12-2020

Approaching Hanabi with Q-Learning and Evolutionary Algorithm

Joseph Palmersten
joepalmersten@gmail.com

Approaching Hanabi with Q-Learning and Evolutionary Algorithm

by

Joseph A Palmersten

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Computer Science

December, 2020

Starred Paper Committee:
Bryant Julstrom, Chairperson
Donald Hamnes
Jie Meichsner

Abstract

Hanabi is a cooperative card game with hidden information that requires cooperation and communication between the players. For a machine learning agent to be successful at the Hanabi, it will have to learn how to communicate and infer information from the communication of other players. To approach the problem of Hanabi the machine learning methods of Q-learning and Evolutionary algorithm are proposed as potential solutions. The agents that were created using the method are shown to not achieve human levels of communication.

Keywords*: Artificial Intelligence, Communication, Evolutionary Algorithm, Gameplay, Machine Learning, Neural Networks, TensorFlow, Q-Learning*

Table of Contents

Chapter Page

List of Figures

Figure Page

Introduction

Machine learning has developed and grown significantly in recent years. With the development of artificial agents reaching superhuman performance in domains of Go, Atari games, and some variants of poker.[1] Artificial agents commonly achieve in areas of zero-sum games. However, an area of artificial intelligence that has seen a lack of development is communication and cooperation between different agents with different policies.

In human society, multi-agent interactions are commonplace in government and economics. The optimal policy of single agents in a multi-agent system depends on the policy of other agents in the same system. In a multi-agent environment, other agents tend to be the most complex part of the environment. Other agents' policies are commonly stochastic, dynamically changing, or dependent on private information that is not seen by others.

Multi-agent interactions of a cooperative nature require the ability to infer the reasoning behind the actions of another agent. All humans are naturally capable of inferring the reasoning for other humans' actions. As an example, a pedestrian is crossing the street. Once traffic has stopped an approaching driver can infer that the other drivers have stopped for a pedestrian even though the approaching driver cannot see the pedestrian. This type of interaction poses a challenge for current artificial agents. For the development of machine learning, Deepmind proposed the game Hanabi as an area of research to address areas of difficulty with multi-agent interactions.

In this work, we will be going over two current machine learning methods Q-learning and evolutionary algorithm. We will then be combining Q-learning and evolutionary algorithm to address multi-agent interactions in the context of an artificial agent that can play the game

Hanabi. The artificial agent will use TensorFlow to control its neural network. For the environment of Hanabi, Deepmind provided the Hanabi Learning Environment to establish consistency in the training of artificial agents.

Literature Review

2.1 Hanabi

Hanabi is a purely cooperative game of imperfect information for two to five players. The structure of the game consists of everyone having a hand of cards of which do not know, but each player knows what cards are in their teammates' hand. As a requirement of the game, players cannot share any information about other player's hands or talk strategy, except during designated times based on the rules of the game. Imperfect information and communication restrictions provide a challenging situation for current AI methods.

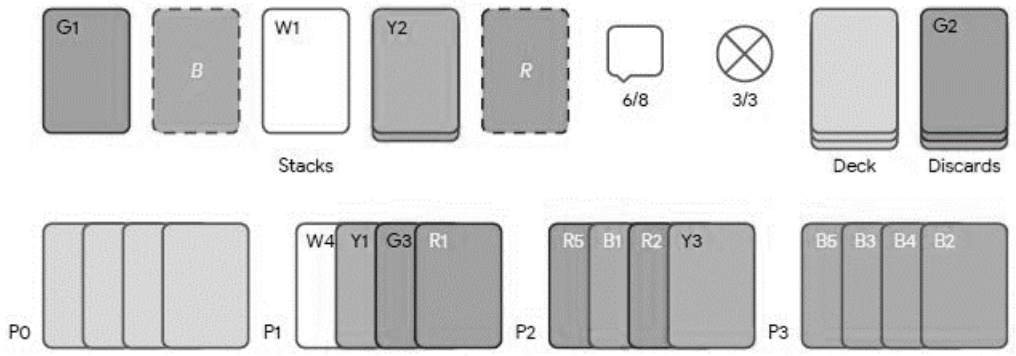
The setup of Hanabi consists of each player starting with five cards, in a game of two to three players. For four to five players the starting hand size goes down to four. Each card has a rank (1 to 5) and a color (red, blue, green, yellow, and white). Each set of cards has three 1s, two 2s, two 3s, two 4s, and only one 5. The sets must be completed in rank order from lowest to highest. The most unique perspective that Hanabi provides is that you do not see your own hand but instead the hands of the other players, Figure 1 has an example game state. The starting setup is each player having a full hand size. The team has eight information tokens and three mistake tokens. The game ends when the deck runs out and each player has one final turn, or when the players have zero mistake tokens left.

Game Actions

During a turn a player has a choice of three different actions: playing a card, discarding a card, or giving a hint. The player who is taking their turn is called the active player.

Figure 1

Example state of Hanabi from player 0 (P0) perspective[1]



Play a Card.

The active player may pick a card from their hand whether it is known or unknown and attempt to play it. To play a successful card the active player would have to lay down the next card in any sequence. Looking at Figure 1, a successful card play could be G2, B1, W2, Y3, or R1 these cards are next in the sequence for each color. Player 1 and Player 2 from Player 0's perspective have potential plays. If a play is successful it goes to the appropriate stack. When a 5 is played and completes a stack an information token is gained. If it is an unsuccessful play the card goes to the discard pile and a mistake token is lost. No matter the outcomes of a player playing a card, that player immediately draws a new card from the deck if it is not empty.

