Exploring Learning Algorithms in Digital Card Games

Aaron Smith
SOIS:Intelligent Computing B.S.
Rochester Institute of Technology

***Abstract*—Artificial Intelligence(AI) is a rapidly growing field of study, and its application to card games has both practical and theoretical use. Learning algorithms can be used to quickly and efficiently implement AI that can be used in the game itself. Additionally, and possible more important, the study of AI in card games can give insight into human decision making, and give possible models to represent this. This paper first provides an introduction to various learning algorithms, and some considerations on how they apply to card games. Then an implementation of a Recurrent Artificial Neural Nets (RNN) that simulates playing Hearthstone is discussed. It was found that an RNN is a good tool to use to make decisions in Hearthstone. The learning time is efficient and improved from only a few minutes of training. The implementation is simple, and does not require any changes to the core algorithm when adding features.**

***Index Terms*—Recurrent neural network, Artificial Intelligence, Hearthstone, Machine Learning**

I. Introduction

This paper explores the the theory behind learning algorithms, why they are useful, some common implementations, and how they can be used to solve the problem of developing a Hearthstone AI. The first section of the introduction describes the Hearthstone card game, and how it is transformed from a game into information that can be process by an algorithm. The next section details Game Theory, the main motivation for attempting these types of problems. This is followed by a description of Machine Learning, its major paradigms, and the strategy of selecting algorithms. Introduction section D, details four potential candidate algorithms for the AI: linear regression, Support Vector Machines, Decision Trees, and Artificial Neural Networks. The remaining sections are dedicated to describing the implementation details of the selected learning algorithm, Recurrent Artificial Neural Nets, and its results.

A. The Game

Hearthstone is a digital trading card game developed by Blizzard Entertainment where two players choose a class, and a deck of 30 cards and attempt to defeat their opponent. Full knowledge of the game is not required for the understanding of the research, and is important only in how it relates to Machine Learning, Game theory, and the implementation. This section will describe how the different parts of Hearthstone relate to the implementation.

1) The Card: In Hearthstone each card has three main stats the cost to play the card, an attack value, and a toughness value. Cards can have additional effects, but for this implementation those extra effects are ignored, a more robust AI could also consider the additional effects, but would require much more time to implement. Additionally cards are limited to two types, spells and minions, Hearthstone has some other card types, but those are ignored again because of time limitations.

2) The Player: Each player has three associated stats, a life total, an armor total, and a mana pool. A player's life starts with and has a max value of 30, if this reaches zero that player loses. A player's armor value is an extension of the life total, and has no limit. The mana pool represents the players ability to play cards, the cost of each cards is subtracted from this total when played, and is refreshed each turn.

3) Game State: This AI assumes it has full knowledge of the deck that both the player's and their opponent's cards are drawn from, but not the order in which they will be drawn. At any given moment the AI can see, the player's hand, and all cards that have been played by anyone. All unseen cards, both decks and the opponent's hand, are represented as probabilities. Each individual card is pre-processed and assigned a numerical value, and a vector of these cards and both players stats represent a state of the game, and is given to the AI. The vector has a set size based on the limitations to the number of cards a player can have in play and in their hand. Each player can at any time have up to 10 cards in their hand, and seven cards in play.

4) A turn: Each turn consists of a series of moves made by the player, this can either be to play a card, or attack with an already played card. In order to play a card, the player must have sufficient mana, and available space to play the card. The exception being that spell cards do not count towards the number in play. An attack can either be made to the opponent directly or to one of the minions they have in play. The attack value of the attacking minion is then subtracted from the target players health and armor, or the minions toughness.

5) Motivation: Later sections will answer the motivation for creating a Hearthstone AI in general, but it is worth mentioning why a simplified game state is a valid representation of the game. Each of the additional effects are that are being ignored can be represented as one of the represented states. For example some extra effects give minions additional attack or toughness, this can viewed as playing a single minion with the larger stats. Some extra effects add additional minions to the

board, and can be viewed as playing more minions with zero cost. In each case the extra effects can be transformed into one of the simplified actions. It then just becomes a problem of mapping all cards to their extra effects, but in every case the same base choices are made and thus the simplified model still accurately represents the decision process of the full game.

