# Training an Agent to Complete Contra(NES) with Reinforcement Learning

Evan Kender
University of Oregon
1585 E 13th Ave. Eugene, OR 97403
ekender@uoregon.edu

## Abstract

*In this paper, we look at using Reinforcement Learning (RL) to train an agent to complete the first level of the NES game Contra. In particular, this study will focus on using Proximal Policy Optimization (PPO) to train the agent. It took over 5,000,000 training steps for the agent to complete the level for the first time, however, due to the difficulty of the game, the completion accuracy remains extremely low for millions of more steps.*

## 1. Introduction

The use of Machine Learning (ML) and Artificial Intelligence (AI) on video games has been very common for a while. Whether it be to train an AI to play against a real person [1], to find a way to achieve a new world record [2], or just to see if it can be better than a professional player [3]. Many ML algorithms are even benchmarked on Atari games [4] due to their simplistic environment and action space. My motivation for this project originally came from seeing a popular Youtuber use ML to complete a level of Super Mario World in 2015 [5]. However, since then many people have shown success with training agents on Super Mario World, leading me to choose a different game, Contra. From what I found only two people have publicly attempted Contra, one was successful [6] while the second was not [7]. This along with the fact that there was already an environment for Contra [8] were the main motivations for this project. My original approach for this project was inspired by Andrew Grebenisan and his success using Double Deep Q-Network (DDQN) with Super Mario World [9]. This however quickly showed poor results which led me to follow uvipen's approach [6] using Proximal Policy Optimization (PPO) instead.

## 2. Background

### 2.1. Game Mechanics and Objective

Contra for NES is not considered an easy game, and this is true for even the first level of the game. In total, the game has eight different levels each with a respective boss, and each increasing in difficulty. The objective of the game is the complete each level in order without dying, the player starts with just three lives, and a single bullet or contact with an enemy will cause you to lose a life. When the first two lives are lost the character is respawned in the top left with a short amount of invincibility, when the third life is lost it game over. To complete the first level the player must move towards to right of the screen dodging or shooting enemies in the way. Once the end of the level is reached the player is presented with a boss that they must defeat to continue to the next level. The boss for the first level is static while the fires out many projectiles and has a small hit box where you can damage it. The player must land 32 shots on the first boss to defeat it.

To move through the level the player has four movement directions LEFT, RIGHT, UP, DOWN, and two button inputs A and B which are jump and fire respectively. These inputs can be combined for example A RIGHT or A B RIGHT DOWN leading to 20 different possible inputs. There are also power-ups throughout the level that give the player different weapons, the first is a machine gun which has large bullets and a faster fire rate, the second is a fire weapon that leaves a spiraling path of bullets, the third is a laser weapon that shoots a short distance straight beam, the final and most powerful is the scattergun which fires 5 large bullets spread over the screen. For mediocre players like myself, the scattergun is almost necessary to complete even the first level consistently.

### 2.2. OpenAI Gym and Gym Retro

OpenAI [10] is an open source Python library original designed to tackle two problems in RL research. The first being the need for better bench marks, while in supervised learning (SL) we have large benchmark datasets like Ima-

geNet [11] but before OpenAI there was not a easily accessible large set of environments for RL. The second problem being the lack of standardization between environments making it hard to benchmark the same model across different environment, or a different model on the same environment. OpenAI provides a standard API to communicate between learning algorithms and the environment, as well as a large collection of environments for training.

Gym Retro builds off of OpenAI's API but adds new games from the Sega Genesis, Sega Master System, Nintendo's NES, SNES, and Game Boy consoles. They include over 1000 different games with complete rewards systems for use in research RL algorithms. They also include a User Interface (UI) integration tool for integrating classic games into retro gym environments. This tool allows you to set custom rewards, end scenarios, and search and save ram registers that store important game information like lives or score. These ram registers are essential in creating a reward system because without them you would have no feedback and you agent wouldn't have anything to learn from.