Discard a Card

The active player may discard a card from their hand when there are less than eight information tokens in play. The card is then placed into the discard pile. When this is done, the player draws a new card and an information token is placed back into the pile.

Give Information

The active player can choose to give information to a player by spending an information token. Note that the active player cannot give information if there are no information tokens left. The information that can be given to a player is limited to either telling a player about a rank or a color. The active player must tell the receiving player every card that matches the chosen piece of information. The active player cannot give information that does not pertain to the player's hand. Looking at Figure 1, Player 0 wants Player 1 to play their R1. To do this Player 0 gives the hint that card four is red. Player 0 could have instead given them information about rank 1, in which Player 1 would say "cards two and four are 1s". The rank play would result in more information but would not achieve the desired result of the R1 being played. The active player could not choose to just say "card four is a 1".

The challenges of the game consist of its required cooperation between players and the imperfect information a move will require. For an AI to achieve a perfect score of 25 in the game, AI will have to make moves based on its teammates and not just its own information. An example play of this would be that Player 1 gives a hint to Player 3 that one of his cards is a 4 and yellow. Player 2 now knows that they have a 3 in their hand. They also know that Player 1's action is a waste of time because the next required yellow card is a 3. From there, Player 2 can infer that the reason Player 1 made their move is hoping that Player 2 would infer that their 3 is a

yellow 3. This is where advance play of the game requires learning and understanding your teammates rather than the game itself. However, there is a danger in the previous example it is instead possible that Player 2 infers incorrectly, and Player 1 was not going for that move because the 3 is red and not yellow.

2.2 Machine Learning

Machine learning is a process in which computers can improve their performance based on data. In a machine learning problem, the agent does not know what the optimal action is in a given problem. An agent's training data is usually be split into two groups: the training data set and the test data set. The training data is used to train the agent to make optimal choices or guesses. From there, an agent's performance is tested with the test data. An agent must never encounter test data during training. An example of a machine learning problem is a program that can automatically identify handwritten postal codes on mail addresses after learning from a training data set. There are several different methods of machine learning. Those learning methods are supervised learning, unsupervised learning, semi-supervised learning, and active learning.[2]

Supervised learning is a form of classification problem. The supervision of an agent is done with labeled examples in the training data set. Following the postal example from before, the training data would have answers connected with each data point in the set. Additionally, the test data would use the labels on the data to access an agent's performance after training.

Unsupervised learning is a form of clustering problem. The learning process is unsupervised because the training data would not have labels or correct answers connected with the data. For example, looking at images of digits, unsupervised learning would cluster together

all imagines that would be similar. An example would be the clustering all imagines of a 9 are in the same cluster. However, an agent using the clustering method as in the postal example from earlier would not understand that the cluster was a collection of imagines of the number 9.

Artificial Neural Networks

Artificial neural networks are a computing system which is inspired by the neural networks in the human brain. An artificial neural network contains several layers of connected nodes called artificial neurons. Each layer of neurons is connected to another layer of neurons, similar to our brain's synapses, axons, and dendrites. Every connection between neurons has an associated connection strength or weight. Figure 2 visually shows what an artificial neural network would look like. In the figure shown, each circle represents a neuron, each arrow represents a connection between neurons. When each node in a layer is connected to every node in the next layer it is called a fully connected layer. At a minimum, each node must have one input connection and one output connection.

Figure 2

Example of a neural network, from left to right: input layer, hidden layer, output layer.[3]

