Chess Anti-cheat Detection Using an Adversarial Network Framework

Osama Dabbousi, Vishesh Jain

January 26, 2024

Abstract

The creation of powerful chess programs in past years has made cheating in the game easy and prevalent. This fact has facilitated a need for high quality chess anti-cheats capable of detecting AI play. While such algorithms exist, they often focus on better-than-human play as opposed to non-human play. To address this issue, in this paper we introduce ChessGAN. ChessGAN is a generative adversarial network whose generator learns to play human-like chess, while its discriminator is trained to distinguish human moves from AI moves. We find that while such a discriminator was capable of distinguishing its generator, this training did not generalize to all human or AI play. However, we did find that the use of this discriminator aided in the training of a generator which played high-quality chess with a move distribution closely resembling that of human players.

Github Repo: https://github.com/OsamaDabb/Chess_anti_cheat

Introduction

As chess engines have become more powerful in the past decades, opportunities for their misuse in casual and professional settings have increased dramatically. As a result, many chess organizations and game hosting websites such as chess.com have had to adapt "anti-cheat" technologies for detecting and penalizing this form of misuse. While these entities rarely discuss the underlying algorithms, it is known that many rely on statistical analysis of move quality as well as machine learning to identify moves that "surpass confirmed clean play." [8] The issue with such algorithms is that while they cannot distinguish general human play, from non-human play. This fact brings to light two rooms for growth in current chess AI: first, the potential for AIs capable of playing high-quality human-like moves, and second, the need for algorithms capable of detecting such AIs to prevent their illicit use.

In this paper we explore both sides of this paradigm. We do this by creating an adversarial system:

1. The discriminator—representing anti-cheat algorithms—will be fed a combination of games from real players and those played by the generator, and will have to discriminate between the human and AI.

2. The generator—representing an engine used to cheat—will be trained to play chess as well as possible while still evading detection by the discriminator.

To this end, we will employ General Adversarial Networks (GANs) to simultaneously train a proficient chess agent and a robust chess detector showcasing their efficacy in enhancing both human-like strategic gameplay and sophisticated detection of chess AIs.

Related Work

Research on chess cheating is relatively scarce. Though large chess hosting websites have their own algorithms, these are never made public. One of the only available resources on the subject comes from Reagen et al. [4], which relies on analyzing move likelihoods conditioned on a players Elo, a rating system which quantifies a player's skill [1]. On the other hand, some articles discuss the inherent risks of judging chess play probabilistically such as Barnes and Hernandez-Castro [3] which emphasizes the inherent uncertainty of such approaches by showing how games that are provably cheat-free had moves or sequences that most methods would classify as engine assisted. The combination of these works emphasizes the significance and demand for new technologies in the field. Currently, publicly available methods of play analysis are underdeveloped and rely on probabilistic models comparing expected level of play to actual level, rather than judging by the nature of the moves themselves.

With regards to chess-playing agents, the state-of-the-art is dominated by tree-search algorithms such as stockfish which use neural network board evaluations in conjunction with minimax or Monte Carlo tree search. However, there is also precedent for chess agents using deep-learning and behavioural cloning. First, Silver et al. [12] showed how convolutional neural networks could be used in grandmaster level chess play. Second, McIlroy Young et al. [9] provides evidence that using behavioural cloning on human chess games creates networks that are significantly better at predicting human-like moves than a traditional tree-search approach. Lastly, Ruoss et al. [11] shows that one can achieve grandmaster level play using imitation learning that uses no tree-search.

On the other hand, the use of a GAN training structure for either chess play or chess cheat detection is a topic that has not been explored to the best of our knowledge. The use of GAN architectures to train a network to avoid cheat detection has been explored in the realm of first-person shooter games (Kanervisto et al., 2023)[6], where the resulting

cheat software was found to evade cheat detection by anti-cheat software as well as human judges. The success of the cheating agent in this work is potentially significant considering the much larger action space for an agent in a first-person shooter game. This result, as well as the findings of [3], both point to discrimination of non-human play being an especially difficult task.

