Exploring Ability to Correct Communication Mistakes with ML Techniques through Hanabi

1st Matt Riedman
CS 229
Stanford University
riedman@stanford.edu

***Abstract*—Communication is an important topic in AI that has not received very much attention [1]. Hanabi is a cooperative card game that requires effective communication between players in order to achieve a high score, making it.a good model for communication as a whole. In this paper, I address the problem of accounting for errors in communication during Hanabi by identifying mistakes using K-Means and the EM Algorithm. I then test these models' ability to increase performance against an agent which frequently makes mistakes, both alone and in combination with reinforcement learning. I found that all combinations of these techniques improved performance to varying degrees in high-mistake settings without sacrificing performance when the cooperator made few mistakes.**

I. Introduction

While communication is an essential tool in everyday human life, the variety of signals a communication agent must learn to interpret and respond with makes it a very challenging subject in artificial intelligence [1]. In particular, training an agent to properly identify and react to mistaken or erroneous signals presents a fundamental liability given the fallibility of any means of exchanging information. One domain in which these problems can present themselves is in the cooperative card game Hanabi.

In Hanabi, two (or more) players are given a hand containing five cards which all players except the one holding them can observe. The goal of the game is for players to give each other hints about properties of the cards, such as their color and rank, in order to place them down in the proper order. The full rules of Hanabi can be found at [2].

In this paper, I explore various techniques of mistake detection and response in games played with error-prone Hanabi agents. I provide as input to the Hanabi agents the full observable board state at each turn (including the cooperator's cards, prior hints given, etc., but excluding the agent's own cards), which is the same information that any human playing the game would receive. To restrict the scope of the problem, the model's output is simply an indicator for whether to treat the cooperator's prior move as genuine or erroneous.

I built separate models using K-Means and the EM algorithm to predict the correct labels for each move. Then, I built two classes of agents to act on these predictions: one which took the predicted label at face value and one which used reinforcement learning with value iteration to determine whether to act on the labels. (I will go into further detail about how these parts interact in the Experiments section.)

With all methods, if a move is interpreted as genuine, the agent will play a move based its cooperator's strategy, likely using implicit information gleaned from prior moves. If it is interpreted as a mistake, then the agent will still adopt its cooperator's hinting strategy but otherwise avoid acting on any information not explicitly included in hints.

While I did a project on Hanabi in CS 221, the only pieces that this report shares in common with that project is the strategy that the agents I have designed are meant to play against and some of the code I've modified from DeepMind to run the games.

II. Related Work

One past approach to crafting a Hanabi agent is to encode agents whose strategies mimic what a human playing the game may choose to do, with varying degrees of sophistication, such as the strategies created by Hirotaka Osawa [3]. These approaches generally make few assumptions about their cooperator's behavior, meaning they are not prone to playing incorrect cards, but their limited hinting strategy makes them prone to misinterpretation by many other agents.

Another approach to achieving high scores in Hanabi involves hand-coding an agent with the specific purpose of keeping track of the likely game state based on the entire move history. These algorithms tend to be more complex and specialized than the principles behind ones like Osawa's, although they often yield performance better than most human players. One example of this approach is David Wu's Fireflower bot, which utilizes long short-term memory (LSTM), whose average performance in 2-player games is 2.5 points shy of perfect [4].

A third approach to creating a Hanabi agent is to train an agent to decode its cooperator's actions with minimal built-in guidance. While training agents to develop their own conventions for communication is difficult using most tools in AI, there has been some success in recent years with deep multi-agent reinforcement learning yielding agents that routinely secure perfect sores, something almost impossible for humans. A pair of examples, both using Bayesian techniques to deduce one's own cards from the cooperator's actions, are [5] and [6].

One final approach is to create a pool of agents in order to evaluate which has the best overall performance relative to its peers, as in [7]. Here, the authors generate a wide variety

of candidate agents using different permutations of a rule set and tweaking other behaviors such as risk aversion and communicativeness. They then have each agent play against all others to determine the one with the highest overall average. Since agents in this method see a wide variety of strategies, including many that are suboptimal, they should be more resilient to games against mistake-prone agents, although it is still not an explicit purpose of the paper to address that problem.

