Paweł Biegun[1]

Opiekun naukowy: Sławomir Herma[2]

WYKORZYSTANIE SKALOWALNYCH TECHNIK TRENINGU MODELU Z KATEGORII REINFORCEMENT LEARNING W GRZE W STATKI

Streszczenie: Zgodnie z tym co zaprezentował Tamburro A.[3] sieć neuronowa wykorzystująca Q-learning do gry w statki z jednym krążownikiem (trzy pola długości) mogła osiągnąć optymalny wynik jedynie na planszy 7x7 i mnieszych. Celem tego artykułu jest osiągnięcie optymalnych wyników na większych planszach.
Słowa kluczowe: Q-learning, Reinforcement Rearning, Convolutional Neural Netowrks, Tensorflow, Keras, Battleship

UTILIZING SCALABLE REINFORCEMENT LEARNING MODEL TRAINING TECHNIQUES IN THE GAME OF BATTLESHIP

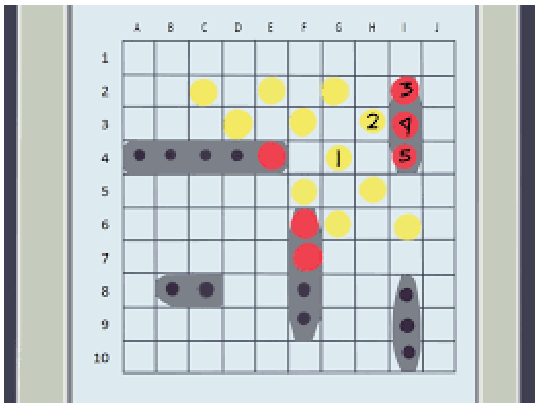
Summary: As demonstrated by A Tamburro[4] a neural network utilizing Q-learning do play the game of battleship with a single cruiser (ship of length of three) was able to demonstrate optimal performance only on boards 7x7 and smaller. The goal of this article is to achieve optimal scores on bigger boards
Keywords: Q-learning, Reinforcement Learning, Convolutional Neural Networks, Tensorflow, Keras, Battleship

1. Literature summary

The problem of creating an artificial intelligence capable of playing battleship has been studied before using both algorithmic1 and deep learning2 solutions. The algorithmic solutions focus on calculating the probability density map through analysis of all possible ship configurations. This approach isn't sustainable in the early phases of the game as it would take around 40 hours to calculate the correct probability density map with the algorithm designed by C Liam Brown. His algorithm was able to process around 270000 boards per second. To achieve a time per move of around

[1] Uczeń klasy 3A – V Liceum Ogólnokształcące w Bielsku-Białej,
[2] dr inż. Katedra Inżynierii Produkcji, Akademia Techniczno-Humanistyczna w Bielsku-Białej
[3] Tamburo A, An artificial intelligence learns to play battleship Medium Philadelphia 2020
[4] Tamburo A, An artificial intelligence learns to play battleship Medium Philadelphia 2020

5s we would have to improve his algorithm 22000 times which seems unlikely to be possible in an environment with limited computing capabilities that could be dedicated to a single game. In the early and middle phases of the game an algorithm randomly sample configurations to check which results in an accurate approximation of probability density map. This is why a machine learning solution that has a constant move generation time for any board, regardless of how many ships may have been sunk, can outperform an algorithmic solution. It should be added that an algorithmic solution utilizing an approach that bases its decision only on the generated probability density map that is the average of all possible configurations is blind to the fact that humans don't place their ships randomly on the board. Humans have the tendency to place their ships far closer to the edge than would be expected purely based on probability3. A deep learning agent could account for this over the course of the game much more easily than an algorithm.



Picture nr. 1 Sample battleship game (source: https://www.wikihow.com/Play-Battleship)

2. Creating an environment

The environment for this project is quite simple and there is no need to discuss it in detail. The agents are taking their turns in parallel. If both were to make a winning move in the same turn player 1 will win due to the order of winner evaluation. It should be noted that the code presented in the kaggle notebook isn't parallelized due to only one physical core being present, There would no point parallelizing it since the cpu utilization is already at 100% when the environment computations are performed. There was no opportunity to develop this model on a more powerful machine but if someone were to attempt further development with more powerful hardware it should be parallelized as it is the main bottleneck of the model training. After model makes it's prediction it is applied by shooting at the point with the highest probability out of those that *haven't* been shot yet. As we don't have a chance there to punish the model for predicting the score that has been already shot we simply demand the model to present all the square where the enemy ships are and the information about the ones that have been shot and thus detected is copied over.

