Deep Q-Learning for Classic Games

*Implementing Artificial Neural Networks with Tensorflow

1st Niklas Schemmer
Institute of Cognitive Science
University of Osnabrueck
Osnabrueck, Germany
nschemmer@uni-osnabrueck.de

2nd Dominik Brockmann
Institute of Cognitive Science
University of Osnabrueck
Osnabrueck, Germany
dobrockmann@uni-osnabrueck.de

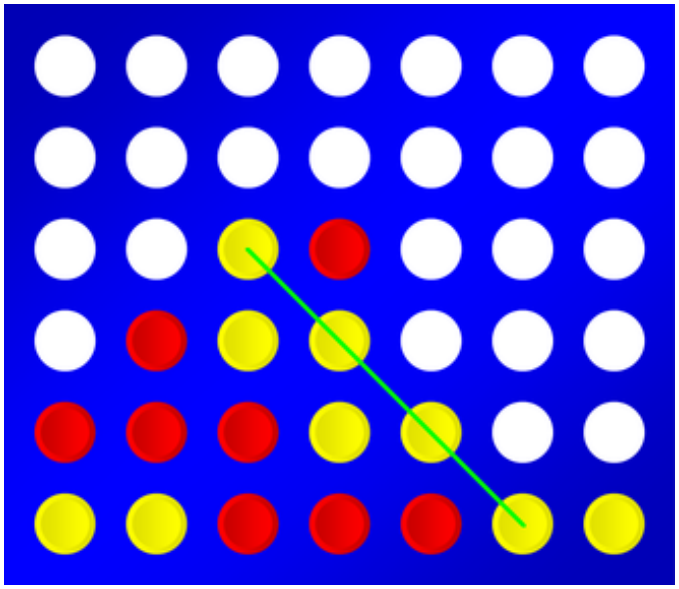

Fig. 1. Representation of the Connect four game, from [15]

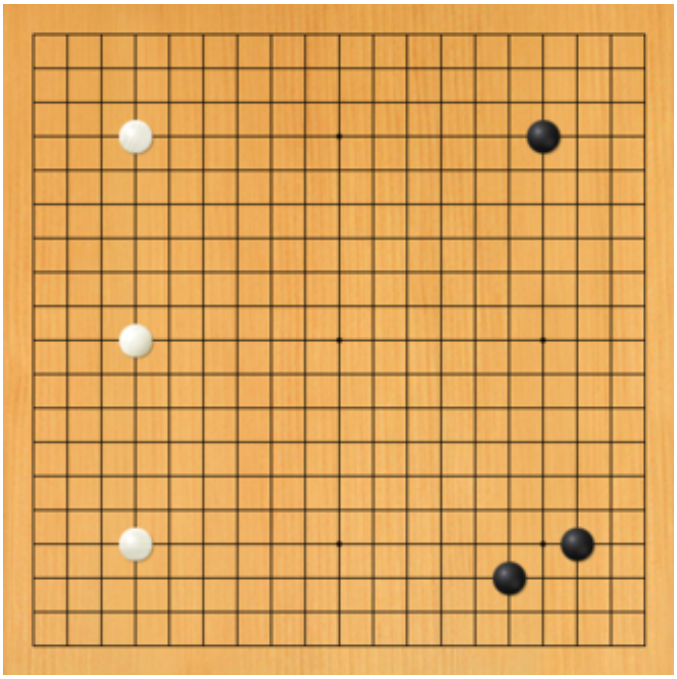

Fig. 2. Representation of Go board game, from [16]

I. Introduction and Background

A. Project

The implementation of this project can be found in our GitHub Repository[1].

B. Connect Four

Connect four is a game which consists of a vertical grid of 7×6 positions (a matrix). It is played with two players, each player has 21 coins of one color, each of them placing a coin one after the other. Coins can only be dropped into the columns that are not full.

Winning player is the one that first connects four coins in a row, either vertically, horizontally or diagonally. A draw can be achieved if none of the players could create a row of four coins and all columns are filled with a coin. An interesting feature of the game is that the coins are not freely positioned on the grid but fall from above into the lowest free position. [4] [1]

[1]https://github.com/niklasschemmer/deep_q_connectfour_go

C. Go

Go is an abstract strategy board game originally from China played by two players, the main objective of the game is to surround more territory than the opponent. In the recent days it has gained popularity through the machine learning community.

