Daniel Chin        1002095

Laura Ong        1002464  
Jiang Jinjing 1002368

# Artificial Intelligence Project

12th August 2019

Objective: to train an AI agent to solve a Minesweeper problem using an 8x8 matrix.

## Introduction

Minesweeper is an NP-complete problem, hence the challenge is to guess the probability that a particular tile has a hidden mine and take appropriate future steps. We apply the Deep Q-learning method in order to evaluate the parameters in solving this problem.

## Loss

**MSELoss** was used as the loss function.   
**RMSprop** was used as the optimizer.

## Training Procedure

Data:

An 8x8 matrix was generated randomly. Five mines were placed at maximum but arranged randomly. We thought about limiting bomb appearances to the inner border, thus reducing our search space to 6x6, but felt that the resulting constraint would also lead to difficulty in approximating the mine’s location as there is lesser room for exploration.

This can be found in minesweeper.py

## Optimum Training Settings

Batch Size: 16

Number of Epochs: 20 000

Learning rate: 1e-3

## Hyperparameters tuned:

* Batch Size: An increase in Batch size brought about a lower testing score
* Number of Epochs: An increase in epochs led to an increase in testing score before it plateaus at around the 15000th step. We feel that it might be due to overfitting.
* Learning Rate: An increase in learning rate to 1e-2 led to slow growth, and thus was stopped midway.
* Number of Mines: Varying mines caused less overfitting but lower reward scores
* Model used: It was found that by reducing the fully connected layers led to faster convergence. Hence, the current existing model uses a CNN to read into 10 channels. First to Eighth channel to account for the number of mines placed surrounding the square, the ninth to indicate if there was a mine, and the last if otherwise.
* Optimizer: We switched from Adam to RMSprop as we found that RMSprop converges at a slower rate but contains less variability

## Results:

The Accuracy and the Rewards would be placed side by side for comparison. Our reward was calculated by the following:

* If no progress was made in increasing the probability, then a deduction of 0.3 is given
* If progress was made, then 0.9 reward was given
* If all empty spaces have been uncovered, than a reward of one was given
* If mine was struck, then deduction of 1 is given

We discussed if we should deduct the reward given that a random click was created. We felt that though it is unlikely to result in winning the game, the agent will eventually figure out that it can get the progress reward by clicking a random unknown field, and human attempts recommend this as a strategy. Therefore, we did not deduct or reward the agent for doing a random action. However, the agent has a 10 percent chance given the current parameters.

In addition, we discussed and felt that since Minesweeper can be divided between the exploration phase and the deductive phase, more exploration should be encouraged in the beginning of the game, hence no deduction is given for random boxes selected. However, as we come close to deduction, we see that there is a lesser inclination for random clicking (and partially due to lack of spaces for random clicking).

A close up of a logo

Description automatically generatedA close up of a logo

Description automatically generated

### Parameters:

Replay Memory of 50 000

Number of Epochs: 20 000

Learning Rate: 1e-3

Model as described in the following paper[[1]](#footnote-1):

self.fc1 = nn.Linear(n\_states, 288)

self.fc2 = nn.Linear(288, 220)

self.fc3 = nn.Linear(220, out\_length) #0 to 63 actions that can be taken

As seen, the high variability led to the understanding that more layers had increased complexity, and thus lost information from the board. We attempted to use a CNN and layer directly.

A close up of a logo

Description automatically generated

A close up of a logo

Description automatically generated

### Parameters:

Replay Memory of 50 000

Number of Epochs: 20 000

Learning Rate: 1e-3

Model:

Conv2D(1,2,3) -> A convolutional layer of kernel size 3, with 2 channels. Unlike earlier described, this was meant to reduce the number of channels and thus hopefully complexity. However, we found that this also reduced the information that we obtained from the board when it is fed through the neural network.

nn.ReLU -> a ReLU function

Linear(72, 512) -> A linear layer that takes in the input of 64 squares -> a flattened 8x8 matrix

Linear(512, 64) -> A linear layer that outputs a list of values. The highest value would be the square selected

A close up of a logo

Description automatically generatedA close up of a logo

Description automatically generated

### Parameters:

Replay Memory of 50 000

Number of Epochs: 20 000

Learning Rate: 1e-3

Model:

Conv2D(1,10,3) -> A convolutional layer of kernel size 3, with 10 channels. By comparing between this and the initial graph, we see a visible improvement in the accuracy.

nn.ReLU -> a ReLU function

Linear(360, 64) -> A linear layer that outputs a list of values. The highest value would be the square selected

A screenshot of a cell phone

Description automatically generatedA screenshot of a cell phone

Description automatically generated

### Parameters:

Replay Memory of 50 000

Number of Epochs: 20 000

Learning Rate: 1e-3

Optimizer: RMSProp (Previous tuning used Adam; while accuracy seems to be similar, the variations in reward has decreased. Thus we recommend RMSProp for the final tuning. )

Model:

Conv2D(1,10,3) -> A convolutional layer of kernel size 3, with 10 channels

nn.ReLU -> a ReLU function

Linear(360, 64) -> A linear layer that outputs a list of values. The highest value would be the square selected

## Files

|  |  |
| --- | --- |
| File Name | Description |
| main.py | Training Procedure |
| miner.py | AI Agent |
| Minesweeper.py | Generator class |
| gui.py | GUI |
| eval.pth | State dictionary of best model |
| Images/ | Images needed for the GUI to perform |

## Bibliography

Luis Gardea, Griffin Koontz, and Ryan Silva, “Training a minesweeper solver”, 2015, accessed at <http://cs229.stanford.edu/proj2015/372_report.pdf>

Jacob Hansen, Jakob Havtorn, Mathias Johnsen, and Andreas Kristensen, “Evolution Strategies and Reinforcement Learning For a Minesweeper Agent”, 2017, accessed at https://github.com/jakejhansen/minesweeper\_solver/blob/master/article.pdf

1. https://github.com/jakejhansen/minesweeper\_solver/blob/master/article.pdf [↑](#footnote-ref-1)