III. Dataset and Features

The training data used in my model comes from 200 games that my Standard Strategy with a 50% error rate played against itself. The Standard Strategy is meant to mimic a simple strategy that I've used in Hanabi, and it is set to have a 50% error rate (meaning that $\frac{1}{2}$ of its moves, rather than following the rules of the strategy, are randomly generated). These games are played in the DeepMind Hanabi Environment [8].

These 200 games contained a total of 3996 moves, for which I saved 7 features and a label. The label took the value 0 if the move was randomly generated, 1 if the move was not random but did not carry implied meaning, and 2 if the move was not random and did carry implied meaning. The seven features, which I hand selected based on my knowledge from playing the game, were based on simulations by an agent of their possible hands. Specifically, at all times, each agent keeps track of hints given to it. For example, it may know that its leftmost card is blue. From this, it generates 50 hands that it may have, meaning that in the previous example, one simulation may have the leftmost card as a blue 1. Then, for each simulated hand, the agent determines what move its cooperator would have made if the agent had that hand and records it. The results of these 50 simulations are used to generate the features below.

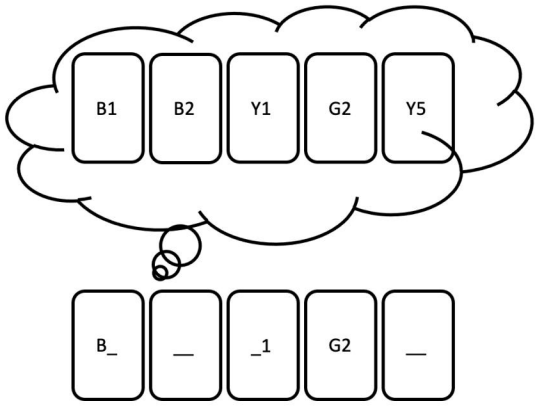

Fig. 1. Hypothetical Card Simulation

1) Proportion of simulated moves matching the cooperator's move
2) Proportion of simulated moves that are the most common move (e.g. if Play Card 1 occurred 5 times and Play Card 2 occurred 45 times, then this feature would be $45/(45+5) = 0.9$)
3) Proportion of moves with the same type as the observed move. (Types are Play, Discard, and Hint.)
4) Proportion of moves of the observed type which were the observed move
5) Proportion of moves from the most commonly observed type
6) Ratio of moves from the observed type to moves of the most commonly observed type

The validation set was obtained by running 300 more games using the same agent and recording the same features. In those 300 games, there was a total of 5677 moves recorded.

IV. Methods

A. EM Algorithm

The EM algorithm allows for unsupervised learning of data under a latent variable model. In this case, the latent variable is whether the cooperator's previous move was genuine or erroneous. Since it would be expected that this distinction would affect the distribution of the features, the EM model is a good fit for this paper's problem. Further, since in many practical problems involving communication error it is difficult to acquire labeled data, the unsupervised aspect makes it more broadly applicable to similar problems.

In order to facilitate better convergence, I used a "semi-supervised" approach in which a small number of data points are given their proper labels. Since in this case we are hoping for the model to pick up on particular clusters of points, this also helps protect against any unforeseen trends in the data changing the cluster assignments.

Given a point $x^{(i)}$, parameter θ, and set of latent variable labels $z^{(i)}$, the log-likelihood for θ is given by

$$\ell(\theta) = \sum_{i=1}^{n} \log \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta)$$

The object of the EM algorithm is to alternately estimate

$$Q_i(z^{(i)}) := p(z^{(i)}|x^{(i)}; \theta)$$

in the E-step and update θ to increase the lower bound on log-likelihood in the M-Step using Jensen's Inequality. Specifically, the M-Step consists of solving for

$$\theta = \arg\max_{\theta} \sum_{i} \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

In the semi-supervised setting, these formulas are augmented using the supervised log-likelihood

$$\ell_{\text{sup}}(\theta) = \sum_{i=1}^{\tilde{n}} \log p(\tilde{x}^{(i)}, \tilde{z}^{(i)}; \theta)$$

and the semi-supervised log likelihood becomes

$$\ell_{\text{semi-sup}}(\theta) = \ell(\theta) + \alpha \ell_{\text{sup}}(\theta)$$

for a hyper-parameter α, with the E-Step and M-Step updated accordingly.