B. Game Theory

Game Theory(GT) is the study of mathematical models of strategic interaction among rational decision-makers[1], and was first formalized by John von Neumann in 1928 with his paper *On the Theory of Game of Strategy*[2]. Although its roots are in two player games with full information available to both, e.g. chess, the field has expanded to cast a net on almost every possible area where humans make decisions. Extremely popular is its use in business and economics; GT is used to model the decision process of investors[3] or deciding prices when merging business[4]. GT is also used in political science to model both voters, and politicians, e.g. GT can be applied to the Cuban missile crisis[5]. In the realm of games GT has expanded beyond just simple games like chess to include any time of game.

Hearthstone is included in this, and is part of a sub class of problems called adversarial search, where there is an agent that is making decisions in an attempt to prevent you from finding the optimal solution. Further Hearthstone can be categorized as both stochastic and imperfect. It is stochastic because the order of the cards is randomized, and is imperfect because, your opponents possible moves are unknown to you. Putting this all together Hearthstone can be formulated into a stochastic imperfect adversarial search problem that can both use and contribute to models of similar problems. The business investor is one such similar problem: other investors decisions are competing against your own, knowledge of companies earnings are unknown (at the time the decision needs made), and based on random events. Knowledge gained while developing a Hearthstone AI could help model investor decisions.

C. Machine Learning

Machine learning(ML) is sometimes used interchangeably with AI, but is better viewed as a sub field of the later. ML focuses specifically on the implementation of models/algorithms that learn over time. AI includes this, but is also concerned with broader philosophical questions such as the one made famous by Alan Turing: "Can Machines Think?"[6]. ML models find solutions to a given problem by reacting to percepts from sensors and modifying themselves based on prior/predicted information. They are considered learning algorithms because, much like students in school, their ability should improve over time, and given enough training able to accurately give output for unseen input.

ML algorithms are very powerful and offer solutions to problems that would otherwise not be solvable. This includes problems that are computationally impossible, i.e. a non learning algorithm would need more time than available in the universe, and ones that there is not a good way to represent the data. One example of the later is image classification, it is easy for a Human to look at a picture of a dog and know that it is not a cat. However it not so easy to answer why you know and because of this it makes programming a traditional algorithm to classify images very difficult. ML makes this problem easy and fast, but at a cost traditional algorithms operate much faster, and require no learning phase. This is why the first step when approaching a problem is to determine if ML is the proper solution.

Machine Learning algorithms come in variety, there is not just "one size fits all", and deciding if ML is appropriate relies heavily on the output data: if it can be modeled, and how an algorithm can be trained. Despite their nuances most ML algorithms can be classified by these two aspects of the output. If the data can be modeled as a continuous number than a regression algorithm is appropriate, otherwise a classification algorithm must be used. In regression, the goal is to find a relationship between the input variables, or features, and their dependent variable, or output[7]. The solution is a line that fits the data, and can be used to predict unseen samples. Like Hearthstone not all data can be represented continuously, what card to play does not fall on a continuous spectrum. Instead the algorithm determines how closely the input features relate to a set of output classes[7], continuing the Hearthstone example the classes would be "play" or "don't play". Representing the data in one of these two forms is not a trivial task, and is done using a process called feature extraction.

Feature extraction is the process of applying simple computations directly to the input of the sensors[7]. The complex input data is reduced into smaller features that can be processed by the algorithm. The chosen features have a direct impact on the ability of the algorithm to complete its goal, and thus they need to be a good representation of the data. This is also one of the few times that you can influence how the algorithm works so the features should be carefully selected. When selecting features it is important to select information that is thought to have correlations to desired output. As an example, if you were attempting to predict whether or not a river will flood from rainfall, you could have the features: current water level, average water level, rate of flow, rainfall per hour, length of rainfall. You may decide that the number of fish in the water, and size of tributaries is irrelevant and not give that information to the algorithm. In this case it is likely that tributaries are an important piece of information, and the algorithm will give poor predictions. To avoid these mistakes, or in cases where it is not possible,the entirety of the input can broken into features. When classifying images it is typical to make each pixel, or group of pixels, a feature. This can greatly increase computation time, but avoids Human bias.