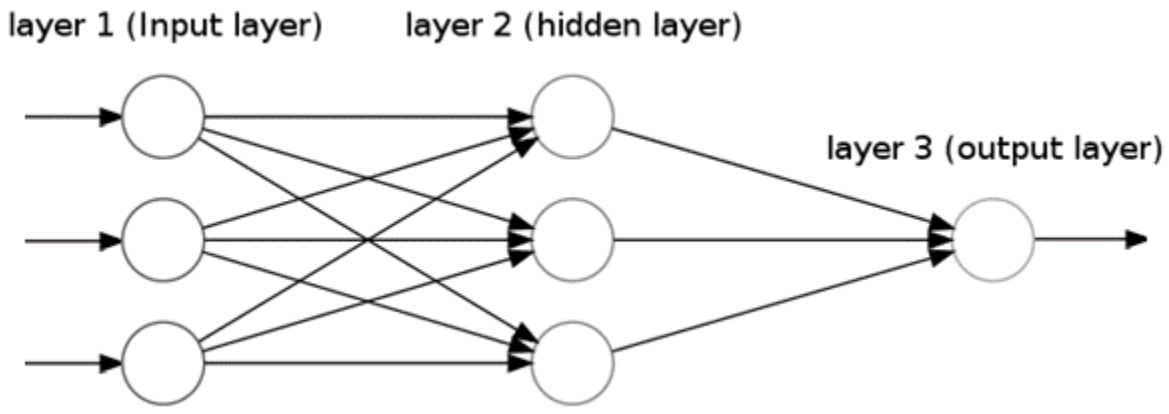
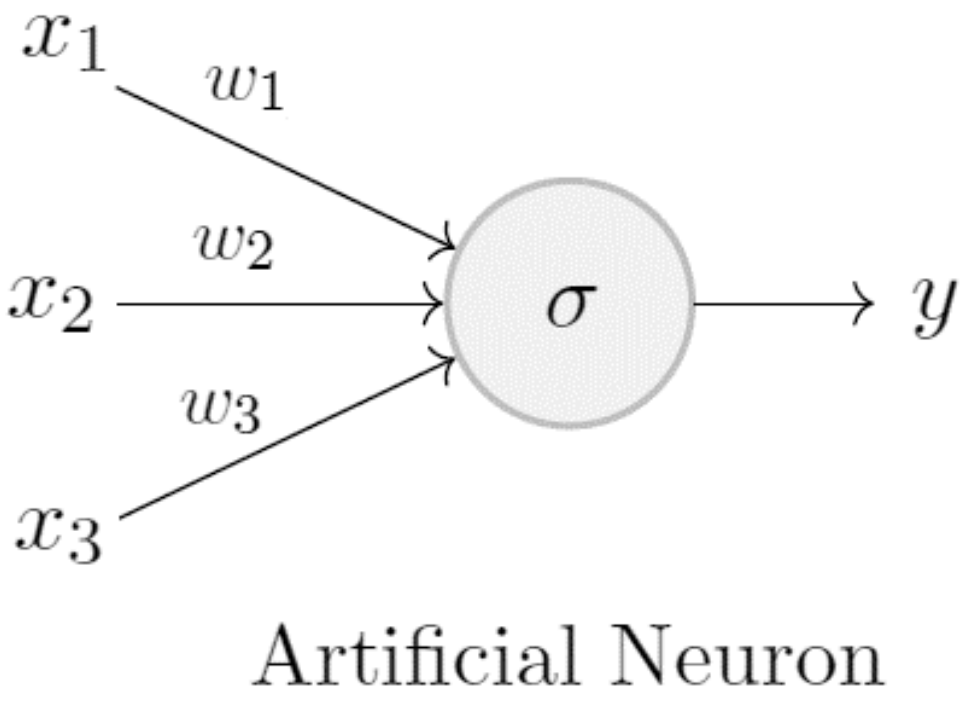


Figure 3 is an example of an individual artificial neuron that has three inputs (χ_1, χ_2, χ_3), three weights (w_1, w_2, w_3), and an output of y. For a neuron to give an output a threshold t must be reached. When the sum of $\chi_1 w_1 + \chi_2 w_2 + \chi_3 w_3 > t$ then the neuron will output y otherwise the output of the neuron is 0. This can be expanded with n inputs with n weights.

Figure 3

An artificial neuron with three inputs, three weights, and one output.[4]



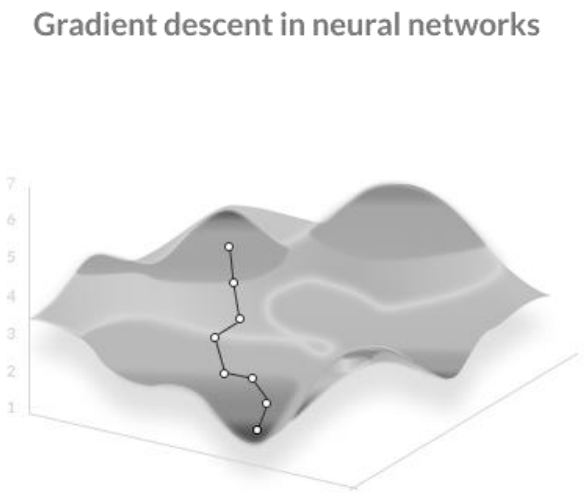
In artificial neural networks, three layers are used to solve problems. The first layer is the input layer which is simply the input data for a given state in a problem. Let's consider an example of a 20x20 pixel image of a digit to be classified. The size of the input layer would be 400 nodes, each pixel would have its own node and that node's value would be either 0 or 1. The

output layer would be the action or classification a neural network would decide for a given input. In the digit classification example, the output layer size would be ten and each neuron in the output layer would represent a different digit 0 through 9.

The final layer is the hidden layer. The hidden layer is the most significant part of a neural network when it comes to learning. The hidden layer contains zero or more layers of neurons. The reason for the name of the hidden layer is that we do not know the reason behind any weights or values placed into the hidden layer. Whereas with the input and output layer we know its output values and input values. For the example of digit classification, finding the correct size and number of layers for the hidden layer would be up to experimentation that produces optimal results.

Figure 4

Visual example of gradient descent[5]

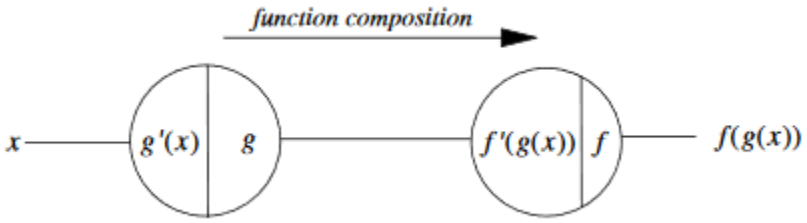


A neural network works by feed-forwarding the input to an eventual output. However, to make changes to a neural network a common technique to use is backpropagation to change the weights of the neurons. Backpropagation works using the premise of gradient descent. Figure 4

shows a representation of gradient descents traversal of a function to the minimum. Gradient descent is a mathematical technique that modifies the parameters of a function to descend from a high value of a function to a low value. It does this by looking at the derivatives of the function with respect to each of its parameters. Finally, apply the gradient descent to an error function to help find weights that minimize the error values.