Then, predictions are made on a new point x using

$$\arg\min_z p(x, z; \theta)$$

B. K-Means

K-Means is another method of unsupervised learning which sorts data into distinct clusters by their features where each cluster is composed of the points closest to its centroid. Following similar reasoning to the EM algorithm, I chose to adopt a semi-supervised approach with a small number of labeled points.

K-Means consists of two steps: one in which points are assigned to their closest centroid (or, in the case of labeled points, assigned to the centroid bearing their label) and another in which centroids are updated to be the mean location of their assigned points, here with a modified weight given to labeled points dictated to the hyper-parameter α. In mathematical notation, given centroids $\mu_1, \ldots, \mu_k$, unlabeled points $x^{(i)}$ with nearest cluster $c^{(i)}$, and labeled points $\tilde{x}^{(i)}$ with correct centroids $\tilde{c}^{(i)}$, first set

$$c^{(i)} \coloneqq \arg\min_j ||x^{(i)} - \mu_j||^2$$

and then update the centroids by

$$\mu_j \coloneqq \frac{\sum_{i=1}^{n} 1\{c^{(i)} = j\} x^{(i)} + \alpha \sum_{i=1}^{\tilde{n}} 1\{\tilde{c}^{(i)} = j\} \tilde{x}^{(i)}}{\sum_{i=1}^{n} 1\{c^{(i)} = j\} + \alpha \sum_{i=1}^{\tilde{n}} 1\{\tilde{c}^{(i)} = j\}}$$

These two steps are repeated until the centroids converge, at which point predictions are made on a new point x by determining which centroid μ_j is nearest to x.

C. Reinforcement Learning

Reinforcement learning, broadly, allows for building a policy over the state space of a Markov decision process (MDP) to maximize future rewards. In this paper, I use the value iteration algorithm, which estimates the value of each state given estimated future rewards. In particular, given a policy π mapping states onto actions, the value function of π is

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{i=0}^{\infty} \gamma^i R(s_i) \middle| s_0 = s, \pi\right]$$

with hyper-parameter γ as the discount factor. This function satisfies the Bellman equations, which say that given transition probabilities $P_{sa}(s') \coloneqq p(s_{i+i} = s' | s_i = s, a_i = a)$, then

$$V^{\pi}(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{\pi}(s')$$

The Bellman equations lead naturally to the value iteration algorithm, in which values are initialized to 0 for all states and then updated using

$$V(s) \coloneqq R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s') V(s')$$

until convergence.

In the context of the Hanabi communication error problem, reinforcement learning is intended to solve the issue of overcorrecting for perceived errors, where in a game with a cooperator which rarely make mistakes, our agent may lose more points by ignoring falsely flagged hints than what is gained from avoiding errors, and its implementation below will reflect this.

V. Experiments and Results

A. Mistake Identification

The first type of experiment I conducted was using K-Means and the EM Algorithm to predict from the features described in the Dataset and Features section whether a move was genuine or not. Here, the objective is to correctly classify each move by a mistake-prone agent as either genuine or erroneous, meaning success is measured by the proportion of moves that are correctly classified. In both categories, I found after experimentation that both models yielded better results when using three labels instead of two: one for mistaken moves (Mistake), one for regular moves (Regular), and one for moves carrying implicit meaning (Info). Using those, I trained my models as described below.

1) K-Means: Here, I used the semi-supervised implementation of K-Means described in the Methods section. I used 200, or roughly 5%, of the training examples for the labeled portion, and set $\alpha = 5$, meaning that the weight given to labeled examples was 5 times the weight of unlabeled ones. Both values were obtained by running the model over the training data with different hyper-parameters to observe which came with optimal performance on the training set.