If the data can be successfully broken into features, and the output can be represented as regression or classification then the final aspect to look at is how the algorithm will learn. Learning can be broken down into three main categories: supervised, unsupervised, and reinforcement[7]. In supervised

ML training sets of data that have associated ground truth values are used to determine the conditional probability, P(x|y), for future unforeseen data. In other words, a supervised approach will find the probability that an input(x) matches its given output(y). The learning is done by adjusting the algorithm based on differences from its predicted value and the ground truths, it can be modeled as a series of inputs and outputs, Figure 1.

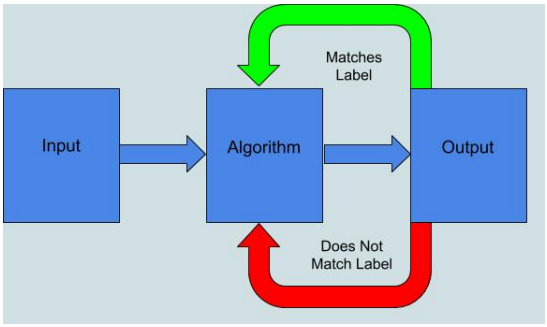

Fig. 1. A sample model of supervised learning.

and This method has the fastest convergence, but requires a large set of labeled training data. Since it is not always possible to get this x->y association, unsupervised learning can very useful.

In unsupervised models only the data for the input is required, and the goal of the algorithm is to find similarity compared to a mean, or in the maximization of a function[7]. This use of the output data is main difference between this method and supervised learning, and can be seen by comparing Figure 2 to that of supervised learning in Figure 1.

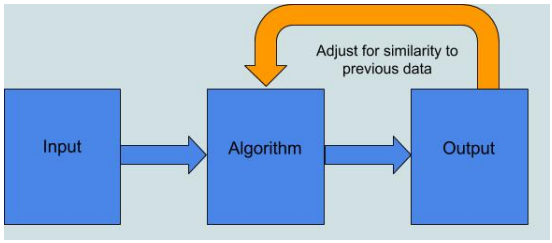

Fig. 2. A sample model of unsupervised learning.

This directly infers the probability of x from the input features. Like supervised learning a training set of data is still needed, but because labels are not needed the training set can be selected as a random subset of all the input data. The algorithm will create groupings of similar inputs from the training set, and those are then labeled. This is useful not only because it eliminates the need to label all the training data, but it can also find patterns and groupings that were not known. Figure 3 illustrates the difference between the results of supervised and unsupervised learning.

Sometimes it is the case that neither of these two methods are sufficient, and it is appropriate to implement reinforcement learning strategies. This requires a small shift in paradigm;

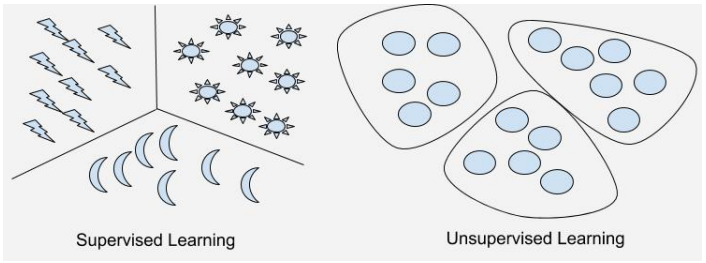

Fig. 3. A graphic depicting how the different types of learning will separate input data.

both previously mentioned learning methods, supervised and unsupervised, make adjustments by comparing a series of inputs to either labels, or previous inputs, respectively. In reinforcement learning the algorithm is modeled as a series of states[7]. The current and previous state are given to the algorithm as input, and the output is used to generate the next state, illustrated in Figure 4.

