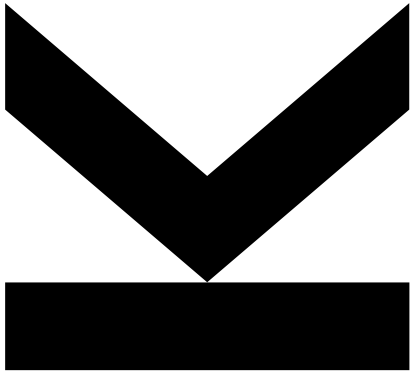
Contextual Preference Ranking and Reinforcement Learning for Gin Rummy



Mykola Zaitsev
k12147393

2025W: 351.072

Overview

- Gin Rummy is a card game where the player constantly has to evaluate whether a new card fits well into the hand she is holding
- Contextual preference ranking can learn such information using Siamese neural networks
- Reinforcement learning can be readily applied as the games are comparably short and there is a clear feedback signal

Task:

- Realize a Gin Rummy player based on a combination of contextual preference ranking and reinforcement learning

Content

- Overview of Gin Rummy card game
- Siamese Neural Network
- Review of the code and methods
- Findings and results
- Conclusions

Gin Rummy

Objective: score more points

- Run: sequences of three or more cards in the same suit e.g. **3♥ 4♥ 5♥**
- Set: three or four cards sharing the same rank e.g. **8♥ 8♦ 8♠**

Standard deck, King high, Ace low

Discarding card until all cards in hand are part of runs or sets

Players receive point based on opponents unmatched cards (deadwood)

Score is the difference between value of unmatched cards + bonuses (e.g. bonus for having no deadwood)

Siamese Neural Network

The SNN takes three inputs:

- **Anchor (Hand/Deck).** The current set of selected cards.
- **Positive (Card 1).** A card that was chosen by the expert/good player.
- **Negative (Card 2).** A card that was available but not chosen.

The model compares the two candidate cards to determine which one is a better addition to the given hand.

Siamese Neural Network

- The model uses a shared embedding network to encode the hand and cards into a vector space.
- Two forward passes are made for each candidate card (positive and negative). The difference between the embeddings of the hand and card pairs is computed.
- The triplet loss function is used to ensure the network learns to bring positive card embeddings closer to the anchor hand while pushing away negative card embeddings.

Siamese Neural Network

Triplet loss

$d(Anchor, Positive) < d(Anchor, Negative) + m$, where:

- $d(A, P)$: Distance between hand and chosen card.
- $d(A, N)$: Distance between hand and rejected card.
- m: Margin, ensuring that the chosen card is always closer to the hand than the rejected card.

Review of the code and methods

Environment: *gin_rummy_v4* environment from *pettingzoo* library

```
from pettingzoo.classic import gin_rummy_v4
```

Action ID	Action
0	Score Player 0 *Used after knock, gin, or dead hand to compute the player's hand.*
1	Score Player 1 *Used after knock, gin, or dead hand to compute the player's hand.*
2	Draw a card
3	Pick top card from Discard pile
4	Declare dead hand
5	Gin
6 - 57	Discard a card `6`: A-Spades, `7`: 2-Spades, ..., `18`: K-Spades `19`: A-Hearts ... `31`: K-Hearts `32`: A-Diamonds ... `44`: K-Diamonds `45`: A-Clubs ... `57`: K-Clubs
58 - 109	Knock `58`: A-Spades, `59`: 2-Spades, ..., `70`: K-Spades `71`: A-Hearts ... `83`: K-Hearts `84`: A-Diamonds ... `96`: K-Diamonds `97`: A-Clubs ... `109`: K-Clubs

Row Index	Description
0	Current player's hand
1	Top card of the discard pile
2	Cards in discard pile (excluding the top card)
3	Opponent's known cards
4	Unknown cards

Column Index	Description
0 - 12	Spades `0`: Ace, `1`: 2, ..., `12`: King
13 - 25	Hearts `13`: Ace, `14`: 2, ..., `25`: King
26 - 38	Diamonds `26`: Ace, `27`: 2, ..., `38`: King
39 - 51	Clubs `39`: Ace, `40`: 2, ..., `51`: King

Review of the code and methods

Data preprocessing

- Collect winning game data
- Select positive and negative examples
- Dataset class

```
def collect_winning_data(num_games=100000):…

def select_training_examples():…

class GinRummyDataset(Dataset):…
```

Review of the code and methods

Siamese Neural Network.

