Danmarks Tekniske Universitet

Pong - Exploring Cooperation with Multi-Agent Deep Q-Learning

Introduktion til Intelligente Systemer - 02461

Christoffer Grauballe, Jonathan Tybirk, Viktor Dyrby Larsen
s234801, s216136, s234957

Abstract
This report aims to examine how differing available information and reward structure affect AI cooperation in a cooperative version of Pong, the classic Atari game. The AIs were trained using deep Q-learning.
We found that the reward structure and available information had a clear effect on how the models played, but it was not shown that either added complexity definitively made the models play better. Further research is needed.

January 19, 2024

Contents

1 Introduction

Most recent research regarding AIs playing games have been on zero-sum games such as Chess, Go and others alike. In this report, we instead put focus on cooperation and how two AI agents can learn to play together. Specifically, we compare whether information about the other agent and a small reward for the other player hitting the ball will positively impact their effectiveness.

With the use of deep Q-learning, we test this question by comparing how well instances of four different models play a cooperative version of Pong where score is rewarded for hitting the ball. In this game, it is beneficial for both players to keep the ball in play. These models represent the combinations of having or not having information about the other player, and of gaining a small reward from the other player hitting the ball.
We hypothesize that agents with information about the other player and a shared reward function will play the game consistently better than those without.

2 Theory

2.1 Q-learning

Like traditional (tabular) Q-learning, a deep Q-network computes the quality Q of state-action pairs in a Markov decision process. At any game state, S, the agent takes an action, A, to reach a new game state, S', and receive an immediate reward, R. The quality of a state-action pair, $Q(S, A)$, is equal to this immediate reward plus the expected reward of taking the best action at the next game state multiplied by a discount factor, γ[1]:

$$Q(S, A) = R + \gamma \cdot \max_{A'} Q(S', A')$$

Here, Q of a state action pair depends on Q of the next state, which is dependent on the next state again, and so on.
A γ value between zero and one will mean that future rewards are valued less than immediate ones, and that far-future rewards are valued less than near-future ones.

The agent will at each state choose to explore (i.e. take a random action) or choose the action with the highest expected reward. In traditional Q-learning, expected rewards for all state-action pairs are initialized to be a predetermined value (usually zero), and are then updated every time the agent takes an action.

This approach works great for environments with a small number of states, as it is possible for every state-action pair to be explored and the expected values to converge until the optimal policy for every game state is known.
In more complex environments however, it becomes infeasible to explore every possible game state.
With deep Q-learning, we do not need to.

2.2 Deep Q-learning

In Deep Q-learning, $Q(S, A)$ is estimated by a neural network.[2] The network is continuously trained to shift its weights and biases to minimize the difference between its prediction of $Q(S, A)$ and the 'true' $Q(S, A)$ computed by the formula. This difference is called the *loss* of the prediction.

[1] Mikkel N. Schmidt, Fundamentals of Artificial Intelligence, 11 - Reinforcement Learning, 2023.
[2] David Silver, Deep Reinforcement Learning, 2016

It might seem peculiar to call this calculated $Q(S, A)$ any more 'true' than the model prediction, as the calculation relies on $Q(S', A')$, which itself is an estimate by the model.
The key to understanding this is to consider 'terminal states'. These are game states where a given action will lead to an environment reset, like getting a game over in a video game.
As there is no next game state S' to reach, there is no possibility of getting future reward no matter what action A' is taken. The value of $\max_{A'} Q(S', A')$ must therefore necessarily be zero.
The quality of a state-action pair that lead to termination will thus not depend on model predictions and be exactly equal to the immediate reward it receives.

A 'true' $Q(S, A)$ can therefore be calculated without relying on model estimations. Minimizing loss *will* therefore train the model to accurately assess the quality of terminal states.
Once terminal states are correctly assessed, states just before a terminal state can also be accurately calculated from the formula. And so, the model will also learn to assess these states correctly, and then the states before these. This process will repeat until the model makes good quality estimates of any game state.

