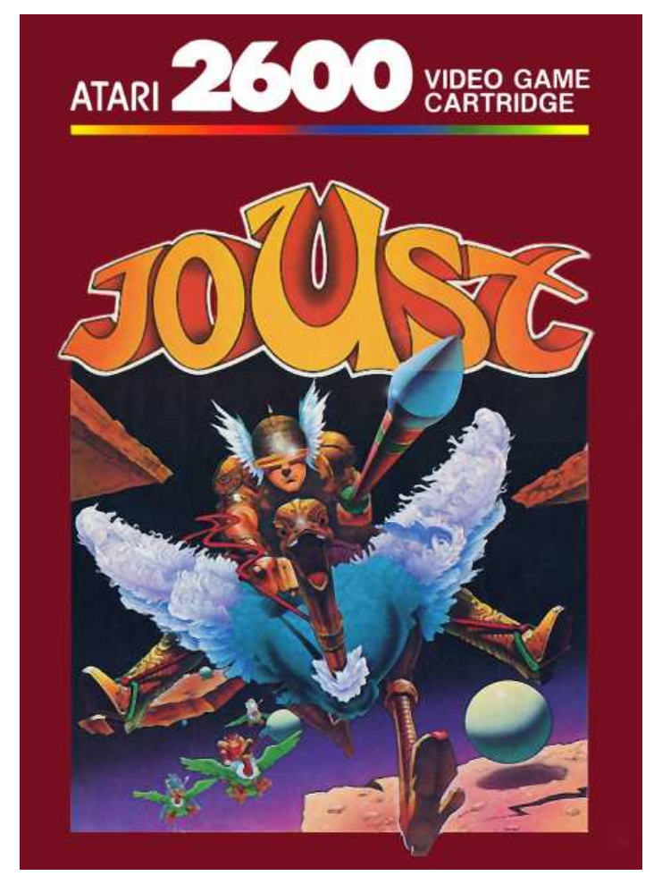
Exploring the Effectiveness of Klijn's Coevolutionary Evolution Strategy for Multi-Agent Reinforcement Learning in Atari's Joust

Jason Zheng and Bryce Anthony

Davidson College

Problem Statement



We use a novel evolutionary strategy alongside neuroevolution to train a neural network to competently play Joust.

