Learning to Play Stratego with Convolutional Neural Networks

Schuyler Smith
Stanford University
skysmith@stanford.edu

Abstract

A considerable amount of progress has been made, especially in the last few years, on developing neural network-based AIs for a number of strategy board games. A large subset of strategy board games have an aspect of chance or hidden information, but to date the effectiveness of convolutional neural networks on such games is mostly unexplored. In this paper we will investigate applications of CNNs to the game Stratego, a popular incomplete information game of this type, and we will compare the effectiveness of two different approaches to using neural networks within a game engine for Stratego.

1. Introduction

Stratego is a popular two-player, incomplete information strategy board game. Unlike some other board games, like chess and checkers, computer AI for Stratego—not just using CNNs but in general—is relatively unexplored. For this reason and due to the inherent complexities of Stratego, the best Stratego AIs are considerably weaker than the best human players today [4]. Hence, it would seem there is potential to apply a new approach to designing a Stratego AI, using convolutional neural networks for the first time.

1.1. Stratego

Stratego is played on a 10×10 board. Each player begins with 40 pieces: one Flag, which must be captured to end the game, several defensive "bombs", troops with ranks 2 through 10, and a "spy". Players may choose the placement of their pieces on their 4×10 side of the board before the game begins. The "neutral zone", the 2×10 region without any pieces initially, contains two 2×2 "lakes", which are impassable.

After the setup phase of the game, players take turns moving a single piece at a time, similar to chess. Each piece may move one square up, down, left, or right each turn, with the exception of bombs and the flag, which are immovable. Pieces may attack other pieces by moving onto their square. The winner of an attack is determined by the following rules, and the loser is removed:

Figure 1. A typical Stratego board game. Note that the pieces are labeled on only one side, so you cannot see your opponent's pieces' ranks. Note also that the ranks in this version are ordered opposite the more common ones described in the text.

1. Higher rank pieces defeat lower ranks (ex. a 7 defeats a 6).
2. If both pieces are of equal rank, both are removed.
3. The special rank "spy" defeats the 10 if it attacks first, and loses all other battles (including against the 10 if the 10 attacks first).
4. All pieces lose to bombs except 3s, the "miners".
5. All pieces win against the flag, the objective of the game.

The game ends when either team has no valid moves (in which case they lose), or when one team captures the other team's flag.

Finally, perhaps the most interesting aspect of Stratego, the ranks of opponents pieces are hidden from the player until they attack or the player attacks them. Hence, a large aspect of the strategy in Stratego involves *bluffing*, and maintaining the odds of certain pieces having certain ranks.

1.2. Why Stratego?

While neural networks have been applied to a number of strategy board games like Chess and Go [3], very little research has been done on games with uncertainty. Uncertainty leads to a vastly larger search space, since modeling and reasoning about a Markov Decision Process is considerably more complex than reasoning about sequences of deterministic steps. This makes traditional game-playing algorithms less effective, and a common solution—Monte Carlo-based methods—is similarly computationally expensive. Hopefully, a neural network will be able to learn patterns that allow it to outperform existing methods.

Additionally, as mentioned above, Human strategies for Stratego are quite complex and the best computers are currently far worse than the best humans, so there is clearly room for improvement.

1.3. General Approach

To train any neural network requires training data. Unfortunately, a good, existing game database for Stratego doesnt exist. I first developed my own non-CNN AI for Stratego to solve this problem, and provide a useful baseline for performance measures.

With a baseline AI to produce realistic game states, good moves to choose from them, and the eventual results of each game, I could train a CNN to either predict optimal moves, or evaluate the desirability of different board states. This will, of course, be discussed in much more detail in the Approach section, below.

2. Background

2.1. Other Stratego AIs

Strong Stratego AIs are difficult to come by, but they do exist. For several years there was an annual Stratego AI competition [4], but it appears to be defunct now. Perennial winners of that competition include most notably Probe [5], which doesn't seem to be actively maintained. While Probe is available for free as shareware, it's only distributed as a Windows binary, cannot be set to play itself, and has no provision for scripting outside its GUI. Hence, while I experimented with it, Probe was not useful while training or evaluating my algorithms. A few other Stratego AIs are playable online but they do not seem to be as strong as Probe, in my tests.

