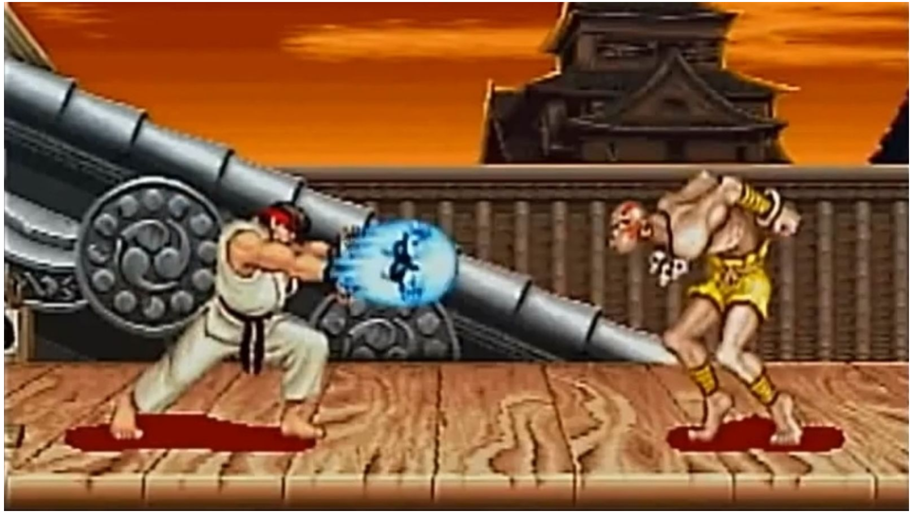
Reinforcement Learning for Street Fighter II

By Sunny Chen, Anis Chihoub, and Stanley Chou

Introduction

- Street Fighter II is a classic arcade game that has led to various grassroots tournaments.
- Known for having a very tough AI to beat.
- ROM is directly compatible with OpenAI gym.



Can we train an AI to beat Street Fighter II?

Deep Q Learning (DQN)

- Approximate a Q function to represent the expected future rewards for all state action pairs
- Policy is inferred from the Q function as $\pi^*(s) = \text{argmax } Q(s, a)$
- Rather old:
  - Unstable: shows drastic decline in performance
  - Very slow to train: may take multiple hours to days to train a model
- Requires that we know what the “best action” is.
  - Not always an objective way to determine the best action.

Proximal Policy Optimization (PPO)

- Developed by OpenAI in 2017
- Policy gradient family
  - Directly approximates policy function, not Q function
  - Stochastic, not deterministic (better for us)
  - Originally extremely slow
- PPO is improvement on previous policy gradient methods
- Policy needs to be constrained to “trust region”, limit the amount that policy will change each time step
- This constraint is expensive to compute, use score penalty as soft constraint

Methodology (Hyper Parameter Tuning)

- For PPO
- 150 Trials, 100,000 time steps
- 5 Variables
  - Learning Rate
  - N_steps
  - Gamma
  - Clip_range
  - Gae_lambda
- Test each set on 10 episodes
- Keep 2 Best Hyper Parameter Sets

```
    #env.close()
    #try:
    model_params = optimize_ppo(trial)

    # Create environment
    env = StreetFighter()
    #Monitor allows us to extract reward and other information from the game in a vectorized form
    env = Monitor(env, LOG_DIR)
    env = DummyVecEnv([lambda: env])
    #frame stack of size 4 is the standard for atari games helps you consider velocity of objects and other things that
    # 1 frame difference cannot observe like we did in our DQN implementation
    env = VecFrameStack(env, 4, channels_order='last')

    # Create algo
    #PPO using CNNs , takes the game environment, logs information to the LOG_DIR , lot's of trials so we don't want prints, and
    model = PPO('CnnPolicy', env, tensorboard_log=LOG_DIR, verbose=0, **model_params)
    #short training time because we need results fast
    model.learn(total_timesteps = 75000)
    #model.learn(total_timesteps=100000)

    # Evaluate model
    # evaluate the model on 5 games of Street Fighter
    mean_reward, _ = evaluate_policy(model, env, n_eval_episodes=10)
    env.close()

    #Saves the best hyper parameters the ones that created the best model for full training later
    SAVE_PATH = os.path.join(OPT_DIR, 'trial_{}_best_model'.format(trial.number))
    model.save(SAVE_PATH)

    return mean_reward
    #Any errors we don't want the hyper parameters to crash and waste training time so -1000 just let's us skip parameters that
    #are generated by Optuna that cause errors
    #except Exception as e:
        #return -1000
```

```
In [170]: # Creating the experiment
# creates the optuna automated hyper parameter testing experiment
study = optuna.create_study(direction='maximize')
# n_trials means we are only trying 10 sets of values in that range, realistically we would try more sets and train longer
#but we simply lack the time to do that currently
study.optimize(optimize_agent, n_trials = 50, n_jobs=1)
```