Motivation

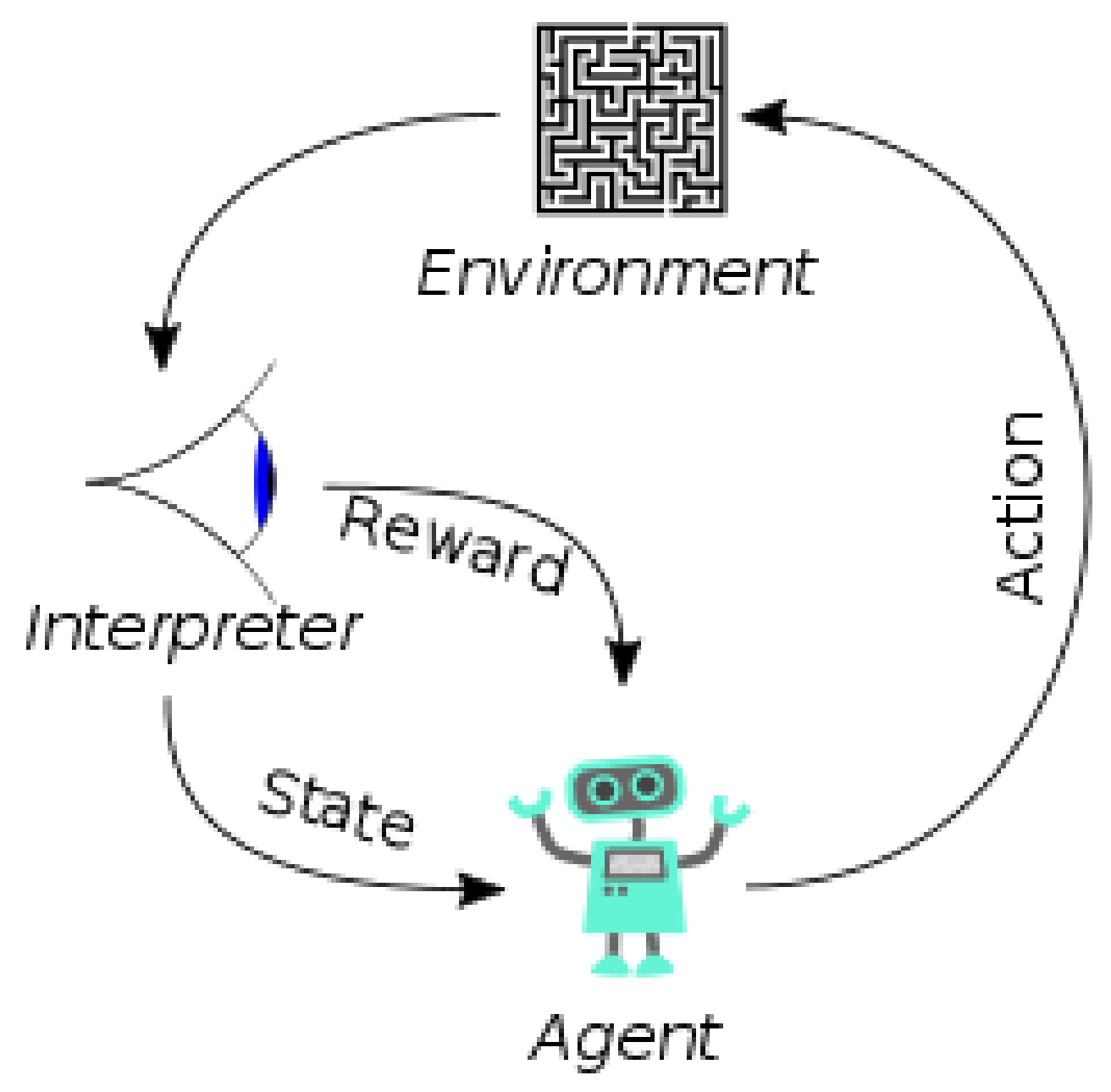
- Atari games provide a controlled environment where agents can learn to maximize rewards by interacting with the game environment.
- Studying how multiple agents can collaborate and compete with each other in these complex environments can lead to insights that can help solve complex real-world problems more effectively.
- Our project is a continuation of the existing literature from [2] that focuses on the application of their Coevolutionary Evolution Strategy for Atari's Joust Game.

Background

Atari's Joust is a mixed competitive-cooperative game where a player(s) must:

- Work together to defend themselves against 3 types of increasingly dangerous computer-controlled "Buzzard Riders" [1].
  Points are scored points by:
  - Unseating opponents in a joust. The winner of a joust is the rider whose mount is highest at the moment of contact. If the mounts are of equal height, the joust is a draw.
  - Grabbing the egg that spawns after unseating a Buzzard Rider in a joust, if the egg isn't caught it will hatch into a more dangerous opponent. [1]
- Compete against each other in "special waves" to achieve goals that differ slightly from the main game

Reinforcement learning is a machine learning training method based on rewarding desired behaviors and/or punishing undesired ones. In general, reinforcement learning has an agent that is able to interpret its environment and interact with it via a set of predefined actions. This ability to award and punish outcomes allows for algorithms to be written that can allow an agent to learn through trial and error.



This continuous learning can be applied to many domains, like Atari games, which have become an important tool for the advancement of reinforcement learning. Atari games present a unique opportunity to apply reinforcement learning algorithms to tasks that are challenging to humans but are also reasonably complex.

Methods

To train our agents we do neuroevolution using the coevolutionary evolution strategy. Neuroevolution is an evolutionary strategy, where the weights of a neural network are changed to train the network. We use one policy and mutate it to create "exploratory policies" that are then used to update the weights of the network. The algorithm is shown in Figure 1

```
Algorithm 1 Coevolutionary Evolution Strategy
Initialize Learning rate $\alpha$, HoF size $k$, noise standard deviation $\sigma$, initial policy parameters $\theta_0$
For $t = 0, 1, 2, \ldots$
    Sample $\epsilon_1, \ldots, \epsilon_n \sim \mathcal{N}(0, I)$
    Compute returns $F_i = \frac{1}{k}\sum_{s=0}^{k-1} F(\theta_{t-s}, \theta_t + \sigma\epsilon_i)$
    For $i = 1$ to $n$
    Set $\theta_{t+1} \leftarrow \theta_t + \frac{\alpha}{n\sigma}\sum_{i=1}^{n} F_i\epsilon_i$
```

Figure 1:Coevolutionary Evolutionary Stratey [2]

Experimental design

For our experiments, we begin by using the hyperparameters specified in [2]; we used a learning rate of 0.1, a HoF size of 1, a noise standard deviation of 0.05, and two neural networks (Model 1 and Model 2).

