Paul Galatic
CSCI-331
Professor Homan
Project 1

Introduction

Artificial intelligence has been described as a robot's ability to surprise a human by performing a task it is assumed that only a human is able to do. Effective game-playing is an example of a task that's designed for human interaction, yet may be effectively performed by a machine, given the proper setup. Digital games are ripe for this type of performance evaluation, as they require no robotics, boasting a digital interface easily manipulable by a software program. In the case of arcade games such as those by Atari, these interfaces can be extremely simple, as was necessary back in the time period they were made. These games are, therefore, ripe for experimentation. In this report, the original Space Invaders will be examined.

Methods

Google Deepmind published a paper in which they assessed the performance of their Deep Q Network algorithm by training it on Atari games (Mihn et. al, 2015). Deep reinforcement learning has since been the standard by which these games are played. However, this algorithm is extremely computationally intensive—most notably in the requirement that frames be added to a set of data that is later used to train the algorithm. I will be testing the necessity of this requirement.

OpenAI's Atari-Gym environment was utilized for this report. The models were trained using Tensorflow via Keras on an Nvidia 1060 GPU. They were each trained for 100 games.

Each frame, Gym provides the program one frame, and requests it take one action. In order to save space, much of the information in a frame is discarded before it is processed; its size is cut in half, and it is reduced to grayscale from color. Its pixels

are also normalized to a value between 0 and 1. The final input shape, to cooperate with Keras, is (105, 80, 1).

I have three neural networks, DQN4, DQN5, and DQN6. DQN4 was a result of experimentation, and DQN5 is the refinement of that model. However, those are both simply convolutional neural networks, and they only train on the most recent frame. DQN6 is my attempt at incorporating deep Q-learning.

DQN4 uses Maxpool primarily to decrease the size of the image, with the idea that focusing on the brighter pixels may allow it to see important details, like enemy bullets.

DQN5 uses a more traditional approach, simply with convolutional layers.

DQN6 has the same structure as DQN5, but with the integration of Q learning. DQN6 is the only network with a memory lasting more than one frame.

All convolutional layers are activated with RELU. The optimizer was RMSProp, and the loss was the mean squared error between the probability vector of actions for a given state and expected reward, either given by one frame or via the Q function.

```
Structure of DQN4:
     INPUT            (?, 105, 80, 1)
     CONV_1           (?, 101, 76, 32)
     MAX_POOL_1       (?, 50, 38, 32)
     CONV_2           (?, 49, 37, 64)
     MAX_POOL_2       (?, 24, 18, 64)
     FLAT             (?, ?)
     HIDDEN           (?, 128)
     OUTPUT           (?, 6)

     Hyperparameters
          GAMMA          = 0.95
          EPSILON_MAX    = 1.0
          EPSILON_MIN    = 0.01
          EPSILON_DECAY  = 0.995
          LEARNING_RATE  = 0.001
          MAX_SIZE       = 2000
          BATCH_SIZE     = 32

Structure of DQN5:

     INPUT            (?, 105, 80, 1)
     CONV_1           (?, 52, 40, 32)
     MAX_POOL_1       (?, 26, 20, 32)
     CONV_2           (?, 13, 10, 64)
     FLAT             (?, ?)
     HIDDEN           (?, 6)

     Hyperparameters
          GAMMA          = 0.95
          EPSILON_MAX    = 1.0
          EPSILON_MIN    = 0.01
          EPSILON_DECAY  = 0.9
          LEARNING_RATE  = 0.005
          MEMORY_SIZE    = 2000
          BATCH_SIZE     = 32
```

Structure of DQN6:

INPUT	(?, 105, 80, 1)
CONV_1	(?, 51, 39, 32)
MAX_POOL_1	(?, 25, 19, 32)
CONV_2	(?, 11, 8, 64)
FLAT	(?, ?)
HIDDEN	(?, 6)

Hyperparameters:

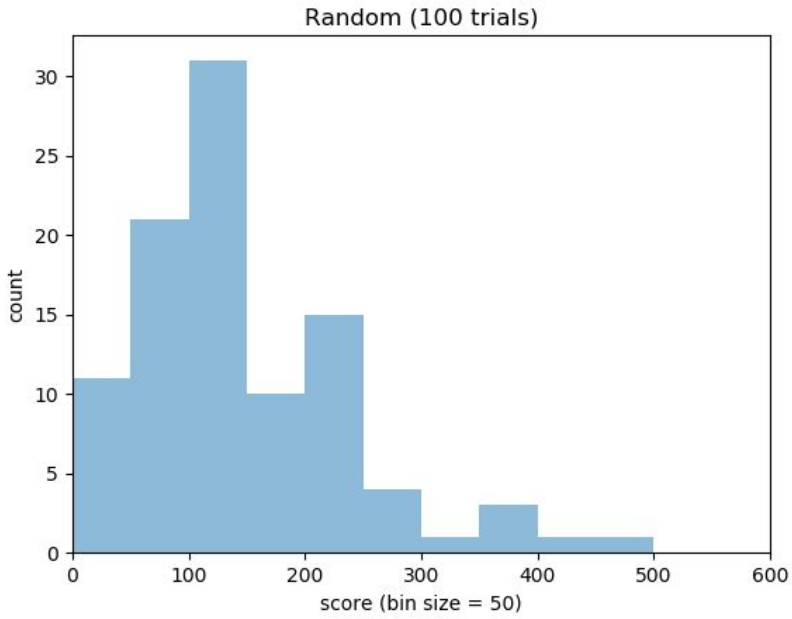
GAMMA	= 0.95
EPSILON_MAX	= 1.0
EPSILON_MIN	= 0.1
EPSILON_DECAY	= 0.9995
LEARNING_RATE	= 0.0001
MEMORY_SIZE	= 2000
BATCH_SIZE	= 32

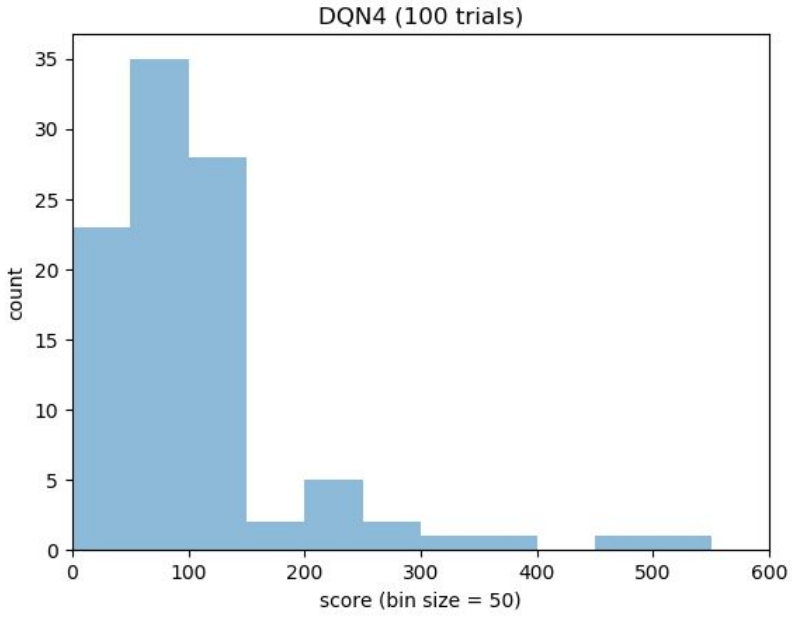
Alll networks then collapse the final convolutional layer into a flat layer, then use one fully connected hidden layer before mapping those nodes to outputs corresponding to actions. Softmax is applied to turn the output layer into a one-hot vector.

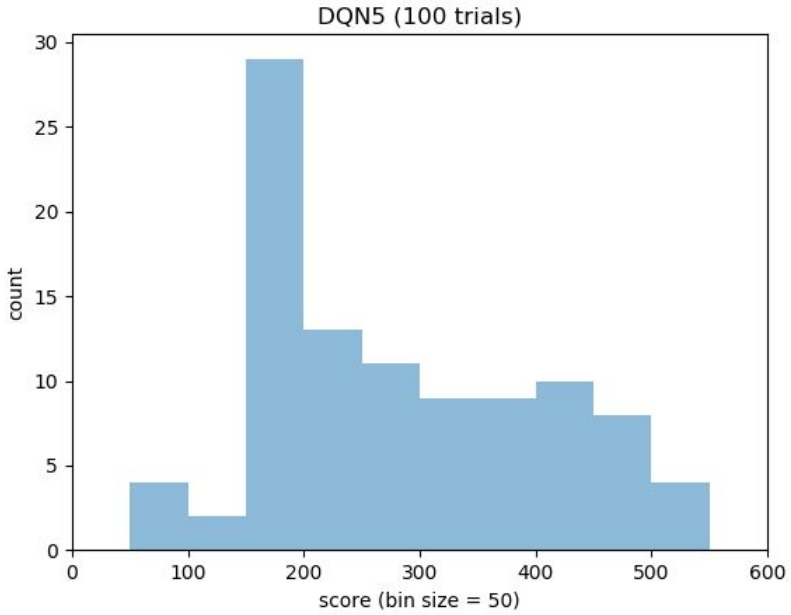
In Q learning, the models are then trained based on accumulated experience. Initially, the model will take mostly random moves, according to the Epsilon of the training session, which decreases over time. However, eventually it will begin to act based on a policy it develops during Q learning. This policy is designed to maximize the agent's reward. In this implementation, Gamma is the discount, and Epsilon is the probability that the agent will randomly sample the action space.

