Reinforcement Learning - Pong, breakout

D. Nandadeep[a] and C. Merlin[b]

(Dated: 19 March 2018)

Keywords: Atari, Reinforcement Learning, Neural Networks

I. INTRODUCTION

When it comes to artificial intelligence, one of the most commonly thought of uses is robotics. Decades of books and movies with cybernetic intelligent machines have led to countless and expensive failures in programming bipedal robotic motion. But in the past few years, new advances in machine learning have led to alternative approaches in motion that could lead to more advanced behavior with significantly less expense and programming time. Instead of prideful scientists believing they have the ability to program movement in a non-programmatic world, allow the robots to learn the way humans learn.

A. Motivation

As babies, we don't learn about motion by having it explained to us, we learn through trial and error. We learn through reward and punishment. Our reward is the satisfaction of successfully taking our first step, our punishment is falling flat on our face. Reinforcement learning takes this same behaviourist psychology to teach computer algorithms how to perform. In an attempt to visualize this learning process we have chosen to implement reinforcement learning algorithms with Atari video games. Video games are a great tool to not only visualize this process of success and failures, but the rewards can be directly linked to the scoring within the game. Using Elon Musk's OpenAI gym, a tool kit developed specifically to practice developing reinforcement algorithms, we can easily focus on programming intelligent agents without worrying about the environment. Being that these are our first experiments with reinforcement learning, we have chosen two games with fairly simple behavior, Pong and Breakout. Both games consist of bouncing a ball off of a paddle, thus our algorithm only has to determine the probability of moving one direction or the other. Andrej Karpathy has written a great article called "Pong from Pixels", that explains this complex algorithm in simple and understandable terms.

II. APPROACH

In the subsequent sections, we define a series of methods adopted straight from the Deep Learning workshop conducted by Andrej Karpathy and also propose some changes or potential improvements which were left to creativity during the workshop. These following sections describe our source of input data, the Neural Network architecture in the base implementation, heuristics from external sources like DeepMind, how we preprocess the input for a better fit, training techniques used.

III. PONG EXPERIMENT

A. Preprocessing

Ideally we'd want to feed at least 2 frames to the policy network so that it can detect motion. To make things a bit simpler we do a tiny bit of preprocessing, e.g. we will actually feed difference frames to the network (i.e. subtraction of current and last frame). We get 100,800 numbers (210*160*3) and forward our policy network (which easily involves on order of a million parameters in W1 and W2). Suppose that we decide to go UP. The game might respond that we get 0 reward this time step and gives us another 100,800 numbers for the next frame. We could repeat this process for hundred timesteps before we get any non-zero reward! E.g. suppose we finally get a +1. Thats great, but how can we tell what made that happen? Was it something we did just now? Or maybe 76 frames ago? Or maybe it had something to do with frame 10 and then frame 90? And how do we figure out which of the million knobs to change and how, in order to do better in the future? This is the *credit assignment problem*. Please see FIG 1 and FIG 2 for the visualization.

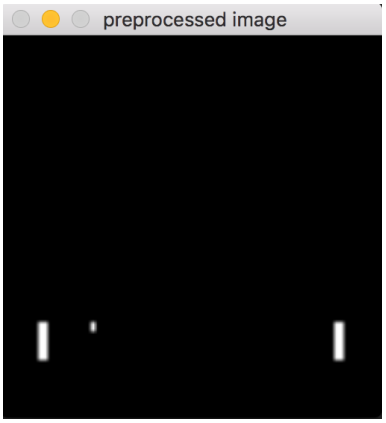

FIG. 1: Image frame during the preprocessing stage

[a] Electronic mail: davuluru@pdx.edu
[b] Electronic mail: mpc6@pdx.edu

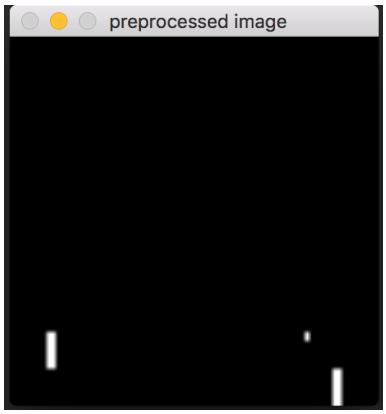

FIG. 2: Image frame during the preprocessing stage

B. Training

Dealing with metrics, the model has a single hidden layer of neurons and we use Xavier initialization of weights as starting point. The network is designed to simulate supervised learning by rolling out a bunch of policies initially to obtain what we call, a fake label for each these runs. If we encounter a win, we expect the network to maximize that pattern in the future. At this point, we have a number of labelled winning moves and losing moves. These initial runs are experimental with respect to the training but create a foundation as to how the network progresses. This progression is described in Fig. 4. We feed forward the bare minimum - raw 80x80 image pixels, fully connected to the 200 hidden neurons using a rectifying linear unit activation function. Finally, we squash the output layer using a sigmoid probability that "predicts" a suitable action for the paddle to either go up or down. The action derived out of the described policy is assigned a reward - if the ball crosses the paddle, we lose, hence a negative reward and positive if otherwise. The error rates are then backpropagated using RMSprop in batches of 10. We hope that tuning the hyperparameters - learning rate, gamma, batch size, hidden units, we hope to achieve/tune the model towards better accuracy, or in our case, winning more games in the future.