Data and Background

Dataset

Significant volumes of chess data are accessible within the research community, with researchers such as those referenced in [9] and [12] commonly leveraging a dataset sourced from the open-access chess platform, lichess.org [2]. This dataset comprises records from billions of human chess games annotated with player ratings, time control settings, and other metadata useful for training. In this work, a set of $50,000$ games was used, filtering out those with short time-controls (games of less than 180 seconds per player), and those of lower player elo to create a dataset of roughly 3M board-move pairs (b_i, m_i^h). Using stockfish, we also generated a set of optimal moves in each board-state m_i^*, so that the final dataset was $\mathbf{Z} = \{b_i, m_i^h, m_i^*\}_{i=0}^N$

Data representations

There are a number of valid ways to represent the boards and moves for use in training. Commonly, the boards are represented as bitboards, which can be seen as treating each square as a one-hot encoding of the 12 possible piece/color combinations, (i.e. white pawns, black pawns, white bishops, black bishops, etc.) resulting in a $12 \times 8 \times 8$ input. We build on thsis encoding method in two ways. First, we flip the boards such that they are always from the perspective of the player whose turn it is. This explicitly communicates to the bot whether it is playing as white or black. Second, we additionally encode information about which squares are under attack by each player in the given board state. This additional feature augmentation conveys useful information to the network, increasing the input size to $14 \times 8 \times 8$.

With regards to moves, they are often represented as one-hot encodings of the 4096 total possible moves as described by the square the piece is moved from, and the square it is moved to. However, this approach produces significantly unbalanced data, as some moves occur more frequently than others. An alternative encoding which addressed this issue is the use of two separate labels corresponding to the move's from-square and to-square. For example, a move from square 12 (e2) to square 28 (e4) would be encoded by two separate 64×1 vectors which encode those two values. To highlight the benefits of the latter approach, consider a dataset which includes a single sample for each of the 4096 labels of the first approach. When encoded with the second approach, the same data

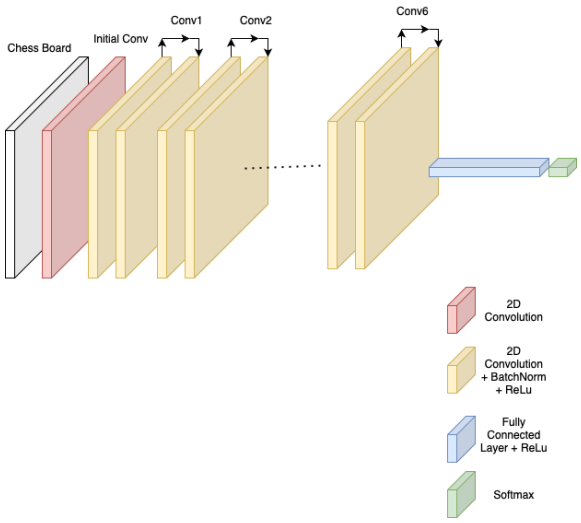


Figure 1

would correspond to roughly 30 samples per label. In this way, the latter encoding reduces the output space and compensates for issues relating to unbalanced data and dimensional sparcity.

Generator and Discriminator Architecture

Convolutional Neural Networks (CNNs) are a class of deep learning algorithms that have revolutionized various fields, including computer vision, due to their ability to effectively learn spatial arrangements of features within the input data. A CNN architecture that has been used in the context of chess is the AlphaZero architecture used by both [12] and [9], which consists of a series of convolutional, batch normalization and fully connected layers. One unconventional aspect of this CNN architecture is the absence of MaxPooling Layers. MaxPooling layers are frequently used in computer vision structures as they reduce the dimensionality of the data while retaining the most salient information—such as edges and depth. In chess, however, boards are of a small and fixed size (8x8 grid), and the spatial relationships between pieces are crucial for determining optimal moves. MaxPooling would reduce the spatial resolution of the board, potentially diminishing the network's ability to accurately capture these relationships, and thus most successful networks in this field elect not to utilize them.