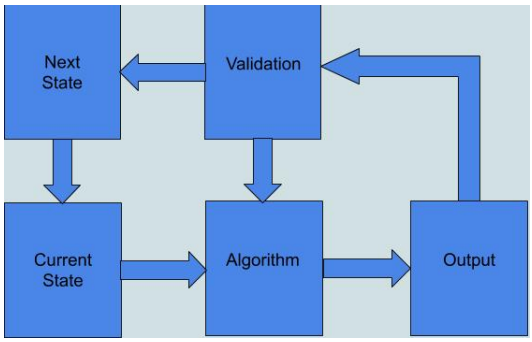

Fig. 4. A sample model of reinforcement learning

The algorithm learns by making adjustments based on the desirability of the new state. This type of learning is particularly useful when the algorithm is modeling time dependent data, each state corresponds to a time step. In games each turn is a new state, and is dependent on the previous and next states.

After careful consideration of the input, output, and learning process an informed decision can be made on whether or not Machine Learning is appropriate. If the problem can be represented as a combination of the described models, and there is not an efficient traditional algorithm a ML algorithm can be chosen to develop a solution model. In the case of Hearthstone, it is possible to extract features from each state of the game, but it is neither clear what information is important nor how to categorize it. So the entire game state is extracted into features. Classification is needed because the output generated is not numerically continuous, and the game is both time dependent and lacking clear labels, thus reinforcement learning is appropriate. The next step is determine which ML algorithm satisfies these conditions.

D. Algorithms

There is no shortage of potential algorithms to generate models, and each has its own set of advantages and disad-

vantages. In general all algorithms take a matrix of input values and matrix of weights to generate a signal output vector. Typically this is done through matrix multiplication of the weights and input to produce the output. The adjustment of the weights is how the algorithms learn, and produces different outputs. Additionally each algorithm has what is refereed to as tuning parameters that can be set to bias the algorithm and improve performance. But perfectly modeling data is a very hard task, and in the cases where it is feasible it is likely that traditional algorithms are easily implemented. This leaves ML algorithms the task of modeling highly complicated data, and thus it is necessary to make assumptions about this data. These assumptions add limitations to the algorithms, but make their implementations, and learning time, much faster. This section will give a brief overview of general ML algorithms, what assumptions they make, and the limitations and advantages they would give to solving the Hearthstone problem.

1) Linear Regression: In linear regression the goal is to find a linear relationship between the input data and the output. The solution will be in the form

$$y = XW + \epsilon \tag{1}$$

where y is output, X is the input, W is the weights, and ϵ is the error. The operation between X and W is matrix multiplication so for each training sample y_i of the vector y the relationship can also be written as

$$y_i = W_0 + W_1 X_{i1} + W_2 X_{i2} + ... + W_m X_{im} + \epsilon_i \tag{2}$$

where m is the number of features[7]. After this is calculation is done an error is calculated from the desired output and the weights are adjusted. As the name implies, this is a regression ML model, so the first assumption it makes is that the data can be represented continuously. In addition to that it assumes that the output's dependence can be formulated from a linear combination of the inputs, and that the error from each feature is independent, i.e. feature 1 does not impact feature 2. Linear regression typically requires labeled data for calculating errors, but it can also be implemented in an unsupervised manner so this is not a limitation. Despite its simplicity this algorithm is very powerful and can has been used for such tasks as supply chain forecasting[8]. However, the assumptions it makes are not appropriate for the Hearthstone problem. It is unlikely that a state in hearthstone can be represented as either continuous or linearly separated, and Independence of the variables is surely not true. This basic model can be expanded to fit non linear data, and consider dependence, but the problem of feature extraction becomes immense, and it is likely that a different model is a better fit.