2.3 Batches And The Loss Function

To find a balance between using computational power on updating the weights and simulating the environment, a *batch* and *mini-batch* is used. The batch is an array of the data that the model is fed, and a mini-batch is a random sample from the batch of some predetermined size. This way, the model can be trained on a lot of data at the same time, instead of having to update its weights on every single data point.

The average loss is then found by applying a mean square error function to the data in the mini-batch, as seen in the following loss function, where n is the size of the mini-batch:

$$\frac{1}{n}\sum_{i=1}^{n}(Q_{pred_i} - Q_{true_i})^2$$

The weights are then updated with this single value instead of every data point separately, saving computations, fastening the learning process.

3 Methods

3.1 Implementation of Pong in Python

We recreated the Atari game Pong in the python module Pygame. In our version, a player receives a score-point when they specifically deflect the ball with their paddle. Deflecting the ball also makes it speed up, and it will move too fast for a bad pass by the other player to be reachable after just a few passes back and fourth. This is to necessitate cooperation in the form of good passes between the players.

3.2 Implementation of deep Q-learning

Two separate instances of a given model controls the left and right paddle respectively. The paddles can every game step stay put, move up or move down. The actions taken by these agents are recorded in the batch, and the models are then trained on the batch data (see model specifications). As the 'state' at each step, an agent's model is fed the position of its paddle (y-coordiante) as well as the position and the velocity of the ball (as x- and y-coordiantes).
In this report, we compare four models: Blind-Base, Sighted-Base, Blind-Nudge, and Sighted-Nudge. Blind-Base is as described above. 'Sighted' models are fed an additional input in state, the position of the other agent's paddle. 'Nudge' models get an additional small reward when the other agent passes the ball. See table 1 in the appendix.

The batch saves the entry (S, A, R, S', T) at every step (see section 2.3). T is a boolean, set to 1 if S is a terminal state. As described in section 2.3, mini-batches are continuously randomly sampled from the batch and fed to the models. To ensure the training data gets better with the agents, the batch has a maximum size of 100,000, continuously replacing entries from old games as newer data is collected.

3.3 Model specifications and training

As described above, only the reward structure and the extra input of the other agent's paddle differed between models.
All models had 2 hidden layers, each with 150 neurons and a ReLU activation function. From 5 (own paddle y-position, ball x- and y-position and -velocity) or 6 inputs (additionally the other paddle's y-position), the model output the predicted Q-values of the 3 possible actions.

The models were all trained for 10,500,000 game steps, getting their weights updated every tenth step to speed up data collection for the batch. This corresponded to roughly 25,000 games. A visualization of the summed score of both agents over time for each model can be found in Figure 1-4.
Epsilon, the likelihood that a random action will be explored instead of the model's suggestion, was initialized to 1. It dropped 10^{-7} per step to a minimum value of 0.1 kept for the remainder (1.5 million game steps) of training.
The weights and biases of the models were tweaked using PyTorch's ADAM optimizer with a mean square error loss function (see section 2.3).

These parameters were the result of trial-and-error throughout our project. The code is public in the GitHub directory, and further information on training and testing models can be found in the README.md file. See appendix.

3.4 Data collection

To determine the effectiveness of the four models, the summed score of the agents were logged over a number of simulated games, and a statistical analysis is done on the gathered samples to analyse their mean and variance.

4 Results

4.1 Sample size calculation

A 95%-confidence interval with width 0.1 would be sufficient to certainly determine an accurate ranking of the models. We ran a pilot study on 100 games to estimate the variance for each model, and then estimated the necessary number of sample games for each model with the following formula:

$$n_{model} = 1.96^2 \cdot \frac{s^2_{model}}{0.1^2}$$

Model	Blind-Base	Sighted-Base	Blind-Nudge	Sighted-Nudge
Pilot-study variance	9.09	21.58	39.03	18.16
Required samples	3492	8292	14994	6975
Actual samples	3500	8300	15000	9200

With the amount of sample games to simulate determined, numerous games were simulated and the data collected.

