Evaluation on Transformers Based Reinforcement Learning For Games

Evan Knobler, ek3093
Johan Sweldens, jws2215

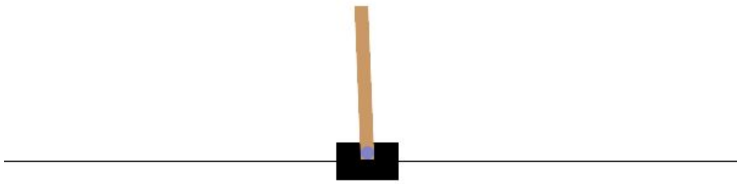
EECS 6892: Reinforcement Learning
Spring 2024

Agenda

- Project Description
- Summary of Paper
- Approach
- Software Implementation
- Results

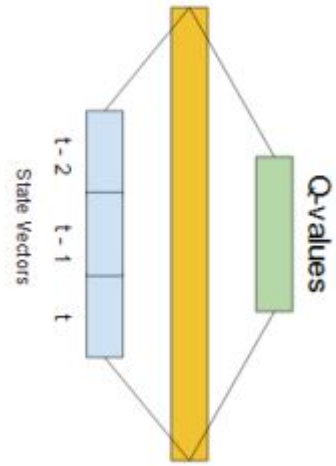
Project Description

- Seek to replicate this paper: “Transformer Based Reinforcement Learning for Games” by Upadhyay et al.
- Compares performance of DQN and DTQN
    - Keep the same training hyperparameters
    - Record different score values
- Simple Classic Control Game: Cartpole-v1
    - Only game from the paper
    - 4 continuous value states
- Complicated RGB Atari Game: ALE/Breakout-V5
    - Expanded to see if results can be extrapolated
    - state_shape: (210, 160, 3)
    - state_size: 100800

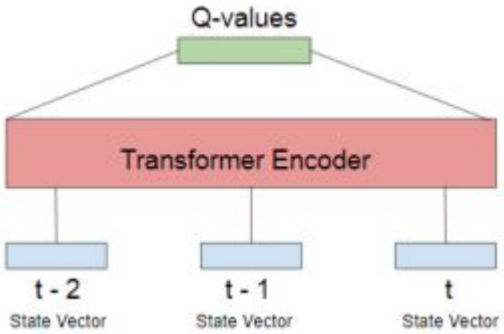


Summary of Paper

- Aimed to train several networks to play Cartpole using actor/critic reinforcement learning
  - Deep Q Network
  - Deep Recurrent Q Network
  - Deep Transformer Q Network
- Hyperparameters
  - Learning rate: 1e-4
  - Batch size: 32
  - Discount factor: 0.99
  - Epsilon decay rate: 5e-6
  - 5000 episodes
- Used the four previous states as input
- Found that the DQN outperformed DTQN



(a) DQN



(c) DTQN

Num	Observation	Min	Max
0	Cart Position	-4.8	4.8
1	Cart Velocity	-Inf	Inf
2	Pole Angle	~ -0.418 rad (-24°)	~ 0.418 rad (24°)
3	Pole Angular Velocity	-Inf	Inf

Approach

Our approach

- Added Breakout Atari Game
  - Flattened to MLP
- Implemented DQN, DTQN
  - Same architecture
- State space is fully observable
- Added play feature to visualize the game
- Used Same hyperparameters
  - Learning rate, epsilon decay, episodes, etc
- Experimented with Biased Memory, but kept standard memory for results

The papers approach

- Solely focused on Cartpole Game
- Implemented DQN, DRQN, DTQN
  - Same architecture
- State space is only partially observable
  - Removed velocity states
  - Multiple frames to “simulate” velocity
- Used same hyperparameters
  - Learning rate, epsilon decay, episodes, etc
- Used standard memory class

Software Implementation

- Separate the repository into 8 Jupyter Notebooks:
  - train_pole_DQN, play_pole_DQN
  - train_pole_DTQN, play_pole_DTQN
  - train_breakout_DQN, play_breakout_DQN
  - train_breakout_DTQN, play_breakout_DTQN
- Utility functions written in python for training, playing, and visualizing the results
- Used the deep neural networks provided by Upadhyay et al.

