Deep Learning: An Inspection of Self-Play and Further Implications

Andre M. Gras[*]

University of St Andrews

E-mail: amg25@st-andrews.ac.uk

Contents

Abstract

This paper discusses Deep Learning and its implications on self-play as well as the evolution of Game AI and the future of Deep Learning. To understand Deep

Learning, one must understand feedforward and recurrent neural networks with only one hidden layer to learn the aspects of backpropagation and architecture. From there, the expansion to Deep Learning is made with the addition of many hidden layers to create increasingly abstract representations. Self-play is a technique of reinforcement learning using Deep Neural Nets that pits a tabula rasa model against itself and iterates successively, converging on an optimized game solution. The methods with which game models do this task varies, as does performance. As we see modern AI delve into Deep Learning, we must take a step back and acknowledge the inherent flaws in the system while appreciating and applying the discoveries made.

Introduction

Neural Networks have been a hot subject in AI for the past half-century, yet while the implications of the field are conflicted, it is safe to say that AI is evolving along with Machine Learning at its forefront in the modern day. Symbolic look-ahead search is still implemented within neural networks but the crux of learning does not come from expert systems, but rather through complex abstract representations and an attempt at feature discovery. Neural networks are composed of inputs, hidden layers, and outputs. Deep Neural networks are networks in which there are many hidden layers (or a layer that can keep internal memory through recurring nodes), that allow for abstraction of complex functions into simpler ones that go building up into the complexity necessary. Deep Learning Neural Nets have been applied to self-play in games, utilizing look-ahead search learning and decision analysis to beat human and computer professionals in Chess, Shogi, Go, Backgammon, and Poker. While the success of the programs vary, they are all taught through reinforcement learning from a blank slate, learning only through self-play. These advancements and developments have lasting implications for the future of Machine Learning and AI alike.

Neural Network Architectures

Neural Networks are an interconnected group of nodes that loosely model how neurons work in the brain, with weights on nodes and their output determining whether or not they “fire.” They are used in myriad Machine Learning applications in attempts to learn from large and act upon large data sets in game-playing, image classification, text recognition, etc. Neural networks are implemented with different architectures to allow for more specialized methods of learning and training.

Feedforward Neural Networks

In feedforward neural nets, The inputs feed into a hidden layer that feeds into an output, but there are no cycles and data progressively moves through the network, thus "feedforward."

For a feedforward network to iterate, one must first randomly intialize the weights. A random uniform distribution is best as it allows for arbitrary conditions that could uniformly speed up the process. One must also intialize the model used, because the network must tune itself to find the desired outcome. Then, one can attempt to calculate the output with the initialized input and random weights. Once the output is calculated, a loss function must be calculated, usually from the sum of squares of the absolute error. The loss is defined as such to increase the emphasis on larger margins of error, forcing a distribution with more small errors as opposed to a few large ones. The goal of the neural network then becomes to minimize this loss function. To do so, one performs a gradient descent or ascent to the loss function. If we can calculate the derivative of the loss function, we can determine the direction we must tune our weights. Much like a ball rolling down a hill, the derivative of the loss function tells us how to tune our weights so as to minimize loss. If the derivative is negative, an increase in weight decreases the loss function. If the derivative is positive, an increase in weight increases the loss function. Therefore, no matter where we initialize our model, the gradients of the derivative roll us back to the minimum. With only one hidden layer, backpropagation is easy as the derivative of the loss function decomposes the layer. Backpropagation with multiple layers performs the same feat, as the derivatives decompose the functions that were called and propagating backwards through the network by destacking the function calls made to get to the output. Backpropagation also allows for fast weight tuning as it decomposes functions using the Chain Rule, allowing for simultaneous computation of all the partial derivatives using one forward pass and one backward pass. Now that we know which direction to tune the weights through backpropagation, the weights must be updated. One knows the rate of error relative to the weight change (the derivative), yet one cannot take large "steps" down or up the hill when performing the gradient descent/ascent or one may end up continually crossing the minimum back and forth. Ergo, the weights are updated using the delta rule, where the new weight equals the old weight minus the derivative rate multiplied with a learning rate that sets the size of the steps. This process iterates over and over until the loss function is 0 and the model has converged on the desired output.