4.2 Mean and standard deviation

95% confidence intervals for mean and standard deviation were computed with the formulas

$$CI_\mu = \left[\mu - 1.96\frac{s^2}{\sqrt{n}},\ \mu + 1.96\frac{s^2}{\sqrt{n}}\right], \quad CI_s = \left[\sqrt{\frac{(n-1)\cdot s^2}{\chi^2_{97.5\%,n-1}}}, \sqrt{\frac{(n-1)\cdot s^2}{\chi^2_{0.25\%,n-1}}}\right]$$

	Blind-Base	**Sighted-Base**	**Blind-Nudge**	**Sighted-Nudge**
CI of mean	[8.71, 8.93]	[10.02, 10,21]	[10.87, 11.07]	[8.92, 9.07]
CI of sd	[3.23, 3.30]	[4.48, 4.62]	[6.17, 6.32]	[3.66, 3.77]

4.3 Graphs

To analyze the training efficiency of the four models, we plotted every 100th game and the associated score during training to visualize how the models learn to predict the correct quality estimations of each state over time. See Figure 1-4 in the appendix.

4.4 Study findings

Our hypothesis was not confirmed by the results. On average, Blind-Nudge performed the best followed by Sighted-Base. Based on the confidence intervals, it can not be determined which of Blind-Base and Sighted-Nudge performed better on average. The standard deviation on Blind-Nudge was the largest, with Sighted-Base second highest, Sighted-Nudge third and Blind-Base fourth.

5 Discussion

5.1 Sources of error

Intuitively, one would think that training the models on a larger amount of data (and in the realm of deep Q-learning over a longer period of time), would lead to the more complex models gaining an advantage over the others. We are unable to disprove this, however, the graph of the training period (figure 1-4) alludes to the fact that the models yield, on average, about the same efficiency in training.

Another point of interest is whether the environment with a fast-increasing ball-speed increases the incentive for cooperation between the agents, if even at all. Perhaps the environment, more so than incentivizing cooperation, merely acts as an increased difficulty for the agents, rather than forcing them to pass the ball to each other.

5.2 Implications

The results don't seem to indicate a linear connection between Blind and Sighted or Base and Nudge, as changing one of these values doesn't unambiguously increase or decrease a model's performance. The data instead suggests, that there is a certain advantage from Blind and Nudge combined in the same model, whereas the two other models containing Blind and Nudge are the worst performing ones. Therefore, rather than blankly disregarding the model setups, could it be that the Sighted and Nudge aspect to the models is an either/or scenario; Having none of the two leads to a model being too simple, and having both makes the model too complex, requiring more training data before the efficiency in the setup comes to light.
For example, it may be that Blind-Nudge learned to make generally easy passes towards the middle of the game area, seeing that those were more likely to give it the small nudge reward, as the other player caught the ball. The 'Sighted-Nudge' might instead use its extra input to try to predict where the other player will be and fail to do so, actually causing it to play worse. This hypothesis would require further testing to confirm.

While the standard deviations of the models do not say anything about how good they are on average, they allude to differences in consistency. A higher standard deviation hints at less consistency in the games, meaning a model might be better in some game states and worse in others. A low standard deviation hints at the model being almost equally good in all game states. This difference in standard deviation shows that while Blind-Base and Sighted-Nudge are about equal in efficiency, the difference in available information and reward structure anyway has a big impact on how they play.

5.3 Further research

Regarding further research on the subject, if the study were to be repeated, more training time would be applied, for the reason that it might be, over time, possible to more definitively distinguish the advantaged models over the others. It is also possible that different instance pairs of the same model could be trained, to determine to what extent the randomly initialized weights influence the training efficiency of the models.

Another idea for further experimentation is to introduce other cooperation-actuating inputs such as the other players' previous action or the like. On the same note, the reward structure could be tweaked to be further based on incentivizing cooperation, with the addition of reward for the sending the ball in the other paddle's direction or other similar measures. Lastly, the environment could be revamped in a way that encourages cooperation even further than the environment used in this research. Perhaps making the ball speed lower while shrinking the horizontal dimension of the environment between the paddles, maintaining the vertical dimension would make it easier for the players to understand the importance of cooperation, as a badly placed pass would leave the other player unable to reach the ball in time.