DQN

- Replicated the DQN from paper exactly
- No hidden layers, pooling, dropout, regularization, etc
- MLP for Cartpole and Breakout, no CNN
- Initialize weights with Xavier uniform distribution
- Two fully connected layers: input size, 128 units, output size

DQN_Model.py 3

Users > johansweldens > Documents > EECS6892.RL > final_project

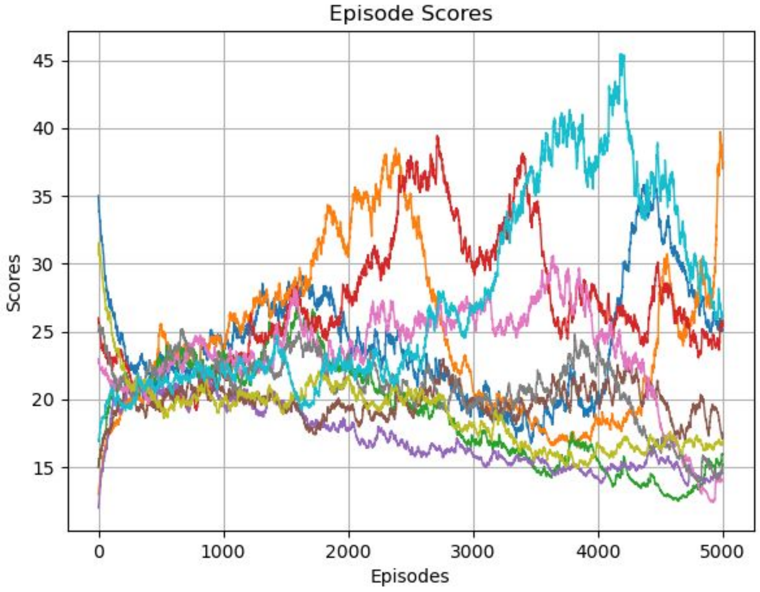
```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class DQN(nn.Module):
    def __init__(self, num_inputs, num_outputs):
        super(DQN, self).__init__()
        self.num_inputs = num_inputs
        self.num_outputs = num_outputs

        self.fc1 = nn.Linear(num_inputs, 128)
        self.fc2 = nn.Linear(128, num_outputs)

        # intialize the weights in xavier_uniform
        for m in self.modules():
            if isinstance(m, nn.Linear):
                nn.init.xavier_uniform_(m.weight)

    def forward(self, x):
        # MLP
        x = F.relu(self.fc1(x))

        x = self.fc2(x)

        return x # should be the qvalue


    def get_action(self, input):
        qvalue = self.forward(input)
        action = torch.argmax(qvalue)
        return action.cpu().numpy()
```

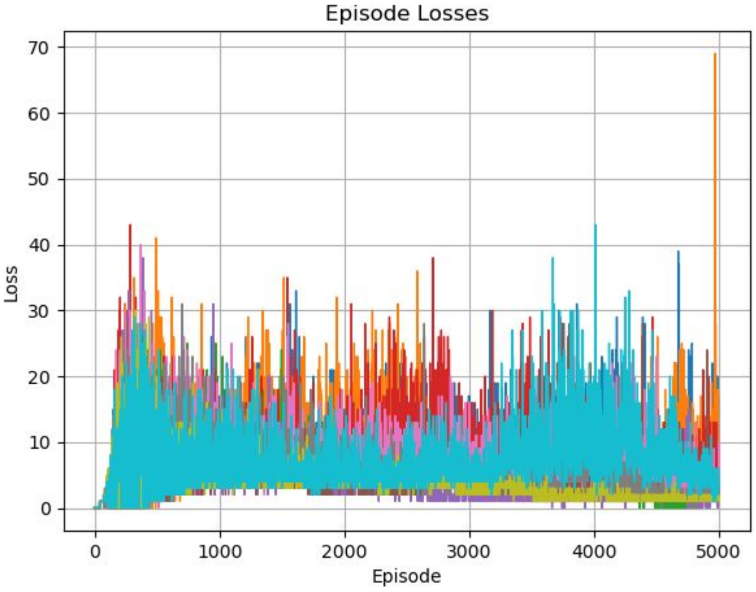
DTQN

- Transformer:
  - 64 nodes
  - 2 heads
  - 3 layers
- Fully Connected:
  - 2 layers
  - ReLU activation