Our final architecture for the generator and discriminator both largely follow the AlphaZero architecture, as seen in Figure 1.

Each convolutional filter is of size 3x3 with a unit stride and zero-padding of size 1. As

the input is of size 14 x 8 x 8, the initial convolutional layer takes in the 14 channels and expands it to 64 channels. Subsequently, the 6 residual blocks each operate with 64 in and 64 out channels. The residual blocks are comprised of two layers of convolutions and layer norms, with LeakyReLU activations. After which, a linear layer takes in the flattened 64 x 8 x 8 representation, (i.e., a 4096 dimensional vector) and outputs a 128 dimensional vector. As a final step, we apply softmax to this vector. Since the first 64 values represent the *from square* and the latter 64 represent the *to square*, we apply softmax to each 64 subvector separately, which allows us to find the best combination of *from square* and *to square*.

Methodology

As previously mentioned, the discriminator and generator are convolutional networks of largely similar architectures. The main difference is that the discriminator takes as input a 16x8x8 vector (a concatenation of the 14x8x8 board and the 128x1 move vector reshaped to 2x8x8) and passes through a sigmoid to output a single value between 0 and 1. The generator takes as input the 14x8x8 board, and outputs the 128x1 move vector. In keeping with findings during research [7] [10], regularization methods such as dropout and weight-decay were avoided, and an optimizer of Adam with $\alpha = 0.0002, \beta_2 = 0.95$ was used for both networks. Additional hyper-parameters included the ratio of iterations with which the discriminator and generator were trained to combat mode collapse.

While the output of the generator in this case is a distribution of probabilities, the human moves are one-hot encodings. This resulted in the discriminator distinguishing between the two output types rather than the moves they represented. A reasonable solution to this issue would be the Gumbel-softmax [5]; however, due to time-constraints we decided to implement a "differentiable argmax" inspired by the reparameterization trick used in variational auto-encoders. This argmax operator would take the product of the generators output vector and a vector of equal shape created to zero-out all non-maximal terms and re-normalize the maximal term to have value 1. For example, given the probability vector

$$x = [0.2, 0.3, 0.5]$$

our equation represents creating the vector

$$x' = [0, 0, \frac{1}{0.5}]$$

and taking their element-wise product to get a one-hot encoding. While addressing the initial issue, this likely leads to issues relating to the variance of the generator's output as well as the quality of the gradients, and represents a potential area of improvement for future works.

As an additional augmentation on the traditional GAN architecture, we implemented a generator loss function that works to incentivize optimal play as well as human-like play. This new loss is defined as

$$L_{G_S}[\theta] = L_G[\theta] - \alpha \sum_m \sum_j m_j^* \log(G(z_j, \theta))$$

where the first term is the traditional GAN generator loss, and the second term represents the categorical cross-entropy loss between the generator's move and the optimal move as determined by Stockfish. The constant $\alpha \in (0, \infty)$ is used as a scaling term to control the influence of each term. This loss was used to train two additional networks, one that used randomly initialized weights and one that used the weights of the behavioural cloning network.

Evaluation

We separate the evaluation into two categories. First, we will evaluate the generator on its quality of play as well as its ability to match the human move distribution. Second, we evaluate the effectiveness of the discriminator in distinguishing between the generator and the human moves.

For the purpose of evaluation we created a number of models. Our first model is "behavioral cloning"—the baseline model—which is a chess agent trained using the CNN (non-GAN) architecture directly on the human moves using a cross-entropy loss. We juxtapose the remaining models against this benchmark to consistently measure the impact of the GAN architecture on training. We test two versions of our GAN architecture: BasicGAN and StockGAN. BasicGAN is a conventional GAN architecture while StockGAN is augmented with the cross-entropy loss on the optimal moves $L_{G_S}[\theta]$, as defined above. In both scenarios, we conducted training on generators with randomized weights, as well as on generators initialized with the weights of the benchmark model. Notably, models initialized with the weights of the benchmark model were designated with the prefix "pre-trained" for clarity.

