CS404 Coursework

George Field
Department of Computer Science
University of Warwick
u2013389

***Abstract*—This report investigates bidding strategies for an auction scenario with the goal of acquiring a target distribution of paintings from different artists. Simple rule based bots like the Greedy Bot and Fastest Plan are introduced as baselines. Reinforcement learning approaches are then explored, including regression to approximate the Greedy Bot, a genetic algorithm, and a Deep Q-Learning algorithm with separate policy and value networks trained through self-play. While the Deep Q-Learning approach shows promise, the inability to perfectly mirror the testing environment in training stunted its progress. Ultimately, the Greedy Bot with a minor optimisation emerges as the top performer among the strategies evaluated.**

I. Introduction

Discussed below is the process of designing and building a bot capable of reaching a target distribution of paintings bought from an auction. The auction is selling paintings by 4 different artists. The goal is to acquire a distribution of paintings with 3 paintings each from two of the artists, and 1 painting from each of the remaining two artists.

A bots starts with a budget of 1001 and will compete against other bots in the same situation. Bids are accepted as integers. This is an incomplete information game with an almost continuous state space. Therefore, finding a perfect solution is an intractable problem.

II. Simple Bots

A. Greedy Bot

Greedy Bot is a simple yet effective algorithm that bids on any painting that it needs to reach its target distribution. Since the target distribution contains 8 paintings, and greedy bot will not bid on any it doesn't need, it can bet 125 ($\frac{1001}{8}$) on every painting.

B. Fastest Plan

Fastest Plan bids in such a way that, if there were no other competitors, it would reach the target distribution the fastest whilst acquiring no extra paintings. Because it doesn't bid for paintings it doesn't need, it can also bid 125 for each desired painting.

C. The +1 Optimisation

Since the budget is 1001, and both bots bet 125 until they have 8 paintings, both simple bots have an unused unit. The '+1 optimisation' utilizes this unit by allowing a bot to bet 126 once on a rare artist, defined as an artist that appears in less than a quarter of the next 25 paintings.

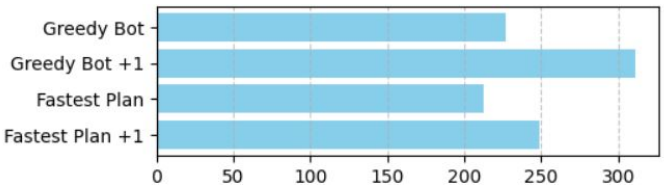

Fig. 1: Result of 1000 auctions run with all simple bots. Greedy Bot +1 performed the best.

III. Measuring Success

Before creating more complex agents, a measure of success must be defined to allow for empirical comparisons between bots.

The bot gauntlet was created. 100 rounds would be played in an auction room containing 3 bots: the test subject, Greedy Bot +1, and Fastest Plan +1. The number of rounds won by a test subject would be used as the primary indicator of that agents fitness.

As a control, the best performing simple bot, greedy bot +1, won 35 out of 100 rounds in the gauntlet.

IV. Reinforcement Learning

A. Why Reinforcement Learning

Traditional techniques like Q-learning would be ineffective for this problem due to the effectively continuous state space. Reinforcement learning (RL) emerged as the ideal solution as it excels in ill-defined problems or when manually specifying a model is difficult [1]. This report details the attempt to use RL to develop a model that, given the current auction state, outputs a bid to maximise the chance of reaching the target distribution.

B. Initial Regression

As a starting point, I implemented a model trained using regression to match Greedy Bot's output for each input auction state. The number of parameters defining an auction state in this model was huge; a 220 dimension array of every bots details and the painting order was input to the network. The regression was performed in batches of 10 auction rounds using the 'Adam'[2] optimiser built into Keras[3]- a python machine learning tool.

Despite the loss dropping to a tiny amount (see figure 2), the bot won 0 rounds when tested in the gauntlet.

C. Improving the Input Data

Minimising a network's input dimension improves training performance for the following reasons:

1) Smaller input dimension ⇒ less network parameters ⇒ faster to predict from and fit model.
2) Large, highly correlated, network inputs train slower when compared to more information dense inputs[4]

In this paper[4], the authors used principal component analysis to reduce the dimensionality of the input to their neural network. This resulted in a large improvement in training speed. Inspired by this, I increased the amount of preprocessing applied to the auction data before it is given to the network.

