Learning Strategies Through Reinforcement Learning

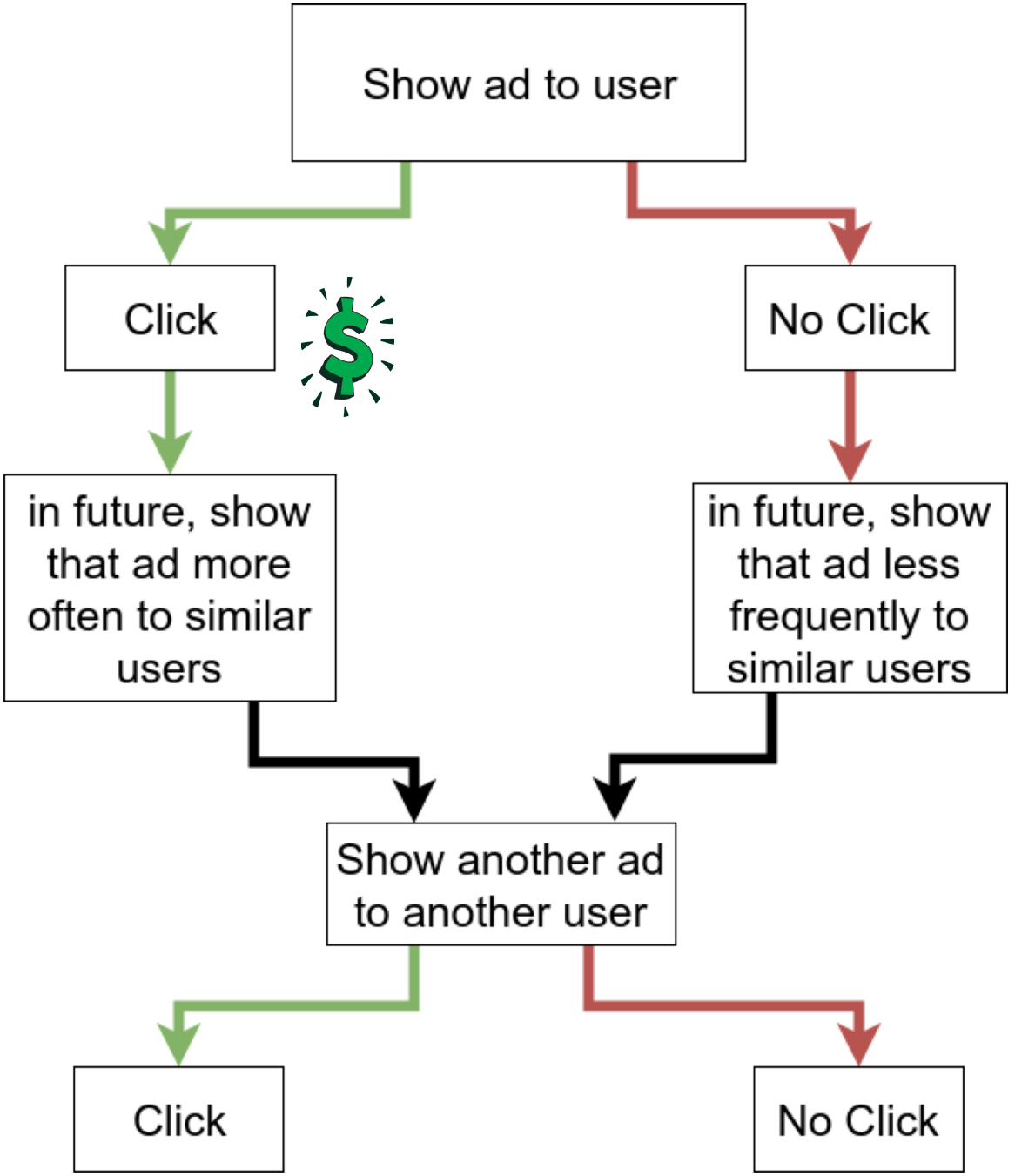
Max Candocia

April 27, 2018

What is Reinforcement Learning?

- Teaching a program to perform a task by
  - Simulate task many times
  - Send feedback on simulations
    - e.g., score or chance of success
  - Simulate again with updated AI that learned from feedback
  - Repeat above
- Task must involve decisions that the program makes

Reinforcement Learning Example: Advertising



Main Considerations

- What is the **input**?
- What is a **response**?
- How does the program **act** on the response?

Main Problem

I want to teach a computer to play board games

AND

I want to be able to learn strategies from it

Ideal Characteristics for First Game

- Few rules
- Simple turn structure
- Easy way to describe game using numbers
- Little, if any, hidden information
- No easy-to-do “bad” moves
- Relatively quick

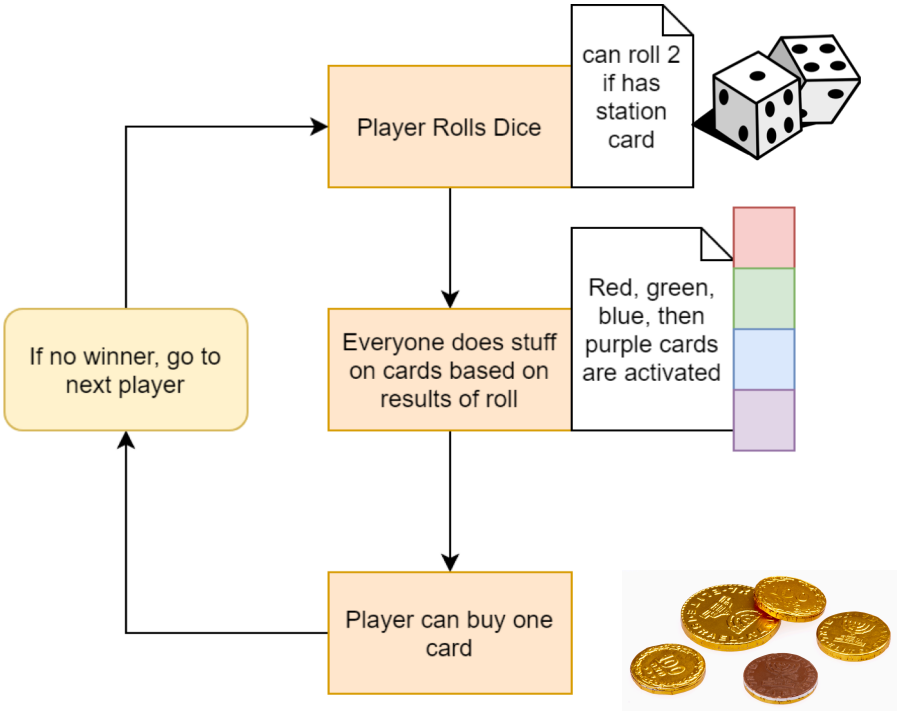
Game 1: Machi Koro

- 2-4 player game
- Uses dice rolling to determine what cards each player has are activated
- Players can buy cards that have different effects on different dice roll values
- Winner is first person to buy 4 specific cards