Mutation

- **Mutation 1: Population-Wise Mutation** -- Each individual policy in the population was a randomly generated Gaussian distribution matrix with a mean of 0 and a standard deviation of 1. Each randomly generated policy was combined with the policy in Model 1 using the noise standard deviation to generate an "explorative" policy based on the policy in Model 1.
- **Mutation 2: Generation-Wise Mutation** -- To obtain the new policy to be used by Model 1 in the next generation, we followed line 6 in 4. This was done by normalizing the fitness-weighted sum of the policies in the population using the learning rate, the population size, and the noise standard deviation and combining it with the current policy in Model 1.

Neural Network Architecture

Each model consists of 3 convolutional layers, and a 512-node densely connected layer followed by a 9-node output layer. The first convolutional layer had 32 channels with a filter size of 8x8 taking strides of 4 pixels. The second convolutional layer had 64 channels with a filter size of 4x4 taking strides of 2 pixels. The third convolutional layer was 64 channels with a filter size of 3x3 taking strides of 1 pixels. Each of these layers used rectifier linear units as their activation function and the final layer was only 9 nodes to represent the 9 unique moves available in the game of joust.

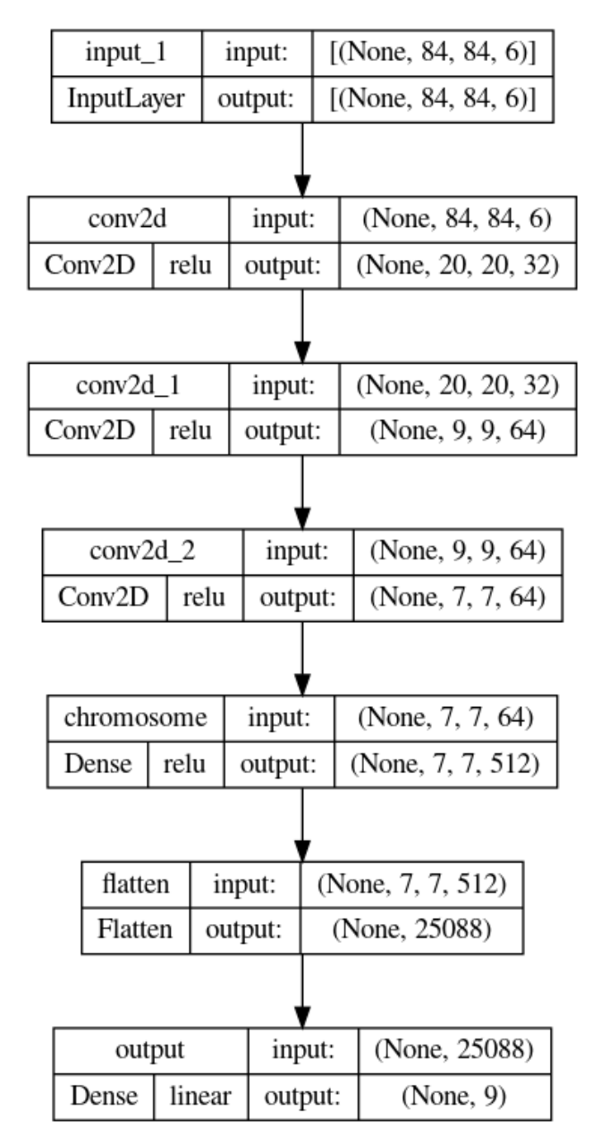

input_1 | InputLayer | input: [(None, 84, 84, 6)] | output: [(None, 84, 84, 6)]
conv2d | Conv2D | relu | input: (None, 84, 84, 6) | output: (None, 20, 20, 32)
conv2d_1 | Conv2D | relu | input: (None, 20, 20, 32) | output: (None, 9, 9, 64)
conv2d_2 | Conv2D | relu | input: (None, 9, 9, 64) | output: (None, 7, 7, 64)
chromosome | Dense | relu | input: (None, 7, 7, 64) | output: (None, 7, 7, 512)
flatten | Flatten | input: (None, 7, 7, 512) | output: (None, 25088)
output | Dense | linear | input: (None, 25088) | output: (None, 9)