The board consists of a grid of vertical and horizontal lines. Two players, using either white or black stones, take turns placing their stones on a board. The goal is to surround and capture their opponent's stones or strategically create spaces of territory. Once all possible moves have been played, both the stones on the board and the empty points are tallied. The highest number wins. [3] [2]

D. Algorithms and Approaches

In the table I you can see different approaches and their outcomes.

In our project, the model we use is a neural network with several fully connected dense layers with ReLu activation, trained using Deep Q learning. The network is fed with the flattened game board and returns an action to choose for the next step. The agent starts playing against itself without having any prior knowledge and only get positive or negative reward for wins/losses.

II. Methods and Models

The method of choice to approach this problem was deep Q learning. Q learning usually works with a Q table, that saves for each state of a problem the most likely Q values for each action that can be taken. Q values therefore state the probable future reward of all future actions that can be taken. Some problems might simply be too complex to save a Q table for each possible state. For the game connect four that would mean saving a table for each combination of all colours of coins on every field. A more memory efficient approach is now to replace the Q table with a neural network that predicts our Q values for every action that can be made in a current board state.

The training procedure works the same in both games. The observations from the game boards are flattened and fed into the network. The network is then trained by predicting future rewards of actions. These are then compared to the next states and the rewards that would be predicted from the next actions. The higher it values actions that also have high Q values in future actions, the better the prediction was and the other way around. To get a proper weight between future rewards and current rewards, there is a gamma value that states how important future rewards are. This also leads the agent to try to find strategies that lead to a win, but also try to prevent the opponent from making good moves in current states.

In our states we used the epsilon greedy strategy to get a good balance between exploitation and exploration. This is important, because if the agent while training would only select the highest Q values in every board state, this would mean that no new strategies or board states would be found and we would end up in a local maximum very soon. The strategy works by simply choosing between the highest Q values the network calculated or a random action. In the beginning of the training we very often take random actions, but as we proceed we more often take the highest Q valued action, to find out good strategies. [7] [8]

A. Model

We will describe the design of our models like this $50 \rightarrow 75 \rightarrow 25$. Every number stands for a hidden layer that is fully connected to every other layer and the number is the amount of neurons in that layer.

The model for connect has the following design: $100 \rightarrow 75 \rightarrow 50 \rightarrow 25$. We chose this, because the first layers need to gather the whole complexity of the game. At some point the game needs to learn that every second input from the board are its own coins and once a coin is dropped by the opponent, its own coin we land on top of it. In our 1D input array this would be a totally different index then. This complexity can be broken down in the lower layers to lead to a strategic decision for the actions.

Go has a bit different model design: $200 \rightarrow 150 \rightarrow 150 \rightarrow 100 \rightarrow 100$. As Go the actions to be made and the situations that lead to a win are much more complex. Sometimes you want to catch a coin of the opponent and in other situations you just want to sit back and do not want to make a move. This requires a much deeper structure of the network.

The activation except of the output layer is the Rectified Linear Unit (ReLu) function. The output is activated by a Linear activation. The ReLu is a more performant activation function than Sigmoid for example and it is helpful especially in Linear Regression networks.

III. Training

To train our model, we use the self play method. This means the model controls both players and tries to beat itself. While the agent plays, states of previous game situations are saved in pairs with their next state. This then builds up a replay memory of previous game situations. Every step the game then samples from these previous situations, to let the agents predict the Q values for those again and improve afterwards.

In table II you can see the parameters that we used for the training of our model.

IV. Results and Analysis

The performance of the model was tested by letting it play against an agent that simply plays random steps, but at least avoids illegal steps. At the start of the training the model also makes random steps, as all weights are initialised randomly.

Firstly the model has to start understanding how to avoid illegal moves, because it would lead to an immediate loss and a negative reward. Afterwards the model will start to understand how to make moves that lead to a win and a positive reward in the future.