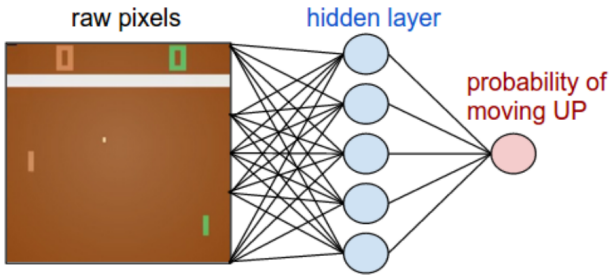

FIG. 3: Our policy network is a 2-layer fully-connected net. It takes an 80x80 input frame and fully connects set of 200 neurons in the hidden layer

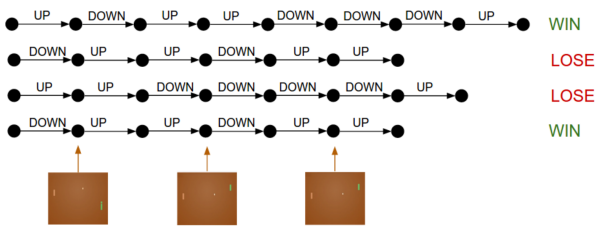

FIG. 4: Cartoon diagram of 4 games. Each black circle is some game state (three example states are visualized on the bottom), and each arrow is a transition, annotated with the action that was sampled. In this case we won 2 games and lost 2 games. With Policy Gradients we would take the two games we won and slightly encourage every single action we made in that episode. Conversely, we would also take the two games we lost and slightly discourage every single action we made in that episode.

IV. OBSERVATIONS

We believe that our observations throughout the experiment was relying on how good the trained model pans out against the hard-coded agent. As an additional visualization, we added a plot of the reward mean over the episodes during training to showcase that the agent actually got better over a period of time.

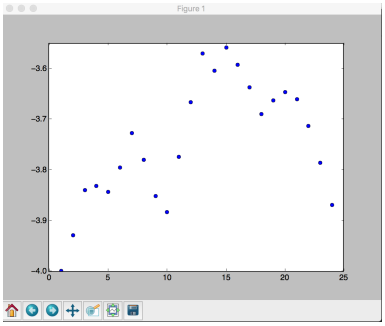

FIG. 5: A plot showing the mean running reward of the agent as the number of episodes increase

FIG. 6: A plot showing the mean running reward of the agent as the number of episodes increase

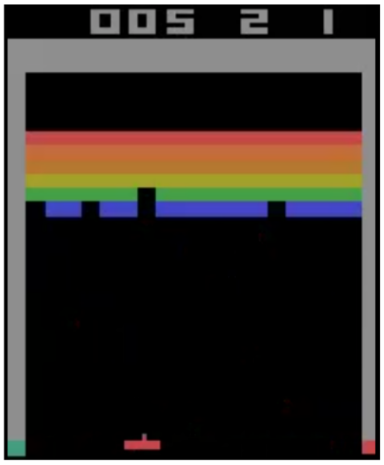

FIG. 7: Atari Breakout environment

V. BREAKOUT EXPERIMENT

A. Approach

After reading the DeepMind Atari reinforcement learning paper and several other more recent interpretations of it, it was clear that the multi-layer perceptron network used for Pong was not enough to learn how to play Breakout. So we went out to recreate the results that DeepMind achieved.

B. Features

1. The Model

The first step was recreating their deep convolutional network. We chose the very popular API Keras to create our model. This API is well known for being easy to use and less verbose than TensorFlow, yet uses TensorFlows backend for calculations and GPU support. Keras has become so popular as an instructional and implementation tool that Google has now included the Keras API within their TensorFlow libraries. With a mere seven lines of code, we were able to recreate the DeepMind deep convolutional neural network. The model consists of an input convolutional layer with 32 filters, that is fed into a 2nd convolutional layer of 64 filters, that is fed into a 3rd convolutional layer of 64 filters, that is flattened and fed into a fully connected layer of 512 neurons, which is fed into a fully connected output layer with 3 neurons for Breakouts 3 possible actions, move left, right, or fire the ball.