2) Support Vector Machines: Another approach is the Support Vector Machine(SVM). In this model the output data is in relation to what side of a dividing hyperplane it lies. In mathematics a hyperplane is a space that is one dimension lower than the data represented, in the case of two dimensions this a line, in three a plane, and in general the hyperplane. For an SVM the hyperplane takes the form of

$$WX - \epsilon = 0 \tag{3}$$

where x is the set of points that satisfy this equation[7]. SVM are a model that requires supervised learning, so the output vector is labels of either -1, or 1. The algorithm's goal is find a hyperplane that best separates the training data into the two categories. This is done by maximizing the distance that each x_i from the training set is from the hyperplane onto the side defined by the its corresponding y_i. Figure 5 shows an example of two dimensional data separated by an SVM. Similar to

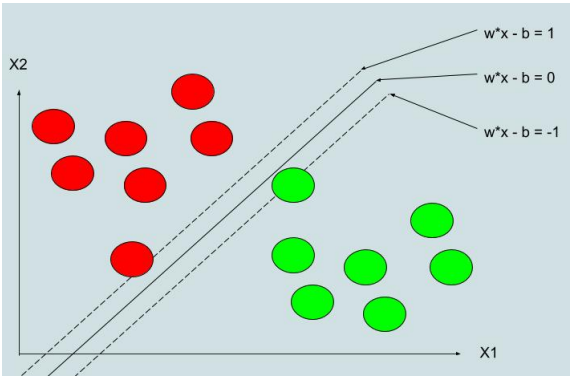

Fig. 5. A sample SVM. The solid middle line represents the hyperplane, and the two dashed lines delineate the classes

regression the SVM is simple, and powerful. In Information Retrieval an SVM can be used to determine ranks for how documents are related to a given query. However its limitations prevent it from being a good candidate for Hearthstone. It's expansion into non linearity is computationally expensive, and even though it classifies data labels are required.

3) Decision Tree: Both linear regression and SVM are effective ML algorithms, but it is hard to extrapolate how they relate to decision making. While both methods are strong at pattern recognition, it is a reasonable assumption that they don't capture the essence of how humans make decision. The decision tree(DT), as the name implies, is a model that attempts to get close capturing that process. A decision tree can be viewed as a series of conditions that are applied to a problem[7]. Like the other models, DTs rely on the updating of the weights applied to an input based on the produced output. Figure 6 shows an extremely simple DT that could be made for Hearthstone. The yes or no lines in the picture, can be adjusted to be probabilities and the choices cane lead to more than two outcomes, and with these considerations a robust decision tree would make an excellent model for a Hearthstone AI. It can in fact accurately model the decision process of a player. However it makes two very important assumptions, that the rules for what constitutes a good decision are known, and they can executed in a reasonable time. The game of Hearthstone is not a solved problem. There are many theories among top players as to what what should be considered when making decisions, but they often conflict, and are constantly adapted as new cards, and methods are introduced. Not only is it most likely too hard to represent the rules, if it were done the decision tree quickly runs into computation problems when attempting to consider the effects of multiple turns. The

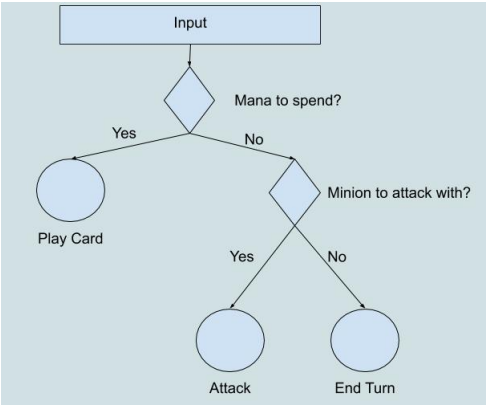

Fig. 6. A simple DT. Diamonds represent choices, and circles represent outputs

limitations of a DT model do not remove it as a candidate, but do pose some issues.