In figure 3 you can see the connect four models performance against the random policy after 8 hours of training. You can see that the peak of the accuracy was reached around the middle of the training and then started to decrease again. The reason might be due to a high learning rate or an exploration phase that was too short. The model for connect four quickly achieves a good accuracy, as the probabibilty of making illegal moves in early states of the game are very low and it very early starts to make random moves that lead to a successful game. These experiences then help the model to understand how it can win a game very quickly.

In figure 4 you can clearly see that the Go model only started learning which move not to make, but could not learn a strategy to win yet. The accuracy only gathered around 0 after being -1 in the beginning.

An accuracy of -1 means that our model lost every game against the random policy. This happens because the random policy at least knows how not to make an illegal move and but for our untrained model in Go it is very likely to make an illegal move at least once until the end of the game.

TABLE I
DIFFERENT ALGORITHMS AND APPROACHES USED FOR CONNECT FOUR AND GO

	Algorithm/Approach	Description/Outcome	Reference paper
1	Multilayer Perceptron architecture with one hidden layer using the Backpropagation algorithm	Five different network architectures were created using this approach with different number of neurons in the hidden layer for each type. Results: The network was shown to be efficient to learn tactical knowledge from saved games	Neural connect 4 – A connectionist approach to the game by Schneider, Marvin & Lus, Joo & Rosa, Joao. (2003)
2	Minimax Algorithm	The original Minimax algorithm was extended by adding the Alpha-beta pruning strategy to improve the computational speed and save memory. Results: The AI player is said to perform exceptionally well and outperform the human, but when tuning the number of depths at the minimax function from high (6 for example) to low (2 for example), the AI player may perform worse.	Alpha-Beta Pruning in Mini-Max Algorithm – An Optimized Approach for a Connect 4 Game by Nasa, R., Didwania, R., Maji, S., & Kumar, V. (2018)
3	AlphaGo	A new approach to computer Go that combines Monte-Carlo tree search with deep neural networks that have been trained by supervised learning, from human expert games, and by reinforcement learning, from games of self-play. The first time ever that a computer program has defeated a human professional player.	Mastering the game of Go with deep neural networks and tree search by Silver, D., Huang, A., Maddison, C. et al.

TABLE II
DIFFERENT PARAMETERS USED IN THE MODEL

Name of the parameter	Description	Values used in Connect Four	Values used in Go
Batch size	size of the batches that are sampled from the replay memory	64	128
Gamma	defines how important future rewards are compared to current rewards	0.985	0.9995
Start/End/Decay	Start/End/Decay value of the Epsilon greedy strategy	1/0.05/0.00002	1/0.01/0.00002
Memory size	size of elements in the replay buffer used	100000	500000
Epochs	number of cycles to train the neural network with the training data	200000	200000
Learning rate	the rate to adjust the weights of our network, that in turn controls the how quickly or slowly the network learns	0.00001	0.000008

The accuracy of 0 means that both the model and the random policy win equally often, this means that our model learned how not to make illegal moves, but it does not have a strategy to win, so the placements of the coins our model makes are just as random as those from the random policy. The reasons might be of a different nature in this case, the design of the model could have been just wrong. A deeper/shallower or wider network could perform better. Another reason could also be that the training duration was simply too short.

V. CONCLUSION

We were able to train the agent on the connect four game with good performance. It learned some strategies, like building up a win situation and it knows how to prevent the opponent from winning as well. Here some hyperparameter optimization and different model types would have led to much better results.

However, we reach the conclusion that neural networks is not the best strategy to solve connect four, as there are other algorithms like minimax that can provide more optimal results.

Using the knowledge from connect four on the other side was a good start to improve the model on Go, because the training procedure is mostly the same for those two games.