Recurrent Neural Networks

Recurrent Neural Networks are neural networks with the nodes froming a directed graph and do not only feed forward to the next layers. Recurrent neural nets can use their internal memory to process sequences of inputs since temporal behavior is exhibited by the cycles' ability to go back and forth between two nodes. The power of recurrent neural nets stems from this ability to operate over sequences of vectors. In recurrent neural nets, the outputs are dependent on previous computations. Therefore, recurrent networks are most often used in speech and text recognition, where it is important to store order information and parse sequences of data. The name "recurrent" comes from performing the same task for every element of the sequence. Typically, RNNs only look back a couple of steps and are different than traditional neural networks in that parameters are shared across every step, instead of changed at each layer. "RNNs are very powerful dynamic systems, but training them has proved to be problematic because the backpropagated gradients either grow or shrink at each time step, so over many time steps they typically explode or vanish."

Deep Learning Architectures

The term "Deep" Learning comes when neural networks have many layers of abstraction, creating a "deep" model. Deep Learning allows for increasingly better performance on increasingly large data. It does this based on a concept called feature learning. With more layers in a neural network, there is more ability to abstract an incredibly complicated concept by compiling and building it out of many simpler concepts. Layers build progressively abstract representations of data which allows for automatically discovering representations needed for feature detection or classification. This abstraction, however, is the Achilles heel of Deep Learning. With progressive abstraction lies the abyssal issue of understanding. How do we know what the process is even doing? How do we formally represent the understanding and outputs developed by this neural network? Despite these flaws, Deep Learning has provided immense practical application in the field of image, text, and speech recognition, as well as game-playing and a host of other tasks.

Deep Learning, Feedforward and Recurrent

Most Deep Learning applications utilize feedforward architectures, so the learning is the same. Feedforward architectures allow for deep abstraction through many hidden layers and are the simplest of neural net architectures. Feedforward networks are simple but allow for incredible power when combined with deep layers, as a problem can be highly abstracted with specific parameters for each layer. Changing parameters per layer allows for incredibly complex representations to be built using feedforward architecture. Most importantly, the many layers allow for various non-linear transformations and relationships to be modeled, which represents a wide array of classical and practical problems today.

Recurrent neural networks work similarly, but with the added temporal dynamic behavior of internal state. Deep Recurrent Neural Nets allow for the same increase in abstraction and complexity, but in this case utilized for sequences of data and allowing for hierarchical processing on difficult temporal tasks, aiding in naturally capturing the structure of time series. This method makes RNNs great at text and speech recognition and classification.

Self-Play in Game AI

The field of AI has always found two-player games to be a great way to develop AI elements through complex rules and myriad possibilities, as well as requiring some sort of "intuition." In games, there is either perfect information, meaning one has all the knowledge available and all the state of the game is public and viewable by both parties, and there is imperfect information, in which players have their own private state along with a public state viewable by all players. Recently, great success has been found having AI play itself and learn from tabula rasa rather than performing the extensive searches along with domain-specific and expert knowledge.