The results on the validation set are in the table below, with each number representing the proportion of examples with a given label assigned a certain prediction (so Row 1 Column 2 is the proportion of Mistakes which were predicted to be Regular):

	Predicted Label		
Actual Label	Mistake	Regular	Info
Mistake	0.59	0.15	0.26
Regular	0.31	0.35	0.34
Info	0.14	0.10	0.76

Overall, 47% of examples were assigned to the correct label, and looking strictly at genuine versus mistaken (meaning labels Regular and Info would be identified), the model predicted 67% of examples correctly.

2) EM Algorithm: For the EM Algorithm, I used the assumption that my data came from a mixture of Gaussians, i.e. that each point of a given class was randomly drawn from a multivariate Gaussian distribution. I used 200 labeled examples with $\alpha = 100$, and I determined both values in a similar fashion as I did for K-Means. The results are in the table below, formatted identically to the table for K-Means:

	Predicted Label		
Actual Label	Mistake	Regular	Info
Mistake	0.25	0.62	0.13
Regular	0.25	0.57	0.18
Info	0.08	0.71	0.21

Overall, 47.1% of examples were assigned to the correct label, and looking strictly at genuine versus mistaken, the model predicted 64.6% of examples correctly.

B. Hanabi Gameplay

Below, I run an experiment in which agents utilizing the models from the section above play Hanabi against a mistake prone agent. The primary metric here is the average Hanabi score obtained, which ranges from 0-25. Even when the agent I have chosen plays optimally without mistakes, it averages roughly 18 points, making that a more realistic ceiling. In Hanabi, if players play three cards which are not allowed, the game is over and the final score is zero. This means that in settings where the opposing agent is more prone to mistakes, there may be no possible way to prevent the cooperator from erroneously playing three incorrect cards in a row, resulting in a score of zero. This means that when there is a large number of mistakes, the ceiling may be lowered further.

In my experiments, I used the following six agents:

1) **Advanced Human Strategy:** This agent plays Hanabi with a strategy I've used in games myself. Whenever this agent gives a color hint, it is a signal to its cooperator that they should play their leftmost card of that color. While this strategy proves itself effective by reducing the number of hints needed to convey information about certain cards, if this agent erroneously hints a color, it may lead its cooperator to play an inappropriate card.
2) **Advanced Human Strategy with Mistake Rate β:** This strategy makes the same move as the Advanced Human Strategy with probability $1 - \beta$, although it instead selects a random move with probability β. The necessary trade-off of wanting to rely on the genuine color hints described above with avoiding compounding mistakes if color hints are given erroneously makes this agent a suitable candidate for testing the remaining strategies against.
3) **K-Means Agent:** Upon obtaining its cooperator's move, this agent uses its trained K-Means model to predict whether that move was genuine or mistaken. If it is determined to be genuine, the K-Means agent will play the strategy dictated by the Advanced Human Agent. Otherwise, it will refrain from playing a card based on a color hint but otherwise behave the same.
4) **EM Agent:** This agent behaves identically to the K-Means Agent except its determination of whether a move is genuine is determined by the results of the EM training.
5) **RL K-Means Agent:** This agent starts by obtaining the predicted class of the prior move using the K-Means model, just like the K-Means agent. However, as indicated by Figure 2, this result is then passed through the RL policy to determine whether the prior move should be *treated* as genuine, regardless of what it was predicted to be. Also, along with the RL EM Agent below, the RL policies used against Advanced Human Mistake Agents with different mistake rates β are trained separately from one another to reflect the possibility that a more conservative approach is warranted when dealing with more mistake-prone agents.
6) **RL EM Agent:** This agent behaves identically to the RL K-Means agent except that it uses the EM model in place of the K-Means model. It maintains a separate estimate of each state's value from K-Means.