### 2.3. Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a policy Gradient RL algorithm inspired by trust region policy optimization (TRPO) but designed to be more general and simplistic [12] and only using first-order optimization. A policy is simply a mapping from action to state space, or in Contra this would be the possible inputs and a certain position in the level. We evaluate an agent based on its performance while following a certain policy and this is where the policy gradient plays a role, when the agent doesn't know what actions to choose to get the best result it calculates the policy gradient. While in some algorithms this policy could lead to massive deviations PPO limits this change by using a clipped surrogate.

$$\mathrm{L}^{CLIP}(\theta) = \hat{E}_{\mathrm{t}}[min(r_{\mathrm{t}}(\theta)\hat{A}_{\mathrm{t}}, clip(r_{\mathrm{t}}(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\mathrm{t}})]$$

$\theta$ is the policy parameter

$\hat{E}_{\mathrm{t}}$ denotes the Empirical exception over time steps

$\hat{A}_{\mathrm{t}}$ is the estimates advantage at time t

$r_{\mathrm{t}}$ denotes the ratio of probability between the new and old policy

$\epsilon$ is a hyper-parameter typically 0.1 or 0.2 which denotes the limit of the range within an update is allowed

**Algorithm 1** PPO

**for** iteration=1,2,... **do**:
  **for** iteration=1,2,...N **do**:
    Run policy $\pi_{\theta\text{old}}$ in environment for T time steps
    Compute average estimates $\hat{A}_1$ ,...., $\hat{A}_{\mathrm{t}}$
  **end for**
  Optimize surrogate L wrt $\theta$, with K epochs and batch size M $\geq$ NT
  $\theta_{\text{old}} \leftarrow \theta$
**end for**

## 3. Approach

### 3.1. Environment

The original plan for the environment was to use the already existing Gym-Contra [8] package which has done all of the work for the rewards and scenarios in Contra allowing you to plug it straight into a model. This did not work out however, the reward and scenario system was not robust enough to train the agent to complete the level, in this testing. The only rewards this package implement were for moving to the right, gaining score, and beating the boss, it also had penalties for dieing or losing the game. The problem that was ran into with this package was while it could occasionally make it to the boss it had no reward for damaging the boss only for beating it. This led to the model occasionally landing shots on the boss, but mostly just dying right away or standing in the corner. So now the option was to using Retro Gyms User Interface (UI) tool to create the reward and done conditions completely from scratch.

To create the reward system from scratch Retro Gym UI has a search feature that allows a user to search for specific values and changes in values that can be monitored while playing through this level. Retro Gym provide some great tips from finding these variables.

General

To define these, you find variables from the game's memory, such as the player's current score and lives remaining, and use those to create the done condition and reward function. An example done condition is when the lives variable is equal to 0, an example reward function is the change in the score variable.

Score

Score occasionally is stored in individual locations — e.g. if the score displayed is 123400, 1, 2, 3, 4, 0, 0 all will update separately. If the score is broken into multiple variables, make sure you have penalties set for the individual digits (such as BOB-Snes). A number of games will update the

score value across multiple frames, in this case you will need a lua script to correct the reward, such as 1942-Nes.

Done Condition

In defining a game over variable, look for a binary value that switches between 0 and 1 – 0 when the game is in play, 1 when the game is over. And make sure to test it by playing a few consecutive levels.

Once you have a functional reward system you can apply wrappers to the environment many useful ones are include in Retro Gym. The most important wrapper for this project was used to change the possible actions the agent could perform. This was used to reduce the possible inputs from twenty down to ten.

Remaining Possible Inputs

[B] [LEFT B] [DOWN B] [A B] [RIGHT] [RIGHT A] [RIGHT B] [RIGHT A B] [RIGHT UP B] [RIGHT DOWN B]

As you can see almost all the possible left movements are removed, this is because the agent rarely need to move left to complete the level and it will only increase the complexity of the action space and training time if not removed.

Another wrapper that was used on the environment was a frame skip wrapper. This speeds up training by replaying every steps actions over the number of desired frames skipped, in this project we used a frame skip of 2.

### 3.2. Rewards

While attempting to train the agent to complete the game the hyper-parameters that were focused on first were the rewards. As mentioned above the orginal reward system came directly from Contra-Gym [8]

| | |
|---|---|
| Score | +$\delta$Score |
| Xpos | +$\delta$Xpos |
| Life Loss | -20 |
| Game Over | -50 |
| Win | +100 |

The next reward system that was implemented using the Retro Gym Integration UI was based off uvipen's reward [6] which he used to successfully complete the first level.

| | |
|---|---|
| Score | +min(max(($\delta$Score), 0), 2) |
| Xpos | +min(max(($\delta$Xpos), -3), 3) |
| Life Loss | -15 |
| Game Over | -35 |
| Win | +50 |

While unvipen was able to complete the first level with this reward system the model was still getting stuck at the boss as there was no reward for damaging the boss, which led to the next reward system.