Figure 2:Neural Network Demonstration 1

Conv2D
Dense
Flatten

Figure 3:Neural Network Demonstration 2

Experimental design

Frame Stacking

We implemented frame stacking based on the article from [4] by creating tensors of 4 layers where each layer contains the pixel-wise maximum of every 3rd and 4th frame/image of the current Joust game, shown in Figure 4. Then based on the implementation from [2] we added two additional layers to represent which player was being controlled.

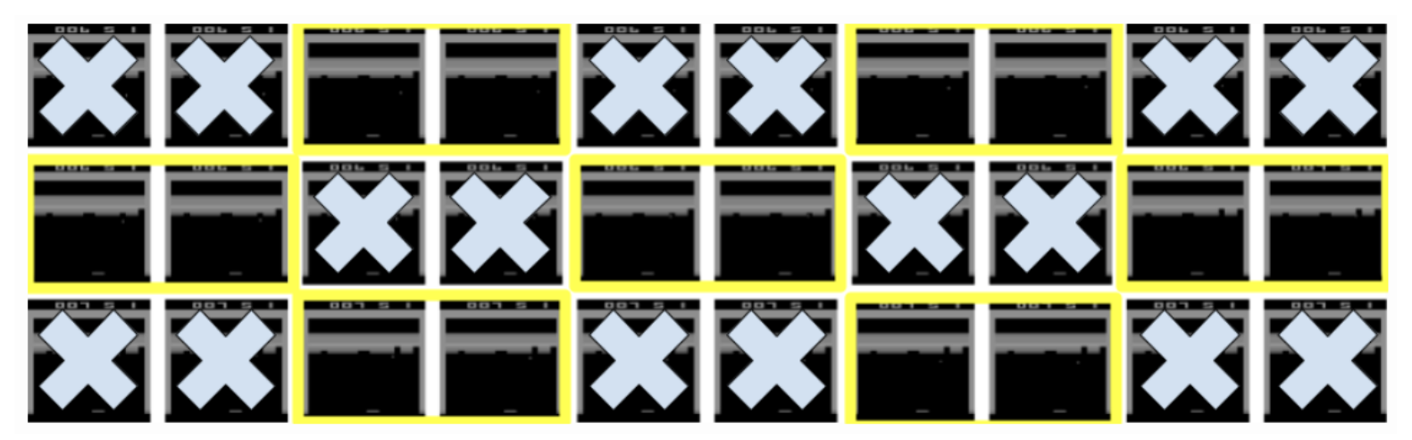


Figure 4:Frame Stacking Visual Demonstration [4]

Evolution Strategy

Our evolutionary strategy starts by generating an initial population of policies to be used in Model 2 and a single randomly generated policy in Model 1. Each individual policy has a mutated policy based on Mutation 1. Since each policy should be able to control the actions of both player 1 and player 2 in the game, the collaborative fitness of the two policies was obtained from playing two games; one where model 1's policy controlled player 1 and model 2's policy controlled player 2 and vice versa. The mean of the scores from both games was used as fitness for the policy in model 2. After this was done for all individuals in the population, the policy in Model 1 was updated using Mutation 2. This process was repeated for the specified number of generations, where for each individual in the population, Model 1 uses the updated policy, and Model 2 uses a combination of Model 1's policy with a randomly generated policy.

Future Works

To build upon this project, we would like to include a virtual batch normalization as it seemed to create more diverse agents for [2] and parralleize the training procedure to experiment more and obtain results faster.

References

[1] Greg Chance.
Joust - atari - atari 2600, 1997 - 1998.

[2] Daan Klijn and A. E. Eiben.
A coevolutionary approach to deep multi-agent reinforcement learning.
2021.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller.
Playing atari with deep reinforcement learning.
2013.

[4] Daniel Seita.
Frame skipping and pre-processing for deep q-networks on atari 2600 games, 2016.