I reduced the dimensionality of the input through my own reasoning in an attempt to very loosely approximate principal-component-analysis. This liberty was taken as dimensionality reduction is a task humans are intuitively good at[5].

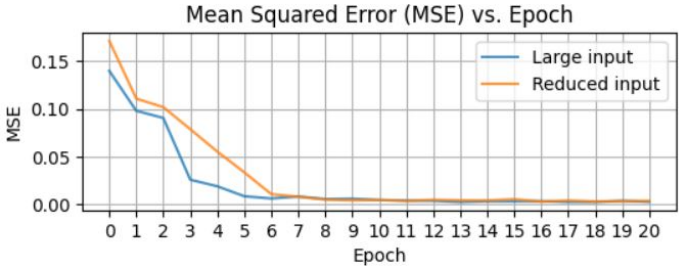

Fig. 2: MSE loss vs. training epoch. Large input and reduced input shown.

Despite having practically equivalent final mean squared error, the regression bot with a smaller input dimension significantly outperformed the larger network, winning 14 out of 100 rounds in the gauntlet. This allowed me to use this bot as a base to improve the network further using genetic training.

V. Genetic Training

In genetic training, a set of bots play against themselves and each is given a fitness score based on how they perform. At the end of each generations testing, the worst performing bots are removed, and the set is repopulated by the remaining bots[6].

The repopulation process is based on evolution. For each space available after culling, two parents are selected at random from the surviving bots and their network parameters are crossed over with a slight mutation applied.

The bots all mutate from the same initial starting point: the Greedy Bot approximator network specified above. I chose to use this network as starting point in an effort to reap the benefits of transfer learning; a technique that has been known to improve training speed for decades[7].

Figure 3 shows the results of the genetic algorithm. The algorithm's parameters were:

- Mutation rate of 0.01 (1% of network parameters mutated)
- Mutation amount standard deviation of 0.05

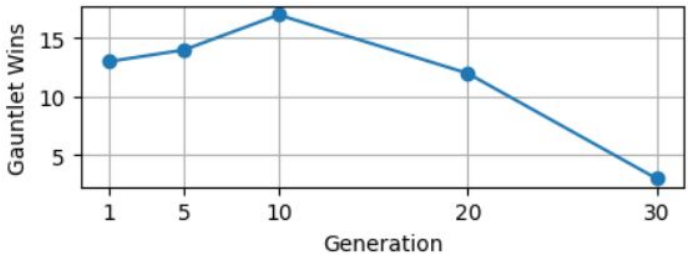

Fig. 3: Performance in gauntlet vs. Generation

A. Genetic Algorithm Results and Analysis

The genetic algorithm showed initial improvement, but then collapsed. The mutation variables are set to very small amounts because the bot's performance is highly sensitive to changes in bidding. This suggests that the initial improvement is most likely just variance from the greedy bot approximator running the gauntlet again, rather than true optimisation progress.

I believe this algorithm was stunted by a severe lack of computing power. Due to performance constraints, per generation a bot would only play in 5 auctions, with each auction being a face-off with 1 other bot.

To demonstrate how poor that is, imagine a bot that beats any other bot in the generation with probability $0.5 + \epsilon$. The probability of that bot losing in the majority of its match-ups is graphed in figure 4.

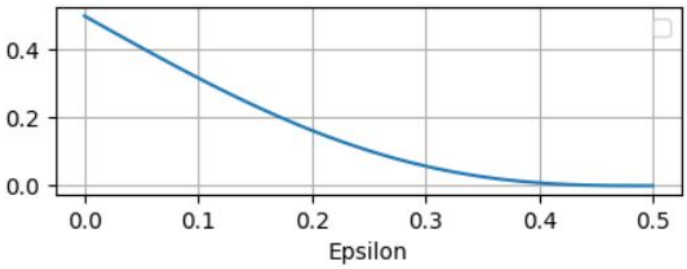

Fig. 4: Epsilon vs. probability the bot wins less than half the auctions

A upper estimate of epsilon for a better bot created through mutation is 0.05 meaning that ≈40% of the time, the much better bot will likely be culled.