Figure 5

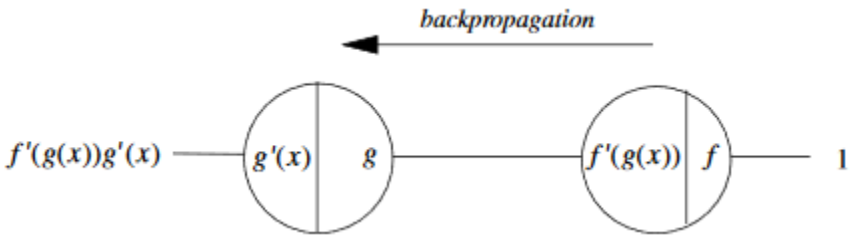
Feed-forward step of a neural network[6]



Backpropagation algorithm is used to find a local minimum of an error function. Taking the simplest neural network of 2 nodes can be viewed in Figure 5 and Figure 6. Figure 5 shows the state of a network after the feed-forward step. Figure 6 shows the result of a backpropagation step with the output of the network being set to a constant 1. Information from the right side is multiplied by the value stored in a node's left side. The final result of the backpropagation is $f'\big(g(x)\big)g'(x)$ which is the derivative of the function $f\big(g(x)\big)$.

Figure 6

Backpropagation step of a neural network[6]



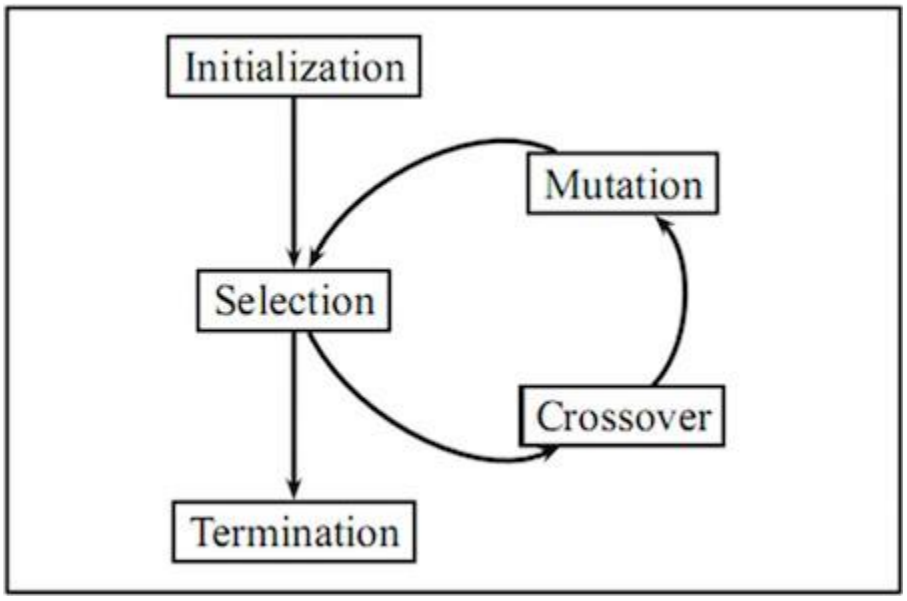
Evolutionary Algorithm

Evolutionary algorithms are a method of function optimization based on the theory of evolution.[7] A common method of evolutionary algorithms is a genetic algorithm. Genetic algorithms use a 'genotype' that is decoded and evaluated. How a genetic algorithm works is by selecting the fittest agent from a large population of agents. Each agent from the population has its own genetic code. An agent's genetic code contains genes that contain data that influences how an agent performs during its lifetime.

At the end of the lifetime of an agent, how it has performed is measured by an evaluation function called the fitness function. Depending on the specifics of the algorithm, a singular best agent is selected, or several best agents are selected for reproduction. The old population is removed and replaced with the selected best agents. The new population will then have each of its agents' genetic codes undergo a random mutation. A mutation is a random change in the data in a genetic code to produce new agents that have a chance of a better fitness score. This process repeats until a satisfactory agent is found. An illustration of a genetic algorithm can be seen in figure 7, with the fitness function being represented by Selection.

Figure 7

Illustration of genetic algorithm [8]



To apply genetic algorithm concepts to a neural network-based agent, the weights of the neural network would need to be the genetic code of an agent. From there the selection process of the agent would still be the same following a fitness evaluation function. The mutation portion of a neural network would randomly change some or all of the weights of a network. How much the weights changer our determined by the programmer.

A note about function optimizers and neural networks is that they do not always yield an optimal solution but instead yield a competitive solution. It is important to view genetic algorithms as a search process rather than strictly as an optimization process.[9] As such, competition implemented by the fitness function is the key aspect of the genetic search.