Play Quality

The first metric we use to evaluate our generators is their win rate against the baseline model. In figure 4, we can see the odds ratio of each generator winning against the baseline model as a percentage:

A primary observation is that the pre-trained versions of the models significantly outperformed the baseline model. It clearly shows that both pre-trained GAN architectures improved play ability over the baseline model. Another important observation is that the

Model	Odds of Winning Against Baseline
BasicGAN	0%
Pre-trained BasicGAN	89%
StockGAN	3%
Pre-trained StockGAN	87.5%

Figure 2

generators initialized with random weights failed to achieve any significant play ability, even after a significant amount of training. This indicates that the GAN architecture on its own is not sufficient to impart the context of chess; however, when also provided an a priori distribution of moves given boards, the GAN is able to fine-tune the weights and substantially improve play quality. The final observation from Figure 4 is that the added cross-entropy loss from StockGAN didn't make a significant contribution in play ability as seen in the decreased win rate.

Distribution Quality

As observed earlier, the pre-trained Basic GAN exhibited the most exceptional performance among all the models. Nevertheless, the objective of this paper is twofold. It aims to develop the most effective generator while also playing human-like. To this end, in this section, we assess the proximity of each model's distribution to that of a human player. In particular, we use the KL divergence, a metric that defines distance between distributions. Thus, to evaluate how well our models compare to a human agent, we calculate the KL divergence between the distributions of each agent with that of the human players. The results can be found in Figure 5:

Model	KL Divergence
Behavioral Cloning	0.0203
BasicGAN	10.3461
Pre-trained BasicGAN	0.02928
StockGAN	0.3855
Pre-trained StockGAN	0.04399

Figure 3

Clearly, simple behavioral cloning had the lowest divergence from the target distribution. This was closely followed by the pre-trained Basic GAN and then by the StockGANs, and finally, BasicGAN had the farthest distribution.

The results above shed light on the fact that a basic GAN—without any chess context and without any aid from stockfish directing optimal moves—fails to emulate the chess

play-style of a human player and also fails to play chess at a decent level (as seen in the previous subsection on play quality).

However, the moment that Stockfish, a "teacher" with optimal moves, is added to the GAN, the KL Divergence drops significantly. Thus, the extra loss term in the StockGAN architecture aided in pushing the generator's play in a much more human-like direction.

Finally, we see that pre-trained models capitalized on their chess context and had quite a low KL divergence. Unlike the previous case, Stockfish was slightly detrimental in the training of the generator. This is reasonable since once the generator is very close to the human distribution, Stockfish's optimal moves begin to differ with human moves causing the generator to stray from the human distribution.

The most interesting outcome of these statistics is the fact that all GAN models had a higher KL divergence than plain behavioral cloning. The GAN architecture was chosen due to its ability to reconcile the generator's distribution with that of humans, however our experimental results indicate otherwise. Combined with the fact that the pre-trained BasicGAN had the highest win-rate against the baseline, it can be inferred that the GAN architecture moved the generators *away* from human play and instead *improved* their level of play. A similar trend held for the pre-trained StockGAN that also had a much better play accuracy but also a more divergent distribution.

To visually compare the distributions, Figure 6 plots the distribution of human moves as well as the distribution of behavioral cloning superimposed on that of the pre-trained Basic GAN:

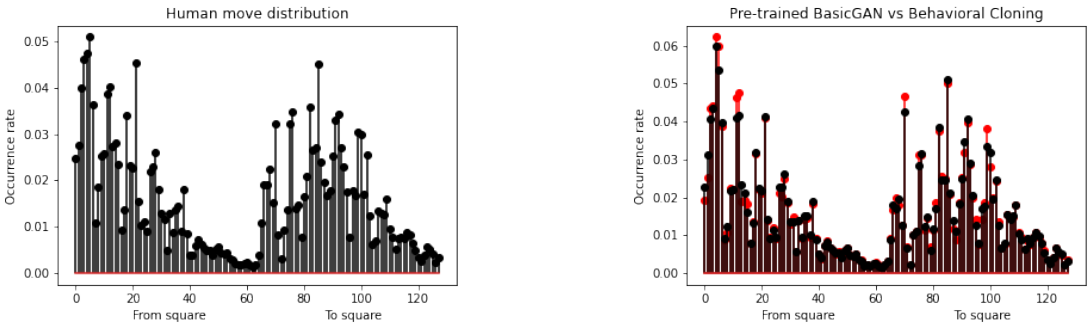


Figure 4: Distributions of Human vs AI Moves

We first note that the distributions of behavioral cloning and pre-trained BasicGAN have a heavy overlap. More importantly, they both significantly resemble the distribution of the human moves. This visually demonstrates the findings produced by the KL divergence statistics. We include the distributions of the other chess generators in the Appendix.

Combining the results from play quality and distribution quality, we find that while pre-trained Basic GAN strayed slightly from the human distribution, it performed considerably better than the baseline model. Overall, the marginal increase in KL Divergence is justified by the superior play of the generator.

Discriminator Quality

As previously mentioned, our interest extends beyond merely creating a chess AI capable of emulating human play to also developing a discriminator proficient in identifying non-human behavior.

For all models, we found that the discriminator was initially able to separate the human and AI moves. One of the drawbacks was that this separation was local to the generator of the GAN. For instance, the discriminator from StockGAN would be apt in detecting the StockGAN generator as AI but would fail to catch BasicGAN's generator (and vice-versa). The conclusion is that given the current discriminator architecture, it can be trained to catch non-human play for select AIs such as Stockfish but doing so across numerous AIs remains unfeasible.

An interesting property noted in the discriminator was its differing ability when given moves vs games. As discussed in the model architecture, the discriminator was trained on individual boards and their moves. When tested on individual move-board pairs, the discriminator sometimes found it difficult to distinguish AI from humans. Conversely, when the discriminator was tested on sequences of board-move pairs (of game-like lengths), it was able to separate the two groups with high probability. This suggests inherent disparities between a human's overall game strategy and that of an AI, indicating promising prospects for cheat detection assessments to be more effectively conducted at the level of entire games rather than individual moves.

Finally, an initial limitation of the discriminator was its display of a memoryless trait—it would lose its ability to distinguish the generator from past iterations after a few epochs. This resulted in oscillatory behavior between the generator and the discriminator. To address this issue, we opted to introduce a substantial set of samples from previous generators to enable the discriminator to maintain recollection of past decision boundaries. This adjustment not only enhanced the discriminator's performance but also bolstered the generator's play quality.

All in all, we found that the discriminators failed to generalize over different chess AIs. Still, we found that the discriminator architecture can be trained on individual chess AIs and when done so, it is most effective in detecting AI play in games as opposed to single moves.

Conclusion

Chess AI is a large and well-developed field whose existence has made it incredibly easy to cheat. As a result, there is a strong need for technologies capable of detecting not just super-human play, but any "non-human" play styles that might be evidence of AI use.

To measure the potential of such an anti-cheat we created ChessGAN, a GAN architecture whose generator learned to play chess while the discriminator attempted to distinguish between human and AI moves. Our experimentation revealed that while the discriminator struggled to distinguish between human and non-human moves universally, it showed promise in learning to identify the particular AIs it was trained on.

On the other hand, the discriminator was able to improve the play quality of the generator when compared to a baseline model. Lastly, the discriminator did not enforce the human move distribution on its generator as effectively as the baseline.

There are potential improvements on our frameworks, including use of the Gumbel-softmax, a larger dataset and a discriminator which takes full games as input. All of which could lead to meaningful gains for the generator and discriminator. Ultimately, there is promise for GAN centered playing agent and anti-cheat architectures.