VI. Deep Q learning

To improve agents more efficiently, I needed to have a way to score 'positions' in the auction. I chose to create a value network, Q, that is trained to score a state-action pair based on the outcome of games played and the actions taken from each state within a game. This is known as deep Q learning. I was inspired by a paper[8] where a network is trained to score state action pairs for any state occurring in the game.

Unlike the paper, I will not be storing the best action from a selection of states in memory because the state space in the auction is too varied. Instead, I will train a policy model[9] P to approximate $max_{a'}\{Q[s, a']\}$.

An arena is initialised with multiple agents, each with a random policy network. An initial, random Q network is created. The agents bid against each other in an auction. At each bidding round they save their current state s along with the action a (their bid) output by their policy network given the state as an input. At the end of the auction, Q is fitted using backpropagation to output 1 for state-action pairs saved by the winning agent, and 0 for state-action pairs saved by the losing network.

The agent's policy networks are also updated by setting the target from each input state to an approximation of $max_a\{Q[s,a]\}$. This approximation a' is a single step of gradient ascent from the current output action.

$a' = a + \alpha \cdot \nabla Q[s,a]$

Then the agent policy networks are fitted using backpropagation to output the a' instead of a from state s.

A. Deep Q Learning Psuedocode

```
agents ← [agent_0, agent_1, ...]
state_action_pairs_dictionary ← {}
while no auction winner do
    state_batch ← []
    target_batch ← []
    for A in agents do
        s ← get_agent_state(A)
        state_batch.append(s)
        action ← A.P(s)
        target ← action+α · ∇Q([s, a])
        target_batch.append(target)
        state_action_pairs_dictionary[A].append([s, a])
    end for
    state_action_pair_targets ← []
    state_action_pairs ← []
    for A in agents do
        A_win ← 1 if A winner, 0 otherwise
        A.P ← A.P.fit(state_batch, target_batch)
        if A is auction winner then
            for [s, a] in state_action_pairs[A] do
                state_action_pair_targets.append(A_win)
            end for
        end if
        state_action_pairs                                  +=
state_action_pairs_dictionary[A]
    end for
    Q ← Q.fit(state_action_pairs, state_action_pair_targets)
```

B. Deep Q Learning Results

Observing the bids of the output model from each of the tested epochs (epoch being the number of times the agent policy network had been updated) gave me insight into how the agent's policy was developing.

For epochs 30 and below, the agent bet a flat amount. Too high at first, then too low as the policy network learned to prioritise not running out of budget.

Epochs 40 to 100, the agent began to play the game. It placed bids of value between 80 and 140. A behaviour emerged where the agent would bid less as the budget got lower.

At Epoch 125 and above, the agent seemed to have reasoning behind its bids and its strategy had become indiscernible from watching its bids alone.

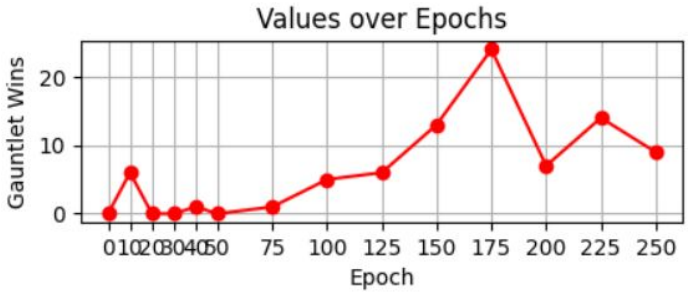

Fig. 5: Epoch vs. number of wins in gauntlet test.

Visible in the results (figure 5) is a positive trend that gives way to varied success. In tested epochs 200 and over, the agent often ran out of money during each auction round in the gauntlet. This method of failure should have been quickly corrected in the training process. I believe differences between the training and test environments meant agents were overfitted to the training environment and stopped improving in the test environment.

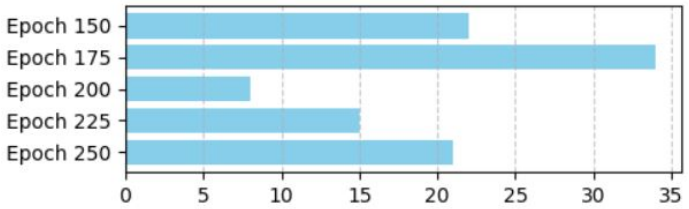

Fig. 6: Epoch vs. number of wins in bot vs. bot arena.