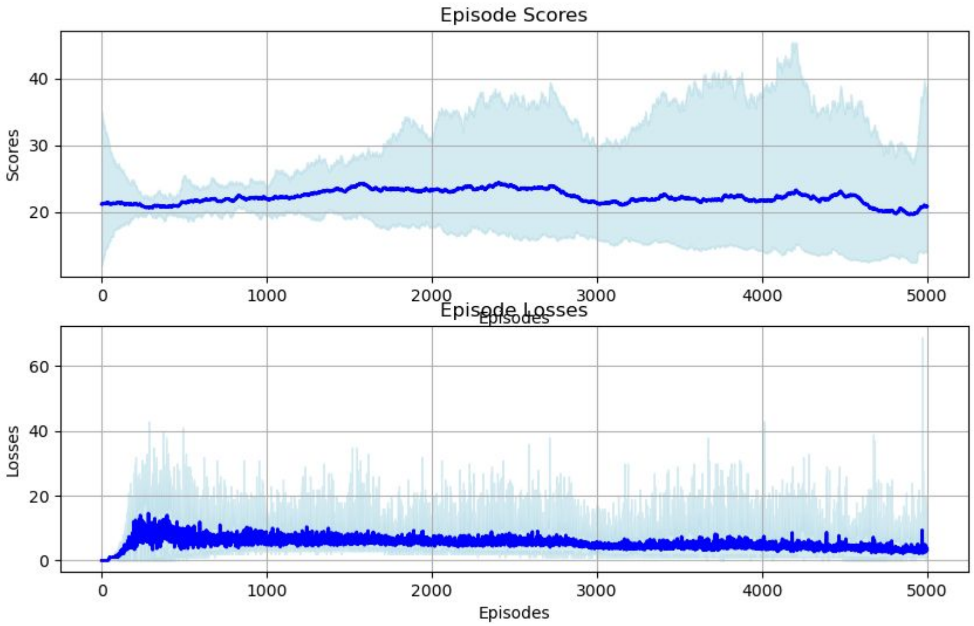
```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class DTQN(nn.Module):
    def __init__(self, num_inputs, num_outputs):
        super(DTQN, self).__init__()
        self.num_inputs = num_inputs
        self.num_outputs = num_outputs

        self.fc = nn.Linear(num_inputs, 64)
        self.Tlayer = nn.TransformerEncoderLayer(d_model=64, nhead=2)
        self.transformerE = nn.TransformerEncoder(self.Tlayer, num_layers=3)

        self.fc1 = nn.Linear(64, 32)
        self.fc2 = nn.Linear(32, num_outputs)

        for m in self.modules():
            if isinstance(m, nn.Linear):
                nn.init.xavier_uniform_(m.weight)

    def forward(self, x):
        x = self.fc(x)
        out = self.transformerE(x)
        out = F.relu(self.fc1(out))
        qvalue = self.fc2(out)

        return qvalue

    def get_action(self, input):
        input = input.unsqueeze(0)
        qvalue = self.forward(input)
        action = torch.argmax(qvalue)
        return action.cpu().numpy()
```