References

[1] Elo rating (chess).

[2] Lichess database. `https://database.lichess.org/`. Accessed: February 10, 2024.

[3] David J Barnes and Julio Hernandez-Castro. On the limits of engine analysis for cheating detection in chess. *Computers & Security*, 48:58–73, 2015.

[4] Giuseppe Di Fatta, Guy McC Haworth, and Kenneth W Regan. Skill rating by bayesian inference. In *2009 Ieee Symposium on Computational Intelligence and Data Mining*, pages 89–94. IEEE, 2009.

[5] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax, 2017.

[6] Anssi Kanervisto, Tomi Kinnunen, and Ville Hautamäki. Gan-aimbots: Using machine learning for cheating in first person shooters. *IEEE Transactions on Games*, 15(4):566–579, 2023.

[7] Minhyeok Lee and Junhee Seok. Regularization methods for generative adversarial networks: An overview of recent studies, 2020.

[8] Liam McGuinness. Chess: How to spot a potential cheat, 2022.

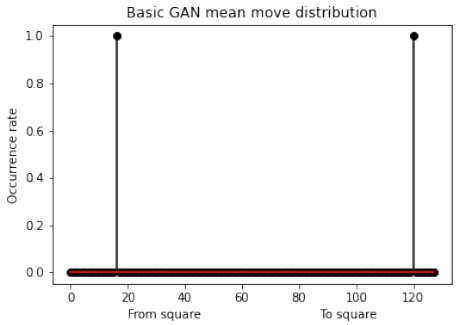
[9] Reid McIlroy-Young, Siddhartha Sen, Jon Kleinberg, and Ashton Anderson. Aligning superhuman ai with human behavior: Chess as a model system. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, KDD '20. ACM, August 2020.

[10] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2016.

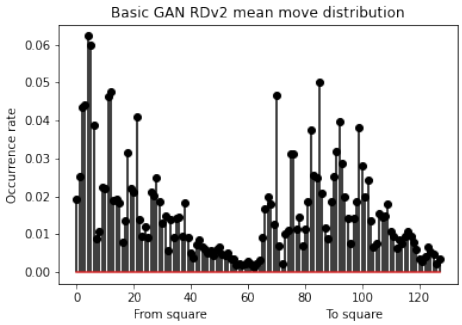
[11] Anian Ruoss, Grégoire Delétang, Sourabh Medapati, Jordi Grau-Moya, Li Kevin Wenliang, Elliot Catt, John Reid, and Tim Genewein. Grandmaster-level chess without search, 2024.

[12] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy P. Lillicrap, Karen Simonyan, and Demis Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

Appendices

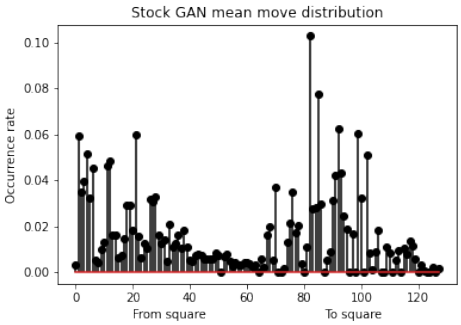


(a) BasicGAN

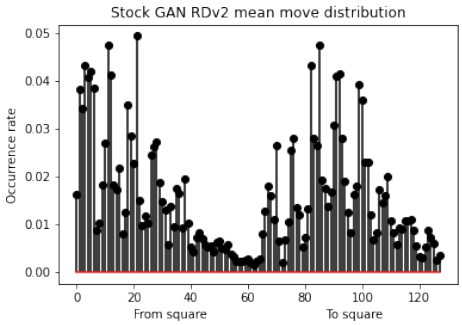


(b) Pre-trained BasicGAN

Figure 5: Distributions of BasicGAN models



(a) StockGAN



(b) Pre-trained StockGAN

Figure 6: Distributions of StockGAN moves