Minesweeper solver using NN

Oleg Yurchenko, Alisher Tortay
CS492(C), Fall 2016

Abstract

Minesweeper is a popular puzzle game developed by Microsoft. For the past two decades a number of approaches to solve the game were presented. Most of them are based on estimating the probability of a covered tile to be a mine. Recent research from Stanford University proposed an interesting attempt to apply modern reinforcement learning techniques to this problem. In this paper we present another approach for solving Minesweeper game using neural network classifier. We were able to obtain results comparable to those of the state of the art algorithms.

Introduction

Minesweeper is an old famous single-player game puzzle. The goal of the game is to clear a rectangular board containing hidden "mines" without exploding any of them, with assistance from pieces of information about the quantity of neighboring mines in every field.

At first player is given an empty board. At each step the player should either pick an unlabeled square to “test” or mark a tile as containing a mine. In the event that the tested tile contains a mine the game is over with loss for the player. Otherwise, tested tile will reveal a number of adjacent mines. If player uncovers the last covered tile without mine, then player wins.

Playing this game requires use of logic and awareness of relations between neighboring fields. It is a game of imperfect information and checking the state of board for solutions is NP complete problem. Therefore, playing minesweeper with neural networks is challenging and interesting research topic in Artificial Intelligence. Our goal in this project was to implement Neural Network based agent that can play Minesweeper.

Literature Review

Recent development of machine learning caused a revival in interest in using machine learning and reinforcement learning techniques in the computer game industry. During the past few years, number of different attempts have been made to solve wide range of games. State of the art algorithms for RL include Q-learning, policy gradient, and number of other approaches. For instance, in 2013 DeepMind in their paper presented a deep convolutional neural network using which they achieved huge performance in playing Atari 2600 games. Similar networks and techniques were used for other games, such as Flappy Bird, Out Run, etc.

As for MineSweeper solver, wide range of methods had been applied starting from analytical solutions, genetic algorithms to Q-learning method. The first attempts to solve MineSweeper game were developed in 1990's. Such that, in 1997, A. Adamatzky showed how to mark all mines populated on n*n board using two-dimensional cellular automaton in $\Omega(n)$time. The state of art method for solving MineSweeper game is limited search with probability estimates strategy proposed by K. Pedersen in 2004. The solver based on this method achieved 92,5% win rate on the beginner board, and 25% on expert level.

There also have been few attempts to use machine to play the game. In 2015 students from Stanford University applied supervised learning using SVM classifier and linear regression model, and Q-learning algorithm based on Q-tables. However, performance of those approaches dramatically decreases as the size of the board increases. For instance, the win-rate for Q-learning algorithm reaches 70% for 4*4 board with 3 mines, but falls to the level of around 5% for 5*5 boards with 5 mines (the density of mines is nearly 20% on those boards). Meanwhile, other algorithms have even lower win rate.

Methods and Approach

Due to the lack of availability of Minesweeper frameworks capable of implementing an automated player, we wrote our own version of minesweeper game. In our game there is no mine at position (0,0). Interface works as follows: we open a square using open(pos) function and mark a tile using mark(pos) function. This functions returns state in the form of array(1,n*m) and whether game is finished. From the game we also can obtain true positions of all mines; we will use this information to train our agents.

1. Policy Based Agent

In policy based model we used two architectures for neural network. In first one we used feed forward network with one hidden layer with 128 nodes. We used epsilon-greedy exploration strategy. We feed network with whole board (4x4 and 8x8). For 4x4 board with 3 mines after 10,000 training games rate of wins is 2%. We would get same rate with making random moves. For 8x8 during 20,000 training iterations we agent could win only two games. In other architecture we added two convolutional layers of size 3 and 5. However, no performance improvement was observed. For 8x8 during 10,000 training iterations our agent could win only one game.

2. Q Learning Agent

In Q learning model we changed usual q learning algorithm to fit our problem. We are not interested in long-term reward. Instead we are more interested in the immediate reward. Thus we removed future rewards. We also used board with true positions of mines to produce our target Q values. During 10,000 training games on board 4x4 with 3 mines agent managed to win 900 games. Since minesweeper has strong local dependencies we decided to add convolutional layers. However, it had no positive effect on performance.

3. Supervised Learning Agent (SLA)

We used supervised learning model to implement this agent. We designed three feed-forward neural networks with different sizes. Input is subboard of size 5x5. A tile has 12 possible values: uncovered, flag, out-of-board, and integers from 0 to 8. Each tile in this subboard is represented using one hot encoding. Output is one number between 0 and 1: probability that central tile of the 5x5 subboard does not contain a mine. We trained our networks with with following values: 0 if central tile of 5x5 input contains a mine and 1 otherwise. We define perimeter as a set of tiles that have at least one uncovered or marked neighbor.