In general, developing a good Stratego AI seems to be very difficult. Probe, the strongest I could find, is about as strong as I am, a human with limited practice. One amusing "paper" I found, written by students at an unknown university, managed to win against a random player only 48% of the time (seriously). It is my lofty aspiration to surpass that in this paper.

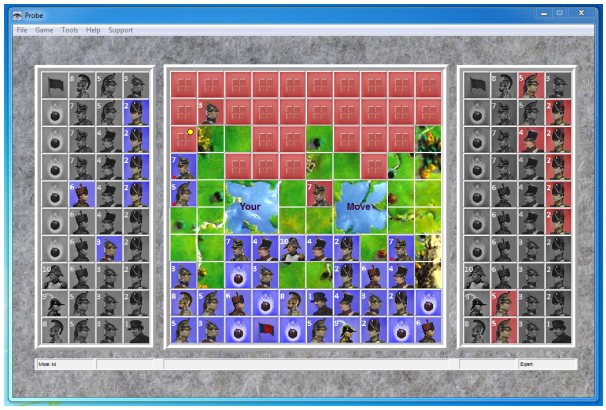

Figure 2. Probe, one of the best currently available Stratego AIs. Only distributed with a Windows GUI.

2.2. Neural Networks for Game Playing

Researchers have been exploring using neural networks for game playing for decades. Chess is undeniably the most studied board game in CS, and was understandably the target of some early research. Chess has movement rules that are more complex than Stratego (but simpler capture rules). As a result, training a network to predict moves in chess is difficult, and instead predicting game state evaluations is more tractable. This was the approach used by [8]. I experimented with both approaches for Stratego.

Backgammon is another strategy game with uncertainty. However, in Backgammon this arises mostly from dice, rather than the opponent, so approaches to reasoning about it are considerably different. One significantly different approach to a neural network-based AI can be found in [7] for Backgammon. In that paper, Tesauro uses temporal difference learning to train a neural network-based evaluation function, similar to the approach mentioned above for chess that I will explore for Stratego below. State-of-the-art Backgammon players to this day make heavy use of neural networks, as well as more traditional precomputed tables.

More recently, applying deep networks to game AI has surged in popularity, just as has applying deep networks to everything else. Bengio [1] is a good overview and example.

One very recent work on applying deep learning to game AI that was particularly relevant and influential was Clark and Storkey's work last year on Go [3]. In Go there is no hidden information, and large amounts of symmetry. Additionally, Go moves are much more easily described, and Go strategy seems to rely much more on pattern recognition than does Stratego strategy, so in many ways Go seems particularly well suited to convolutional neural networks. Some difficulties remain when adapting CNNs to Go, however, and most of these difficulties are shared with Stratego. My general approach is similar to the one described

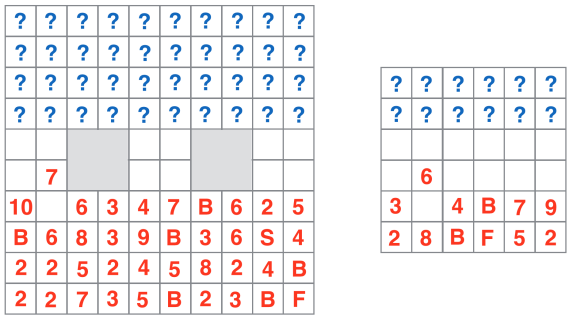

Figure 3. A comparison between Stratego (left) and the mini version I used for some testing (right). Both boards are shown from Red's perspective after making the first move. All of Blue's pieces are initially hidden.

by Clark and Storkey, and in particular my approach to data representation was inspired by theirs. I also found that the types of networks they used (deep convolutional chains with relatively small filters, followed by only one or two fully-connected layers) worked well for Stratego.

The goal of this project then, is to fuse these two fields—the state of the art in Stratego AI, and the state of the art on convolutional networks—in a new and hopefully interesting way. My first approach to playing Stratego is inspired by the one used in [3], while my second approach of predicting board evaluations rather than moves is inspired by [8].

3. Approach

For simplicity, I focused my investigations on the playing phase of Stratego, rather than the setup phase. In human tournaments, there are a small number of classes of initial setups that are widely accepted to be good, and there is little innovation. At the same time, we don't want to train anything into a network that is specific to a certain initial setup, so for each game I use a randomized initial setup, which enforces only a few common-sense constraints, such as placing each flag on the back row.