4) Artificial Neural Network: The final class of algorithms attempt to go one step past the decision trees, and not just model how Humans make choices, but model the Human brain itself. An Artificial Neural Network(ANN) is a model inspired from the biology of the human brain. A collection of nodes simulate the network of neurons in the human brain. Each node is defined by an activation function with an input of the weighted sum of all its connected nodes[9]. In an ANN each feature that is extracted is represented by an input node, each of those nodes is then connected to each node in a hidden layer. The updating of these weights all the algorithm to learn, and the activation function determines if the node will contribute to the next layer. A neural net can consist of any number of hidden layers, each connected in the same fashion as the input to the first layer. The final hidden layer connects to the output at which point a decision can be inferred. Figure 7 illustrates a simple ANN with three input features, one hidden layer, and a binary classification. Mathematically this can be represented

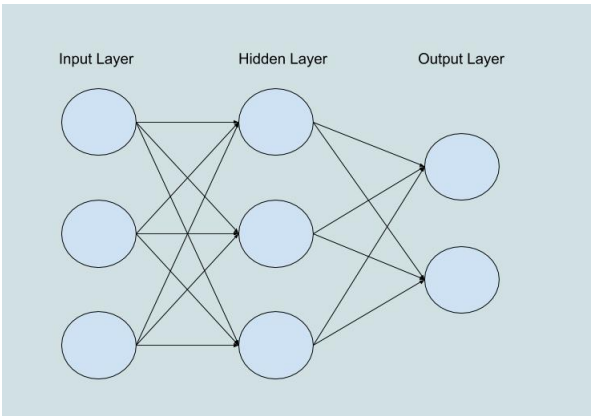

Fig. 7. Example ANN with one hidden layer

as

$$\begin{aligned} a_1 &= f(XW_1) \\ a_2 &= f(a_1W_2) \\ a_n &= f(a_{n-1}W_n) \\ y &= f(a_n) \end{aligned} \tag{4}$$

where X is the input vector, W_i is the weight matrix of a layer, a_i is the hidden layer output, y is the final output vector and f() is the activation function. Once an output vector is found a cost function is used to calculate the error between the predicted and expected output, this function can be implemented using any of the learning methods. The calculated error is then feed to the ANN through back propagation. Starting with the last hidden layer the weights are updated based on their impact on the Cost function, and subsequently each hidden layer is updated based on its contribution. The ANN allows for any type of learning model, which includes reinforcement learning, that output data does not need to be continuous, and any amount of features we do or do not extract from the data can be implemented[7]. Unlike decision trees an ANN does not need a robust set of defined rules, and when expanded to consider time dependence remains relative low computational complexity. Based on these considerations it an algorithm from the ANN family will be the best candidate for Hearthstone. In this case a Recurrent ANN will be used.

II. Recurrent ANN

A recurrent artificial neural network(RNN) is a subset of ANN algorithms that is suitable for modeling problems that have time dependence[7]. In an RNN the output of one state, in addition to back propagation, is used as the input for the next state. Figure 7 shows an overview of a single state transition for the RNN used. The current game state is processed by

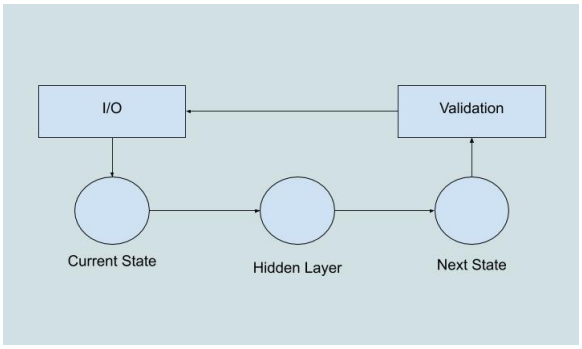

Fig. 8. Figure 7: An overview of the Hearthstone RNN

the hidden layers to produce the next state. This next state is then sent to the validation function to calculate Cost for back propagation, after validation the state is sent to user via the I/O function, and resulting game state, plus any changes by the adversary becomes the current state.