Supervised Learning Algorithm

```
1.  begin by probing upper-left corner
2.  while not game over do
3.      foreach tile in the perimeter
4.          predict probability
5.          if probability < 0.1
6.              mark the tile as mine
7.          else if probability > 0.1
8.              unmark the tile
9.          end if
10.     foreach tile in the perimeter
11.         predict probability
12.         save this training sample into experience buffer
13.     end foreach
14.     open a tile with maximal probability
15. end while
16. train network using random sample from experience buffer
```

As you can see from the algorithm we save training samples into experience buffer and trained our networks by randomly choosing samples from the buffer. Moreover, we predict probabilities for each tile of perimeter twice. In first round we put or remove flags. In second round we use information from uncovered tiles as well as flag information form first round. After predicting probabilities from second round we open the tile with highest probability (of being safe).

Input size is 25 * 12 = 300. We designed three feed-forward neural networks for supervised learning model.

1. **Small:** Three hidden layers of sizes 300, 256, and 128, respectively
2. **Large-A:** Four hidden layers of size 300, 256, 128, and 64, respectively
3. **Large-B:** Four hidden layers of size 300, 256, 256, and 128, respectively

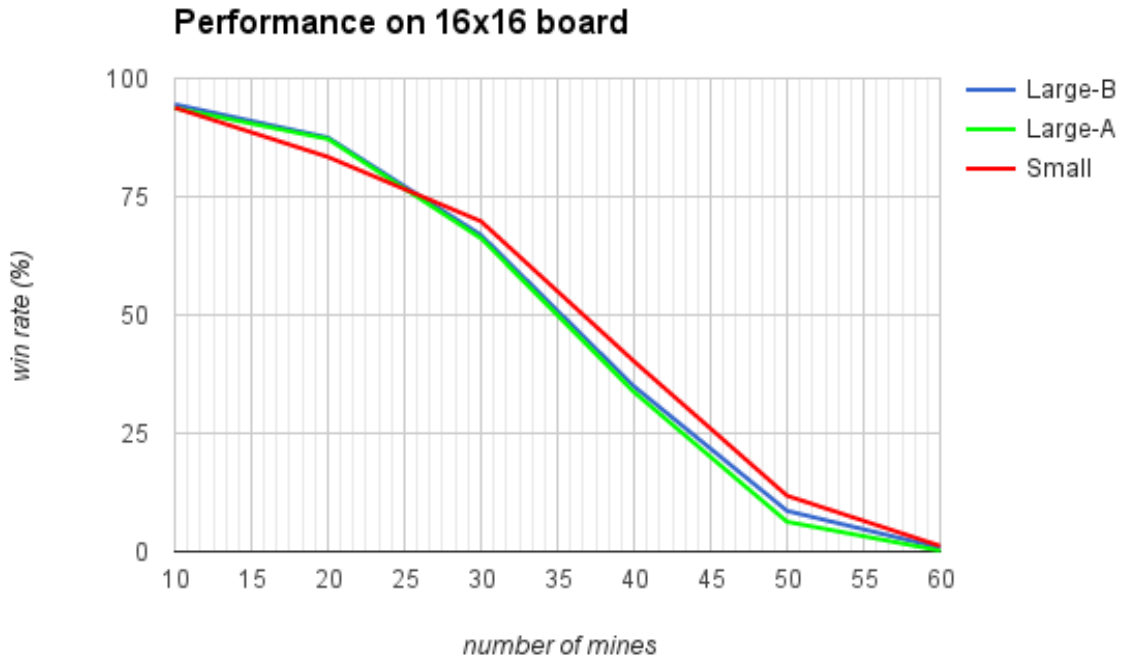
It is important to point out that this model scales perfectly on a board of any size because we always train the same network. That is, training and prediction time is same for all boards.