3. Model architecture

The board is presented to the model as a 10x10x2 matrix. The depth channel (length 2) on position 1 it has information whether we have shot there or not. On position 2 there is information whether there has been an enemy ship sunk on this square. The information about our fields and enemy shots isn't presented to the model. In further research the model designed to play against humans should be presented with information about enemy shots and their sequence as it probably would impact the ship distribution. The 3d convolutional layers with filter of size 3x3x1 have been used in order to capture the relationship between the different patterns that form on the boards as the game is being played. The dense layers are later used to capture the relationship between the patterns found in the board. The final layer is a dense layer with 100 connections which after applying the softmax activation function form a probability distribution.

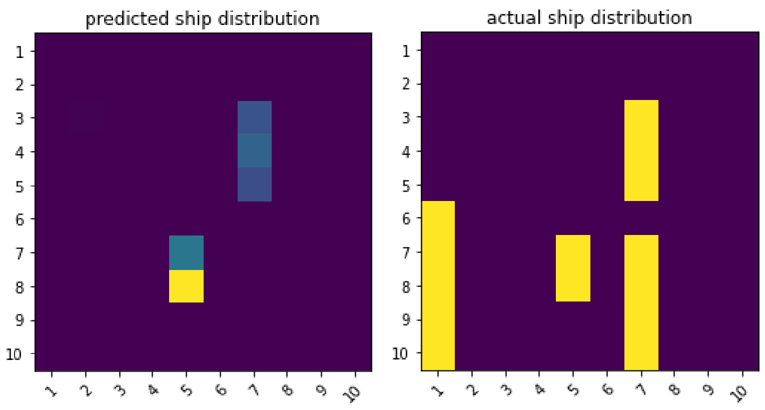
4. Training process

The process of training the model is based on playing a set of games in parallel during a training cycle and repeating the training cycle a set number of times. The model that has been evaluated has been trained for 20 training cycles. A training cycle consists of the model predicting the probability distribution of enemy ships (including the ones that have already been detected), the square with the highest probability out of those that haven't been shot is marked as shot and the state of the game being evaluated to see if the game is over. Training on 256 games in a training cycle has been found to allow for convergence in optimal time. The model sees the result of each move 10 times. After 10 training cycles the average move count calculated on 1000 games is 77.6 and after 20 training cycles it drops to 77.5. This result is well within the expected error and clearly shows that the model ha

5. Results

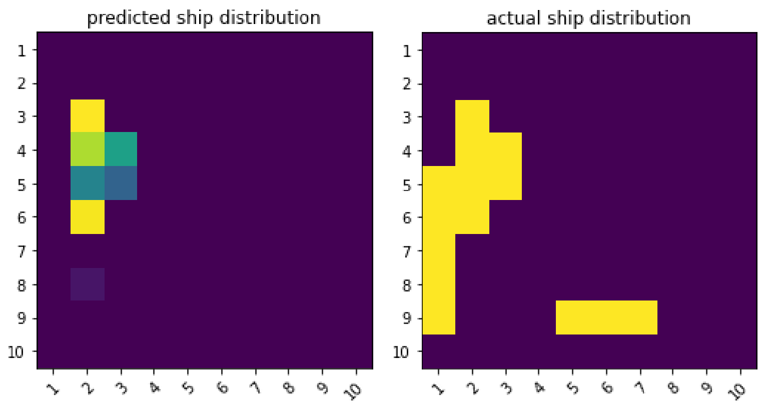
The model has achieved an average game length of 77 rounds which a lot better than making random moves but far from the more basic algorithmic solutions that usually end the game in around 65 moves and really far from top algorithmic implementations that end the game on average in 43 moves. There is certainly a lot room for improvement. Further improvements to this project should focus on parallelizing the environment code to accelerate training in environments with more robust cpu computing capabilities. Training for longer intervals of time should also be beneficial. Unfortunately, although the Kaggle Jupyter Notebook provides sufficient GPU capabilities only two logical cores and the 30 min idle timeout make it not feasible to run longer workloads there. The following examples have been obtained on round 40 it can cleary be seen that the model knows where the shot enemy ships are and that the differences between the non-shot squares are very small.

Example 1:



Picture nr.2 Example of model prediction after 40 shots

Example 2:



Picture nr.3 Another example of model prediction after 40 shots

AUTHORS

1. V Liceum Ogólnokształcące w Bielsku Białej, biegunpawel900@gmail.com
2. dr. inż., Katedra Inżynierii Produkcji, Akademia Techniczno-Humanistyczna w Bielsku-Białej, sherma@ath.bielsko.com

LITERATURE

1. C. Liam Brown, battleship, web app
2. Tamburo A, An artificial intelligence learns to play battleship Medium Philadelphia 2020
3. Keras documentation https://keras.io/about/
4. Kaggle https://www.kaggle.com/

CODE

1. https://github.com/Anakin100100/battleship-CNN-/tree/main