Perfect Information

The start of game AI began as early as Turing with analysis on how to play Chess. The true beginning point would be the seminal paper by Shannon, "Programming a Computer for Playing Chess," wherein Shannon described the game of Chess as ideal due to the sharp definitions of allowed operations and ultimate goal, the difficulty of the game, and the general idea that it requires an effort of "thinking" or skill that would prove mechanized thinking.[1] As a perfect information game, Chess was massively complex with a large amount of moves and skill required to be a grand master. The crux of Chess skill lies in being able to see moves ahead (look-ahead) and make sound strategies by seeing as far as possible ahead and adapting strategy as game went on. This fact caused Chess to begin as a symbolic look-ahead search game, where the point was to search extensive possibilities and provide domain-specific knowledge to have computers beat out human decision-making. The problem therein lies in the sheer amount of possibilities computers had to check. Early Chess AI could not outperform humans. When alpha-beta pruning was developed, there was a possibility to prune some of the branches off the search trees, allowing for quicker analysis of all opportunities, but humans still knew better from years of experience and intuition from having played the game. Alpha-beta pruning stop completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. It was difficult to determine which order to evaluate the branches of the tree even with these pruned branches from alpha-beta. As computers became more and more powerful, better and faster searches allowed computers to start beating professional humans. Finally, Deep Blue defeated the reigning world champion Gary Kasparov once. Deep Blue was developed with Good Old-Fashioned AI, executing an alpha-beta search with extreme parallelism, searching possibilities with a brute force method.[2] The change came not from theory, but from increased capability of computers. Up through Stockfish, the current leader of look-ahead search Chess AI, Chess, and by proxy all perfect information games, seemed to most effective with the exhaustive, analytical approach of look-ahead searches.

Deep Learning and Self-Play

As time progressed, the study of Artificial Intelligence evolved, and Machine Learning started becoming prevalent. With Machine Learning comes this idea of Deep Neural Networks made to solve complex abstract functions by abstracting out the complexities to simpler functions and building the function back up. Deep Neural Nets work on a more "intuition" based design, as it is based on human neuron firing. In Game AI, the model learns through playing itself and reinforcement learning from tabula rasa, meaning no previous domain knowledge or expert systems.The first successful application of Deep Learning to perfect information games came from the game of Go. Go is like chess where there is perfect information, but one can rotate the board and see the same game state and pieces may be placed wherever. Go had always been seen as the most difficult, however, due to the massive search space and difficulty evaluating positions and moves. The first computer Go Champion was AlphaGo, a development utilizing both deep neural networks as well as tree search. While moving into Deep Learning, AlphaGo employed a tree search that evaluated positions and selected moves using deep neural nets. These neural nets were trained through supervised learning from human expert moves, as well as self-play reinforcement learning.[3] This algorithm beat the reigning Go Champion mixing deep learning and Good Old-Fashioned AI search. However, supervised learning systems trained to replicate human experts can only go so far. Expert data is expensive and unreliable. Not to mention, human expertise models cannot help us in places where humans are not experts or not good at the field or subject. Reinforcement learning allows these models to exceed human capabilities, as they are trained from only their own experience, developing many different strategies and moves. Thus, AlphaGo Zero was born, a completely deep neural network that trained solely through self-play. AlphaGo Zero dominated the previously published AlphaGo through much simpler methods and only one neural network that took the board pieces as input features. AlphaGo Zero did not make use of the Monte-Carlo rollouts. Rather, they incorporated “lookahead search inside the training loop, resulting in rapid improvement and precise and stable learning.”[4] The neural network is essentially trained by a self-play reinforcement algorithm that uses Monte-Carlo Tree Search to play each move. Self-play in Go and solely reinforcement learning is made easier since the board may be rotated without affecting game state, allowing for greater transformations on data (meaning more data) to train the model. Training data was augmented by providing 8 symmetries for each position, and, during MCTS, board positions were transformed using random rotation and reflection.[4] The games of Shogi and Chess are not so forgiving for training. They are less suited to neural network architectures and thus had not been seen to much success with only self-play. However, this changed when the developers of AlphaGo Zero decided to make a general algorithm and program that could outperform the reigning champions of Chess and Shogi. In Shogi, the problem is similar to chess, but more complex due to the larger board and captured pieces being placed back on the board. AlphaZero was developed as an attempt to learn solely through self-play as AlphaGo Zero did, but applied to harder to model games such as Chess and Shogi, which were only recently defeating world champions. AlphaZero surprisingly outperformed the top Shogi and Chess programs, Elmo and Stockfish, respectively.[5] Both these programs use similar algorithms to computer chess programs, based on highly optimized alpha-beta search along with domain-specific adaptations. This result is unexpected as Chess and Shogi are more difficult to model than Go since the games are position dependent, meaning different actions can be performed based on rank (see pawn), asymmetric (castling on queen verus king side), include long-range interaction (queen checkmate across the board), and the action state includes all legal destinations for all players’ pieces on the board (including captured pieces for Shogi). All these features make Shogi and Chess seem solvable solely through domain-specific knowledge and adaptation as well as extensive lookahead search. In AlphaZero, there is a single neural network that is updated continously, rather than waiting for an iteration to complete before deciding the “self-play winner,” as was done in AlphaGo Zero. “Self-play games are generated by using the latest parameters for this neural network, omitting the evaluation step and the selection of best player.”[5] Without going heavily into the details of AlphaZero’s methodology, it was able to employ self-play and beat out all the reigning computer champions in Go, Chess, and Shogi, all using the same model and algorithm, with different exploration noise based on the amount of legal moves. The craziest part, however, comes from the fact that AlphaZero outperformed Elmo in only 2 hours of training, and Stockfish just after 4 hours of training. This method shows that tabula rasa self-play can outperform humans as well as computers in playing these perfect information games.