Image sources: http://www.theboardgamefamily.com/2016/02/machi-koro-card-game-review/

Image sources: http://www.theboardgamefamily.com/2016/02/machi-koro-card-game-

Machi Koro Turn Order



Machi Koro Input

Variables:

- Player cards
- Player coins
- Cards on board (optional)

Player Data – Readable Input

- There should be a simple way to represent game data
- Programming is easier when you understand these values
- These cannot be directly input into a model, however

```
game_data = list(
    player_1=list(
        coins=2,
        cards=list(
            wheat_field=1,
            bakery=1,
            ranch=0,
            ...
        )
    ),
    player_2=list(
        …,
    ),
    ...
)
```

* Original program in Python 2, so code is an approximate translation

Player Data – Vector Input

- Model input is almost always vector-like
- Writing functions to convert readable to vector helps avoid mistakes
- Having constants describe the order

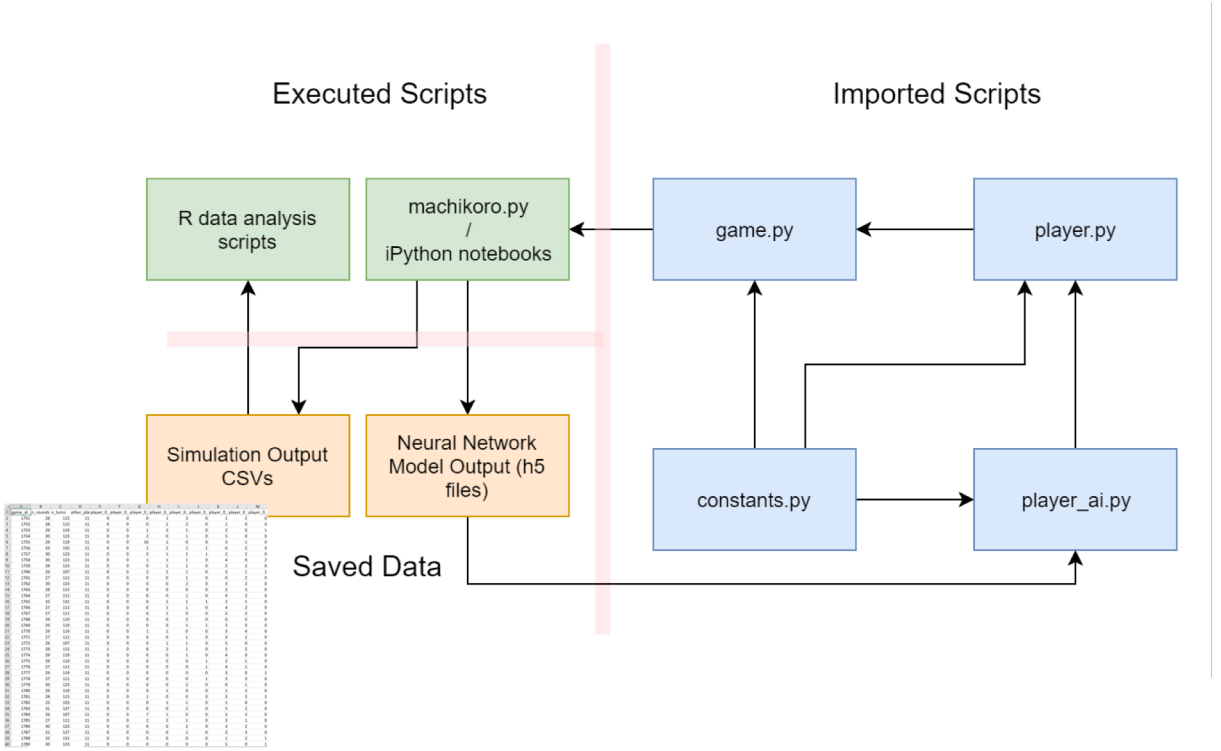
```
card_order = sort(c('bakery','ranch','mine','forest','stadium',...))
player_list = c('player_1', 'player_2', 'player_3', 'player_4')

serialize_player_input <- function(player){
 input = game_data[[player]][['coins']]
 for (card in card_order){
  input = c(input, game_data[[player]][[card]])
 }
 input
}

game_input_serialized = unlist(sapply(
  player_list, serialize_player_input
))

# game_input_serialized initial value:
# c(2,1,1,0,0,0,0,0,0,0,0,0,...,2,1,1,0,0,0,...)
```

Project File Structure

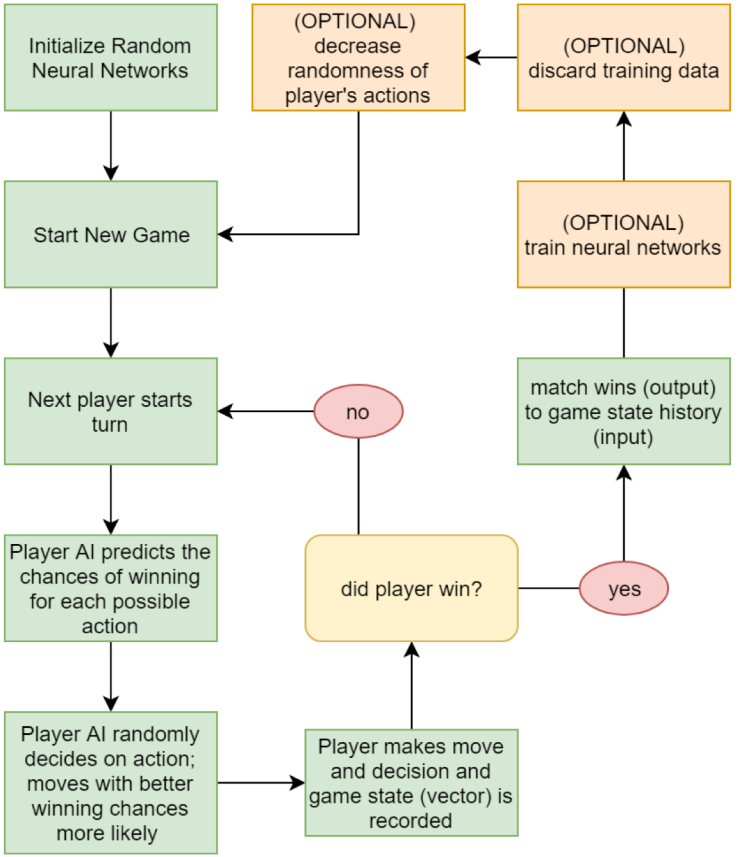


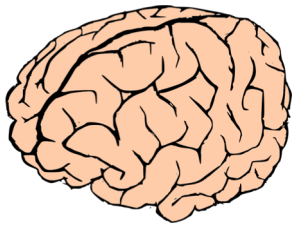
Jupyter Notebooks

- Browser notebooks that can run blocks of code at a time
- Originally developed for Python, but can be used for R, Julia, and many other languages
- Useful for sharing code, output, and visualizations
- Allows others to replicate your results

https://github.com/mcandocia/machi_ai/blob/master/exploration_second_round.ipynb

Learning Process



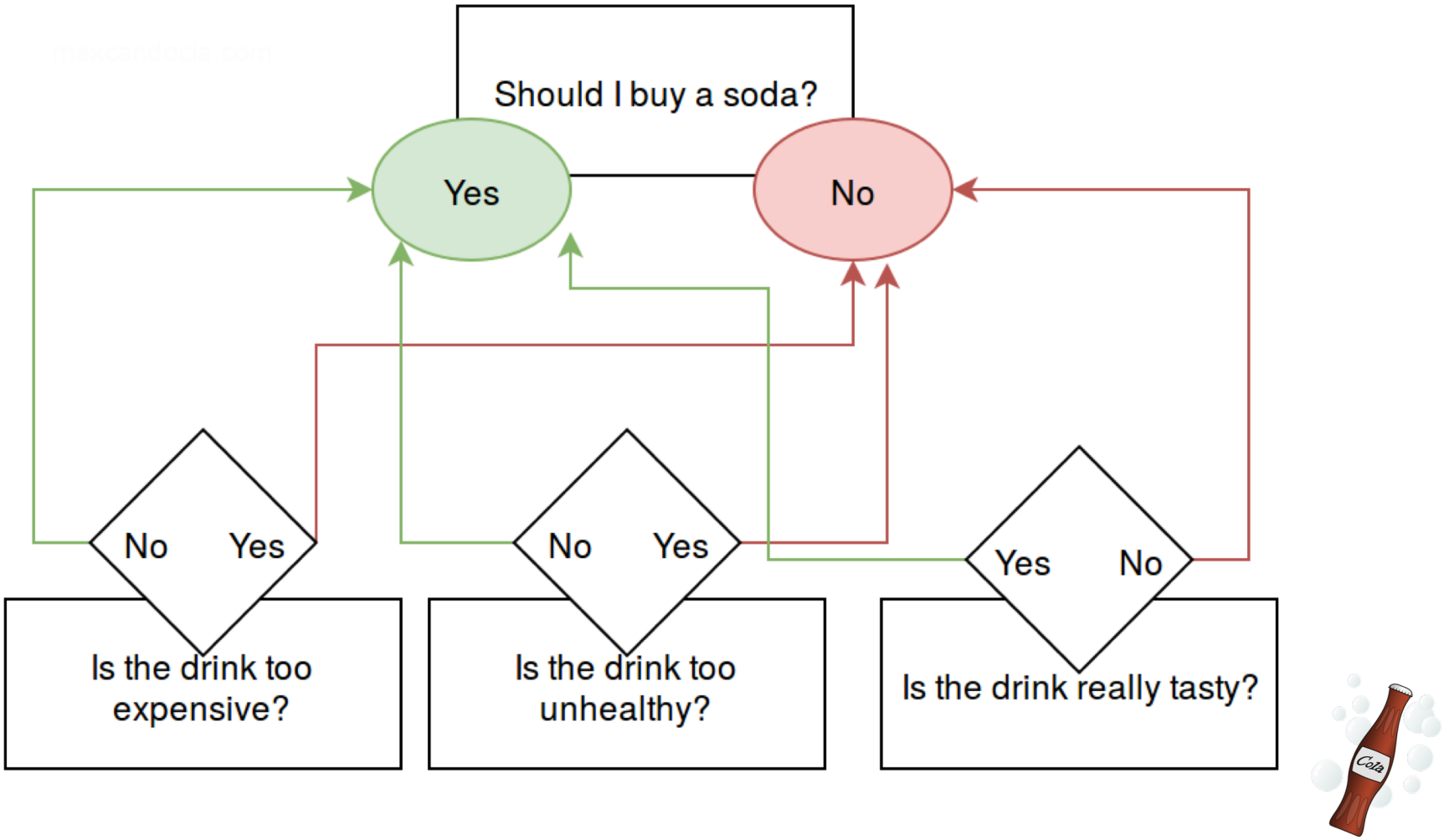


Neural Networks

- Inspired by brain “neuron” architecture
- Can solve very complex problems
- Can take arbitrary numeric input
- Can layer on top of each other
- Internal workings are often indecipherable, especially with many layers

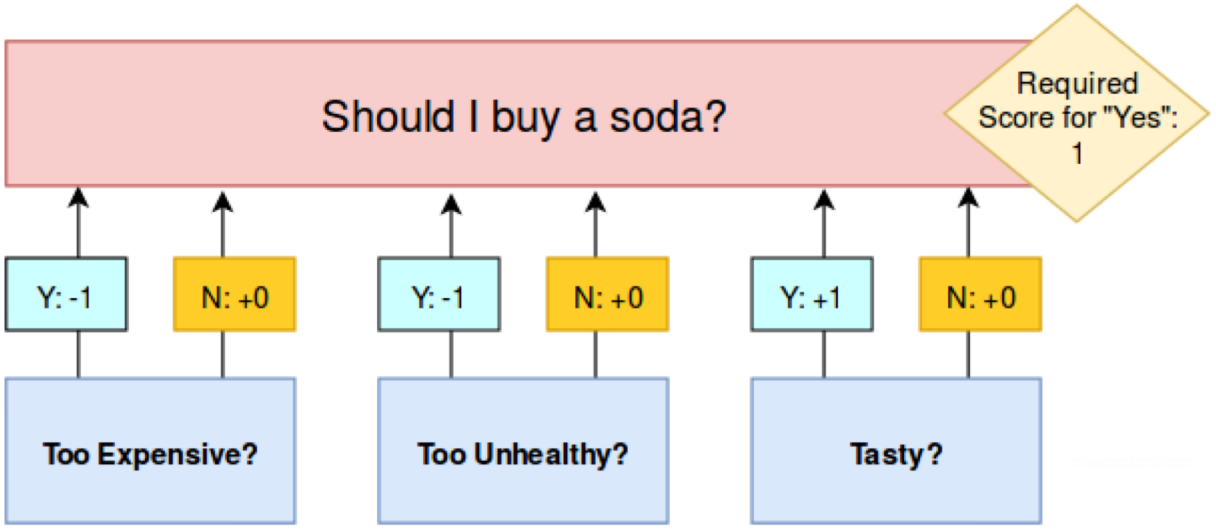
Describing a Flow Chart as a Neural Network

The below is a flowchart I sometimes use when deciding whether or not to buy a soda



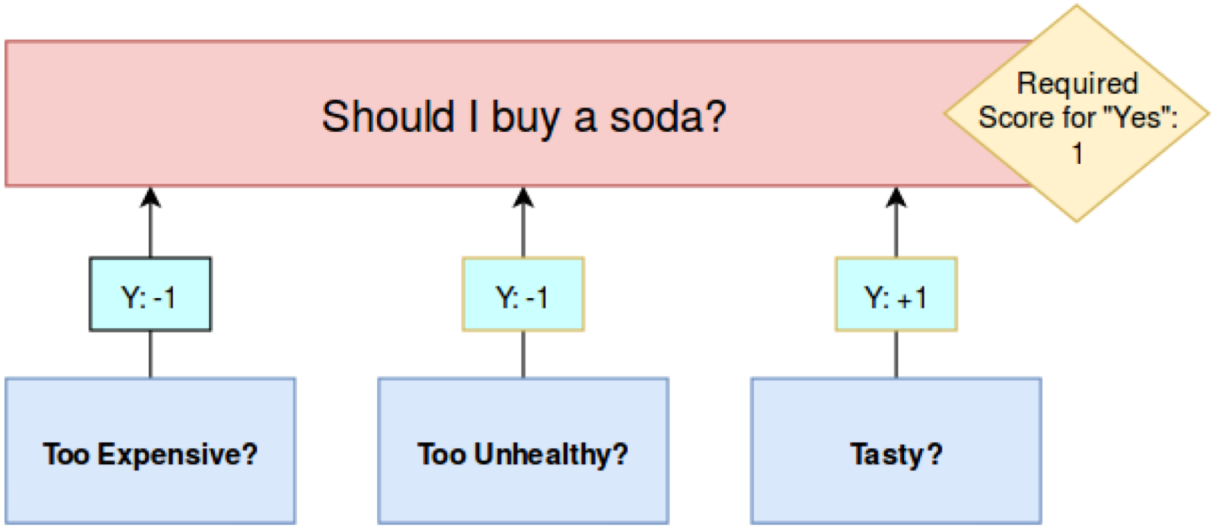
Describing a Flow Chart as a Neural Network (2)

The chart can be simplified with a scoring system



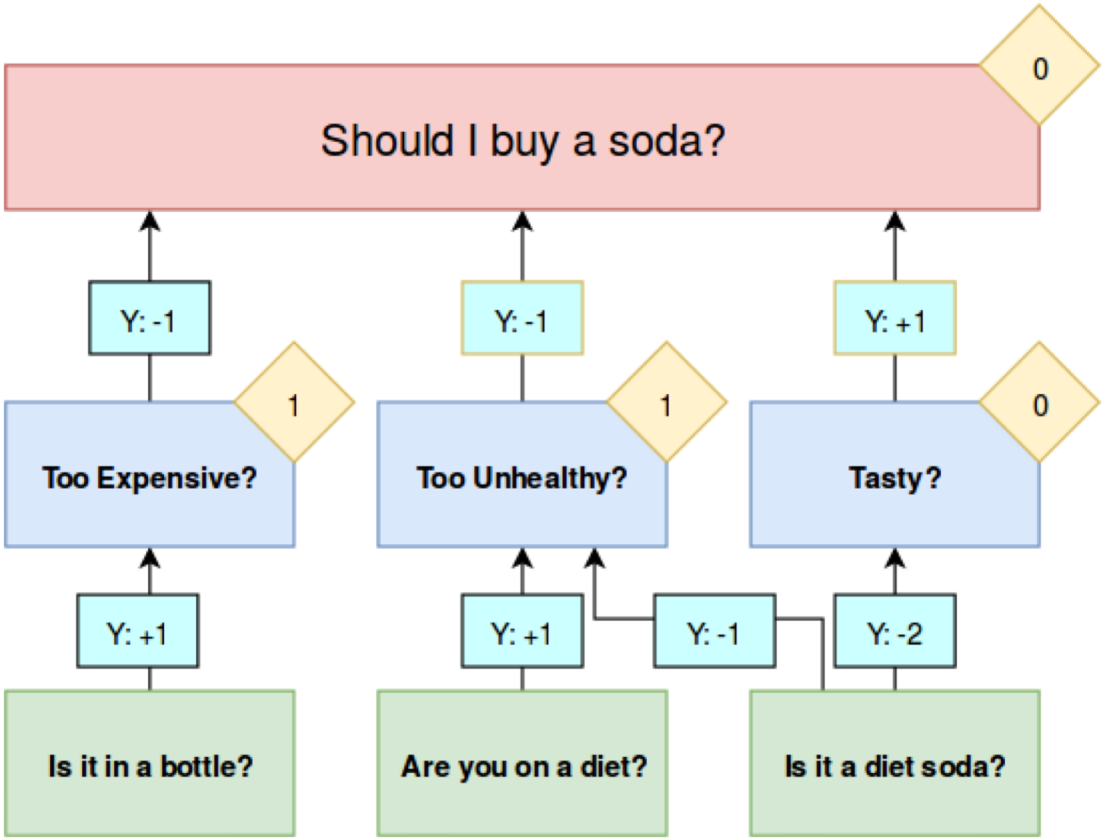
Describing a Flow Chart as a Neural Network (2)

Since the “No” values are all 0, we can just look at the “Yes” scores



Describing a Flow Chart as a Neural Network (3)

We can add a non-subjective input layer to help calculate the middle layers

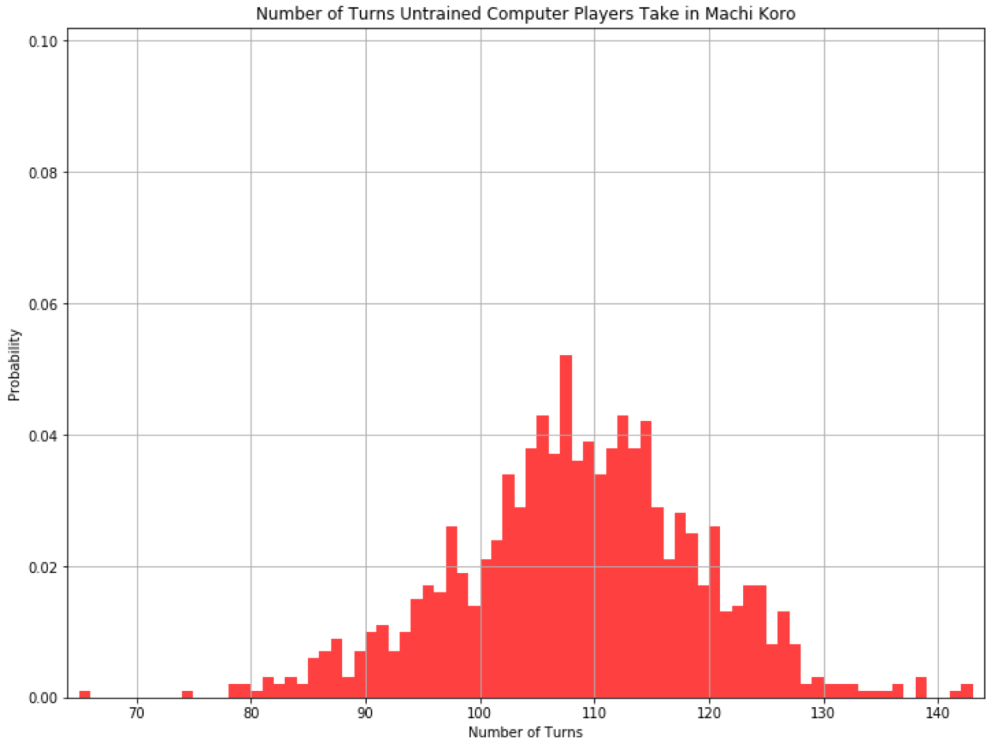


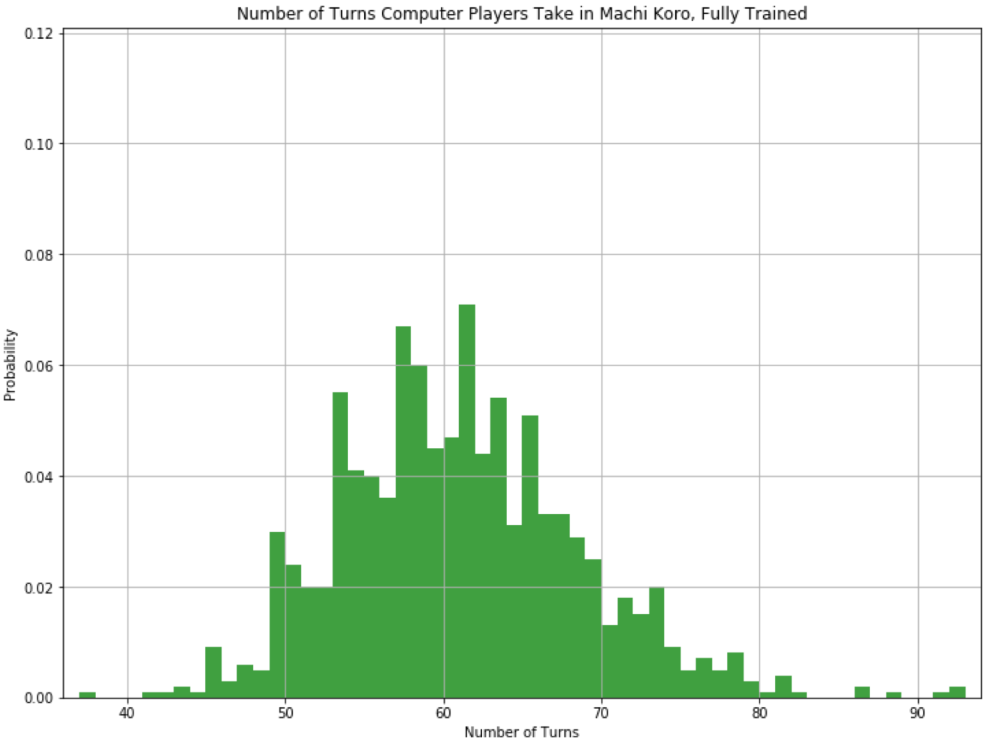
Note: the final layer in this chart has a lower score requirement than of the previous models

Measuring Performance

- How many turns does the AI take?
- How well does the computer perform against less well-trained (or random, untrained) AIs?
  - Wins 98.5% of games against untrained

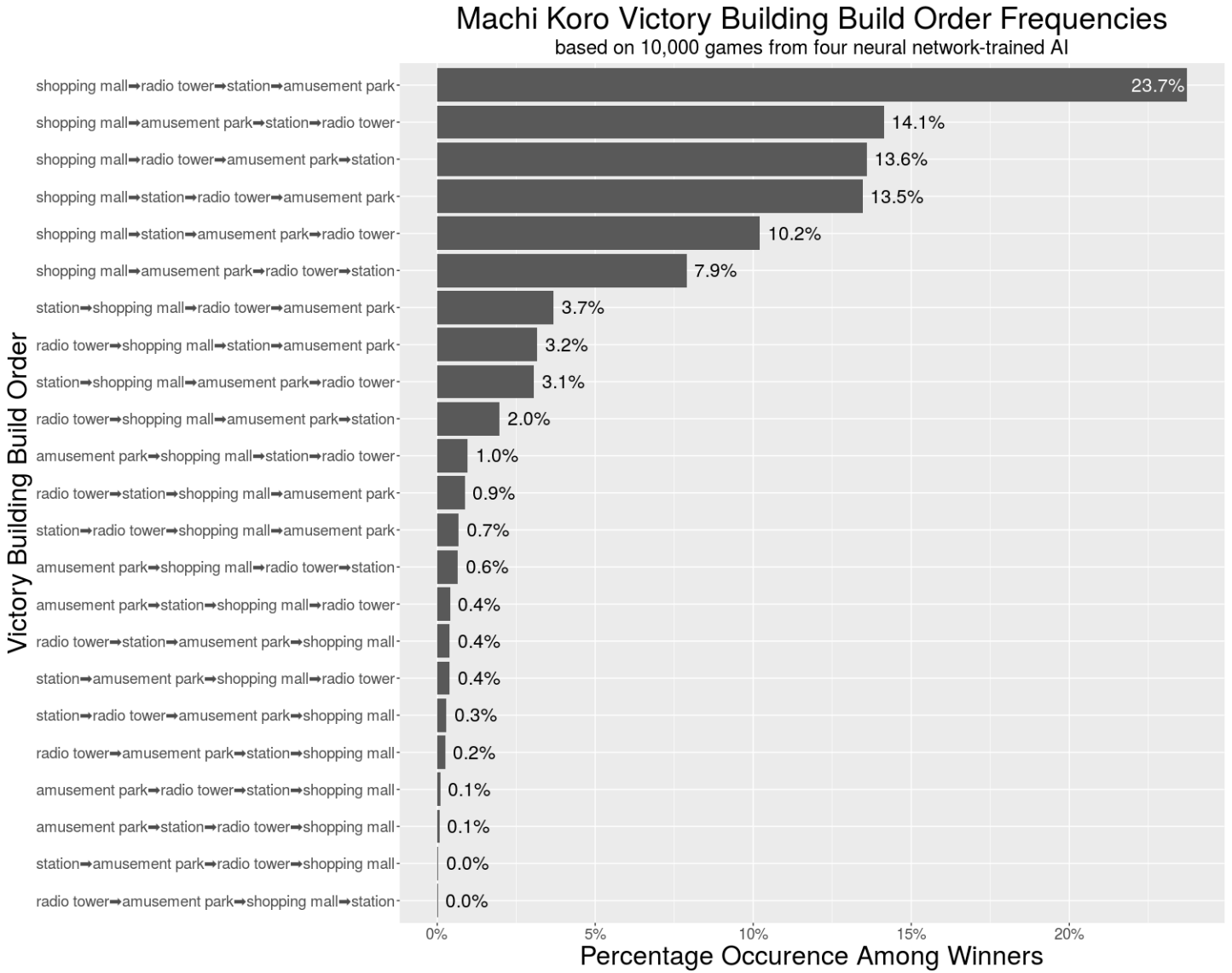
Number of Turns in Untrained vs. Trained AI Games





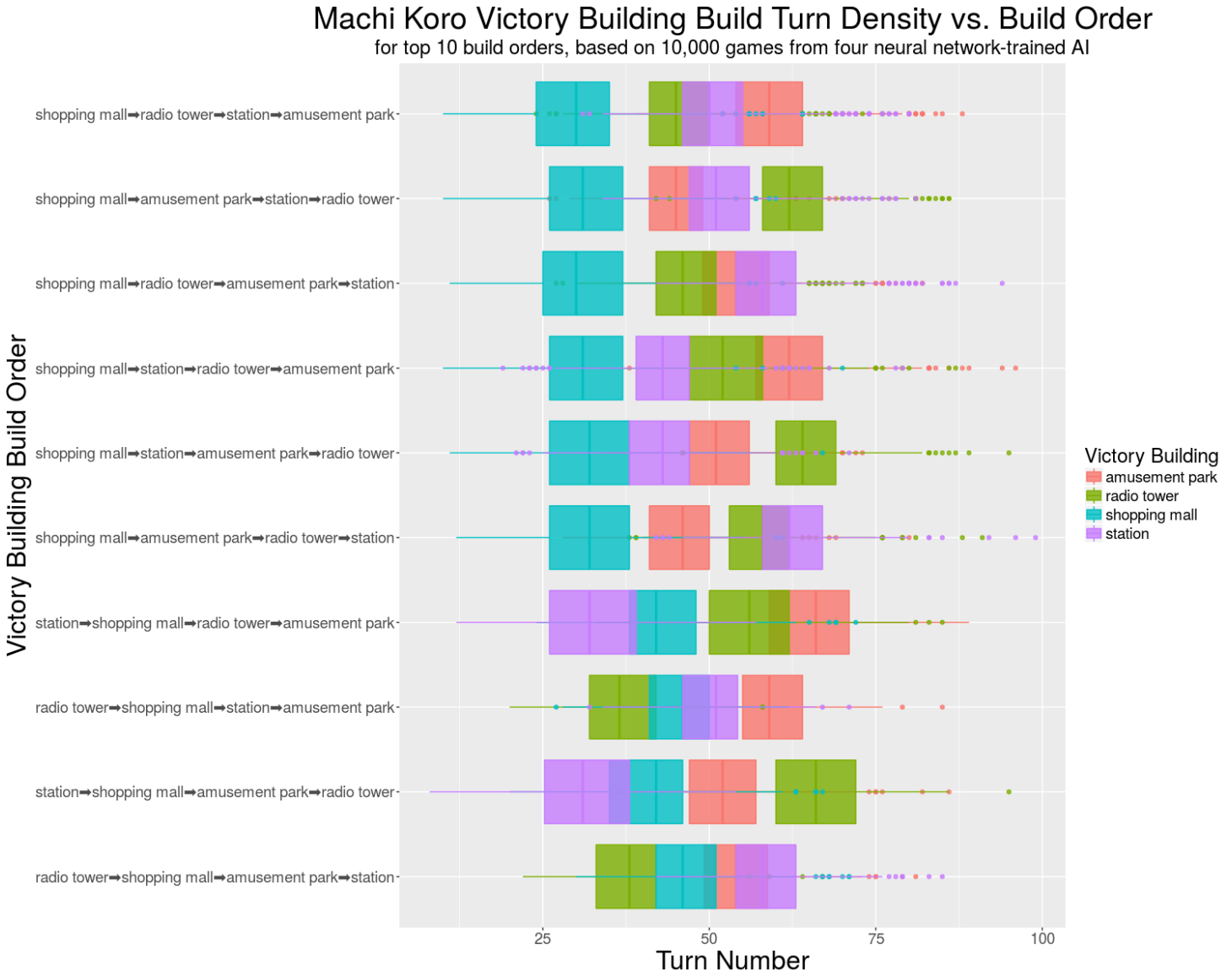
The untrained (red) network takes about 110 turns (27 per player) on average, while the trained network takes around 60 turns (15 per player) on average

Strategic Insights – Build Order



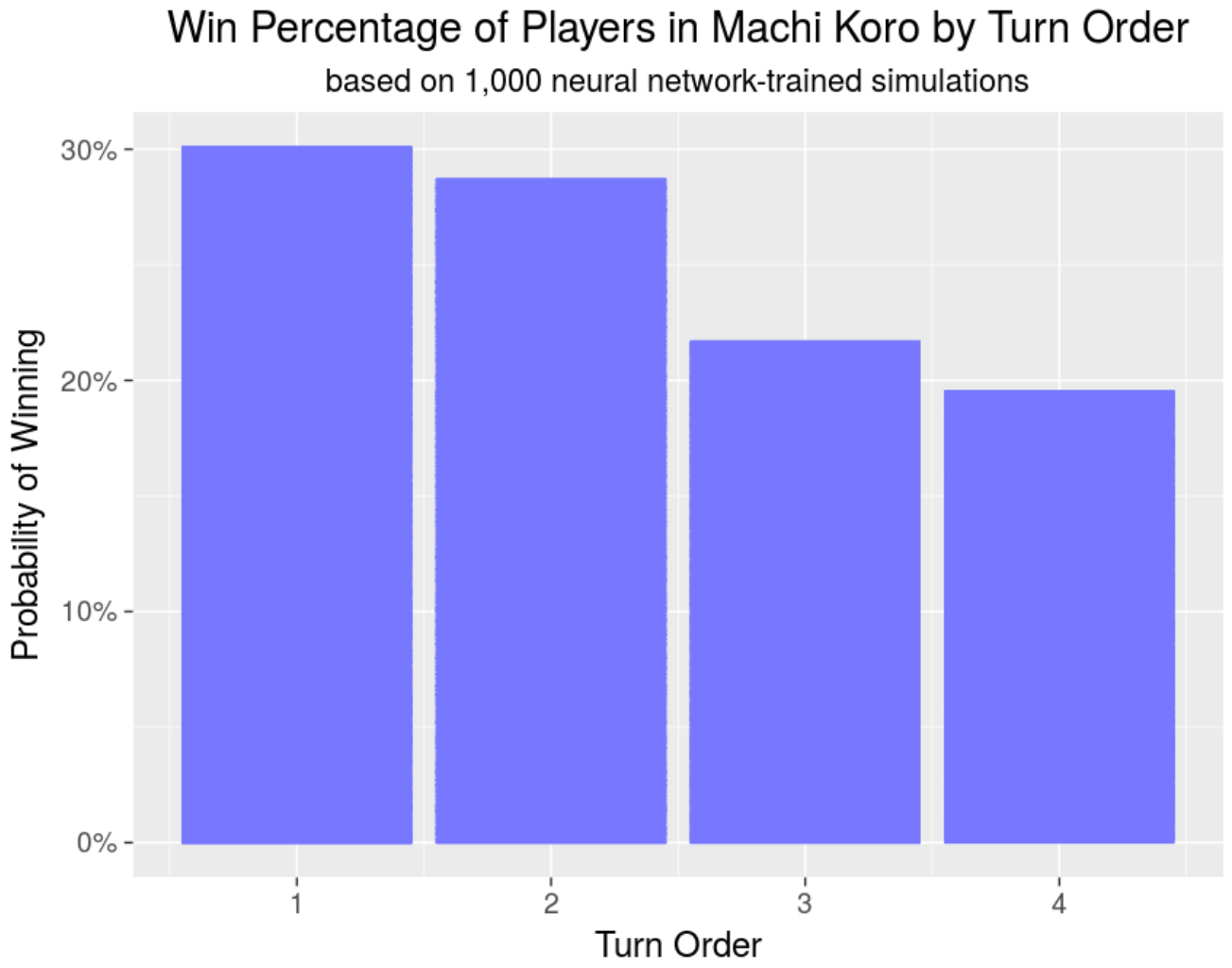
These are the buildings built by the winning player over simulated games

Strategic Insights – Build Order vs. Turn



Shopping mall is popular, but an early radio tower speeds up the game significantly

