Treasure Hunt Game: AI Design

Matthew Pool

Computer Science Undergraduate, Southern New Hampshire University

CS 370: Current & Emerging Trends in Computer Science

Professor Obafemi Balogun

April 19, 2024

Human vs. Machine

A human would start off with a random choice, using a bit of "intuition". This is somewhat similar to how the ML algorithm would start off via random exploration of the environment. With an epsilon value ϵ (exploration factor) that starts at a value close to 1.0, a machine would (most likely) take another random leap of faith. A human may continue in the same direction on their next turn, although it is possible that a human would go a different route. If the algorithm had a low epsilon value to boot, then it would probably continue in its original direction, using its past experience of success found in its developing policy. The ML policy is similar to the human's intuition; however, it is more concrete and based on exact mathematical calculations involving loss and value functions.

Suppose that each player takes an action (move) that leads directly to an obstacle. The agent would know to never take this action while in this exact state (position) due to the penalization that results in a lower cumulative reward (Q-value). Given a maze that is magnitudes larger than this one, a human would not be able to keep track of his/her exact position in the maze and may make the same mistake more than once. A machine, on the other hand, would not make this mistake. The advantage I think a human does have though is the understanding of directionality. For example, a machine is going either: LEFT, RIGHT, UP, DOWN, but a human may be headed in the "down-right" direction. If an obstacle were to be in the way, the human may start going "up-right" or "up" because the human understands that the general direction is blocked. Going back to the example of an extremely large and complex maze, after winning the game several times (episodes), the machine would eventually win the game with a perfect path (max

cumulative reward); however, our human friend is bound to repeat the same (or discover new) mistakes. Although, our machine has the disadvantage of requiring lots of data and time – not to mention computational resources. A human does much better with one-shot and few-shot training – especially for simpler, smaller tasks. A human doesn't need (or want) to discover every single action in every possible state, as the algorithm would want to discover or approximate. But again, as the maze grows, so does the human's inability to successfully complete it.

Exploration vs. Exploitation

The probability of an agent taking an action in any given state is partially dictated by the agent's exploration factor (epsilon value). This value determines the probability of an agent choosing to explore its environment to learn new information versus exploiting knowledge from prior experiences to determine the appropriate action in that state (Beysolow II, 2017). If exploiting its knowledge, the agent uses its policy to determine the action that probabilistically results in the highest cumulative reward for the current state. Exploration means the agent randomly chooses a direction. This can result in a poor decision (action), which is why the epsilon value in an epsilon-greedy approach should be relatively low (closer to 0 and farther from 1.0). A final epsilon-value of 0.5 was used for this deep Q-learning network to ensure that approximately every other move involves exploring and then exploiting the environment to discover the best path, but to primarily rely on its accumulated memory for the highest reward. Once the agent reaches a win rate of 90% (or higher), the exploration rate is lowered to 0.05, meaning the agent should rely on its developed policy to dictate its actions about 95% of the time. This approach assumes that the agent should basically know by now what actions to take in any given state. Alternatively, a slow decay could be used to gradually lower the exploration rate over time.

Reinforcement Learning

Reinforcement learning penalizes the agent by -0.25 if the agent moves to a square that is has previously visited. If the agent runs into an obstacle (or wall), then it is penalized by -1.0 to indicate that this action is a completely wasted move and worse than moving to a previously visited square. The agent is penalized by -0.04 for every time step. This makes the agent learn to arrive at the treasure (goal) in as few steps as possible. When the pirate (agent) reaches the goal, it is rewarded by 1.0, indicating that the highest cumulative reward is received upon reaching the goal in as few steps as possible.

Deep Q-Learning

This neural network is a sequential model, which consist of a linear stack of layers, starting with an input layer and a PReLU (parametric rectified linear unit) activation, which directly outputs the value of any positive input or outputs 0 for any negative input (Gulli & Pal, 2017). This feeds data into a dense, hidden (processing) layer with PRelu activation. Finally, the output layer consists of four classes (LEFT, RIGHT, UP, DOWN). Gradient descent and backpropagation are used to minimize the MSE (mean squared error) loss function, which is the difference between the target Q-values and current estimated Q-values (calculated using the Bellman equation). The model is compiled with the Adam optimizer to provide an adaptive (dynamic) learning rate, in order to adjust the parameter weights θ and maximize the Q-values (value function).

The Q-learning algorithm consists of creating an instance of a maze (environment) and experience (memory). For each epoch (training dataset pass), the agent starts by taking a random action. Then using the epsilon value as the exploration weight for a randomized action, the agent

explores the environment or exploits its prior knowledge (policy). The state and reward are updated, based on the action taken in the previous state. If the agent's cumulative reward falls below the negated value of half the size of the maze, then the game is over, and the agent loses. A tally of game wins (1) and losses (0) are kept in a list, with each experience saved in the agent's memory.