A. Implementation

1) Feature Extraction: Each Game state (input matrix) is represented by the cards, and the player stats, and is represented by the vector:

$$GameState = [PlayerHand, OpponentHand, PlayBoard, OpponentBoard, Playerstats, OpponentStats] \quad (5)$$

Each element in the Game State is a vector of size 10 that contains the information of the individual cards. Both player stat vectors are of size 10 but contain, the duplicate information. It was determined that the information that is most valuable for a a cards is its cost, attack, health, number of turns in hand(or play), and probability this results in the following vector representation:

$$Card = [id, cost, attack, health, turns, probability] \quad (6)$$

The id is a unique identify that allows the AI to distinguish between cards with the same stats for use by the IO function but its not processed by the RNN. When all cards are combined with the player stats the the resulting input matrix is:

$$\begin{aligned} GameState = [& \\ & [card_1, card_2, ...card_{10}], \\ & [card_1, card_2, ...card_{10}], \\ & [card_1, card_2, ...card_{10}], \\ & [card_1, card_2, ...card_{10}], \\ & [health, armor, mana]x10], \\ & [health, armor, mana]x10] \\ ] & \end{aligned} \quad (7)$$

The empty values in the matrix represented with a value of -1, and ignored by the algorithm.

2) Activation Function: The game state is then sent to the hidden layer, that consists of 16 nodes, and matrix multiplied by the weights of the node, and subject to the following activation function:

$$a_i = \sum_{n=1}^{6} e^{X_i n(1-W_i n)} \quad (8)$$

The output of the hidden layer is a vector corresponding to a card in the hand or in play. Each node is selected at the beginning to correspond to either the hand, the opponents board, or the players board. The result from this is that over time, each node will select cards from different learned criteria.

3) Cost and Validation: The output vector of the hidden layer is sent to the validation function, which determines if the selected moved is valid, and how it impacts the board.The move is scored based on the following equation:

$$\Delta Y = \sigma(\Delta life + \Delta attack + \Delta toughness + \Delta hand) \quad (9)$$

and is determined as good if it increases the differential. $\sigma()$ is the sigmoid function defined as follow:

$$\sigma(x) = 1/(1 + e^{-}x) \quad (10)$$

The differential is used as the expected output for the cost function. Cross entropy was chosen as the it represents the change in information from the previous state:

$$Cost = Ylog(a) - (1 - |Y|)log(1 - a) \quad (11)$$

where |Y| is the normalized ΔY and a is the output from the hidden layer.

4) Back Propagation: The new weights are calculated using the following equation:

$$w^{'} = w - [2\sigma(Cost) - 1][ln(x)/60(1 - w)] \quad (12)$$

The activation function requires that the weights be between 0 and 2, and even though the differences are small each iteration after thousands of iterations it is possible to move out of that range. It is therefore necessary to constrain them further by:

$$\begin{aligned} w^{'} &= 0, w^{'} < 0 \\ w^{'} &= 2, w^{'} > 2 \end{aligned} \quad (13)$$

III. The AI

This section describes the flow of information for the AI and how the user sees data.

1) Processing Game Data: First the data has to be gathered from the game client. Every event that happens in Hearthstone is sent to a log file by the game client. A separate function crawls this log file for information, finds when cards have been played, or added to the hand, and keeps track of the current state of the actual game. When the RNN gets to the I/O portion it asks if anything has changed, and updates if it has. Because the game itself is subject to Human interaction it operates much slower than the RNN. Having this task done in parallel allows the AI to do calculations for future turns while it waits for response from the opponent.

2) Calculating moves: After the I/O step the AI predicts a move, as described in the previous section, and validates it. When it returns back to the I/O if the game is in the same state the AI continues, attempts to refine its decision. It does this by simulating a series of moves, based off the cards in play, and the probabilities of likely cards that each player will draw next. The AI slowly builds a tree of possible scenarios. The AI will continue to do this until the I/O function returns that state of the game has changed, at this point the AI will return the move that will result in the best future state, and then start the process over. A sample set of moves may look something this:

$$\begin{aligned} p_1 &= [[], [playA, playB], []][.02] \\ p_2 &= [[playC], [playA], []][.01] \\ p_3 &= [[playC], [attackC, playB][.04] \\ p_4 &= [[playA], [attackC, attackPlayer][.06] \end{aligned} \quad (14)$$

where p_i is a series of turns followed by the resulting , and empty turn ([]) means nothing happened on that turn. Since p_4 has the best score when the I/O changed it would the first move from that list would be returned as the next play, and the the RNN would use that state as its next inputs.