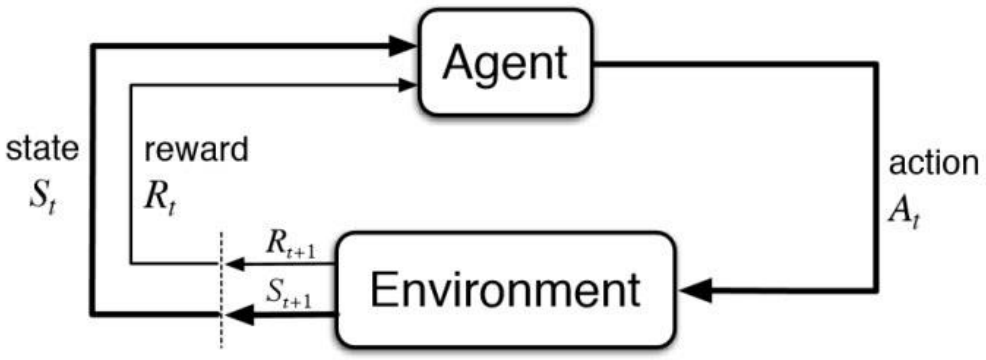
Reinforcement Learning

Reinforcement learning is a general feedback-based method of machine learning. In reinforcement learning an agent makes decisions and predictions for the optimal action without

the need for a supervisor. Instead, how an agent learns is by receiving rewards or reinforcements as a result of the agent's actions. The reward is typically a scalar value based on the outcome of the interaction between the agent and the environment. Figure 8 is a basic model of reinforcement learning, initially an agent receives the state S_t and reward R_t. Based on S_t the agent takes action A_t on the environment, after A_t the environment responds with a new state S_{t+1} and a new reward R_{t+1}

Figure 8

Model of Reinforcement learning[10]



Q-learning is a model-free reinforcement learning algorithm that allows an agent to decide on an action given a state. For any finite Markov decision process, Q-learning finds the optimal action policy using a system of rewards and punishments. The system of rewards and punishments allows Q-learning to maximize the expected value of the total reward over the current step and all successive steps from the current state. A strength of Q-learning is that it does not need a model of the environment and it can be used for on-line learning.[11]

To solve sequential decision problems, Q-learning can learn to estimate the optimal value of each action based on the potential of future rewards. A value can be determined from an action a and a state s as defined in Equation 1.

Equation 1

An equation of the quality of a state given an action. R *reward,* γ *discount factor*

$$Q(s, a) \equiv R_1 + \gamma R_2 + \gamma R_3 + \cdots + \gamma R_n$$

A discount factor γ is given between [0, 1] which creates a tradeoff between immediate rewards and future rewards. A low discount factor would encourage immediate rewards over future rewards.

An agent would learn using Q-learning through trial and error experimentation. An agent will gather information to make connections between reward values connected with actions and states. The agent will update its Q values $Q^{new}(s_t, a_t)$ based on the current Q value $Q(s_t, a_t)$, learning rate α, potential future Q values $max\big(Q(s_{t+1}, a)\big)$, current reward R_t, and a discount factor γ as shown in Equation 2. Learning rate determines the rate in which new information overrides the old information, the value of which is typically between [0, 1]. An additional part of an agent's trial and error is the exploration factor ϵ. The ϵ represents a percentage chance of taking a random action during any given state. If ϵ is 0.1, there is a 10% chance of the action being random. This is important in Q-learning because it allows an agent to learn and experiment with different actions that have potentially better outcomes.

Equation 2

Calculating the new Q value for a given step. t is the current step, α is the learning rate

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot \Big(R_t + \gamma \cdot max\big(Q(s_{t+1}, a)\big) - Q(s_t, a_t)\Big)$$

For an example of an agent's perspective during a Q-Learning process. Figure 9 shows the starting perspective an agent has, and Figure 10 has the ending perspective for the agent. In this example, an agent starts in the bottom left corner and its goal is to reach the positive top

right corner. The agent can only travel up, down, left, or right. Additionally, the agent wants to avoid the -1 because it represents a negative reward. Based on Figure 10 an agent learns the optimal route of up, up, right, right, right. Also, the agent does not prefer the route right, right, up, up, right because that route increases the potential of encountering the negative reward.

Figure 9

Starting perspective of an agent in Q-Learning

0	0	0	+1(e)
0	Blank	0	-1(e)
0(s)	0	0	0

Figure 10

Ending perspective of an agent in Q-Learning

0.812	0.868	0.918	+1(e)
0.762	Blank	0.660	-1(e)
0.705(s)	0.655	0.611	0.388

Q-learning has been commonly used with tables filled with Q-values. The entire table would represent each possible state and action. For most problems, the size of the Q table would render Q-learning impractical to use. A common means of avoiding Q tables is the use of neural networks as a function approximation. Q-learning uses the method of backpropagation to update a neural network.

Methods

3.1 Project

For my approach, I will combine Q-Learning and Evolutionary algorithm. Q-Learning has been used to approach systems that favors future rewards over immediate rewards. The reason I chose Q-learning is because of its success in Atari games and DOTA 2 Bots that beat human players.[12] However, a problem noted about Q-Learning is that it takes a significant amount of time to learn a game.[13] Additionally, a problem with Q-Learning concerning Hanabi is that Hanabi is an environment that contains many local minimums for optimal gameplay. I will use the evolutionary algorithm approach to allow Q-Learning agents to take larger steps to escape the current local minimum to find a new local minimum.