Results Pole - DQN Training

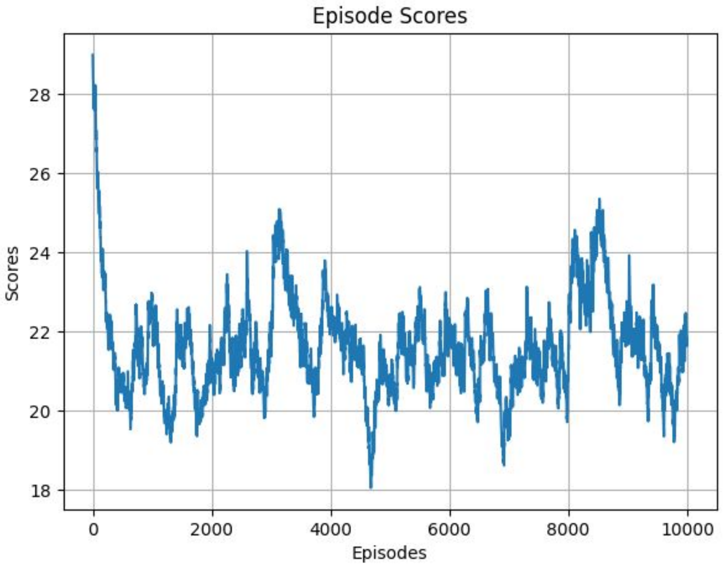


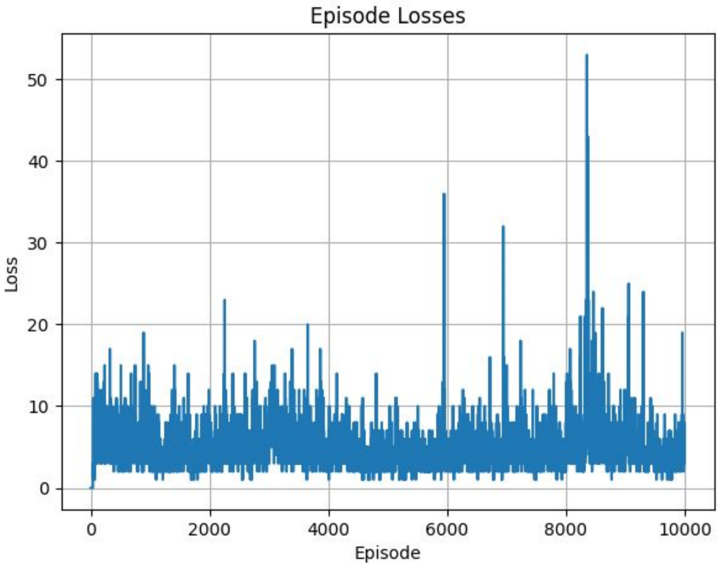


Results Pole DQN Training

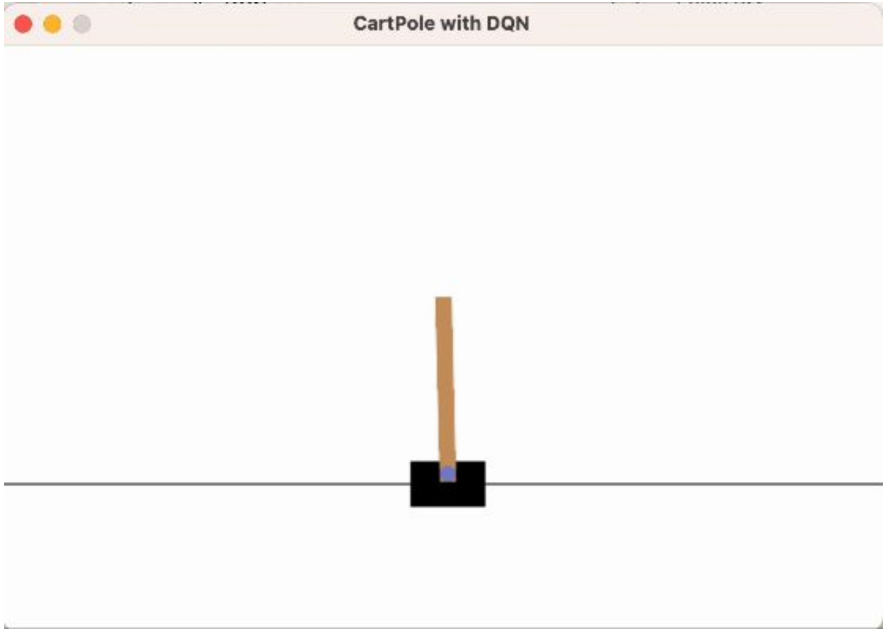


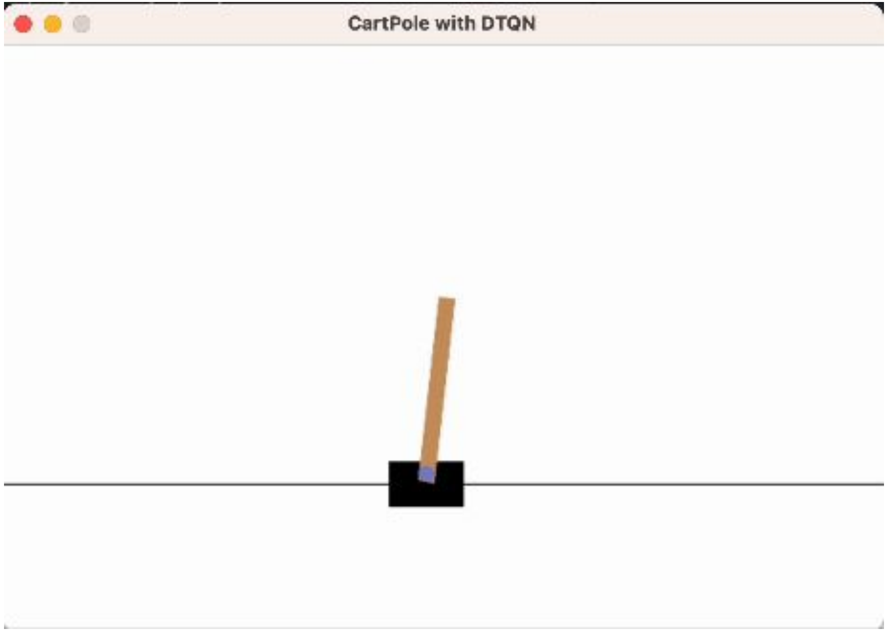
Results Pole DTQN Training



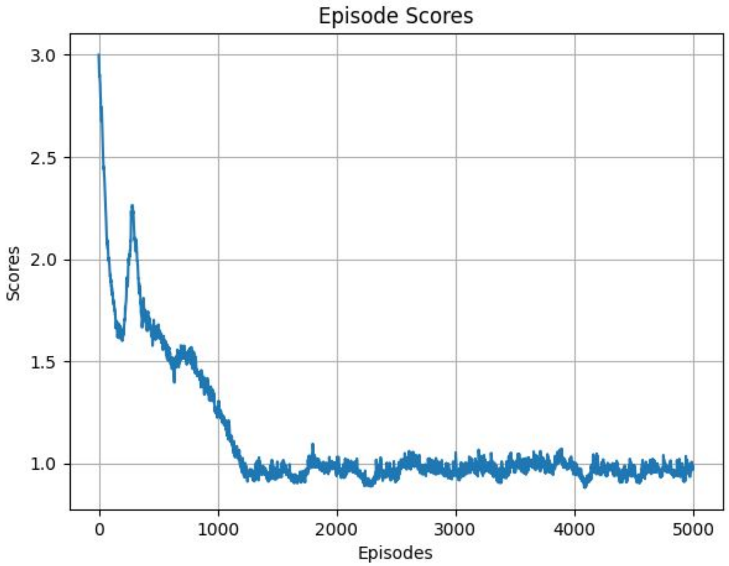


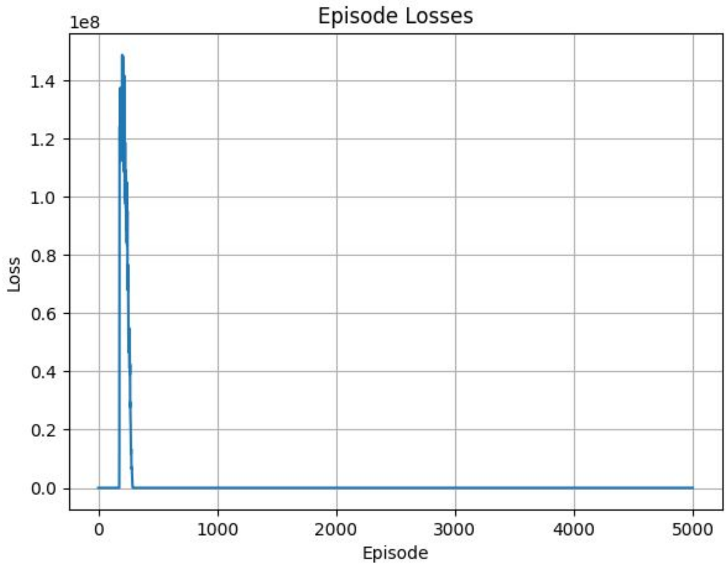
Results Pole - GIFS



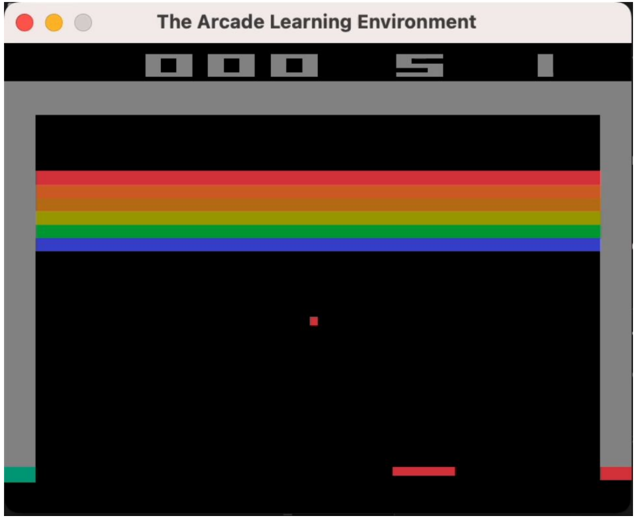


Results Breakout - DQN Training

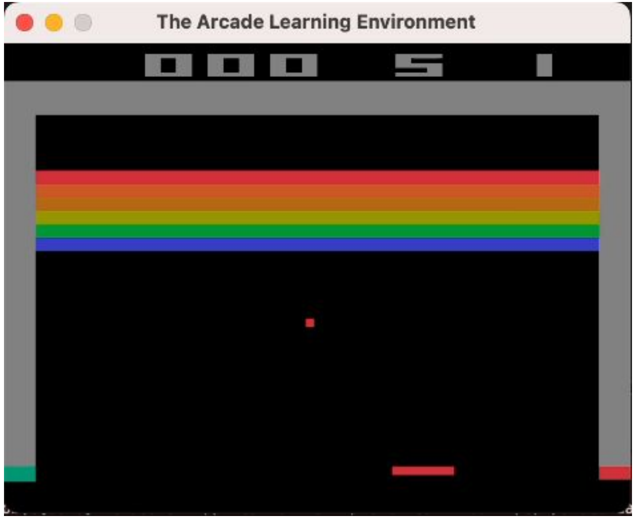




Results Breakout - DQN