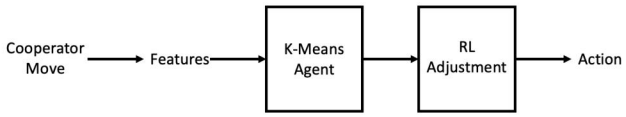

Fig. 2. Visual depiction of RL K-Means Agent setup

Each RL agent has a state space with 134 states. 108 of them are in-game states determined by the number of life tokens left (i.e. the number of incorrect cards played before the game is over), the score rounded to a multiple of 3, and the predicted label of the previous move. All of these states are assigned reward 0. The other 26 states are the end-game states which are solely determined by the game's final score. The reward for each of these states is the game score. Since games are rarely longer than 50 moves, I chose to use a discount factor of $\gamma = 1$.

For this experiment, I first trained both of the RL models by running 100 games at each mistake rate β in order to determine an optimal action strategy based on the predicted labels from the K-Means or EM model. Then, for each

$$\beta \in \{0, 0.01, 0.05, 0.08, 0.1, 0.13, 0.15, 0.18, 0.2, 0.25, 0.3\}$$

I ran 100 games pitting the Advanced Human Mistake Agent at mistake rate β against each of the Advanced Human Agent, the K-Means Agent, the EM Agent, the RL K-Means Agent, and the EM K-Means Agent and recorded the scores of each game. The results are in Figure 3.

VI. Discussion

A. Mistake Identification

Given the amount of complex logical reasoning that goes into playing Hanabi, even at a basic level like the Advanced Human Agent, the fact that the seven features I've selected are able to significantly outperform random choice with both algorithms should be taken as a positive sign.

A large plurality of examples were labeled Regular, which could go some way to explaining the mistakes in both models, especially given that they were likely more "spread out" across features both as a result of their number and the number of distinct situations from which a Regular move can arise. In K-Means, the labeled examples for Regular may have been spread out, leading that cluster to capture many of the examples that it should have and resulting in the near-uniform

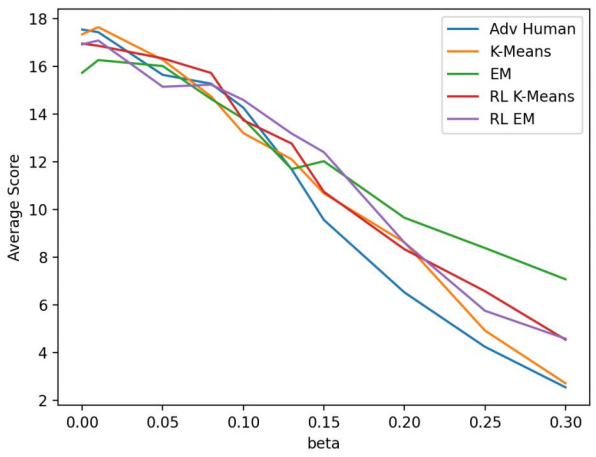

Fig. 3. Results of gameplay with various agents

distribution of examples that should have been labeled Regular across all predicted labels.

Meanwhile, in the EM model, the primary observable consequence was that most examples of all types were labeled Regular. Because the EM model estimates the proportion of examples carrying each label, this could have been a source of error here, too.

Because of the relatively small number of features used and the large number of training examples, I am not especially worried about my models overfitting the data.

B. Hanabi Gameplay

In the gameplay setting, the Advanced Human Strategy served as a good baseline because it effectively followed the same outline as the other agents, except instead of asking whether a move should be treated as genuine or not, it would always assume everything done was intentional. In less mistake-prone environments, this would generally work to its advantage since its assumption of correctness would match the real world, although it would become much more vulnerable in more mistake-prone environments whereas my agents were designed to mitigate that effect.

Hence, while it would be incredibly difficult to beat the Advanced Human Agent using mistake correction techniques for low β, when there are hardly any mistakes to correct, a good model fulfilling the purpose I have set out should outperform it at higher βs, which is exactly what all of my agents did. In fact, for $\beta > 0.1$, every single model of mine outperformed the Advanced Human Agent at every β, meaning that despite the less-than-stellar performance of the mistake detectors, when put into practice, they produced good results. Yet, even at lower mistake rates, none of the four models did significantly worse than the Advanced Human, suggesting that the mistake correction did not carry a significant negative impact on scoring in *any* conditions.

