Fox Net: A Deep Learning Agent for Nintendo's Star Fox 64

Kevin Looby | Josh Grinberg | Kat Gregory
Stanford University | CS231N | June 5, 2017

Introduction

Autonomous vehicle navigation is a common problem in the field of artificial intelligence control. Video games provide an excellent framework for testing and evaluating control models, as they represent an encompassing simulation environment that approximates realistic vehicle navigation in repeatable training scenarios. We present a deep learning model that pilots a video game aircraft through a hostile environment, seeking both to eliminate adversarial agents and navigate past obstacles. Even with only minimal amounts of training, our best model yields promising results.

Problem Statement

Develop an AI for Star Fox 64 that can outperform a human player.

Dataset Processing

Framework: In order to simulate a Nintendo 64 game console's hardware, we use an emulator called mupen64plus. The emulator outputs video frames and receives commands from the trained agent via a TCP socket. The framework for enabling this connectivity was graciously provided by Alexander Dewing and Xiaonan Tong.

Dataset: Our labeled dataset comprises 16,000 timesteps of games played by a strong human player. For each timestep, we record a snapshot of the screen, compressed by a factor of 10, along with the corresponding "correct" action (of 7 possible) taken at that timestep. We run this frame through an OCR model and additional processing to extract the score and the agent's health from the upper-left corner.

Figure 1: Sample training image from Star Fox 64.

Approach

Classification: Develop and evaluate four different models with on-policy classification, optimizing for consistency with a human player's actions.
Q-learning: Improve the best-performing of these models, DQN, with off-policy Deep Q-learning.

Evaluation Metric

Classification: Accuracy given labeled data.
Q-learning: Maximum score achieved by the agent after losing three lives.

Method

Classification Goal: Predict the human's actions.
Pseudocode:

```
model.train(states_train , actions_train)
actions = model.predict(states_eval)
acc = sum(actions == actions_eval / len(actions)
```

Q-Learning

Goal: Improve a game-playing policy over time, optimizing for max_score + health (0-10).
Pseudocode:

```
init state , Q
while True:
        Pick action a using e-greedy for state
        Observe reward r, new state s'
        target = r + max_a' Q(s',a')
        loss = 0.99 (target - Q(s,a))^2
```

Models

Layer	Number of Nodes
Affine ReLU	1024
Affine	7

Two-layer fully-connected neural network architecture

Layer	Num Nodes	Kernel Size	Other
Conv-ReLU	32	7	-
Batch norm	-	-	-
Conv-ReLU	32	9	-
Batch norm	-	-	-
Pool	-	2	Stride 2
Affine-ReLU	1024	-	-
Dropout	-	-	Rate 0.5
Affine	7	-	-

Simple CNN architecture

Results: Comparing Models

Figure 2 compares the accuracies of various models trained offline to perform classification:

Model	Train	Validation
Linear	*64.1*	*41*
Fox CNN		
DeepMind DQN	*98.6*	
DeepMind DQN Multi-Frame		

Figure 2: Accuracies of four models after offline training.

As illustrated in Figure 3, our classification models demonstrate a smooth increase in training accuracy and decrease in training loss over 20 epochs of training. This means that they accurately learn to mimic human players - but offline training alone is unlikely to create an agent that can outperform a human.

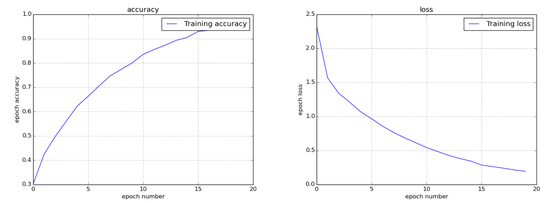

Figure X: Classification accuracy and loss for Fox CNN.

The saliency maps in Figure 4 indicate that our models will require more training time to identify the most relevant features of a given situation - right now, their only insight is to prioritize the center of the screen.

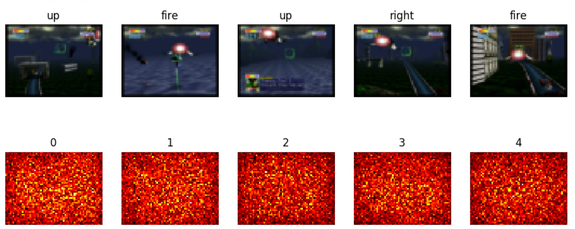

Figure 4: Saliency maps from Fox CNN.

Results: Online DQN

Online results: Our online training yields promising results. Figure 5 charts a training period where a model learned from a human player's choices for 210 time steps as a warm start, and then began to play on its own. The agent only achieves 4 points before it dies on its first solo game; however, its score on its second attempt surpasses that of the human who trained it. High loss values indicate time steps at which the model makes a large update, such as when the agent lost a life at time step 210 and receiving a reward of -100.

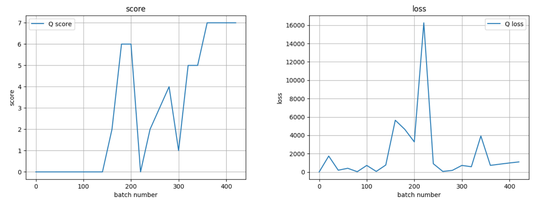

Figure 5: Online score and loss.

Conclusion

Although more training time is required to explore the full extent of its potential, our initial results suggest that online Deep Q-Learning can yield promising results.

Future Work

- Increase training time.
- Add DQN Target Network to increase stability.
- Demonstrate that each online Q-learning enhancement independently improves results.
- Optimize hyperparameters for classification and online Q-learning.
- Decrease exploration over time by decaying the value of epsilon in e-greedy.