2. Frame Stacking

The agent is only capable of learning based on one fundamental property, video games are animated. If you're using static frames to make predictions, the agent has no idea if the ball is moving up or down and we don't want it to fly across the screen to hit a ball moving towards the bricks when the ricochet would lead the ball back to the previous location of the paddle. Thankfully, unlike the multi-layer network, a convolutional model allows us to pass in multi-dimensional vectors. Infact a convolutional network can processes three dimensional images. Seeing that color is irrelevant to the agent, we pre-process each frame of the game to not only be a smaller 80x80 image for computational savings, but we convert the images to grayscale since the model can still infer shades without caring about their true color. So, for position we start with a single frame of the game. For movement, we infer the first derivative, velocity, by appending a 2nd frame to our image vector. But even this is barely enough, so we append a third frame so the model can learn from the second derivative, acceleration. And while probably not necessary for Breakout, a fourth frame is added to the image vector. Stacking these frames along the color channel and inputting them into the model allows for predictions based on time series data. Thus, the model can learn from the game's animation.

3. Epsilon Greedy

With the model created and the simple to use OpenAI gym environment, it was very easy to begin training an agent. Since models are stochastically initiated, the first few episodes the agent makes erratic and uniformed moves until it gets lucky enough to hit the ball by accident. This is when the algorithm receives its first reward and begins to learn how to behave. However, after a few hundred episodes, it happens upon the first more valuable strategy. Hugging the wall allows it to bounce released balls against the bricks in the same location every time, breaking through to deeper layers worth more points. This being the first time it uncovers a strategy that allows it to gain more rewards than clumsily attempting to volley the ball, it believes this to be the best possible strategy. This is what is called a greedy strategy. So we went about implementing one of the most important features of the DeepMind algorithm, epsilon greedy. Epsilon greedy is an algorithm that encourages exploration of the game state while the agent is still fairly uneducated about the mechanics of the game. Epsilon is a hyper-parameter that is used to determine the probability of each action being the greedy action or a randomly selected action. Early on in the game epsilon is set high, usually at 100% to begin with, and through annealing is decayed over episodes of the game so as the agent's predictions are used more often than random moves as it learns the environment and better long term strategies.

FIG. 8: Agent showcasing greedy strategy to gain points through the corners

4. Experience Replay

Using an online learning method, learning while playing the game, the model spends too much time learning small incremental steps and thus forgets other, more long term information. Models need to be trained stochastically, yet because the state space is so large, we cannot use batch training in the traditional sense. So instead of training in real time, the agent plays the game remembering each state, action, transition and reward. Once this memory is full, a random batch of it is selected, the model is fitted, then resumes playing the game again remembering its experiences.

5. Yet another Model

Even though we have created batches, the state space is so large that it must fit the model for each training example and prediction. But sometimes these predictions are not the best long term strategies and everytime the model is fit it gets closer and closer to converging with these possibly poor short term strategies. So, a second model must be used. This model is used as the target model, a stable value approximator. Each prediction is made by prompting the target model for the most valuable move based on the game's state, while the agent model is the one fitted. This keeps the target model from fluctuating while the agent is learning. Once the batch is completed, the updated weights from the agent model is uploaded into the target model for the next iteration of training.

C. Training

With all of the fundamental features of the DeepMind learning algorithm implemented it's time to get back to training. Unfortunately the convolutional network is deep and computationally expensive and these games have extremely large state spaces, every permutation of every pixels color values, (80x80x255)!. This of course is an NP hard problem, that is why we use neural networks to generalize the value of each state. Thankful, in the case of Breakout there is a much smaller subset of these permutations. The paddle only moves horizontally and the bricks don't move, only the ball traverses the entire state space. Still, with high performance GPUś the learning process can take as much as 10 straight days of training. So, having finished implementing the features of the algorithm just three days prior to the presentation, we've created a smaller more economical version of the model with only two convolutional layers and 256 neurons in the hidden layer. The model might not ultimately perform as well, but it's the only way to achieve any decent results within this limited amount of time.

VI. CONCLUSIONS

All and all we are fairly happy with our results at this point. In Pong we have an agent that has shown the capability of beating a hard-coded computer opponent and with Breakout we've seen a score as high as 19 points. However, neither are state of the art. DeepMind is capable of beating the Pong computer opponent 100% of the time and score 240 points in Breakout. But I think from our results, it's clear that our agents did learn. And for our first attempt at such a complex algorithm, we have learned a lot and look forward to more successful implementations of reinforcement learning in the future.

VII. ACKNOWLEDGEMENTS

We would like to thank our course instructor, Prof. Anthony Rhodes to give us this opportunity to explore our interests along with extending his support to full capacity when needed. Our appreciation definitely goes out to Andrej Karpathy for the wonderful workshop exposing the details of implementation of Policy gradients, networks which made our task a lot easier along with OpenAI, Google/Deepmind and the Atari learning environment.

VIII. REFERENCES

[1]Deepmind Atari - Reinforcement Learning.
https://arxiv.org/pdf/1312.5602v1.pdf
[2]OpenAI gym
https://gym.openai.com/
[3]Andrej karpathyś deep learning workshop,
http://karpathy.github.io/2016/05/31/rl/
[4]ALE,
http://www.jair.org/papers/paper3912.html
[5]Human Level Control Reinforcement Learning,
https://storage.googleapis.com/deepmind-data/assets/papers/DeepMindNature14236Paper.pdf