VII. Conclusion and Future Work

The core findings of my paper relate to how communication errors can be inferred from a cooperator's behavior under several different sets of assumptions in ways that significantly impact performance in Hanabi. Both K-Means and EM proved sufficient when semi-supervised at correctly identifying most most examples as either the intended result of communication or an unintended error, both properly classifying roughly two-thirds of examples. Even though the confusion tables were different, I suspect that because the misclassification rates were so similar, the performance is more due to insufficient features than the models themselves.

In the gameplay section, all agents outperformed the baseline in high mistake settings, demonstrating the efficacy of all of the algorithms tested. Surprisingly, the EM Agent outperformed all of the other agents with cooperators with high mistake rates, including the RL EM Agent. It is possible that the RL EM Agent was not sufficiently trained when it was evaluated, although I do feel that some further investigation may be warranted.

Lastly, as previously mentioned, an ideal extension to this paper would be more robust feature generation, possibly including some sort of deep learning, since the complexity of Hanabi likely requires similarly complex features to fully maximize the potential of mistake-correcting agents.

VIII. GitHub Link

https://github.com/mriedman/cs229-project

IX. Contributions

I (Matt) am the only author, and hence contributed all of the report and code (aside from what was contributed by DeepMind, NumPy, and Matplotlib, as well as parts of `run_game_mdk.py}`and `adv_human.py}`which I adapted from DeepMind in conjunction with a partner for a fall quarter project).

REFERENCES

[1] N. Bard et al., "The Hanabi challenge: A new frontier for AI research," Artificial Intelligence, vol. 280, p. 103216, 2020, doi: https://doi.org/10.1016/j.artint.2019.103216.

[2] A. Bauza, "Hanabi." Spillehulen, 2016. [Online]. Available: https://www.spillehulen.dk/media/102616/hanabi-card-game-rules.pdf

[3] H. Osawa, "Solving Hanabi: Estimating Hands by Opponent's Actions in Cooperative Game with Incomplete Information," AAAI Workshop: Computer Poker and Imperfect Information, 2015, Accessed: May 31, 2021. [Online]. Available: https://aaai.org/ocs/index.php/WS/AAAIW15/paper/view/10167/10193

[4] D. Wu, A rewrite of hanabi-bot in scala. 2018. Accessed: May 31, 2021. [Online]. Available: https://github.com/lightvector/fireflower

[5] J. N. Foerster et al., "Bayesian Action Decoder for Deep Multi-Agent Reinforcement Learning," arXiv:1811.01458 [cs], Sep. 2019, Accessed: May 31, 2021. [Online]. Available: http://arxiv.org/abs/1811.01458

[6] H. Hu and J. N. Foerster, "Simplified Action Decoder for Deep Multi-Agent Reinforcement Learning," presented at the International Conference on Learning Representations, Sep. 2019. Accessed: May 31, 2021. [Online]. Available: https://openreview.net/forum?id=B1xm3RVtwB

[7] R. Canaan, J. Togelius, A. Nealen, and S. Menzel, "Diverse Agents for Ad-Hoc Cooperation in Hanabi," in 2019 IEEE Conference on Games (CoG), Aug. 2019, pp. 1–8. doi: 10.1109/CIG.2019.8847944.

[8] DeepMind Hanabi Learning Environment. DeepMind, 2019. Accessed: Apr. 17, 2021. [Online]. Available: https://github.com/deepmind/hanabi-learning-environment

[9] Numpy. NumPy, 2021. Accessed: Jun. 03, 2021. [Online]. Available: https://github.com/numpy/numpy

[10] Matplotlib. Matplotlib Developers, 2021. Accessed: Jun. 03, 2021. [Online]. Available: https://github.com/matplotlib/matplotlib