Even against earlier iterations of the agents, the longer training time shows no improvement.

VII. Improving the Training Environment

The failure encountered previously is an instance of domain shift, where the training and test environments differ significantly – a notoriously challenging problem for reinforcement learning algorithms[10, 11].

Instead of training agents against agents modelled by the same Q-network, a set of secondary static agents (not updated) will also be used during training. The hope is that the secondary agents will allow the primary agents to be trained for a more general environment, not just to beat agents playing similarly to itself.

In the first setup, two primary agents trained against three static agents:

- Bot epoch 150
- Bot epoch 175
- Greedy Bot

Greedy Bot was included as a secondary bot to attempt to specialise agents for the gauntlet.

A. Results

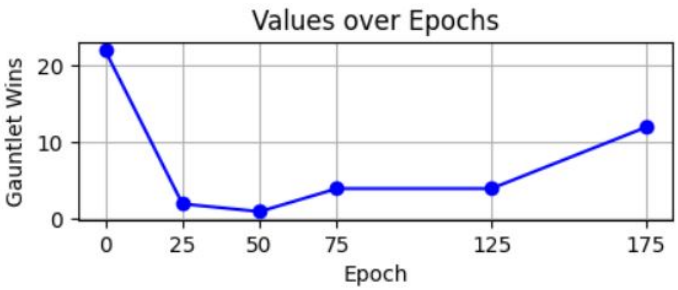

Fig. 7: Epoch of new process vs. number of wins in gauntlet test.

The two primary agents were initialised with the networks from 2 of the agents in epoch 175 of the initial DQL setup. That is why the agent wins starts out so strong at epoch 0.

Despite the positive trend, training was cut off as the agents were stuck in a local maxima where they outbid greedy bot at the beginning and then hoped that they could grab the paintings that they needed with minimal cost when greedy bot didn't bid.

This is a case of overfitting and means the AI would not generalise well.

VIII. Conclusion

Despite extensive efforts to develop reinforcement learning strategies for the painting auction scenario, the algorithms investigated in this report were unable to consistently outperform the simple rule-based Greedy Bot + 1 approach. The deep Q-learning algorithm showed promising initial progress, demonstrating the ability to learn bidding strategies beyond just flat bids. However, issues with overfitting and domain shift between the training and test environments hindered further performance gains.

Attempts were made to improve the training process by introducing static agents. While this seemed to help mitigate overfitting initially, the agents still eventually converged to locally optimal but suboptimal strategies that did not generalise well to the gauntlet test environment.

Ultimately, none of the reinforcement learning approaches were able to surpass the simple yet robust Greedy Bot + 1 strategy. Therefore, the Greedy Bot + 1 remains the top performing agent among the strategies evaluated for acquiring the target distribution of paintings in this auction scenario.

Going forward, further research could explore more advanced domain randomisation techniques to provide better representations of the real life environment. The testing process could also benefit from being made more thorough although this would require increased computational resources.

References

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[2] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[3] François Chollet et al. Keras. https://keras.io, 2015. Accessed: March 26, 2024.

[4] Junita Mohamad-Saleh, Brian S Hoyle, et al. Improved neural network performance using principal component analysis on matlab. *International journal of the computer, the internet and Management*, 16(2):1–8, 2008.

[5] Le Chang and Doris Y Tsao. The code for facial identity in the primate brain. *Cell*, 169(6):1013–1028, 2017.

[6] David J Montana, Lawrence Davis, et al. Training feedforward neural networks using genetic algorithms. In *IJCAI*, volume 89, pages 762–767, 1989.

[7] Weike Pan, Erheng Zhong, and Qiang Yang. Transfer learning for text mining. *Mining text data*, pages 223–257, 2012.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[9] Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, Dumitru Erhan, Chelsea Finn, Piotr Kozakowski, Sergey Levine, et al. Model-based reinforcement learning for atari. *arXiv preprint arXiv:1903.00374*, 2019.

[10] Chhavi Yadav and Léon Bottou. Cold case: The lost mnist digits. *Advances in neural information processing systems*, 32, 2019.

[11] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 23–30. IEEE, 2017.