Hand Embedding Layer. Converts the 52-card hand representation into a 128-dimensional feature vector. Uses ReLU activations and progressively reduces the size to 32 dimensions.
Action Embedding Layer. Converts actions (110 possible actions in Gin Rummy environment) into 32-dimensional embeddings. Uses one-hot encoding before passing it through a ReLU-activated linear layer.

We use Triplet loss for training. The model is trained so that good moves are consistently closer to the current hand representation than bad moves, by at least the margin.

Review of the code and methods

Setup:

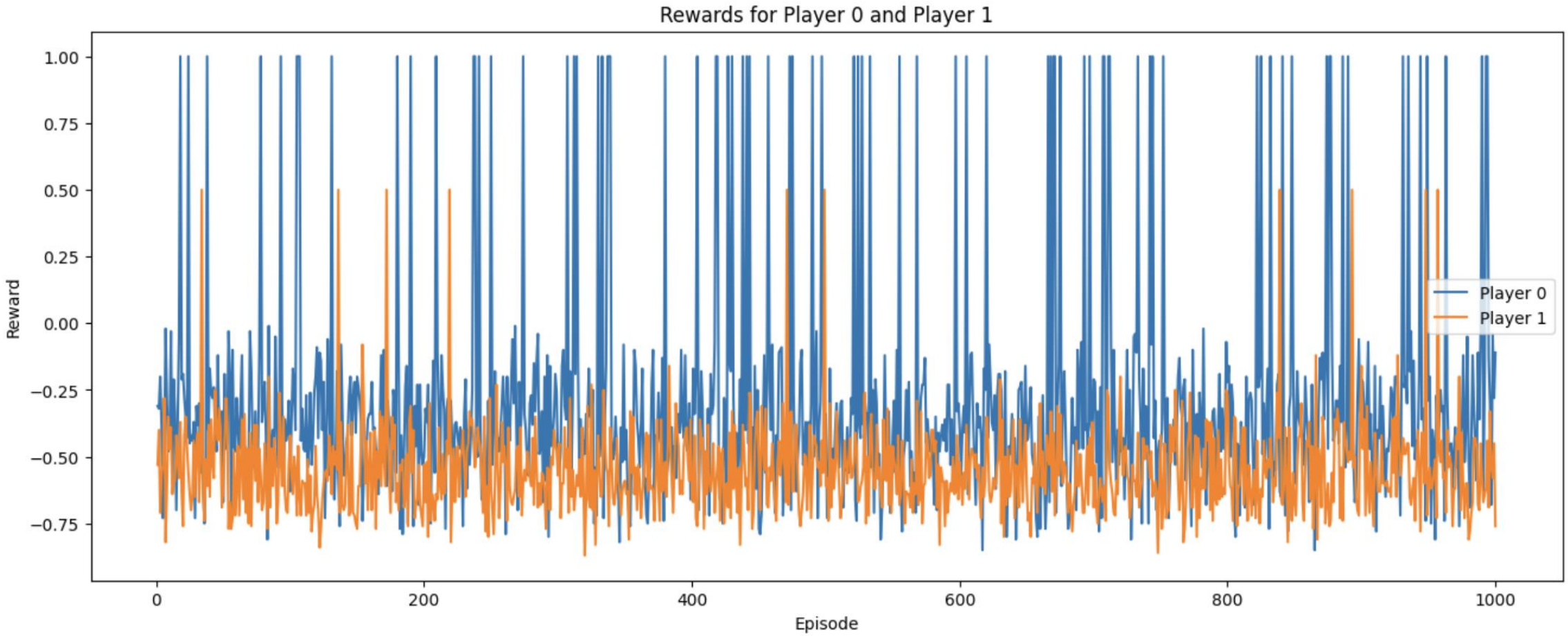
- Player 0 uses Siamese Neural Network to evaluate card usefulness
- Player 1 uses random policy

Evaluation:

- Win percentage
- Number of times Player 0 went Gin (best outcome)

Findings and results

- Player 0 **went Gin 81 times** (for comparing Player 1 went Gin 0 times)
- Win percentage: **75.3 %**



Conclusions

- ***gin_rummy_v4*** environment from *pettingzoo* is quite convenient to use for our task
- Obtaining training data for SNN manually from played games is quite time-consuming
- Using SNN significantly increases the number of times a player goes gin (achieves the highest reward).

For future work, we can consider using data from games played by expert players, to improve the quality of training data. Also, we can consider some experiments with hyperparameters.