Imperfect Information

While perfect information seems the best games to employ Artificial Intelligence, imperfect information allows us to see how well computers can intuit and decide based on factors out of control, such as a dice roll in Backgammon or a flop in Poker. Von Neumann, computing pioneer, wished to see reasoning in games such as these with imperfect information, stating "real life consists of bluffing, of little tactics of deception, of asking yourself what is the other man going to think I mean to do. And that is what games are about in my theory." Backgammon cannot be solved using the same brute force tree searching common in Chess since the probabilistic nature of the dice creates a severe branching factor. Poker also has the same problems, adding also the issue of a public and private state where a player must decide an action based on intuition of another player's cards.

Deep Learning and Self-Play

Backgammon saw the first large leap ahead when TD-Gammon was introduced by Gerald Tesauro. TD-Gammon employed temporal difference learning along with self-play reinforcement learning. The goal of temporal difference learning is to "make the learner's current prediction for the current input pattern more closely match the next prediction at the next time step."[6] TD-Gammon was also capable of automatic feature discovery, one of the long-standing goals of game AI. The program progressed to a point where it can match human champions, but does not necessarily beat them every time as AlphaZero has shown. This fact shows imperfect information games are harder to develop AI for. The ultimate lesson comes from the temporal difference learning aspect of TD-Gammon, which showed that learning with delayed rewards could be applied successfully to prediction learning and more general-purpose techniques as well.

Poker is another game that has eluded successful AI for many, many years. Poker is comparable in size to Go, with decision points exceeding 10^{160}. As the quintessential imperfect information game, it is only natural to see what the field has done with respects to this game. DeepStack in 2017 is the first program to defeat pro players with statistical significance in Heads-Up No-Limit Texas Hold'em.[7] "It combines recursive reasoning to handle information asymmetry, decomposition to focus computation on the relevant decision, and a form of intuition that is automatically learned from self-play using deep learning."[7] This is largely due to the fact that imperfect information games require more complex reasoning than similarly sized games due to information asymmetry. In imperfect information games, the correct decision relies on "the probability distribution over private information that the opponent holds, which is revealed through past actions. However, how our opponent's actions reveal that information depends upon their knowledge of our private information and how our actions reveal it."[7] This recursive reasoning disallows reasoning about game situations in isolation, which is the crux of heuristic search in perfect information games. Competitive AI in imperfect information games has usually reasoned the entire game out and produced a complete strategy prior to playing.[8] Although this method can work, it squeezes the situations down into less, abstract situations (from 10^{160} to 10^{14}). As a result, AI had always been behind professional players. DeepStack approaches this problem using the same counterfactual regret minimization that other self-play techniques in this paper has used where recursive reasoning adapts its strategy against itself over successive iterations. However, DeepStack takes the approach a step further. Since it cannot compute and store a complete strategy prior to play, instead it considers each particular situation as it arise in play, not in isolation. So as not to slow down and bog down the look-ahead with too many plays and combinations, DeepStack avoids reasoning about the remainder of the game and substitutes the computation beyond a certain depth with a fast approximate estimate. This method acts as a sort of "gut feeling" about the possible cards in the opponent's hand. The algorithm is trained with deep learning using examples generated from random poker situations and is composed of three ingredients: a sound local strategy computation for the public state, depth-limited lookahead using learned value function (to avoid reasoning to end of the game), and a restricted set of lookahead actions. Until DeepStack, no sound application of heuristic search had been applied to imperfect information games. At the heart of heuristic search is continual re-searching, or "a sound local search procedure is invoked whenever the agent must act without retaining any memory of how or why it acted to reach the current state."[7] Conversely, DeepStack relies on continual re-solving, "a sound local strategy computation which only needs minimal memory of how and why it acted to reach the current public state."[7] Every time the program needs to act, the strategy is re-solved, which means there is no maintained strategy throughout the game. This method differs greatly from what we see in perfect information games, but shows that one can employ deep learning for imperfect information games with huge amounts of decision points as well.