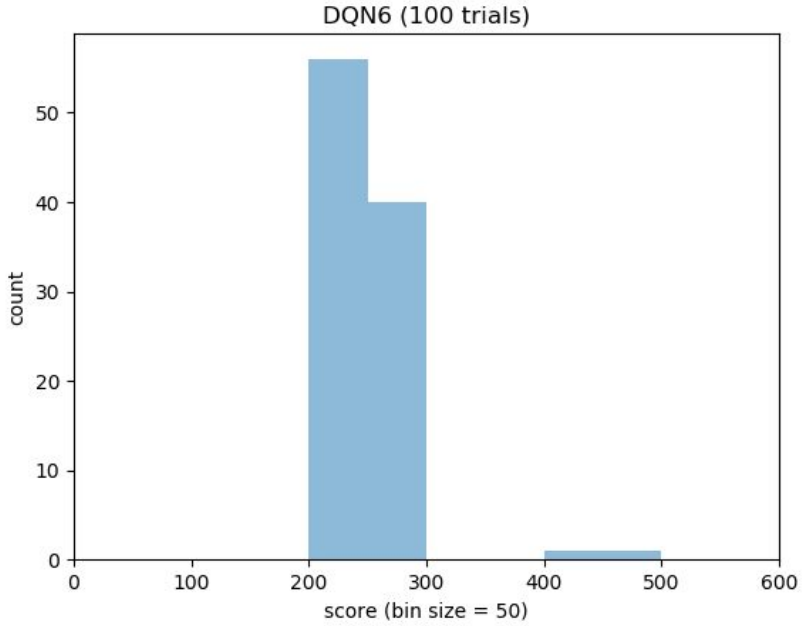
However, experience replay takes an enormous amount of time. So, for the first training sessions, only dummy memories were used that would always return the most recent action. This is to see whether or not the agent can learn a policy even though it does not have access to its memory, and to see whether or not training with a memory could be equivalent to training without one.

Results


Random (100 trials)
count
score (bin size = 50)
0
5
10
15
20
25
30
0
100
200
300
400
500
600


DQN4 (100 trials)
count
score (bin size = 50)
0
5
10
15
20
25
30
35
0
100
200
300
400
500
600


DQN5 (100 trials)
count
score (bin size = 50)
0
5
10
15
20
25
30
0
100
200
300
400
500
600


DQN6 (100 trials)
count
score (bin size = 50)
0
10
20
30
40
50
0
100
200
300
400
500
600

Conclusion

While it is true that deep neural networks can approximate a game and play it reasonably well, in this case it certainly landed on a strange solution. It is likely that the neural network could not properly see bullets—so, it learned to rapidly cross the map toward the rightmost edge and stay there in order to survive longer, inadvertently racking up kills as it did. Both DQN5 and DQN6 converged to this strategy. In particular, DQN6 converged so harshly that it overfit, creating extremely regular patterns of play (and, by extension, score).

Each neural network performed significantly better than Random after only 100 games of training. DQN5 and DQN6 generally earned a score of 280, though DQN5 sometimes scored much higher.

The overall high score, however, was DQN4, with a score of 1000 if the environment seed is set to 0.

Limitations

The most obvious limitation is the lack of computational power. With more resources, a more complex set of neural networks could be used, increasing the amount of information the agent has access to and/or decreasing training time.

These time limitations meant that there is no way to tell whether or not the memory-enabled agent would have eventually converged to an even higher average score. However, it is likely that in order to converge, the model would have required days of continuous training, which was not available at the time of this experiment.

References

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., . . . Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529-533. doi:10.1038/nature14236 . Retrieved from https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf.