Solitaire Using a Deep Q RL Network

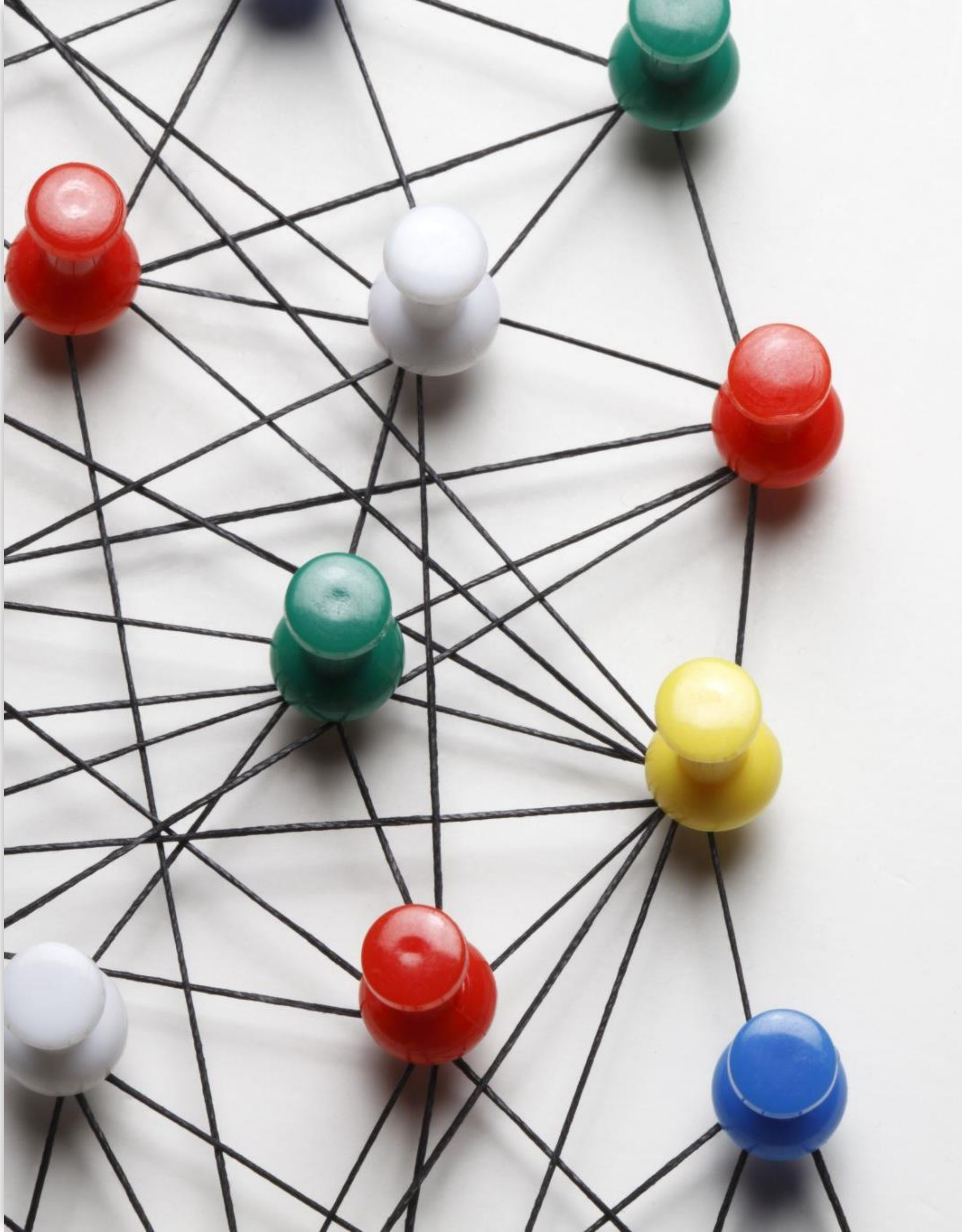
Grant Deljevic

Solitaire DQN - Introduction

- 3-card Draw Solitaire
- Develop a DQN model to play Solitaire
- Using Reinforcement Learning
- Learn an ‘objective’ strategy

Solitaire DQN - Deep Q Networks

- DQNs are a class of networks designed for complex state spaces
- Able to make long-term strategic decisions based on card positions and game rules.



Solitaire DQN - Environment: Overview

- Custom built Solitaire environment to simulate the game
- Initially intended to port another environment
- Structure

Solitaire DQN - Environment: Structure

Deck

- Cards
  - Suit (0 – 3)
  - Value (1 – 13)
  - Face up (0 or 1)
- Group Cards
- What can be drawn?

Table

- Cards taken from deck
- 7 Columns
- 4 "top" piles
- Varying cards per column
- Face up or face down

```python
def move_possible(self, fromCard, destCard, top=False):
    if not fromCard.face_up:
        return False

    elif isinstance(destCard, int):
        if destCard == -1:
            if fromCard.value == 13:
                return True
        elif fromCard.value == 1 and fromCard.suit == destCard:
            return True

    elif not destCard.face_up or fromCard.value != destCard.value + 1:
        return False

    elif top:
        if fromCard.suit == destCard.suit:
            return True

    elif fromCard.suit % 2 ^ destCard.suit % 2:
        return True

    return False
```

Solitaire DQN - Environment: Possible Moves

Solitaire DQN - Environment: Functions

Any_legal_moves	• Checks every possible move • Returns if there's a possible move
Get_possible_actions	• If there are legal moves, creates a list of all possible moves for the network
Check_game_over	• No moves are possible • Winning or losing conditions are fulfilled
Move_possible	• Checks all game rules for each possible move
Move_card	• Executes the move after checking if it's possible
Set_table	• Sets the initial game state in solitaire format

Environment: Network Interactions

- Get_reward
  - Based on how many cards are face up on the table
  - Large bonus for winning
- Get_state_representation
  - Encode the position of every card
    - Simpler than encoding an action
- Encode_action
  - Encodes possible moves
    - First 2 digits are card's value
    - Next digit is card's suit (red or black)
    - Next digit is the column we're moving a card from
    - Next digit is the column we're moving a card to
    - Next digit is whether or not the destination is the 'top' pile
- Decode_action
  - Decodes action decision
  - Receives values above

Solitaire DQN - Model Architecture

Input of encoded choices & game state

- State size is 26,624

Output of action choice

Training adjusts based on reward

Several dense linear layers

- Hidden layers

ReLU Activation Function

- Allows more complex learning

Final Layer

- Reduces hidden layer to number of possible actions

Solitaire DQN - Policy & Target Network

Policy network acts as the agent and makes decisions, weights are frequently and heavily updated. Learns quickly.

Target network is there to keep a more stable version of weights, and ensure the policy network doesn't collapse.

This allows the policy network to explore aggressively without risk of collapse

Solitaire DQN - Training

Initialize replay memory to store experiences

Training Loop

Reset the environment & receive new initial state for each episode

Process the game state through **get_state_representation** for input into the DQN.

Action Selection (Epsilon-Greedy Strategy):

With probability ε, select a random action for exploration.

Otherwise, choose the best action based on the model's prediction (exploitation).

Gradually decrease ε over time to shift from exploration to exploitation.

Pass action to environment, receive reward & next state

Solitaire DQN - Training (cont.)

- Store the transition (state, action, reward, next state) in replay memory.
- Batch Learning
  - Generate random batch sample
  - Calculate loss & update the model
- Every few steps, update the target network with the weights of the policy network
- Log training metrics such as loss and rewards
- Assess if the model's performance is converging

Solitaire DQN - Challenges & Solutions

Challenge: Representing the complex state of a Solitaire game

Solution: Develop a function to encode the state into a numerical format understandable by the DQN.

Challenge: Ensure the model chooses valid actions

Solution: Find all possible actions and separately encode them, allowing the network to have the current state while narrowing down the actions it can take to those possible

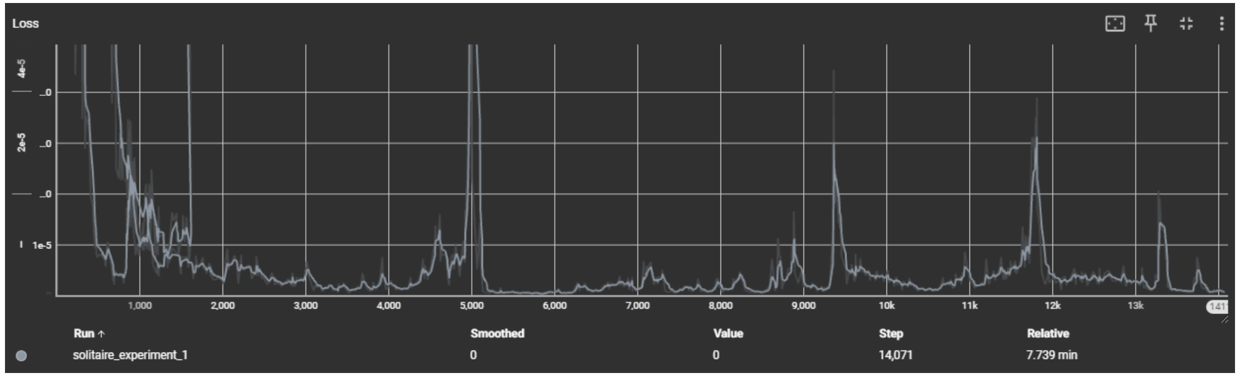
Challenge: Representing consequences of actions to help the network learn