IV. Testing

The ideal scenario for training the RNN would be to input entire games of Hearthstone to allow it to infer information from each decision made, and understand its impact at each step. Because training can take many thousandths of iterations, and each game needs to be played by humans and takes about 30 minutes to complete developing this training data would take a long time to gather. Fortunately Hearthstone is a very popular game, and this data is already collected, however it was not possible to obtain it in time to train the RNN. In lieu of actual games, training was done by generating random games states and allowing the AI to make decision from those. Although the AI will not be able to infer long term strategies it can still generate the best solution for a single state. It was feasible to create a handful of games to use to test the algorithm, and using game knowledge label what the optimal move would be for those turns. The AI was trained using five different number of iterations of updating weights 50, 100, 500, 5000 and 50000, each time starting with randomized weight matrices, and processing random game states. Each iteration value was tested 100 and produced, and the average answer analyzed for its accuracy. For each test case the AI performed exactly the same; it initially selected very poor moves, after both 50 and 100 iterations the selected moves were better, but not ideal. With 500 training iterations the AI selected the optimal move in each case, for 5000 and 50,000 iterations the only difference was the ten minute wait added. The low iteration values had a large variance in the selected outcomes, suggesting that they were in large part selecting random choices, but for 500 and greater a different choice was made less than 10 $\%$ of the time.

V. Conclusion

This paper proposes Recurrent Neural Networks(RNN) as a possible solution to modeling the decisions made in playing a game of Hearthstone. The implementation depicts one strategy that can be used to extract features from the the game state, and a strategy for processing those features through a neural net. It was found that the RNN can be successful at modeling a simplified version of the game, and learn from training data to improve the chosen play.

There are many avenues for future work. First the model can be expanded to incorporate the non basic features of the cards that add a lot of complexities to game, but are needed to be a fully functioning production AI. Second, Decision Trees(DT) were proposed as a potential solution, but overall too complex. A solution that implemented a DT in conjunction with RNN could improve on the ability for the AI to develop long term dame strategies. Finally, this implementation showcases the robustness of RNN, which may prove useful for problems outside of games.

References

[1] R. B. Myerson, *Game Theory: Analysis of Conflict*, Harvard University Press, 1991.

[2] J. V. Neumann, "Zur Theorie der Gesellschaftsspiele", *Mathematische Annalen*, 1928. English translation: A. W. Tucker, R. D. Luce, et al., "On the Theory of Games of Strategy", *Contributions to the Theory of Games*, pp. 13–42, 1959.

[3] C. F. Camerer, "Progress in Behavioral Game Theory," *Journal of Economic Perspectives*, vol. 11, pp. 167–188, 1997.

[4] N. Agarwal and P. Zeephongsekul, "Psychological Pricing in Mergers Acquisitions using Game Theory", School of Mathematics and Geospatial Sciences, RMIT University, Melbourne, Dec. 2011.

[5] S. J. Brams, "Game theory and the Cuban missile crisis", *Plus Magazine*, Jan. 2001.

[6] A. M. Turing, "Computing Machinery and Intelligence", *Mind*, vol. 49, pp. 433-460, 1950.

[7] S. J. Russel and P. Norvig, *Artificial Intelligence A Modern Approach*, Pearson, 2015.

[8] R. Carbonneau, K. Laframboise, and R. Vahidov, "Application of machine learning techiniques for supply chain demand forecasting", *European Journal of Operational Research*, vol. 184, pp. 1140-1154, Feb 2008.

[9] D. J.C. Mackay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, 2003.