Deep Learning Implications

With the advance of deep learning applications within both perfect and imperfect information games as well as in various fields such as image and text classification and recognition, comes severe implications for the future of AI and how we pursue the evolution of the field. These deep learning models for games are being developed currently, with AlphaZero's and DeepStack's releases being only a year behind present day.

Deep Learning and Symbolic Look-Ahead Search

As we see Deep Learning grow, we wish to see how symbolic look-ahead search grows and is applied within it. Most Deep Learning approaches still train on tree searches, the focus is on learning how to choose the right paths and options. At its core, Deep Learning always must apply some sort of representation of search in AI to be able to access the complex features and tasks we wish to improve upon. The drawbacks of Deep Learning are the same drawbacks of Neural Networks as a whole. It is difficult to see how these representations are mathematically logical or sound or what is going on inside the network. We are still unable to formally represent what an image of a cat is in pure mathematics. Therefore, it is comforting to see elements of Good Old-Fashioned AI continuing to improve through Monte-Carlo Tree Search variants and optimization of alpha-beta searches.

Conclusion

Deep Learning continues to grow and expand into many fields, but we are yet to discover what the full capabilities of the technology are. Leaps are made daily, and, with low barriers to entry and a diverse populous of users, the growth rate will rise quasi-exponentially. With advances in image classification and speech recognition, the machines around us will become increasingly intelligent and adaptive, showing a sense of "intuition" found through tabula rasa reinforcement learning. These machines have learned from their *own* experiences.

References

(1) Shannon, C. E. In *Computer Chess Compendium*; Levy, D., Ed.; Springer-Verlag: Berlin, Heidelberg, 1988; Chapter Programming a Computer for Playing Chess, pp 2–13.

(2) Campbell, M.; Hoane, A.; hsiung Hsu, F. *Artificial Intelligence* **2002**, *134*, 57 – 83.

(3) Silver, D. et al. *Nature* **2016**, *529*, 484–489.

(4) Silver, D. et al. *Nature* **2017**, *550*, 354–.

(5) Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; Lillicrap, T. P.; Simonyan, K.; Hassabis, D. *CoRR* **2017**, *abs/1712.01815*.

(6) Tesauro, G. *Commun. ACM* **1995**, *38*, 58–68.

(7) Moravcík, M.; Schmid, M.; Burch, N.; Lisý, V.; Morrill, D.; Bard, N.; Davis, T.; Waugh, K.; Johanson, M.; Bowling, M. H. *CoRR* **2017**, *abs/1701.01724*.

(8) Burch, N.; Johanson, M.; Bowling, M. Solving Imperfect Information Games Using Decomposition. Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence. 2014; pp 602–608.