| | |
|---|---|
| Score | +min(max(($\delta$Score), 0), 2) |
| Xpos | +min(max(($\delta$Xpos), -3), 3 |
| Boss | +(5*$\delta$Boss)+!$\delta$Boss |
| Life Loss | -15 |
| Game Over | -35 |
| Win | +1000 |

The above reward system was successful at the training the model to beat the first level, however the completion accuracy remained extremely low due to the difficultly of beating the boss with the default gun. This inspired creating a reward system to rewarded the agent for picking up the scatter gun. This is the final reward system that the model was trained on.

| | |
|---|---|
| Score | +min(max(($\delta$Score), 0), 2) |
| Xpos | +min(max(($\delta$Xpos), -3), 3 |
| Scatter | +10 for picking up the powerup |
| ScatterLoss | -15 for losing scatter without dying |
| Boss | +(5*$\delta$Boss)+!$\delta$Boss |
| Life Loss | -15 |
| Game Over | -35 |
| Win | +1000 |

Scatterloss is there to penalize the agent from replacing the scattergun with another power up

### 3.3. Model

The first model used in the attempts to train an agent was a Double Deep Q Network (DDQN) based off of Andrew Grebenisan [8] but slightly adapted to work with Contra environment, since this was unsuccessful and was quickly replaced with PPO this paper will focus on that instead.

The PPO implementation in this paper came from the StableBaselines3 [13] Python package. This package includes numerous different RL algorithms as well as tools for modifying the environment, modifying training, and more. While other packages include the PPO model StableBaselines3 is based around OpenAI allowing the custom Contra environment to be integrated easily for training and testing. They also have TensorBoard integration as well which allows monitoring the training graphs live, this proved extremely useful in modifying the reward system and other hyper parameters.

From the Stablebaseline PPO page

The Proximal Policy Optimization algorithm combines ideas from A2C (having multiple workers) and TRPO (it uses a trust region to improve

the actor).The main idea is that after an update, the new policy should be not too far form the old policy. For that, ppo uses clipping to avoid too large update.

### 3.4. Hyperparamters

The reward system is certainly crucial in training a successful model using RL the other hyper parameters are crucial as well. The model was run on StableBaseline3's defaults hyper parameters until the final reward system was in place. Those parameters are below.

| | |
|---|---|
| Learning Rate | .0003 |
| N_Steps | 2048 |
| Batch Size | 64 |
| Gamma | .99 |
| Entropy Cof. | 0.0 |
| Clip Range | .2 |
| N_Epochs | 10 |
| Gae Lambda | .95 |
| Max Grad Norm | .5 |
| VF Cof. | .5 |

These hyper parameters were not effective at training the agent to complete the first level, which led to needing to try many different variations of different hyper parameters. There are a many different ways to test hyper parameters but in this study Optuna [14] was used. At first a problem was encountered trying to use Optuna with a custom Retro environments, while Optuna does work with OpenAI environments Retro ones are slightly different. Luckily others have encountered this issue before and already had developed a solution [15]. After running Optuna on 100 trials with differing hyper parameters the best one was selected and those are below.

| | |
|---|---|
| Learning Rate | 1.2e-05 |
| N_Steps | 2048 |
| Batch Size | 512 |
| Gamma | .98 |
| Entropy Cof. | .045 |
| Clip Range | .1 |
| N_Epochs | 10 |
| Gae Lambda | .95 |
| Max Grad Norm | .7 |
| VF Cof. | .942 |

By comparing the two tables the only hyper parameters that remained the same were N_Steps at 2048, Gae Lambda at .95, and N_Epochs at 10

## 4. Results

After training the model for 8,000,000 steps the agent is able to complete the level for the first time, however the win accuracy remains extremely low even up to 20,000,000 steps. In early training around 3,000,000 steps the agent is making it to the end of the level [Fig. 1] fairly consistently but is only able to land a few shots on the boss before getting reset. From this point on the consistently reaching the boss increases as well as the damage done to boss. While to agent can consistently make it to the boss there are a few places where it tends to die figure 2-5 show common spots where the model dies. The first two figures show pits that the agent got stuck at during earlier training, but still fall victim too at latter stages. Figures 3-4 show two other spots where the agent commonly dies, as you can see the same turret enemy is present in both, which is the cause of the death.

