M. Hunter Klein

Case Studies Final Report: Playing Atari with Deep Reinforcement Learning

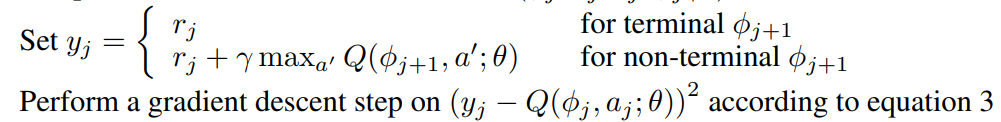
**Introduction**

Deep Mind’s seminal paper *Playing Atari with Deep Reinforcement Learning* by Mnih et al. 2013 takes a new approach to generalizable reinforcement learning. Prior to this paper, most reinforcement learning approaches utilized hand-crafted features which were painstakingly derived for each new application. For a game like Ms. PacMan, this may involve recording the trajectories of each ghost, locations of the remaining dots and super dots, etc.Others that focused on gradient policies struggled to converge for diverse applications. Temporal difference learning stands out as the first model to succeed well at playing backgammon.

Deep Mind’s model aims to take in a general vector input, in this case visual inputs of an Atari game, and develop a policy that performs well in new environments. Using a new technique, they call experience replay, Deep Mind reported performance nearly matching that of a human in many cases and even exceeding it in others. As a testament to the value of their strategy, Deep Mind used the exact same architecture and learning rate for every game with only some tweaks to the reward signal and frame skipping strategies.

**The Model**

Deep Mind’s model combines several strategies shown in previous work. First, they utilize Q-learning which involves learning a function, called the Q function, that estimates the value of a given state, action pair. Q-functions are trained using a combination of current observed rewards r as well as a discounted estimation of future rewards also estimated by the Q function. Secondly, they use a CNN architecture to extract features from the Atari game. The model is not privy to any RAM stored information in the model like the location of certain sprites or the current score of the game. The model is also not privy to the current score of the game. Only a solitary reward signal is provided which is clipped to a 1 for positive reward, -1 for negative reward, and 0 for no reward. As mentioned in the introduction, the truly novel aspect of this method is the utilization of experience replay. States, actions, rewards and state transitions for each step of play are saved into a large array of previous experiences. The Q function is then updated using minibatches randomly sampled from the experience replay. This helps to address the sparsity of the reward signal. The objective function of the model is shown in the snapshot from the source paper below.



Targets for the MSE driven loss function are estimated using the current reward as well as a discounted estimated future reward potential. The idea here is that given a particular action a, the Q function will be updated to maximize current and estimated future rewards. The future rewards are discounted which prevents the model from becoming overly “optimistic” and ignoring the current reward signal.

For my implementation, I chose to use TensorFlow 2 and Keras. To simulate the Atari environment, I used OpenAI’s gym Atari package which is found here: gym.openai.com/envs/#atari. I was unable to find source code for the original paper. Deep Mind’s only published code is for a 2015 update to the method that includes many improvements for training stability – however that specific repo hasn’t been updated since 2017 and many of the default versions of dependent packages have changed and minimal documentation is provided on the site. This left me primarily using *Hands on Machine Learning with Scikit-Learn, Keras and TensorFlow 2* as a reference, however their low-level example only used a linear model with the extremely simple cart-pole problem. Later on, to utilize newer methods that improve training stability, I also used Google’s TFAgents environment (<https://www.tensorflow.org/agents/overview>) which wraps OpenAI’s Atari environments in TensorFlow friendly wrappers and utilizes an abstracted implementation of Deep Q nets. Other articles were used as reference including a helper function for making .gif files of rendered environments with a trained agent. These websites are listed at the end.

**Dataset**

As mentioned earlier, I used environments provided in OpenAI’s gym Atari package as well as the TFAgents wrapper. I trained using environments the environments BreakoutDeterministic-v4, MountainCar-v0, PongDeterministic-v4 and Cart-Pole-v0. The Deterministic argument indicates that the environment has no built in epsilon exploration (taking a random action with probability epsilon). V4 Indicates that 4 frames are skipped between each observation. Neighboring observations are highly correlated with each other, so this helps with training. MountainCar-v0 and Cart-Pole-v0 both originate from the base control package in gym. Neither have built in observation skipping or epsilon exploration. Each environment has a unique number of possible actions. Depending on the environment, the number of output nodes for the Q function was changed to reflect the number of possible actions.

**Training**

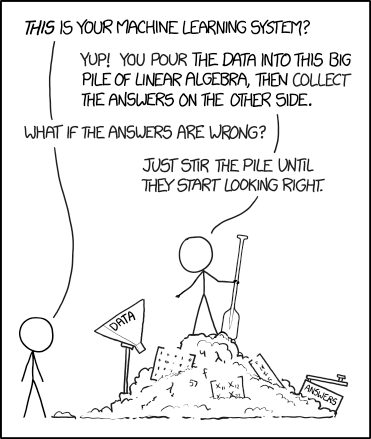
I chose to code up the model from scratch using only the paper as a reference but found that the model was not training well and that I was quickly running out of memory for the experience replay. I was only able to store around 80,000 observations in contrast with Deep Mind’s 1,000,000 frames. Additionally, the paper neglects to mention any of the hyperparameters used in their optimizer, RMSprop, where the effectiveness of RMSprop can vary greatly depending on the learning rate and momentum parameters. The paper also failed to mention how reward signals were tuned for each game. A significant part of the reason why Breakout was failing so terribly was because a negative reward only occurs when you lose the game, and a positive reward only occurs when you win. This means that epsilon exploration never experienced positive reward and failed to learn that losing lives was bad. To get around this, many RL researchers train the model assuming that once a single life is lost, the game is over – however no mention of this was ever made in the paper – nor the structure of the reward signal for any environment for that matter.

With the amount of things I didn’t know that weren’t listed in the paper, I turned to *Hands on ML* for help. Using a linked-list based deque data-structure and by storing the previous frames as unsigned 8-bit ints instead of 32 bit floats, I was able to increase my experience replay size to 320,000 previous steps. I trained the simple Cart-Pole example using only a linear model, and a shorter run time, but otherwise reflecting the core algorithm of Deep Mind’s paper. Loss function plots are rarely helpful in Deep Q Net reinforcement learning. Because the loss function targets and predictions are both estimated using the same Q function, it is possible to simultaneously get microscopic loss while achieving terrible performance. Therefore, the reward function is typically what is used to gauge performance over time:

A screenshot of a cell phone

Description automatically generated

As shown here, the reward signal for each training step can be incredibly volatile. This can be helped by using averaged smoothing, but that is not done here. This plot is left unsmoothed to demonstrate how DQNs can easily fall into local minima to gain good performance and then immediately “forget” a lot of what it learned already. Much of this is due to the objective function “chasing it’s own tail”. This structure can lead to the performance of the model being highly dependent on your random seed for training. It is extremely difficult, and I’d say nearly impossible to verify good training from the training metrics alone. Given it’s volatility and lack of interpretability, to me the 2013 implementation of DQNs by Deep Mind is the epitome of the following xkcd comic:

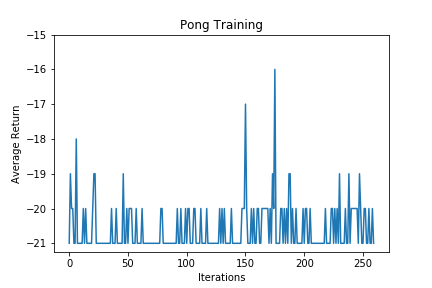


This felt even more true for me as I began training a model for Pong. After training for 3 million iterations, my model was still unable to score even a single point. Regrettably, I was so frustrated with my poor results, I neglected to save the reward plot for this model. However, it’s poor performance is demonstrated in a .gif in the PowerPoint included in appendix B. After 2 days of training, my agent could do no more than crash into the wall of the screen, making only random movements every few seconds due to the 0.01 epsilon parameter.

I theorized that these poor results could be coming from a handful of places. In order of easiest to most difficult I ensured the following: 1st, that the experience replay was being correctly sampled. This was easy enough as I just took multiple samples and ensured that they weren’t identical. 2nd that the model was being passed by pointer and not by reference when using the training function. While I know that pass by pointer is the default for most python frameworks, I didn’t know if the interaction with OpenAI gym had caused anything strange to happen. To test this, I removed helper functions and ran everything in a single loop. Lastly, I theorized that I simply had a bad random seed and that I was getting stuck in a local minimum. Essentially, the model learned that not playing the game was the easiest way to not lose the game. To change this I tried many random seeds and retrained the model nearly a dozen times (each run taking over 24 hours). With no improvement, I turned to updates made in Deep Mind’s 2015 paper *Human Level Control Through Deep Reinforcement Learning* and their Dueling DQN paper from the same year.

Essentially, Deep Mind improved their model by prioritizing high impact samples from the experience replay, making a separate target function from the Q-function for the objective function, and lastly adapting the Q function to estimate the value of the current state and the advantage of a given action separately. These improvements provided added stability to the training where it was less dependent on random seed.

In short, I tested these new methods on the cart pole and on Pong with the results shown on slide 8 of appendix B as .mp4 and .gif files. Training was still hard to follow for Pong. I cancelled training early because it looked like it wasn’t working but was pleasantly surprised to see it working when I watched the video of the best policy. Shown below is average reward signal over thousands of iterations. As explained earlier however, the signal can still be erratic and fails to explain how well the model is doing at surviving and hitting the ball before eventually losing:



**Testing**

Testing was extremely easy and short for this model. Considering how long winded my training section was, I will keep this one brief. Testing each model was as simple as running it through multiple iterations of new environments seeing how it performed. For the Cart-Pole example, performance on new examples was about 50-50 between being perfect and under 50. This would likely stabilize with more training. With more time, I would have provided confidence intervals, but with all my stress trying to get Pong to work, I forgot.

For Pong, I didn’t know it had worked till I already saved the video and closed the model, so regrettably, the .mp4 in Appendix B is the only example I have of Pong working. It just started working on the 29th, 14 hours before the final. If I had more time, I would get plots of its average performance over many episodes in addition to confidence intervals.

**Conclusion**

I’m excited to keep playing around with reinforcement learning in the future. There has been a lot of progress made since 2013, so hopefully some of those methods are a bit less of a headache to train. Overall this was a great experience though.

THANKS SO MUCH FOR AN AWESOME CLASS!!!

**References**

-<https://towardsdatascience.com/tutorial-double-deep-q-learning-with-dueling-network-architectures-4c1b3fb7f756>

-<https://becominghuman.ai/beat-atari-with-deep-reinforcement-learning-part-2-dqn-improvements-d3563f665a2c>

- *Hands on Machine Learning with Scikit-Learn, Keras and TensorFlow2* – Geron

-TFAgents official documentation

-OpenAI gym official documentation

**Appendix 1:**

* Cart\_Pole\_Example.ipynb – Cart pole model inspired using *Hands on ML with Scikit Learn, Keras and TensorFlow 2*
* AtariTFWrapper.ipynb – Pong model using 2015 DQN updates. Can be adapted to many Atari games with a few changes, hence the general name
* Pong\_Fresh-Copy2.ipynb – Pong using pure 2013 DQN that did not perform well at all
* Inline\_DDQN – Pong using the DuelingDQN 2015 update by DeepMind. No helper functions used to test memory handling
* CartPole\_TFAgents.ipynb – Cart-Pole using TFAgents API for 2015 updates
* Raw .gif and .mp4 files included

**Appendix 2:**

* Slides 6,7, and 8 contain .gif and .mp4 files of model performance. Hence the low number of figures here.