Space Invaders with Deep Q Learning

Nick Bernstein

I. Introduction

ON On January 1, 2013, before being acquired by Google, the UK based artificial intelligence company, DeepMind Technologies, published a paper detailing the process of how they were able create a Deep Learning framework that could learn to play seven popular Atari 2600 games using reinforcement learning. In some cases it performed better than a human. For my final project, I decided to recreate their framework specifically for one of the seven games, Space Invaders. I also wanted to attempt to determine if some of the advancements made in AI and machine learning in the last seven years will allow me to create a framework that could perform as well or better than DeepMind's using my personal computer that I built in 2016 as opposed to the vast computational resources that an AI research company has at their disposal. The main specifications of my computer are as follows: Intel i7-6700K 4.0 GHz Quad-Core CPU, 32 GB DDR4 3200 MHz RAM, and Nvidia GeForce GTX 1080 GPU.

II. DeepMind's Deep Q-Learning

The DeepMind framework created a modification to the Q-Learning algorithm that uses deep neural networks, called Deep Q Networks (DQN), and is now appropriately known as Deep Q-Learning. They also added a technique they call "experience replay," where the agent's experiences are stored for each time step in the form of a tuple containing the current state, the action to be taken, the reward of that action, and the state resulting from that action. A predetermined number of recent experiences are stored in a buffer that is then randomly sampled from to train the network. To fit the network they used the RMSprop optimization algorithm and the mean-squared-error loss function on a batch size of 32 experiences. Since reinforcement learning generates its own dataset from which to train, it can be rather noisy. To stabilize this, a copy of the network is created to generate target values for the training network. This target network is updated every so often with the more optimized weights from the training network.

For the network to be able to "see" the game's state, the RGB screenshot frame was fed into a preprocessing algorithm. Each raw frame contains a 210 x 160 pixel image integer values across the three color channels, red, green, and blue. This frame is converted to grayscale by taking the average pixel value across all three channels. Additionally these values are rescaled to float values between 1.0 and 0. This is an important step because neural networks are more effective processing numbers between these values [7]. Then unnecessary areas of the frame are cropped out. Finally, the remaining area is transformed into an 84 x 84 matrix (Fig 1).

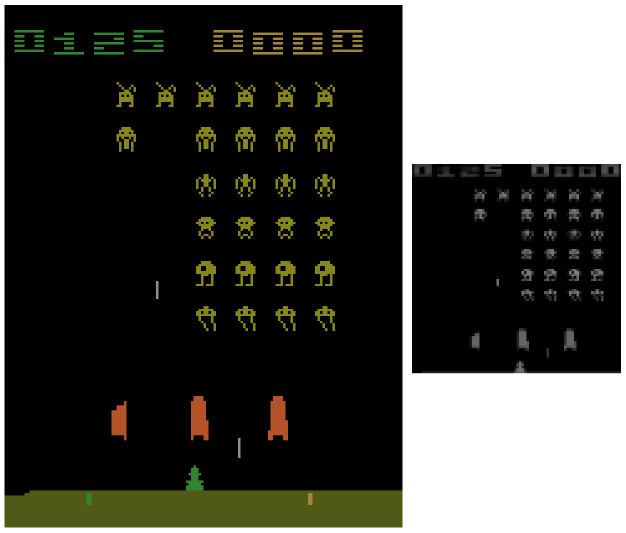

Fig. 1: Preprocess: Grayscale and frame transformation.

This process reduces the frame from containing 100,800 pixel values to only containing 7,056. Since important information such as direction or speed of movement cannot be determined by a single frame, they create a stack of the four most recent frames to feed into the network (Fig 2). This 84 x 84 x 4 tensor is passed through the following