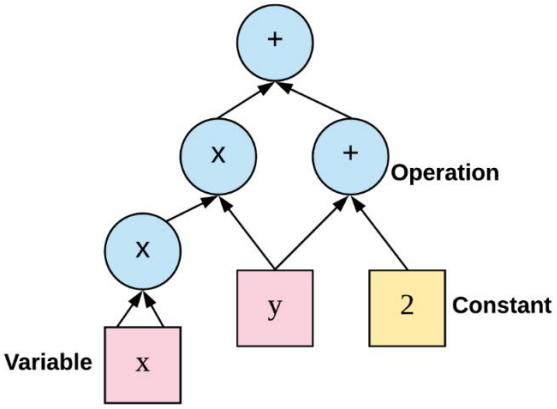
TensorFlow Technology

The project will use Python as the coding language with TensorFlow and Hanabi Learning Environment. TensorFlow is an open-source machine learning library that I will be using to build the neural network with. TensorFlow computations are expressed as stateful dataflow graphs. TensorFlow works by using graphs and sessions. A graph represents the data flow of the computations and a session executes the operations in the graph.

A TensorFlow graph will manage all computations that take place inside the model. An example of a graph can be seen in Figure 11 for the computation f(x, y) $= x^2y + y + 2$. Each operation of a graph is placed into a node of the graph. The arrangement of the nodes represents the operations of the model.

Figure 11

An example graph for the computation of $f(x, y) = x^2y + y + 2$ [14]



A TensorFlow session must be created to perform any operations on a TensorFlow graph. A session places the graph operations onto hardware such as CPUs or GPUs and provides methods to execute them.

To create the structure of a model using TensorFlow a placeholder is used to initialize the structure. In the first half of the Figure 12 code, a model of a neural network is created using placeholders with TensorFlow. When using a placeholder, TensorFlow must be told what type of data each element within the tensor is going to be. In the code example, *tf.float32* is used as the expected data type. Following the example, a three-layered neural network is created with an input layer of *s_size*, a single hidden layer of *h_size*, and finally an output layer of *a_size*. Finally, the output of the network is the node with the highest value from the output layer. In the example, all values are placeholders with a data type of *tf.float32*. In the second half of the Figure 12 code, an update step is created for future use in Q-Learning. With the Adam optimization algorithm being used for the gradient descent.

Figure 12

A class in Python in which s_size represents input layer size, h_size represents hidden layer size, and a_size represents output layer size.

```
import TensorFlow as tf
import tensorflow.contrib.slim as slim

#These lines established the feed-forward part of the network. The agent takes a state and produces an action.
    self.state_in = tf.placeholder(shape=[None, s_size], dtype=tf.float32)
    hidden = slim.fully_connected(self.state_in, h_size, biases_initializer=None,activation_ fn=tf.nn.relu)
    self.output = slim.fully_connected(hidden, a_size, activation_fn=tf.nn.softmax, biases_initializer=None)
    self.chosen_action = tf.argmax(self.output, 1)
    ⋮
#These lines established an update step for the gradient descent in Q-Learning process.
    tvars = tf.trainable_variables()
    self.gradient_holders = []
    for idx,var in enumerate(tvars):
        placeholder = tf.placeholder(tf.float32,name=str(idx)+'_holder')
        self.gradient_holders.append(placeholder)

    self.gradients = tf.gradients(self.loss,tvars)
    optimizer = tf.train.AdamOptimizer(learning_rate=lr)
    self.update_batch = optimizer.apply_gradients(zip(self.gradient_holders,tvars))
```

Hanabi Learning Environment

Hanabi Learning Environment (HLE) was developed by Deepmind to create consistency between research on machine learning in the game Hanabi. The HLE is a library of functions and classes for the game Hanabi that uses C++ for speeding up processing speed. For my program, I will be using the Python shell that calls the C++ library.

Program

The program uses a neural network with an input node set of 763. The input string is fully connected to the next layer of 183 nodes, followed by a layer of 29 nodes, finally to an output layer of 30 nodes. The input string size is determined by the HLE and is consistent for a three-player game. The second layer size was chosen based on deck size, played ranked cards, and knowledge tokens. The second layer is determined by player hand size, colors, ranks, and information token. The output string is all potential moves in three player game that an agent can take.

This section will follow the steps of the program.

1. Setup Hanabi game class with three players expected.
2. Create agent models and graphs in TensorFlow.
3. Prepare an array of agents' different weights in the graphs.
4. Select three agents at random, have them play x games, then repeat this until all agents have played with one group.
5. After each game take the Pre-State, Action, Reward, State relationship history for each agent and take a training step for this game (potentially do this over more than 1 game).
6. Repeat Steps 4 through 5 y amount of times with different random groups of three agents.
7. At this point determine the best players using a fitness function.

8. From the top agents' weights, crossover and mutate into a new population for repeating Steps 4 through 7. Do this until a desired amount of generations has been reached or a desired play pattern is reached.

For this program, I decided to simplify the scope by keeping the number of players constant. The attributes that I will be measuring in the game of Hanabi are score, turns taken, information, and best score. The reason for each attribute: score is how you win, turns taken allows you to have more good opportunities, information is how to make a safe play, and best score as a tie breaker. In my program I had my agents play ten games with each group during Step 4. Then the agents would take the average of each important key aspect of Hanabi, except best score. During Step 6, I had the agents play with six different groups.

The fitness function is $fit(s,t) = s + 0.5 * t$ with s being the average score and t being the average turns. Finally, we use the best score as a tie breaker for the fitness function. The reason I did not include all the tracked data for each agent in the fitness function is that score and turns taken determine the best players overall. Instead, all the tracked data is used in the reward calculation for a given state during reinforcement learning. Once the fittest agents were selected, a crossover rate of 50% and a mutation rate of 1% were used.