We can see how the average length of an episode increases and decreases throughout training in Figure 6. The length starts high but quickly decreases, this is because the model is spending time standing still at the start of the episode or at some point throughout the level. As training continues the agent no longer hangs around the starts but there are still some major increases in episode length due to getting stuck somewhere else, the most notable being at 12million steps, the length drastically increases from 1900 to 2200 in a few 100,000 steps [Figure 7]. While the agent did get stuck it quickly resolved itself and got back on track in around 800,000 steps.

Figure 8 shows the average reward per episode which starts extremely low but slowly increases as the agent learns to consistently move farther throughout the level while losing fewer lives. We can see that up until around 6million steps the average reward increases fairly slowly, this is due to the agent not reaching the boss consistently and also not damaging the boss consistently. After 6million steps however the reward see a sharp increase until around 10million steps where the increase begin to slow again. This behavior is due to the agent constantly reaching the end but not damaging it much or at all, but as training progress the agent learning to consistently land more and more shots on the boss increasing its reward.

Figure 9 and 10 don't show the models performance but rather how the models parameters change throughout training. In Figure 9 we see the entropy loss, which increases as the reward does. This means the as the average reward increase the entropy decreases meaning the models probability of choosing a new action is decreased. This is not a strict relationship, we can see that the entropy has way more variance then the mean reward, which is good as it means the model is still trying new things despite increasing average reward.

In figure 10 we see the clip fraction which also increases as the model learns and the average reward increase. The

clip fraction is how much of the old policy to include and since this is increasing it means the amount of the previous policy kept increases as well, which is crucial in the model not making drastic changes later in training.