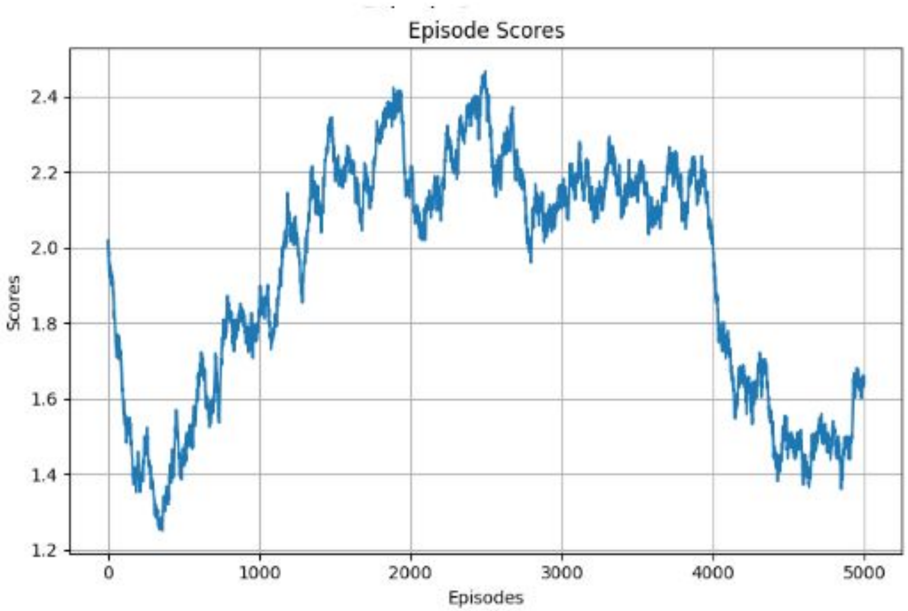


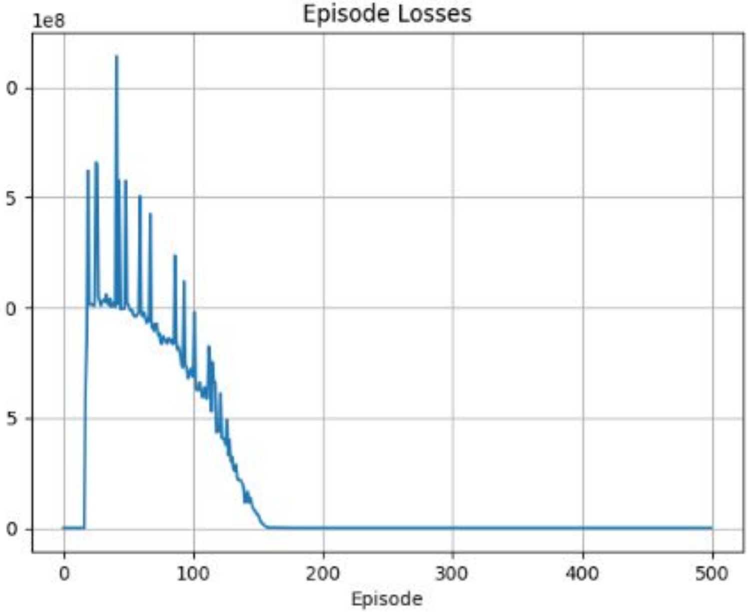
DQN, 500 episodes



DQN, 5000 episodes

Results Breakout - DTQN Training





Results Breakout - DTQN

DTQN, 500 episodes

DTQN, 5000 episodes

Results Comparison

- Our Cartpole implementation outperformed that of Upadhyay et al.
  - Ours: stayed up for 5 seconds on average
  - Theirs: stayed up for 4.5 seconds on average
- Why?
  - We included all four states in our network inputs
  - We tweaked the hyperparameters
    - Epsilon decay rate → 1e-5

Discussion

- The DTQN model outperformed the DQN model
  - Cartpole
  - Breakout
- Transformer architecture is most robust

Future Work:

- Experiment with more complex games:
  - Chess, Tetris, etc.
- Use convolutional neural networks for Breakout
- Fine tuning hyperparameters

References

https://arxiv.org/pdf/1912.03918v1 - main paper

https://machinelearningmastery.com/the-transformer-model/

https://arxiv.org/abs/2307.05979

https://github.com/sweldensj2/eecse6892-spring24-final-project-jsek - Github

https://github.com/mayankamedhe/Transformer-based-Deep-Reinforcement-Learning-for-Video-Games/tree/master/src - Github, Upadhyay et al.

https://www.youtube.com/watch?v=-SPxtoknbOE&ab_channel=JohanWillemSchulzSweldens - youtube

Thank you!

Appendix - Task Accomplished by Each Team Member

Johan	**Evan**
DQN implementation for training, playing, and evaluating for Cartpole and Breakout	DTQN implementation training, playing, and evaluating for Cartpole and Breakout