The reward for each state that the agent encounters will be based on the criteria of playing correct cards, discarding cards that are no longer useful, and amount of information given. Based on a play an agent makes, a different evaluation is given. If the agent plays a card, an agent will receive a reward if the card play was correct and the play was not without information about the card. For discarding a card, if the card is still a potential correct play no reward is given. When giving information, there is no negative reward, only a greater reward

based on amount of information given. If an agent were to have three rank 1 cards that they have no information about, that would be more valuable than two rank 2 cards.

The program applies the reward function in Q-Learning with an exploration rate of 5%, a learning rate of 10^{-6}, and a discount factor of 90%. The reason for choosing an exploration rate of 5% is that in Hanabi you only have three allowed mistakes, so a small exploration rate would reduce the number of random moves during a game to about two or one. The discount factor was chosen because the value of the end of the game overrides any early moves. The main goal of an agent would be to first start by learning how to reach the end of a game and then begin to improve its score beyond finishing the game. The reason for the learning rate is more of a technical limitation with float values and TensorFlow. If a larger learning rate is used, there is a greater potential for a number to be divided by zero, thus crashing the program and removing any values in the agent. Otherwise, the learning rate should be small to allow for incremental growth rather than large steps that would prevent the overlearning from good or bad actions.

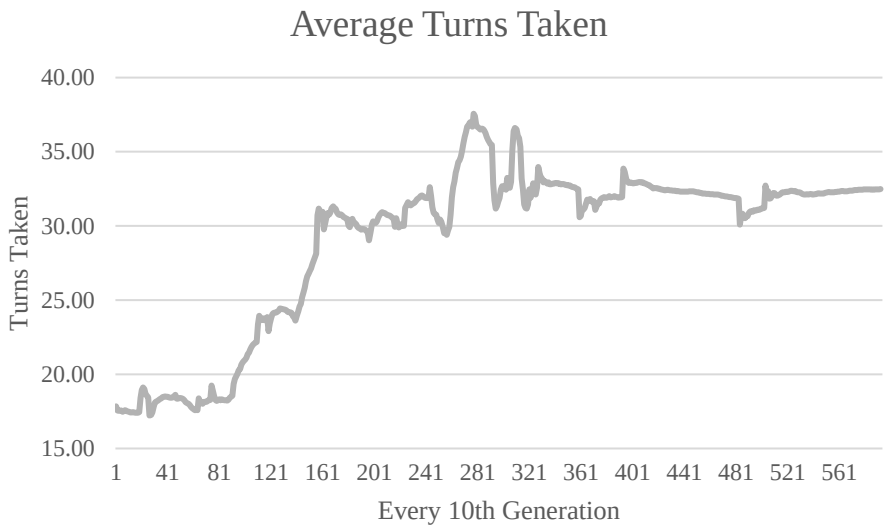
Results

4.1 **Data**

The performance of the best agent did not reach satisfactory levels of improvement. The current abilities of the best agent are capable of reaching the end of the game without losing by mistakes. So far, an average score of 1.55 is obtained consistently, which is not optimal in a game of Hanabi with the max score being 25. For a random agent, the score has an equivalent average. However, a random agent only reaches about 14 turns of the game on average, whereas the best agent reached an average of 30 to 35 turns in a game. The agent has proven to be better than a random agent but still needs to grow. The average length of a game of Hanabi that does not lose to mistakes is 70 turns. A few groups of agents managed to reach the end of a game but could not do it consistently.

Figure 13

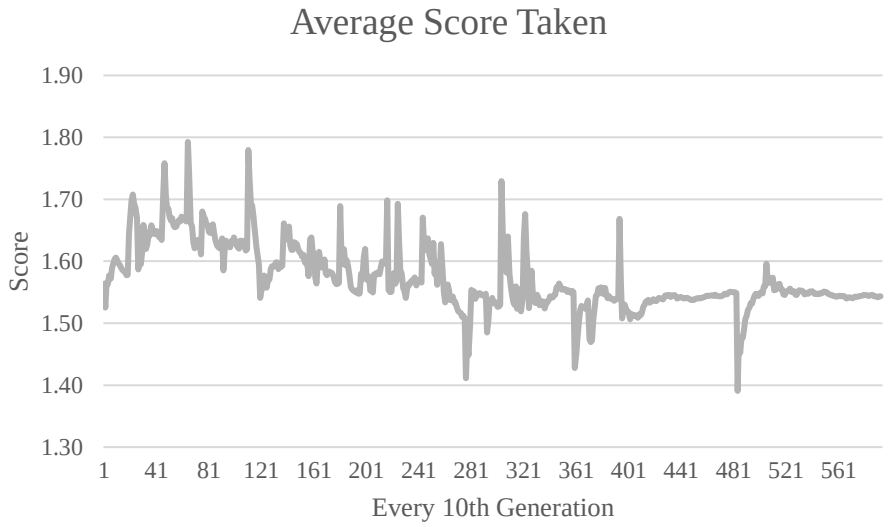
The average turns taken before the end of a game over 5000 Generations.



Comparing the program's AI agents to human players a score of 1.55 is negligible. Human players from my personal experience achieve a score of 15 on the first game. Additionally, it is common to get scores of 22 after some experience with the game. The best Hanabi players can reach a score of 25 70% of the time.[15]

Figure 14

The average score taken before the end of a game over 5000 Generations.



