_steps = 1
target_update_interval = ~~10000~~ -> 1000
exploration_fraction = 0.1
exploration_initial_eps = 1.0
exploration_final_eps = ~~0.05~~ -> 0.01
device = "cuda"                    # (CPU:"CPU", GPU:"cuda")
```

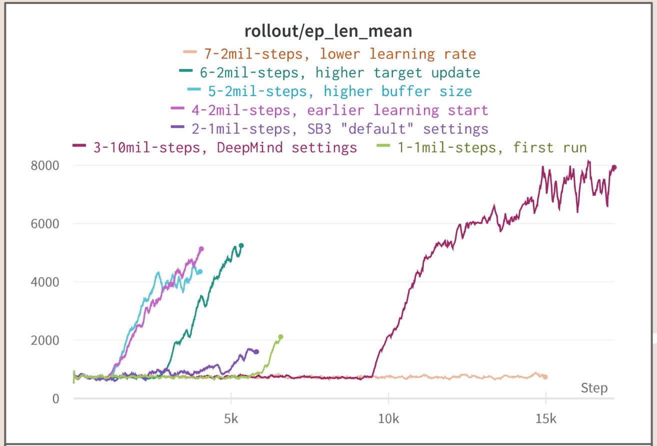
```
total_timesteps = ~~1000000~~ -> 10000000
log_interval = 1000

model.learn(total_timesteps=total_timesteps, log_interval=log_interval,
            progress_bar=True)
model.save("dqn_breakout_06")
```

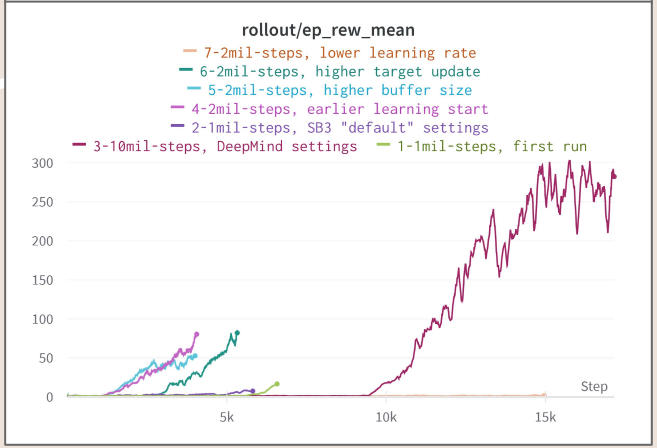
Training Plots

After implementing Weights & Biases Plots

Results
Weights & Biases Plots



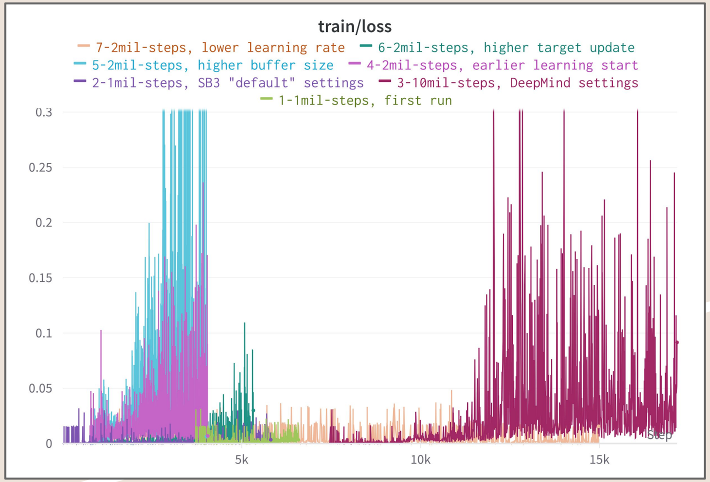


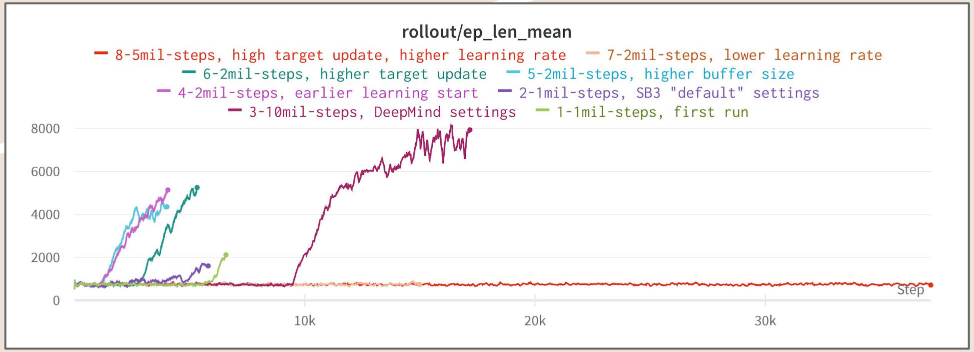


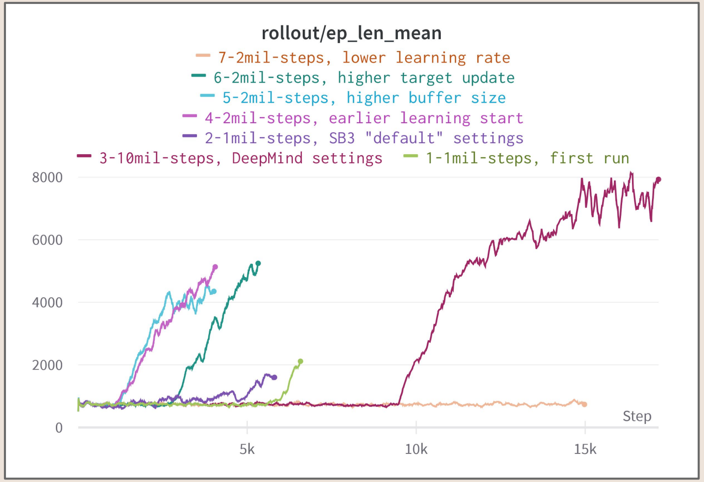
