1 Opponent Hand's Estimation

1.1 Introduction

Hand estimation is a crucial part in estimating player's strategy. When successfully estimating a small portion of opponent's hand, we can choose to discard wisely and safety based on the fact that the discarded card can form a meld with cards that estimated to be in the opponent's hand. With that in mind, we need enough information from each different turn in a match to do our job. Because when we play a turn, we see that opponent does not pick in the discard pile, then we know that they probably do not have any cards that are in meld with out discarded card and vice versa. Additionally, in the discarding phase, we see that the opponent discard a particular card, it can be in the situation that they do not have any cards that are in meld with that card. Therefore, our information will contain a representation of cards that the opponent does pick/ not pick, and cards that are discarded by them.

1.2 Turn State Representation

There will be 4 inputs. Each is a vector of length 52 (one-hot encoded).

- Vector of cards the opponent discarded this turn
- Vector of cards that the opponent does not care about (not picking up from discard pile) this turn.
- Vector of cards that the opponent does care and pick up from the discard pile.
- Cards that are on this player's hand and in the discard pile.

1.3 Model

The four input collected in the games are feed into the model. First, each input vector go to a lstm cell that remember the predicted opponent hand in relation with the each input. The output of all lstm cells are then concatenated into a single vector. The vector is then feeded into the feed forward dense network with dynamic layers.

1.4 Loss Function and Gradient Update

Custom Loss Function - Augmented Categorical Cross Entropy:

$$-\sum_{c=1}^{M} y_{o,c} \log p_{o,c}$$

The problem of this loss function is that it consider all errors at every classes the same. However, in this particular Gin Rummy case, we only want to estimate

each card if we have some clue of them in the input. Say it another way, we can only be able to estimate which card the opponent have in hand if we know the existence of some of the cards.

Therefore, to solve this problem, we need to make cards that are relevant to the input cards, and we also want to make unrelated card more trivial. Therefore the formula of the loss function should evaluate to:

$$-W_o * \sum_{c=1}^{M} y_{o,c} \log p_{o,c}$$

2 Abstraction of hand using VAE

2.1 Introduction

One another approach of defining strategy for a gin rummy player is to go through every situation in the game and continuously traverse the game tree recursively toward the end. For the core of an imperfect information game, we do not know the reward until we reach the end of the game tree. Only when we do so, we are able to update information set and history with some values saying whether some particular playing paths are winnable or not.

However, the number of information set combinations is very large in Gin Rummy. Throughout the game, for one turn, we can have $52C10$ cards combination posibility in hand, $52C32$ cards in draw pile, and $52C10$ cards combinations in the opponent's hand. We can put it another way: in each turn, if we put player's hand, opponent's hand, draw pile and discard pile each a vector of 52 cards, we would have $4^{52} = 2 * 10^{31}$ possible situation (for each card, one can have 4 different possibilities to be in one of the four card vectors)

Therefore, we need to minimize the number of situation is to abstract the situation and put them into bucket that map to some situation values. We can easily map our calculated value back to the abstracted situation so that when we are in a particular public state, we know which information set we are in and can have proper policy. One way of achieving this is to use an auto-encoder.

2.2 Auto-encoder

An auto-encoder consists of 2 parts: encoder and decoder. An encoder consist of neural network layers that reduce dimensionality when the network goes forward, and the decoder takes its input from the reduced encoder's layer and map the features back to an output. Traditionally, autoencoders were used for dimensionality reduction or feature learning. In computer vision, autoencoders are often used in the process of pixelizing or denoising images.

In Gin Rummy in particular, we will apply autoencoder concept by first encoding a public game state with linear neural networks, then we start to decode them to our state label which have been defined to be similar to the input public game state. After that, when we actually traverse the game tree,

we start encoding the public game state to a tensor, and have a comparison that map that tensor to a particular bucket information set, to update and evaluate a particular policy.