The agents have gone through five thousand generations to reach their current capabilities. Which would mean that the agents have played 300 thousand games each. The total games with all agents would be millions of games. It took about two thousand generations before the agents appeared to stop improving, became consistent, and reached their limit.

Discussion

5.1 Conclusion

Using the machine learning methods of Q-Learning and Genetic Algorithm for the game Hanabi was not successful. I made two different attempts to use these learning methods. One of the methods we have already covered, but an additional attempt was made to just have the reward function be determined by the end of game score. In both cases, the agents grew to the ability to just reach the end of the game but were still not able to develop a basic understanding of how to gain a score greater than random play.

Potential areas of growth and change would be to change the reward function, time, and preplanned training. The most basic area of change would be allowing the agents to play more games. Because Hanabi is a game with many potential game states, the agents would have to encounter a large number of potential states consistently to establish a pattern in situations. However, based on the data flattening this would not be an effective approach. Another idea would be to modify the reward function to change based on the current abilities of the agent. Once an agent learns to survive to the end of the game, I would change the reward function to begin to focus on rewarding score instead of reward turns and score.

Another idea would be using a preplanned training data set. The data set would contain situations with labeled correct moves. The training of an agent would start with the training data and the goal of the data would be to get the agent past learning the basics of the game. The game basics would be playing safe cards that are known correct plays, giving information so that a player can play a card safely, and discarding a card when the card is no longer playable. With the basics covered the agent can attempt to learn the more advanced moves in the game.

5.2 Future Works

For any future work on this project, the computation should be done with multiple processers running in parallel. The main reason for this is because any neural network inherently takes a significant amount of computation. Increasing the amount of games played in a shorter amount of time will not change any results of the neural network. However, it would allow for additional testing of any neural network.

An important note is once an agent begins to achieve human level of scores in the game. New parameters should be introduced to see if the agent can be adaptative to new players. This could be done by having a human player standing in or having a directly coded AI to participate and see if the agents can adapt to their gameplay.

REFERENCES

[1] N. Bard *et al.*, "The Hanabi challenge: A new frontier for AI research," *Artif. Intell.*, vol. 280, p. 103216, Mar. 2020, doi: 10.1016/j.artint.2019.103216.

[2] Jiawei Han, Micheline Kamber, Jian Pei, "Data Mining," in *Data Mining Concepts and Techniques*, pp. 24–26.

[3] thiagogm, "How to draw neural network diagrams using Graphviz," *Thiago G. Martins*, Jun. 12, 2013. https://tgmstat.wordpress.com/2013/06/12/draw-neural-network-diagrams-graphviz/ (accessed Dec. 08, 2020).

[4] chm, "Perceptron: The Artificial Neuron," *mc.ai*, Aug. 12, 2018. https://mc.ai/perceptron-the-artificial-neuron/ (accessed Apr. 05, 2020).

[5] "Backpropagation in Neural Networks: Process, Example & Code," *MissingLink.ai*. https://missinglink.ai/guides/neural-network-concepts/backpropagation-neural-networks-process-examples-code-minus-math/ (accessed Dec. 09, 2020).

[6] R. Rojas, "The Backpropagation Algorithm," in *Neural Networks*, Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 149–182.

[7] M. McGlohon and S. Sen, "Learning to cooperate in multi-agent systems by combining Q-learning and evolutionary strategy," The University of Tulsa.

[8] D. Soni, "Introduction to Evolutionary Algorithms," *Medium*, Jul. 16, 2019. https://towardsdatascience.com/introduction-to-evolutionary-algorithms-a8594b484ac (accessed Dec. 06, 2020).

[9] D. Whitley, "An overview of evolutionary algorithms: practical issues and common pitfalls," *Inf. Softw. Technol.*, vol. 43, no. 14, pp. 817–831, Dec. 2001, doi: 10.1016/S0950-5849(01)00188-4.

[10] "5 Things You Need to Know about Reinforcement Learning," *KDnuggets*. https://www.kdnuggets.com/5-things-you-need-to-know-about-reinforcement-learning.html/ (accessed Nov. 09, 2020).

[11] G. Tesauro, "Pricing in Agent Economies Using Neural Networks and Multi-agent Q-Learning," in *Sequence Learning: Paradigms, Algorithms, and Applications*, R. Sun and C. L. Giles, Eds. Berlin, Heidelberg: Springer, 2001, pp. 288–307.

[12] I.-A. Hosu and T. Rebedea, "Playing Atari Games with Deep Reinforcement Learning and Human Checkpoint Replay," *ArXiv160705077 Cs*, Jul. 2016, Accessed: Sep. 26, 2020. [Online]. Available: http://arxiv.org/abs/1607.05077.

[13] P. Chopra, "Reinforcement learning without gradients: evolving agents using Genetic Algorithms," *Medium*, Jan. 07, 2019. https://towardsdatascience.com/reinforcement-learning-without-gradients-evolving-agents-using-genetic-algorithms-8685817d84f (accessed Apr. 09, 2020).

[14] S. User, "Graph and Session." https://www.easy-tensorflow.com/tf-tutorials/basics/graph-and-session (accessed Apr. 17, 2020).

[15] Q. Wong, "Facebook's new card-playing bot shows AI can work with others," *CNET*. https://www.cnet.com/news/facebooks-new-card-playing-bot-shows-ai-can-work-with-others/ (accessed Dec. 08, 2020).