For a lot of my initial testing I used a smaller, 6×6 version of Stratego with only 12 pieces per player and no lakes. This greatly sped up the games, which made it easier to iterate quickly on different designs. However, all of the experimental results reported below are for the full, 10×10 version of the game.

All code for this project was written in Python, using and adapting (and fixing bugs in) the class neural network implementations.

3.1. Baseline AI

I first developed a baseline AI using a Monte-Carlo algorithm, a well-established technique[2]. It is straightforwardly designed, and intended to be simple: for each possible move in a turn, it chooses a random board state consistent with all known information about the opponent, and simulates a number of games from that state for 10-50 turns. Then, it averages the value of the resulting states, where value is determined to be large if the player wins, small if the player loses, and moderate otherwise, with small bonuses for having more pieces of value left than the opponent. This algorithm is far worse than a human player, as expected, but beats a random player 94% of the time in 100 trials.

3.2. Representing the Board

An essential part of playing well in Stratego is using all the information revealed throughout the game. For example, if you attack a piece revealed to be a bomb, it would be foolish to forget this reveal and attack it again later. Neural networks don't inherently have any way to encode this learned history, so it must be built into the game state representation fed to the network.

To achieve this I use a method similar to the one used in [3]. In particular, I break the game board up into six *channels* or layers, each containing different types of pieces. This preserves the spatial associations between pieces, while enabling the network to treat them differently as it sees fit.

The channels I use are as follows, and also illustrated in Figure 3.

1. The player's immovable pieces (their bombs and flag).
2. The player's movable pieces.
3. The opponent's known, movable pieces.
4. The opponent's known bombs.
5. The opponent's unknown movable pieces (which, notably, must not be bombs or the flag).
6. The opponent's unknown, unmoved pieces.

Layers 2 and 3 are encoded with the pieces' ranks, while the other layers are encoded in binary. The final result, the input to the convnet, is a $6 \times 10 \times 10$ matrix with at most 80 nonzero values.

It's important to note that only first-order inferences are encoded in this way. Successful human players, and the best AIs, are capable of making significantly more in-depth inferences [6]. For example, the convnet would be able to directly infer that since a given piece has moved before it must not be a bomb, but a human may also infer that it has rank at most 8, if the pieces with rank 9 and 10 are known to be elsewhere. This seems fair, since while the the neural network is being given "additional" information, it's

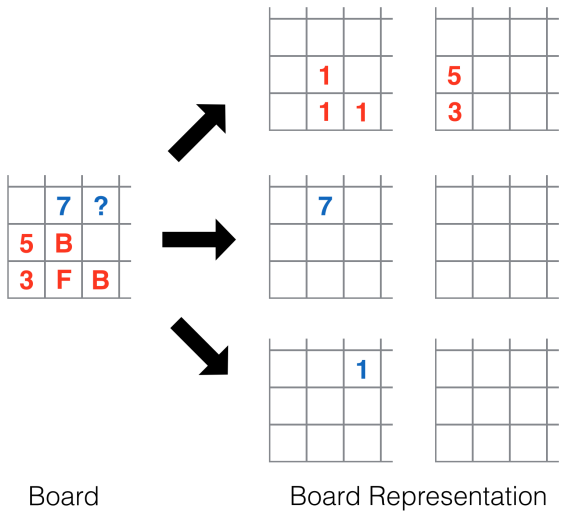

Figure 4. Visualization of the way a 3×3 subset of the board is represented as input to the neural network. The initial board (left) contains five player pieces (red), and two opponent pieces (blue). The player's pieces appear on the first two layers. One opponent piece is known to be a 7, so it appears on the third layer, while the other in known only to be movable, so it appears on the fifth layer. Blank entries in the representation would be 0.

only given information that is obvious, necessary, and trivial to keep track of. It's clear, however, that approaching human-level skill requires deeper inferences, and CNNs are not very well suited to doing this on their own. On the other hand, removing these supplementary channels completely destroys the network's training performance.

Finally, the network must be able to play as either player. To make its life a little simpler, when the network must play as the "top" player, I rotate the board $180°$. Hence, the network always perceives its pieces to be at the bottom of the board. While this shouldn't be strictly necessary, since the pieces are separated on different channels and thus always distinguishable, this should simplify training somewhat since it reduces the variety of scenarios the network must learn to interpret.