We notice interesting relationships between training loss and avg. episode performance (length and reward), specifically that overall decrease in loss doesn't correlate to decrease in performance

Discussion

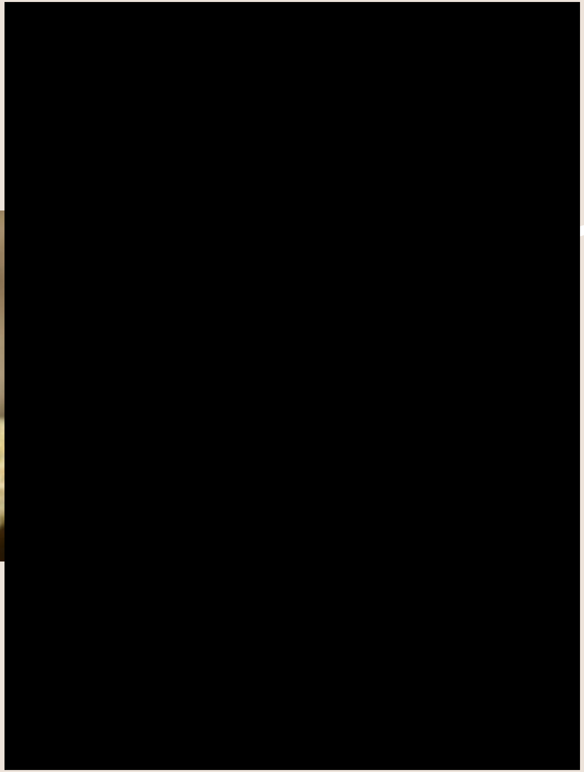
- From training different models, we paid attention to a few hyperparameters and their behaviors
  - **Learning rate** - overfitting vs. learning efficiency
  - **Buffer size** - training loss vs. performance improvement
  - **Exploration rate** - how fast/slow to decrease ε
  - **Learning start**: - start learning before/after exploration?
- Comparison with original DeepMind model
- Limitations and Future Directions







Two Bugs (collusion bugs and corner strats)



Conclusion

- The objective of our project was to use Deep Q-learning to train an AI model to play Breakout inspired by Google’s Atari Deepmind Paper

One more thing…

Thank you!