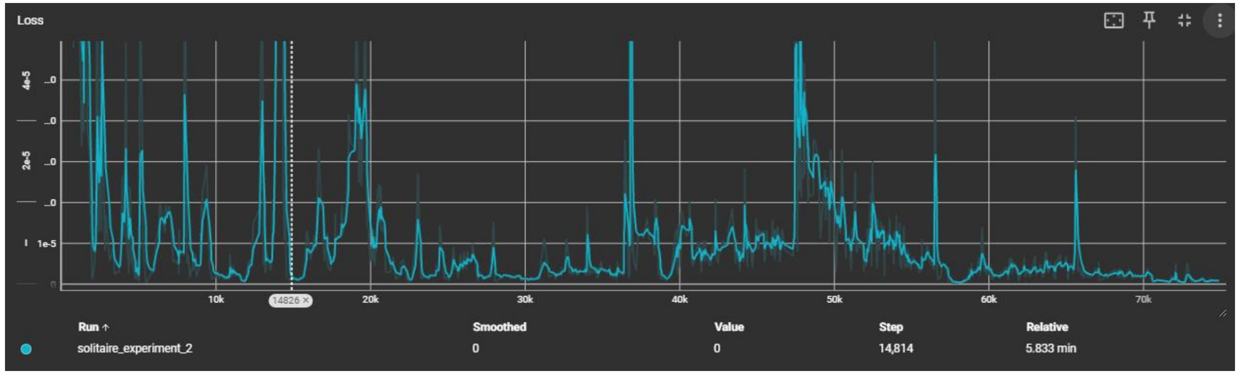
Solution: Store future states in an action state tree, with probabilities of what revealed cards would be

Note: This is hypothetical

Solitaire DQN - Results and Performance

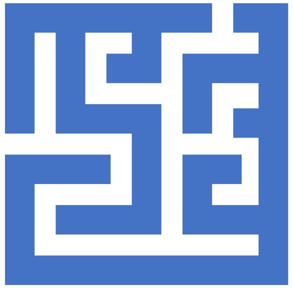
- Monitoring loss and reward progression.
- Loss was good, reward was not





Solitaire DQN - Conclusions and Learnings

Successfully developed a DQN, though teaching it to play Klondike Solitaire will take more work



Learned about the complexities & pitfalls of applying reinforcement learning to a traditional card game, as well as some of the more intricate parts of DQN

Solitaire DQN - Future Work and Improvements

Hyperparameter tuning to optimize performance

Finish encoding of actions and joining it with the state representation

Creating the action space tree

Optimization – making functions faster

Full GPU utilization

Solitaire DQN -
Questions?

Solitaire DQN - Q&A

Q1: What is a DQN good for?

- A1: Complex spaces with many options, where a model will need to prioritize long-term strategic thinking

Q2: What was the biggest challenge in training an AI to play Solitaire?

- A2: Creating a quantifiable environment which followed the rules of Solitaire

Q3: How do you measure the success of a trained model?

- A3: Monitor loss, reward, and when that fails, watch it play!

Q4: What is the difference between a policy and target network in DQN?

- A4: The policy network is the more rapidly-updated network which makes action decisions, the target network is there to keep a more stable version of weights.

Q5: What was the ML algorithm used with DQN?

- A5: Reinforcement learning

Solitaire DQN - Citations

1. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533. https://doi.org/10.1038/nature14236

2. Sutton, R. S., & Barto, A. G. (2018). Reinforcement Learning: An Introduction (2nd ed.). MIT Press. http://incompleteideas.net/book/the-book-2nd.html

3. PyTorch. (n.d.). Documentation. PyTorch. https://pytorch.org/docs/stable/index.html

4. Python Software Foundation. (n.d.). Python 3.9.1 documentation. Python.org. https://docs.python.org/3/

5. TensorFlow. (n.d.). TensorBoard: TensorFlow's visualization toolkit. https://www.tensorflow.org/tensorboard

6. Python Software Foundation. (n.d.). Unittest — Unit testing framework. Python 3.9.1 documentation. https://docs.python.org/3/library/unittest.html