3.3. Strategy 1

First, I experimented with training a CNN to predict optimal moves. This is essentially a classification problem where each possible move is a class, and the network must choose the correct class/move for each board. For this, I used a network with several convolutional layers, followed by two fully-connected layers, and a softmax loss function. I don't use any pooling layers, since reducing the dimensions of the board is undesirable with such a small board.

To generate training data, I allowed my Baseline AI to play itself several thousand times, and recorded the moves it chose at each board position. I then used the same gradient descent-based classifier training algorithm used in some of the problem sets to train the network on these board/move pairs. This leads to an extremely unforgiving evaluation criteria, since the network only receives credit if it predicts exactly the expected move.

Finally, the resulting Stratego AI uses a single forward pass through the network each turn to choose a move. Since the network's prediction is not always a valid move, the valid move with highest softmax probability is chosen instead. This results in an algorithm that must be at most as good as the baseline AI it was trained against, but extremely fast.

3.4. Strategy 2

My first strategy has several downsides: since it's being trained to match the moves of the baseline AI the resulting network can't easily exceed baseline performance, and predicting moves directly is inherently difficult and unforgiving, since the number of possible moves in Stratego is quite large, and which moves are valid on a given turn is non-trivial to determine. Hence, I also attempted an alternate strategy: rather than predicting moves directly, the network is trained to predict board evaluations. This complicates the resulting Stratego AI, which now requires many board evaluations to choose a good move, but since forward passes remain extremely fast, this isn't a problem. Additionally, the network now has the potential to exceed the performance of the baseline AI, because they are now decoupled.

For about 15000 positions that appeared in simulated games, I simulated a number of random games to completion to estimate the probability that each board position was a winning position. Then, I trained a CNN on these board/winning probability pairs. Training was performed similarly to before, namely gradient descent with momentum. The loss, rather than a softmax loss, was chosen as the L2 norm of the difference between the predicted and expected evaluations. The network architecture is also very similar to before, except that the final softmax layer is replaced with a single output neuron.

Finally, I use the network in a new algorithm to actually play Stratego. Unlike before, I use a minimax algorithm (with alpha-beta pruning) to search the game tree, and after a certain depth use the network to evaluate the resulting board state. This algorithm would likely benefit from much more fine-tuning than I performed, as there are a number of well-known optimizations to improve the minimax algorithm.

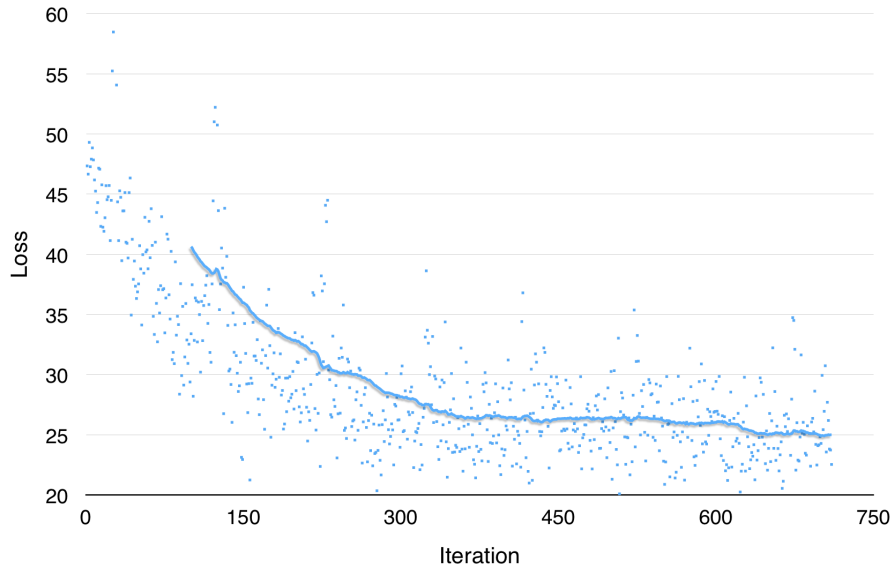

Figure 5. Training loss over time for the best Strategy 2 network. The solid line is a running average, since individual dots are each over a small sample of the training set.

4. Experiments

4.1. Evaluating Strategy 1

Measuring the success of Strategy 1 through training was difficult. Training and validation accuracy remained dismally low (around 5%) throughout, since expecting the network to predict the exact correct move was unforgiving. Validation accuracy would drop further to 0 without strong regularization.

A more relevant performance measure is the network's performance in real games against both a random player and the baseline AI. A comparison of the CNN and the baseline's performance can be seen in figure 5.