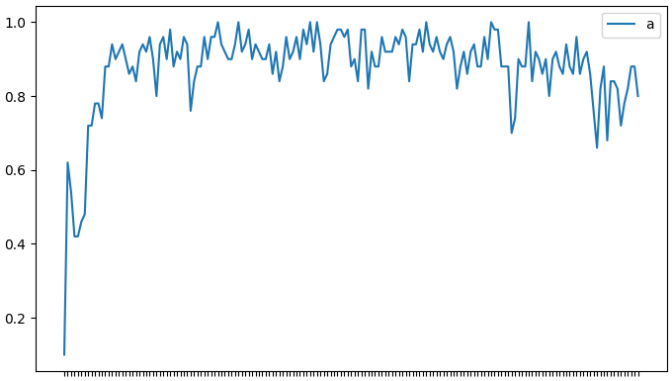

Fig. 3. Accuracy graph of the connect four model

The problem in Go was that the only way to end the game and not to get a negative reward is if the model finds out how not to make a move in case both agents do not make a move at the same time step, the game will end and one of the agents will be rewarded with a positive reward. This can only be achieved by a high exploration rate, where the model tries many different moves and they often start ending the game

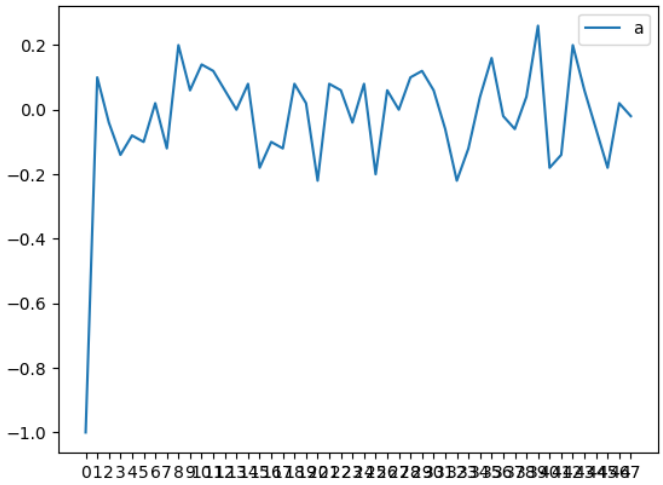

Fig. 4. Accuracy graph of the Go model

randomly. The random policy of the Go training was therefore changed to not make a move every second step. Having much more time and resources to train would have led to a better result in Go as well, because in the used training time the model simply did not have enough time to explore the whole action space in every situation to adjust its Q values.

REFERENCES

[1] Wikipedia contributors. (2022, March 23). *Connect Four.* Wikipedia. https://en.wikipedia.org/wiki/Connect_Four

[2] Wikipedia-Autoren. (2002, May 13). *Go (Spiel).* Go (Spiel). https://de.wikipedia.org/wiki/Go_(Spiel)

[3] AlphaGo. (n.d.). AlphaGo Introduction to Go. https://www.deepmind.com/research/highlighted-research/alphago

[4] Schneider, Marvin Lus, Joo Rosa, Joao. (2003). Neural Connect 4 – A Connectionist Approach to the Game.

[5] Nasa, R., Didwania, R., Maji, S., Kumar, V. (2018). Alpha-beta pruning in mini-max algorithm–an optimized approach for a connect-4 game. Int. Res. J. Eng. Technol, 1637–1641.

[6] Silver, D., Huang, A., Maddison, C. et al. Mastering the game of Go with deep neural networks and tree search. Nature 529, 484–489 (2016). https://doi.org/10.1038/nature16961

[7] Schmalz, L. (2021, December 15). *Playing Connect 4 with Deep Q-Learning – Towards Data Science.* Medium. https://towardsdatascience.com/playing-connect-4-with-deep-q-learning-76271ed663ca

[8] Gupta, A. (2021, December 14). *Deep Q-Learning with Tensorflow 2 - Aniket Gupta.* Medium. https://medium.com/@aniket.tcdav/deep-q-learning-with-tensorflow-2-686b700c868b

[9] PettingZoo: Gym for Multi-Agent Reinforcement Learning https://arxiv.org/abs/2009.14471 https://www.pettingzoo.ml/

[10] https://www.cs.cornell.edu/courses/cs1110/2021fa/materials/style/statement-comments

[11] https://www.voigtstefan.me/post/connectx/

[12] https://www.tensorflow.org/agents/tutorials/0_intro_rl

[13] https://www.tensorflow.org/agents/tutorials/1_dqn_tutorial

[14] https://mathworld.wolfram.com

[15] https://commons.wikimedia.org/wiki/File:Connect4_Wins.png

[16] https://commons.wikimedia.org/wiki/File:Fuseki.png