6 Conclusion

The data did not not support the research hypothesis and we can therefore not confirm. Our results indicate that the 'sighted' and 'nudge' qualities interact in a complex way. Further research would be needed to definitively specify the relationship between the two variables. In addition, the models should probably be trained for longer to more definitely be able to reject that the Sighed-Nudge model eventually would become more skilled than the other models.

7 References

Mikkel N. Schmidt
* Fundamentals of Artificial Intelligence, 11 - Reinforcement Learning

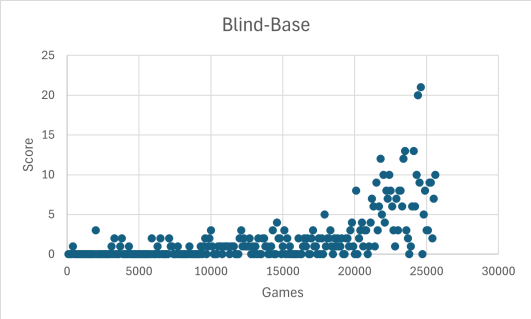
David Silver
* Deep Reinforcement Learning
* 2016
* url: https://deepmind.google/discover/blog/deep-reinforcement-learning/

8 Appendix

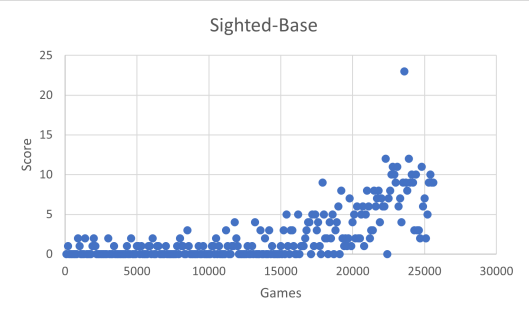
Table 1:

	No other player input	Other player input
No reward for other player hit	Blind-Base	Sighted-Base
Reward for other player hit	Blind-Nudge	Sighted-Nudge

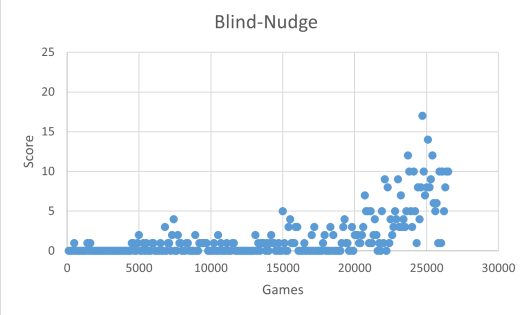
Figure 1-4:



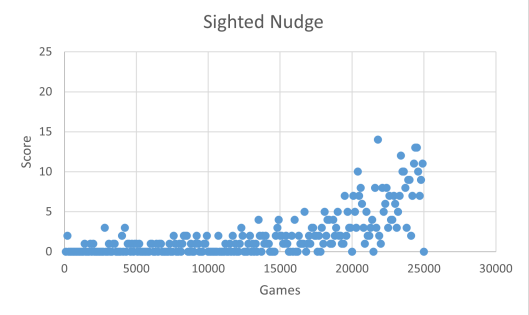
(1) Blind-Base training data visualization



(2) Sighted-Base training data visualization



(3) Blind-Nudge training data visualization



(4) Sighted-Nudge training data visualization

8.1 Code on GitHub

URL: https://github.com/jonathantybirk/Multi-Agent-Pong-Project/

8.2 Responsibility distribution

	Christoffer Grauballe	**Jonathan Tybirk**	**Viktor Larsen**
Introduction	30%	40%	30%
Theory	30%	40%	30%
Methods	40%	30%	30%
Results	40%	30%	30%
Discussion	30%	30%	40%
Conclusion	30%	30%	40%