	Baseline	Strategy 1	Strategy 2
vs. Random	94/100	87/100	96/100
vs. Baseline		37/100	61/100
vs. Strategy 1	63/100		74/100
vs. Strategy 2	39/100	26/100	
Time per move	$500ms$	$1ms$	$800ms$

Figure 6. Comparison of The Baseline AI and both CNN-based strategies. Strategy 1, predicting moves directly with a CNN, is nearly as good as the baseline while making decisions about 500 times faster. Strategy 2, using a CNN as a board evaluator with Minimax, performs somewhat better than the other two, though no faster.

4.2. Evaluating Strategy 2

Strategy 2 was considerably easier to train, since it produced only a single output. However, I had to take care filtering the training data, because the overwhelming majority of board positions (ex. early in the game) don't have clear leaders, and thus have expected valuations near 0 and noisy. If trained on all of these, the network could do quite well predicting 0 for everything, which is undesirable. To avoid this, I restricted the training data to only boards with at least a 20% difference between either side's chance of winning. I also limited the number of boards considered from each game, since some games could go on far longer than others which led to biases in the training data. After this filtering, I was left with 15000 training pairs from 2500 different simulated games. Note that data augmentation is possible (by, for example, flipping the board), but I did not explore it, because it was just as easy to generate more real data.

The best performing network consisted of five convolutional layers followed by two affine layers (all with ReLU, but no dropout). The training loss over time can be seen in Figure 4. Since we're no longer classifying, accuracy is not meaningful, but the loss function computed on the validation set closely tracked (within 1, or 5% on average) the training loss as before.

Finally, we can compare the performance of all three algorithms, as in Figure 6.

5. Conclusions

Overall, the experiments were successful. Both strategies resulted in CNN-based AIs that were competent, though not astonishingly so. As expected, Strategy 1 could not exceed the performance of the baseline AI, though it got relatively close, while Strategy 2 performed a bit better. It's likely that a heavily tuned, hand-designed evaluation function could beat the one trained in Strategy 2, though that is of course time consuming.

I found that in both designs moderately deep networks performed well. Increasing the filter size beyond 3-5 had no effect except on training time, and increasing the number of filters in each layer beyond about 32 also had negligible effect on accuracy. Adding convolutional layers only improved performance for the first few—after about 5 layers improvements were minor. Hence, the design used for evaluation rested on the large end of the sweet spot between training speed and performance.

5.1. Future Work

There are a number of improvements that could be made to the overall system.

In particular, I believe the board representation could be improved. While it currently encodes some information about what the player knows about the opponent, it doesn't encode anything about what the opponent knows about the player, or any deep insights. This makes bluffing, an essential part of human strategy, impossible, and a number of other deductions more difficult for the network. The representation I used proved to be a good compromise between expressiveness and simplicity (which affects ease of training), but a way to deal with the complexity of a more expressive representation would lead to a superior AI.

There are also other directions in which the project could be taken. Strategy 2 is trained on board scenarios that it could encounter itself. Closing this feedback loop and training on the positions it encounters gives the algorithm an opportunity to learn from its mistakes as it plays, and would be an interesting application of unsupervised reinforcement learning.

Alternately, a number of other games may benefit from convolutional neural networks. Some, like Backgammon as discussed above, already have a number of neural network-based AIs, but it's unclear whether convolutional networks would provide any benefit.

References

[1] Y. Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[2] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4(1):1–43, March 2012.

[3] C. Clark and A. J. Storkey. Teaching deep convolutional neural networks to play go. *CoRR*, abs/1412.3409, 2014.

[4] S. Jug and M. Schadd. The 3rd stratego computer world championship. *ICGA Journal*, 32(4), 2009.

[5] I. Satz. Probe. *http://www.probe.imersatz.com/*, 2014.

[6] J. A. Stankiewicz and M. P. Schadd. Opponent modeling in stratego. In *Proceedings of the 21st BeNeLux Conference on Artificial Intelligence (BNAIC09)(eds. T. Calders, K. Tuyls, and M. Pechenizkiy)*, pages 233–240, 2009.

[7] G. Tesauro. Temporal difference learning and td-gammon. *Commun. ACM*, 38(3):58–68, Mar. 1995.

[8] S. Thrun. Learning to play the game of chess. *Advances in neural information processing systems*, 7, 1995.