4. Modified Supervised Learning Agent (MSLA)

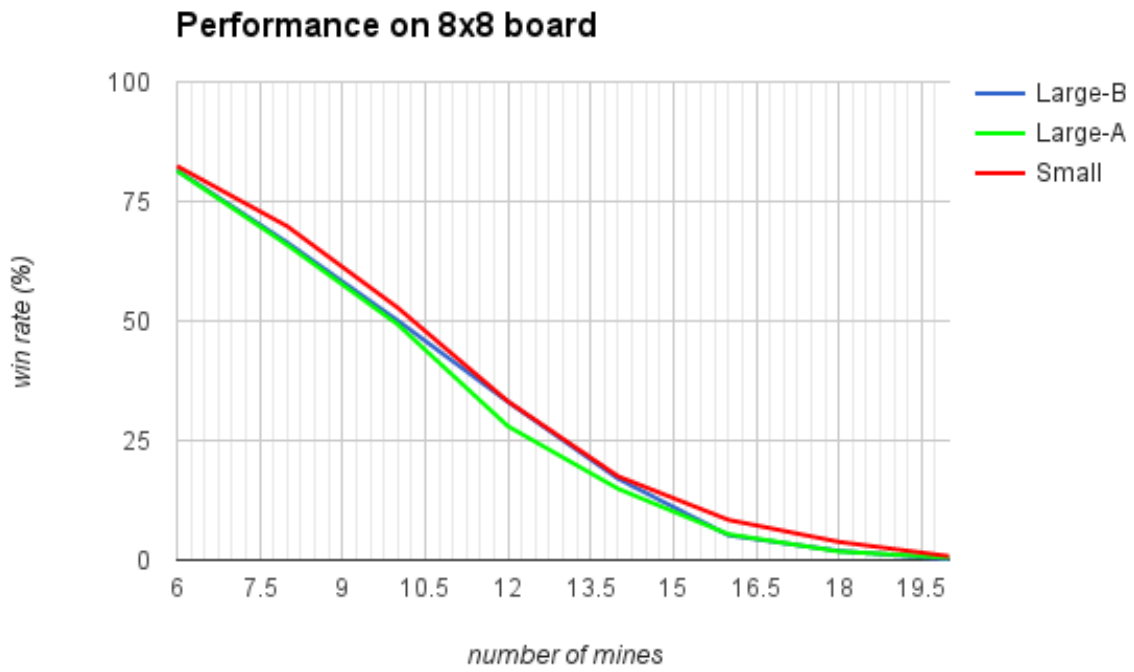
In some cases, it is impossible to say if a given tile contains a mine. Therefore, in the model above we train our network with values that are sometimes infeasible. For example, suppose we have two covered tiles with same “true” probability of being safe. Ideally in this case our model should return 0.5 probability for each. However, we always train our model only with 1 or 0. To solve this problem we implemented modified version of supervised learning model. In this modification instead of training with 1 and 0, we use probabilities from other solver. In other words, we approximate another solver using neural networks. This approach is useful when computation time of original solver is too long. In this model we used very simple solver based on additive properties of Minesweeper game.

Results and Analysis

Since only supervised models were successful, we provide tests only on supervised learning agents (SLA) with 3 different Neural Network architectures and modified supervised learning agent (MSLA). Before testing we trained each of three supervised learning networks on 16x16 board with different number of mines.



We tested all three SLA Neural Networks on 16x16 boards with different mine numbers. From the line chart above we can see that neural networks with 4 hidden layers (Large-A and Large-B) perform better that network with 3 hidden layers (Small) on boards with low density of mines. However, on boards with high density of mines Small performs better than both Large-A and Large-B. In general, Large-B and Large-A have similar win rates over all mine densities.



We also tested all three SLA Neural Networks on 8x8 boards with different mine numbers. Small has the highest win rate and Large-A has the lowest win rate over all mine densities. In general, Large-B perform similar to Large-A, however, on boards with moderate density it matches with Small.

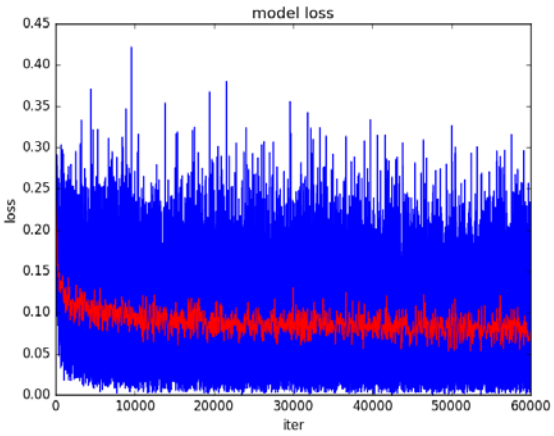
These results were quite surprising; we expected that NNs with more layers will perform better since they have higher complexity. However, more layers take more time to train. That could be a reason of their poor performance. It is also important how NNs are trained. If NN is trained only on boards with low density of mines, it will have very low win rate on boards with high density of mines.