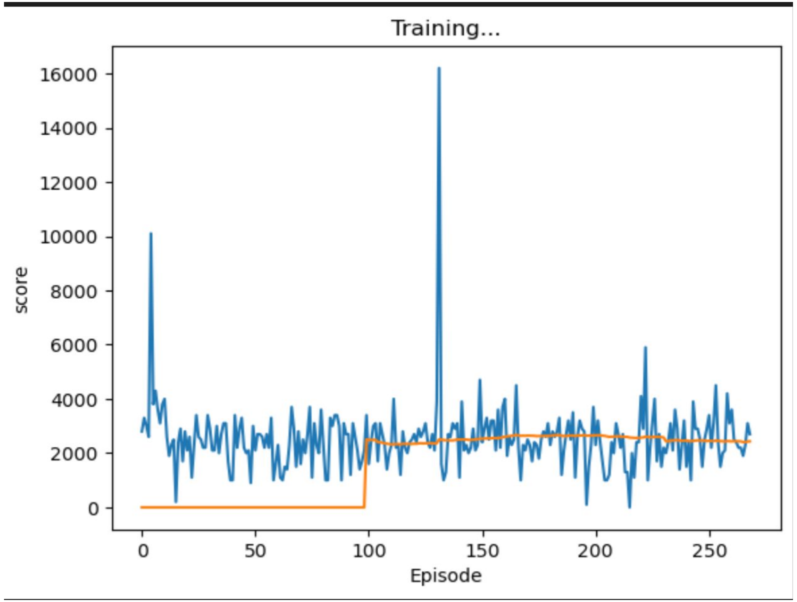
Methodology

- DQN:
  - Trained it for 1000 Episodes, and a Data_Memory of 1,000,000
  - Stopped due to Poor Performance
    - 250 Episodes ~8 hours
- PPO
  - Trained in 10 million episode blocks
    - Time ~10 Hours
  - Hyper parameter tuning
  - Performed Well

```
In [44]: #create model with these weights
         model = PPO('CnnPolicy', env, tensorboard_log=LOG_DIR, verbose=1, **model_params)
         #load up the best model taht was trained for some time already
         #model.load(os.path.join(OPT_DIR, 'best_model_2010000.zip'))
         model.load(os.path.join(CHECKPOINT_DIR, 'best_model_2010000.zip'))

         Using cuda device
         Wrapping the env in a VecTransposeImage.

Out[44]: <stable_baselines3.ppo.ppo.PPO at 0x1c880028430>

In [45]: model.learn(total_timesteps=10000000, callback=callback)
```

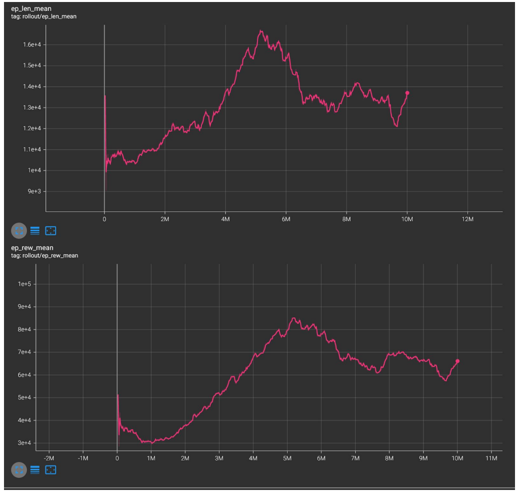
Results (DQN)

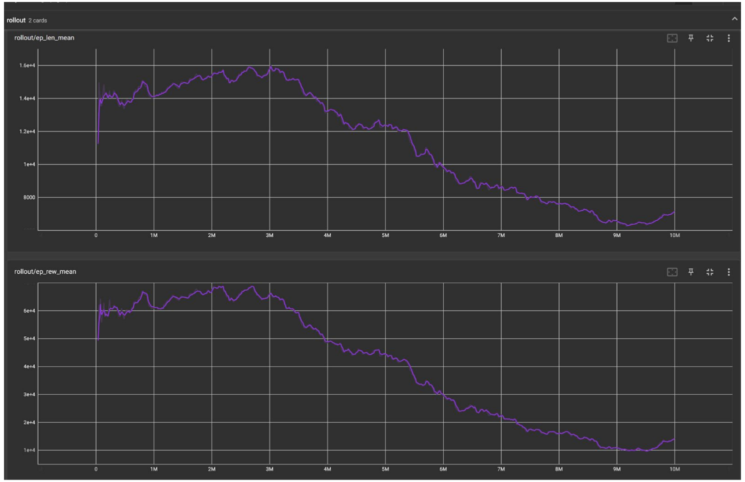
- Unstable results
- Poor Learning
- Took 8 hours for 250 Episodes



Results (PPO)

- Great Results
  - The Collapse is likely due to Learning Rate being too high
  - Learning Rate was in the 2.75e-7
  - 2.75e-5 caused unstable poor results
- 10 Hours Computation Time





Demo

Before Training

After Training

Future Work

- Specialized strategies to beat characters.
    - In real life, fighters adopt specific strategies to beat other fighters.
- Include color information instead of jumping to grayscale.
    - Allows agent to identify different projectile attacks and create a concrete strategy.
- Train against itself or humans
    - Single player AI may not be reflective of human competitive play

Citations

1. Yu-Jhe Li, Hsin-Yu Chang, Yu-Jing Lin, Po-Wei Wu, Yu-Chiang Frank Wang: “Deep Reinforcement Learning for Playing 2.5D Fighting Games”, 2018; [http://arxiv.org/abs/1805.02070 arXiv:1805.02070].
2. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller: “Playing Atari with Deep Reinforcement Learning”, 2013; [http://arxiv.org/abs/1312.5602 arXiv:1312.5602].
3. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov: “Proximal Policy Optimization Algorithms”, 2017; [http://arxiv.org/abs/1707.06347 arXiv:1707.06347].

RUTGERS
UNIVERSITY