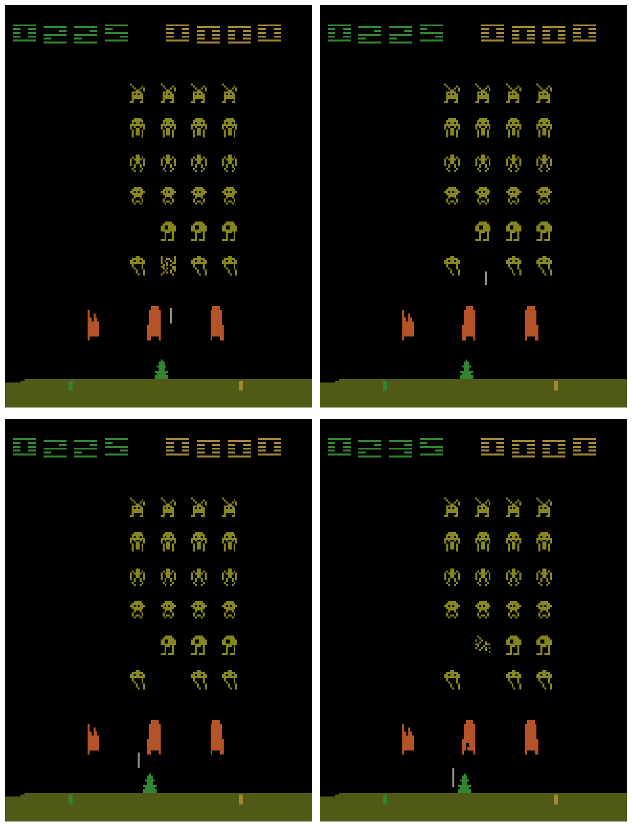

Fig. 2: Stack of frames to show movement. The frames have been kept in RGB form to better display the sense of motion gained by stacking them.

network: ”The first hidden layer convolves 16 8 x 8 filters with stride 4 with the input image and applies a rectifier nonlinearity. The second hidden layer convolves 32 4 x 4 filters with stride 2, again followed by a rectifier nonlinearity. The final hidden layer is fully-connected and consists of 256 rectifier units. The output layer is a fully connected linear layer with a single output for each valid action”[1]. For Space Invaders, there are six valid actions: stay still, move left, move right, stay still and shoot, move left and shoot, and move right and shoot.

During training on 10 million frames, the ε-greedy method is used. This method decides the next action by generating a random number that if greater than epsilon, a random action is chosen, and if otherwise, an action based on the network's predicted Q values for the current state. Gradually ε is decreased so that the agent is more often choosing actions based on it's learned behavior. DeepMind linearly annealed ε from 1 to 0.1 over the first 1 million frames and then kept constant for the remainder of the session. An additional consideration was made to accelerate the process by only adding every third frame to the stack of frames input into the network and repeating the recommended action for the subsequent three frames.

III. My Modifications

After implementing DeepMind's configuration of the network, I made a few slight alterations. I increased the final hidden layer's number of rectifier units from 256 to 512, I used the Adam optimization algorithm that was first introduced in 2014 as an improvement to RMSprop, and I used the logcosh loss function because it minimizes the magnitude of network updates when the difference between the predicted value and the target value is larger than 1 (Fig 3).

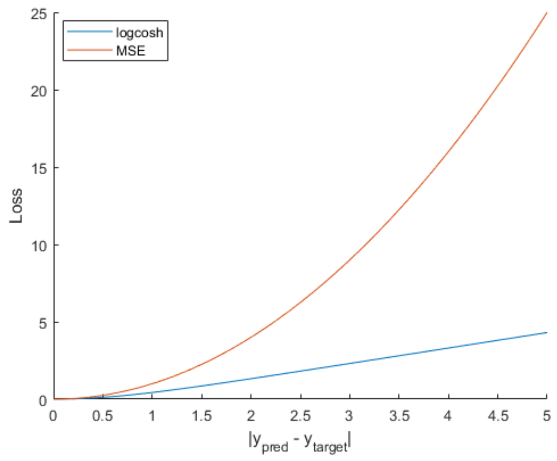

Fig. 3: mean-squared-error vs. logcosh loss.

Then I began experimenting with larger changes, the first being manipulation of the image frame from the Space Invaders environment. I saved the preprocessed images so that I could see what the agent ”saw” and identify potential issues. One such issue was that it didn't seem like the agent was able to notice enemy lasers as it was consistently moving right into their path. I noticed that these lasers were a very dark grey in the images I had saved, so I came to the conclusion that the number representing that grey was close enough to 0 that it was lost when processed by the network. To counter this, I modified the image by changing every value less than 0.05 to 0 (black) and all others to 1.0 (white)(Fig 4) . I trained a network on 1 million frames using this change and compared it to my original model trained on the same amount