Figure 1. Boss

Figure 3. Pit 2

Figure 4. Turret 1

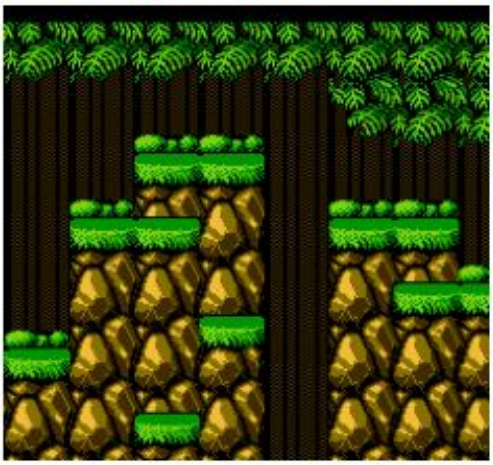

Figure 2. Pit 1

Figure 5. Turret 2

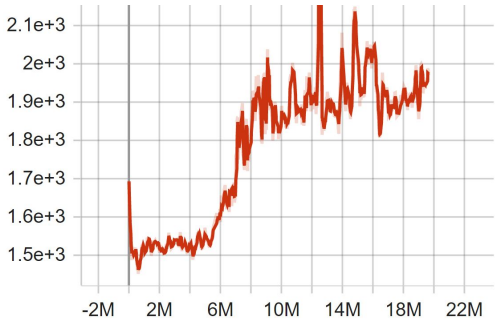


Figure 6. Episode Mean Length

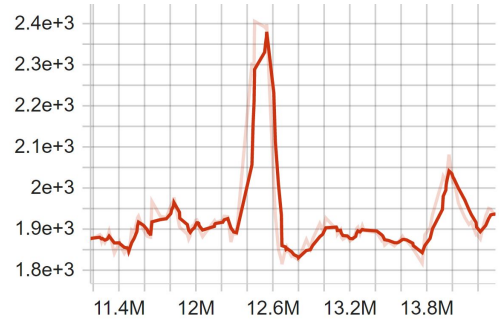


Figure 7. Episode Mean Length Focused on 12-13 million steps

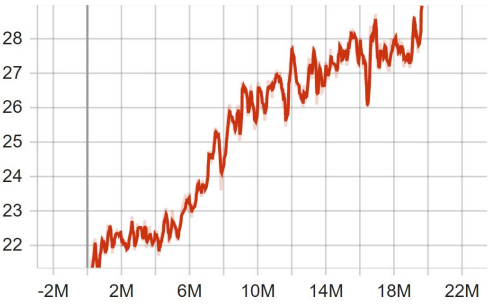


Figure 8. Episode Mean Reward

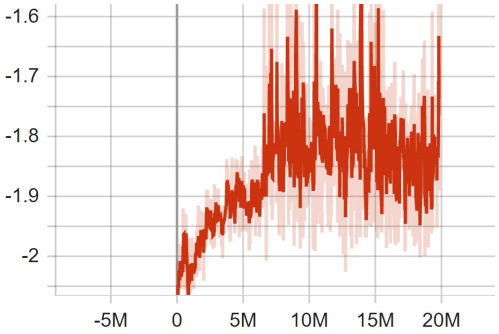


Figure 9. Entropy Loss

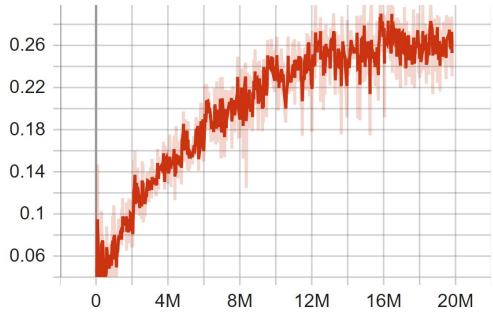


Figure 10. Clip Fraction

## 5. Conclusion

In this project we looked at training an agent using RL to complete the first level of the NES game Contra. While there were many different RL algorithms that could have been chosen we used PPO due to its proven success with Contra as well as its simplicity. In training the model we ran into several hurtles that prevented the agent more consistently completing the game. The first of these hurtles came from the orginal model choice DDQN, while this model has shown success with Super Mario World [8] the same can not be said for Contra. By switching from DDQN to PPO the models training progressed more but still struggled to beat the boss.

To be able to train the agent agent to successfully complete the game took tuning the hyper parameters firstly was the reward system. We started off with a very simply reward system which wasn't enough to train the agent. By adding the boss's health to the reward function we were able to train the agent to be able to damage the boss consistently but not beat it. When we tuned the models hyper parameters like learning rate, clip range, ects. the agent was finally able complete the boss once. The completion consistency still remained extremely low so we tuned the reward system more by adding reward for picking up the scatter gun. This increased the completion consistently however the improvement was very slim, and at 20million training steps the agents completion accuracy is still much lower than I had hope.

In the future I would like to continue training and tuning the hyper parameters until the agent can consistently (>50%) complete the game. Some possible steps to achieve this are below.

- Continue training up to 100million steps or completion accuracy is (>50%) with the current hyper parameters
- If the agent is able to consistently complete the first level then attempt to train the agent on the next 7 levels
- Try adding a time penalty to encourage the agent to beat the level faster

## References


[1] Sebastian Risi Mads Johansen, Martin Pichlmair. ”video game description language environment for unity machine learning agents”, 2019. https://ieeexplore.ieee.org/abstract/document/8490422/. 1

[2] James Vincent. ”a video game-playing ai beat q*bert in a way no one’s ever seen before”, 2018. https://www.theverge.com/tldr/2018/2/28/17062338/ai-agent-atari-q-bert-cracked-bug-cheat. 1

[3] ”openai five”. https://openai.com/five/. 1

[4] ”playing atari with deep reinforcement learning ”. https://paperswithcode.com/task/atari-games. 1

[5] SethBling. ”mari/o - machine learning for video games”, 2015. https://www.youtube.com/watch?v=qv6UVOQ0F44. 1

[6] uvipen. ”proximal policy optimization (ppo) for contra nes”, 2021. https://github.com/uvipen/Contra-PPO-pytorch. 1, 3

[7] twistronics. ”an attempt to playing contra with machine learning”, 2016. https://blog.old.standwith-ukraine.me/blogs/an-attempt-to-playing-contra-with-machine-learning/. 1

[8] ”gym-contra”. https://pypi.org/project/gym-contra/. 1, 2, 3, 6

[9] Andrew Grebenisan. ”play super mario bros with a double deep q-network”, 2019. https://blog.paperspace.com/building-double-deep-q-network-super-mario-bros/. 1

[10] ”open ai gym”. https://gym.openai.com/. 1

[11] ”imagenet”. https://image-net.org/. 2

[12] Prafulla Dhariwal John Schulman, Filip Wolski. ”proximal policy optimization algorithms”, 2017. https://arxiv.org/abs/1707.06347. 2

[13] ”stablebaselines3”. https://stable-baselines3.readthedocs.io/en/master/. 3

[14] ”optuna”. https://optuna.org/. 4

[15] ”retro optuna”. https://github.com/araffin/rl-baselines-zoo/issues/29. 4