From the table below we can see that both MSLA and solver for MSLA have low win rate compared to our SLA networks. However, it is worth noticing that MSLA approximates its solver quite well. Therefore, solver is the bottleneck for MSLA. This problem could be solved using a better solver to approximate.

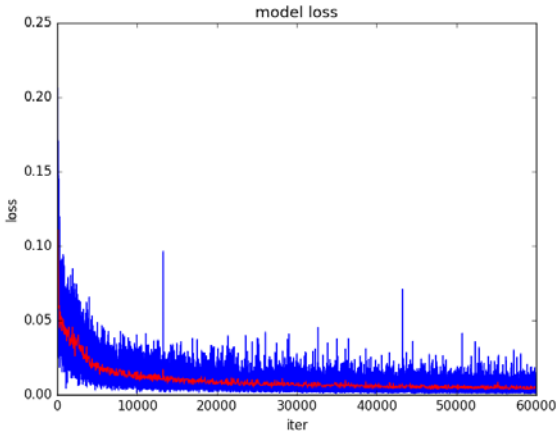
Additionally, the table compares our models to limited search with probability estimates strategy (Pedersen) for two boards. In both cases limited search with probability estimates strategy outperforms our models. However, our models, once trained, are faster than the limited search. Overall, proposed models show significant improvement compare to previous attempts of using neural networks for Minesweeper agent.

Board size and # of mines	Density	Small	Large-A	Large-B	MSLA	Solver for MSLA	Pedersen
8x8, 8	12.5%	**69.8%**	65.9%	66.4%	30.7%	33.6%	92.5%
8x8, 10	15.6%	**52.8%**	49.3%	50.1%	11%	15.3%	-
10x10, 10	10%	-	-	-	39.2%	40%	-
16x16, 20	7.8%	83.4%	87.2%	**87.5%**	40.4%	47%	-
16x16, 40	15.6%	**40.2%**	33.7%	34.9%	0%	0%	67.7%

We also implemented convenient interface for the game to visualize the process of 'thinking' or estimating probabilities. In our game implementation we used different colors that would represent predicted probabilities. In figure above green color represents "safeness" and purple color represents mines. Using this representation, we are able to understand how our model "thinks". We can see that in certain tiles our agent is more confident than in others.



Supervised Learning Agent



Supervised Learning Agent

As we can see from the graphs above, proposed models show significant fluctuations of the loss function. We think that such variations are caused by inaccuracy of values that we use to train our models.

Further Research/Work

Since our approach is based on the approximation of ‘safe’ probability, one obvious way to improve success rate is to use better solver. For instance, we propose to use the limited search with probability estimates strategy (K. Pedersen) as a true oracle. Also, this approach will give us a better insight into situations in which it is potentially impossible to be certain about the true probability of the safe choice. This will prevent the cost function from significant fluctuations and possibly improve and speed up learning process. In addition, we believe that it would be beneficial to try to approximate other probabilistic methods, such as SCP, rejection method (for consistent belief state estimation), MCMC, etc.

Links

https://www.youtube.com/watch?v=io00NDkawgs
https://www.youtube.com/watch?v=m93DKCp-P0c
https://github.com/Chizhik/minesweeper

References

1. Adamatzky, Andrew. "How cellular automaton plays Minesweeper." *Applied mathematics and computation* 85.2 (1997): 127-137.
2. Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).
3. https://en.wikipedia.org/wiki/Q-learning
4. https://en.wikipedia.org/wiki/Reinforcement_learning
5. Hamadi, Youssef, and Marc Schoenauer, eds. *Learning and Intelligent Optimization: 6th International Conference (p. 223), LION 6, Paris, France, January 16-20, 2012, Revised Selected Papers*. Vol. 7219. Springer, 2012.
6. Pedersen, Kasper. "The complexity of Minesweeper and strategies for game playing." *Project report, univ. Warwick* (2004).
7. Gardea, Luis, Griffin Koontz, and Ryan Silva. "Training a Minesweeper Solver."

Tasks

1) Research on the topic (Oleg 15, Alisher 16)
    a. Minesweeper overview
    b. Probabilistic solvers
    c. Neural Networks
    d. Q-learning

    e. Policy based models
2) Game development (Oleg 11h, Alisher 5h)
3) Implementation of Q-learning (Oleg 7h)
4) Implementation of Policy based model (Alisher 6h)
5) Implementation of supervised leaning agent (Oleg 6h, Alisher 8h)
6) Implementation of solver (Alisher 2h)
7) Designing NN architectures, training, and further improvements (Oleg 6h; Alisher 8h)
8) Presentation, midterm report and final report preparation (Oleg 13h; Alisher 13h)