Deep Reinforcement Learning and Transfer Learning with Flappy Bird

Cedrick Argueta\textsuperscript{1}, Austin Chow\textsuperscript{1}, Cristian Lomeli\textsuperscript{1}

Department of Computer Science\textsuperscript{1}, Stanford University, Stanford, CA 94305

Motivation

Reinforcement learning is a technique for solving certain decision tasks where an agent learns how to act in a real world environment. In recent years, major breakthroughs in RL have come from common video games. Since we grew up playing tons of video games, we wanted to explore the applicability of RL to the games Flappy Bird and Pixel Copter.

Problem

- We wish to use reinforcement learning to play the games Flappy Bird and Pixel Copter, determining whether RL can beat us and/or an expert in score.
- Q-learning does not generalize well to large state spaces, since many states would be left unexplored. We can use deep Q-learning, which uses a neural network to approximate the Q-value function and allows us to generalize to unseen states.
- Instead of using the game's screen as input, we use feature engineering, which allows us to achieve the same level of performance without having to train a CNN to learn features for each game.
- We also aim to explore the impact of transfer learning to our task. In our case, transfer learning would be starting an instance of Pixel Copter training with weights already trained on Flappy Bird.

Challenges

- Training required extensive time and compute
  - Opted to forgo image input and use feature engineering instead
- Difficult to pull together several packages for this task
  - Used ALE, PLE, Keras-rl, OpenAI Gym, tensorflow
- Transfer learning was traditionally used for CNNs
  - Changed observation spaces of both games to be as similar as possible
- Couldn't use lots of data we collected since we changed our training and testing process several times

References


Methods and Models

Deep Q Learning with experience replay

We use a neural network to approximate the Q function, and perform weight updates based on mini batches drawn from a cache of (s, a, r, s') tuples.

For each episode, we do:

1. Act
2. Observe
3. Update

Linearly annealed \(\varepsilon\)-greedy

Store in experience replay

Perform gradient descent w/ Adam optimizer on minibatch from memory

Neural Network Architecture

- Input in \(R^3\) and output in \(R^5\), which are the Q-values for the two actions, jump or not jump.
- Each hidden layer pictured here has dimensionality of \(2^{n}\) mm nodes pictured.
- The input layers and hidden layers are followed by ReLU layers, while the last layer is followed by a linear activation layer.

Feature Engineering

We had to come up with a way to make the state spaces similar to facilitate transfer learning, but didn't want to use images as input.

We used the following features for both games:

- y position and velocity (orange arrow)
- distance to next terrain (blue line) and next next terrain (red line)
- absolute y positions of the next terrain (blue dot) and next terrain (red dot)
- additional, Pixel Copter had obstacles that had no analogue in Flappy Bird
- distance to next block obstacle (purple line)
- absolute y positions of the next block obstacle (purple dot)

Table for hyperparameters

<table>
<thead>
<tr>
<th>Hoparameters</th>
<th>Flappy Bird</th>
<th>Pixel Copter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target Model Update</td>
<td>1e-3</td>
<td>1e-2</td>
</tr>
<tr>
<td>Learning Rate</td>
<td>1e-3</td>
<td>1e-4</td>
</tr>
<tr>
<td>(\varepsilon)</td>
<td>0.99</td>
<td>0.99</td>
</tr>
<tr>
<td>Exploration Policy</td>
<td>Linearly annealed (\varepsilon)-greedy</td>
<td></td>
</tr>
<tr>
<td>Annned (\varepsilon)</td>
<td>0.2</td>
<td>0.05</td>
</tr>
<tr>
<td>Annned Steps</td>
<td>50,000</td>
<td>10,000</td>
</tr>
<tr>
<td>Warm Up Steps</td>
<td>50</td>
<td>100</td>
</tr>
<tr>
<td>Training Steps</td>
<td>100,000</td>
<td>50,000</td>
</tr>
</tbody>
</table>

Table for hyperparameters

<table>
<thead>
<tr>
<th>Reward Profile</th>
<th>Flappy Bird</th>
<th>Pixel Copter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tick</td>
<td>0.3</td>
<td>0.3</td>
</tr>
<tr>
<td>Passed Obstacle</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Colled</td>
<td>-10.0</td>
<td>-10.0</td>
</tr>
</tbody>
</table>

Results

6 runs of Flappy Bird training with the hyperparams to the left. We don't show testing data here since each model can run indefinitely.

6 runs of Pixel Copter training with the hyperparams to the left. Test data is shown below, demonstrating little to no improvement for average ability of the agent when using transfer learning, but improvement in the best performance for each agent.

Conclusion and Future Work

- Deep reinforcement learning was able to play both Pixel Copter and Flappy Bird better than we could, and for Flappy Bird in particular our agent reached superhuman levels of ability.
- We did see transfer learning improve training times for Pixel Copter and absolute performance slightly, but only after we used some tricks to ease the process.
- We expect that transfer learning when using images as input would be much more impactful, since we wouldn't need to relearn as much to interpret the game's screen.
- In the future, we'd like to play around with games that have higher dimensionality in terms of observation and action spaces. There are many cool things in RL right now, like OpenAI's Dota bot and DeepMind's AlphaGo Zero!