Results

The batch size used for training was fixed to a value of 16, which is the number of passes through the training data before updating the parameter weights, θ. Since the aim of this project is to train an agent for the singular task of path-finding in a static maze, then the number of epochs taken by the agent to reach a 100%-win rate may be the most critical factor to consider, aside from the agent choosing one of the optimized paths to the goal; however, total training time may be more important if training this model on different mazes within a limited development timeframe.

Various numbers of game epochs and epsilon values were tested to optimize the model. 1000 game epochs with an epsilon value of 0.1 resulted in the model reaching a 100%-win rate within an unimpressive 313 model epochs, taking almost an hour and a half to complete. Lowering the game epoch value from 1000 to 64, 32, 16, 12, and 8 yielded better results, with 16 epochs giving the best stats , reaching a 100%-win rate in only 69 epochs and 14.82 minutes.

game epochs = 1000:

```
Epoch: 313/14999 | Loss: 0.0000 | Episodes: 39 | Win count: 302 | Win rate: 1.000 | time: 1.48 hours
Reached 100% win rate at epoch: 313
n_epoch: 313, max_mem: 512, data: 32, time: 1.48 hours
```

game epochs = 64:

```
Epoch: 094/14999 | Loss: 0.0002 | Episodes: 32 | Win count: 91 | Win rate: 1.000 | time: 23.11 minutes
Reached 100% win rate at epoch: 94
n_epoch: 94, max_mem: 512, data: 32, time: 23.12 minutes
```

game epochs = 32:

```
Epoch: 101/999 | Loss: 0.0001 | Episodes: 27 | Win count: 92 | Win rate: 1.000 | time: 18.93 minutes
Reached 100% win rate at epoch: 101
n_epoch: 101, max_mem: 512, data: 32, time: 18.94 minutes
```

game epochs = 16:

```
Epoch: 069/999 | Loss: 0.0000 | Episodes: 23 | Win count: 55 | Win rate: 1.000 | time: 14.82 minutes
Reached 100% win rate at epoch: 69
n_epoch: 69, max_mem: 512, data: 32, time: 14.84 minutes
```

game epochs = 12:

```
Epoch: 133/999 | Loss: 0.0010 | Episodes: 20 | Win count: 115 | Win rate: 1.000 | time: 17.43 minutes
Reached 100% win rate at epoch: 133
n_epoch: 133, max_mem: 512, data: 32, time: 17.44 minutes
```

game epochs = 8:

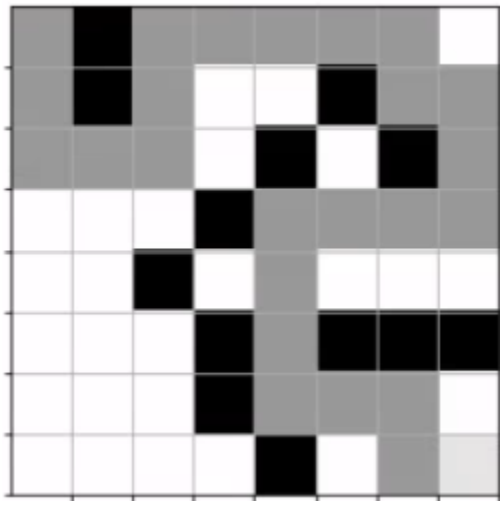
```
Epoch: 191/999 | Loss: 0.0002 | Episodes: 3 | Win count: 180 | Win rate: 1.000 | time: 15.42 minutes
Reached 100% win rate at epoch: 191
n_epoch: 191, max_mem: 512, data: 32, time: 15.43 minutes
```

With 16 epochs, raising the exploration factor from 0.1 to a value of 0.5 (equal exploration and exploitation probabilities) gave the best results, taking only 56 epochs in a little over 15 minutes.

epsilon $\epsilon = 0.5$

```
Epoch: 056/999 | Loss: 0.0030 | Episodes: 23 | Win count: 39 | Win rate: 1.000 | time: 15.28 minutes
Reached 100% win rate at epoch: 56
n_epoch: 56, max_mem: 512, data: 32, time: 15.30 minutes
```

While testing the agent (playing one game), the agent chose the optimized path, represented by the dark (80%) gray squares, with the goal (and final destination of the agent) in the bottom-right square:



Seeing is believing!

References:

Beysolow II, T. (2017). *Applied Reinforcement Learning with Python: With OpenAI Gym, Tensorflow, and Keras*. Apress L.P.

Gulli, A. & Pal, S. (2017). *Deep Learning with Keras.* Packt Publishing.