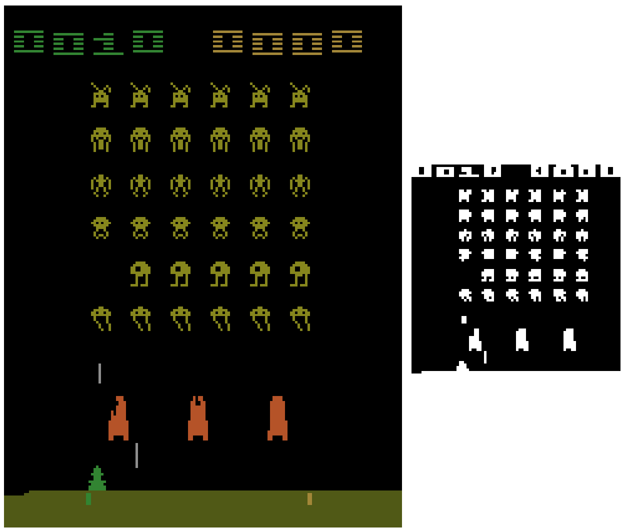

Fig. 4: My grayscale modifications to make lasers more visible.

of frames. Unfortunately, it was hard to make a definitive determination due to how noisy the results are, but it seemed like this change had potentially made the agent less effective. To make matters worse, since the number of frames processed is so high, this change increased the training time from roughly 12 hours to 16 hours, a 33% increase.

Next I noticed that at the start of each episode and after each death, there were roughly 40 ”dead” frames where no input action changed the game state. The total of 120 frames per episode accounted for at least 20% of the frames for most episodes. This meant that not only the agent was wasting time choosing moves that had no effect on the game, but the replay buffer contained a significant amount of useless information that the model would train on. Removing these frames did not speed up computation but after training another agent on 1 million useful frames, it was significantly better than the original model. My theory is that the model was able to train on more meaningful information throughout those 1 million frames.

My final change was the introduction of a dueling architecture found in a 2016 DeepMind paper. This network creates two optimized functions instead of one. One determines the value of the current state and the other determines the advantage of each action at the current state. These values are combined to create the Q function used to determine which action to take (Fig. 5)[3]. This method was incredibly effective and created an agent that was able to clear the first stage of the game after training on 3 million frames, which took approximately 36 hours.

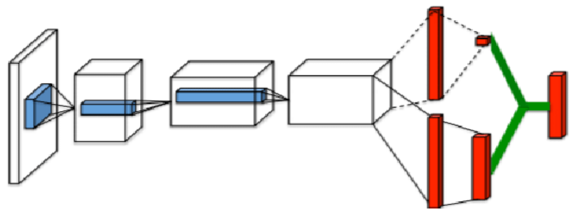

Fig. 5: Diagram of the Dueling DQN [3]

IV. Conclusion

Unfortunately, due to the time spent with experimentation, some inopportune computer crashes, and my need to use my computer for studying for other finals, I was unable to train my network on 10 million frames like DeepMind did. However, according to their paper, their agent’s best Space Invaders game achieved a score of 1075 points while my agent that trained on 20% fewer frames achieved a best score of 935. Because of this, I believe that if allowed to train on the same number of frames, my agent would have surpassed DeepMind’s.

References

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra and Martin Riedmiller. *Playing Atari with Deep Reinforcement Learning*, 2013; arXiv:1312.5602.

[2] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver and Daan Wierstra. *Continuous control with deep reinforcement learning*, 2015; arXiv:1509.02971.

[3] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot and Nando de Freitas. *Dueling Network Architectures for Deep Reinforcement Learning*, 2015; arXiv:1511.06581.

[4] J Grigsby. *Advanced DQNs: Playing Pac-man with Deep Reinforcement Learning*, June 29, 2018. Accessed on: Dec 20, 2019 [Online]. Available https://towardsdatascience.com/advanced-dqns-playing-pac-man-with-deep-reinforcement-learning-3ffbd99e0814

[5] A.L. Ecoffet, *Beat Atari with Deep Reinforcement Learning!*. Aug 21, 2017. Accessed on: Dec 20, 2019 [Online]. Available: https://becominghuman.ai/lets-build-an-atari-ai-part-0-intro-to-rl-9b2c5336e0ec?

[6] A. Juliani, *Simple Reinforcement Learning with Tensorflow Part 4: Deep Q-Networks and Beyond.* Sep 2, 2016. Accessed on: Dec 20, 2019 [Online]. Available: https://medium.com/@awjuliani/simple-reinforcement-learning-with-tensorflow-part-4-deep-q-networks-and-beyond-8438a3e2b8df

[7] Harrison, *Reinforcement Learning* May 30, 2019. Accessed on: Dec 20, 2019 [Online]. Available: https://pythonprogramming.net/machine-learning-tutorials/