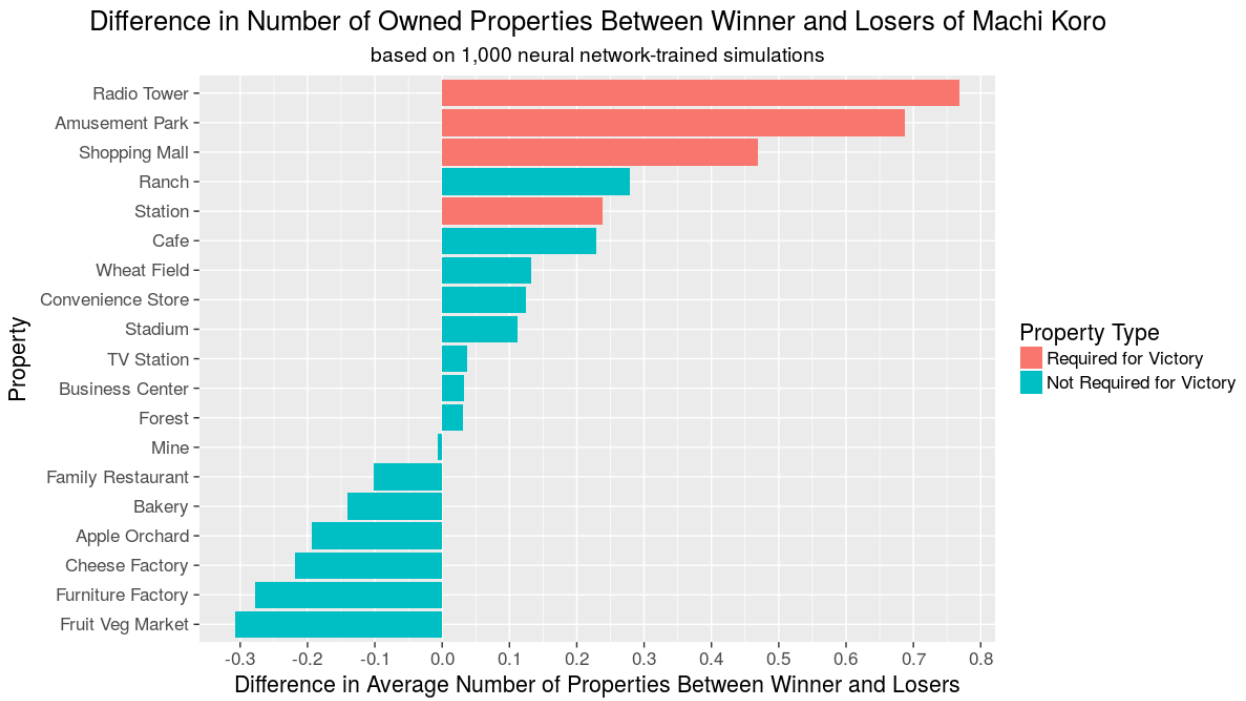
Strategic Insights – Turn Order



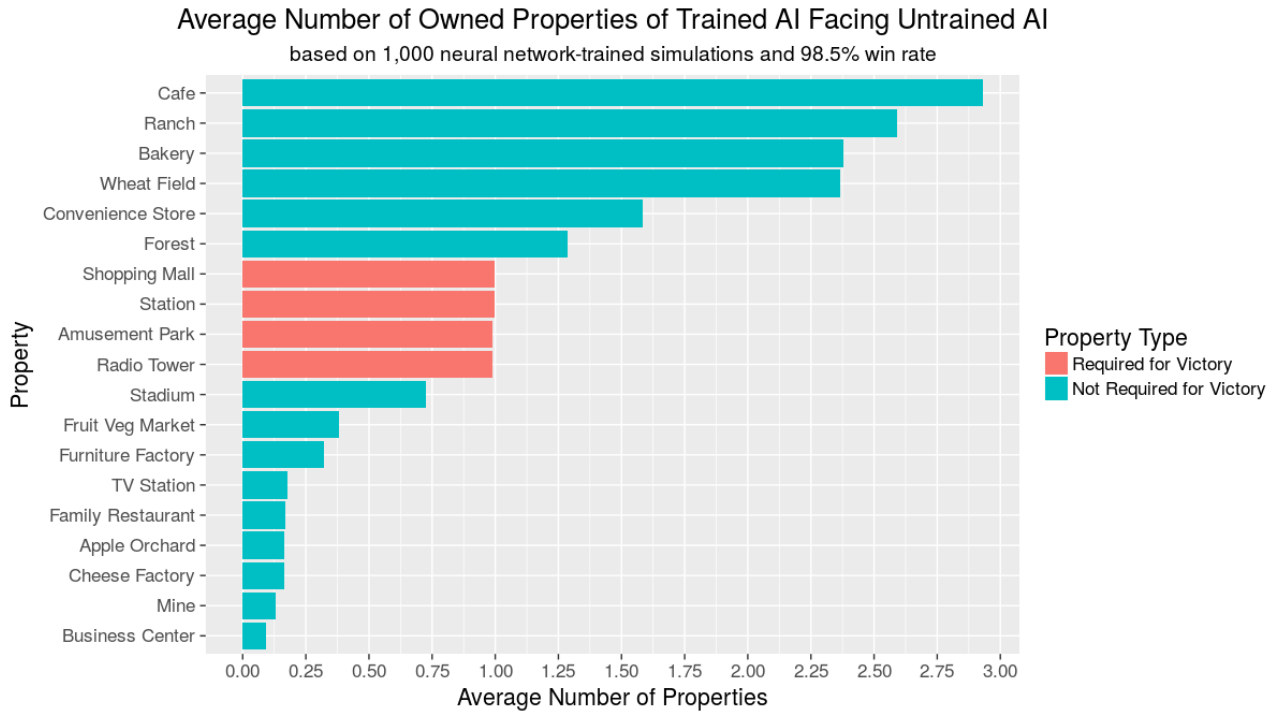
Game is very biased

Strategic Insights – What to Build

Strategic Insights – Difference Between Winners and Losers



Strategic Insights – Difference Between Winners and Losers



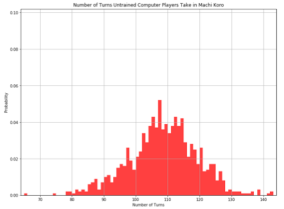
Comparison of Programming Languages Used

Python

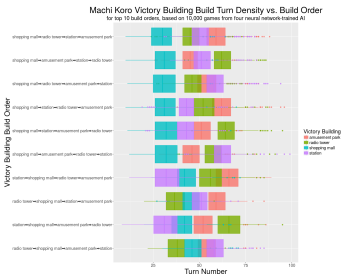
- Better with complex data structures
- Naming problems less common
- Better for general programming tasks

R

- Much easier to work with tabular data
  - Summarizing data
- Much easier and better results for most visualizations



Same number of lines of code



Game 2: Splendor

- Simple rules, more complicated planning
- Objects on board have more complex representations
- Points are awarded, which gives a well-defined short-term goal

http://meepletown.com/2014/06/game-designer-interview-marc-andre/

Splendor Rules

- One action each turn
  - Take 2 gems from 1 pile
  - Take 3 gems from different piles
  - Reserve a card and take a gold gem (topaz)
  - Purchase a card
- Points
  - Points on cards
  - Objectives
- Discounts

http://meepletown.com/2014/06/game-designer-interview-marc-

Applying Lessons Learned From Previous Attempts

- Unsuccessful attempts at games of *Ticket to Ride* and *Settlers of Catan*
- Multiple objectives
  - Points in 1, 3, and 5 turns
  - Winning the game
- Embedded Representation
  - Represent complex objects, such as players and cards, as a few numbers
- Limit number of models used
  - Represent the predicted board state instead of the current one + action

Multiple Objectives and Rewards

- 4 scores are used:
  - IN ONE TURN: 1*points + 0.1*cards_owned + 0.01 * gems_owned (Q1)
  - IN THREE TURNS: 1*points + 0.1*cards_owned (Q3)
  - IN FIVE_TURNS: 1*points + 0.05*cards_owned (Q5)
  - 15*probability of winning the game (WIN)
- Weighted sum of scores used
  - Sort of arbitrary
  - E.g., 0.3 * Q1 + 0.2 5* Q3 + 0.25 * Q5 + 0.2*WIN
- Probability of taking an action
  - $\frac{e^{score(action_x)/T}}{\sum_i e^{score(action_i)/T}}$
  - *T is "temperature"*
    - *Higher = more random*
    - *Lower = less random, "better" decisions more likely*
  - *Inspired by Boltzmann distribution in thermal physics*

```
> probs <- function(x, T=1){exp(x/T)/sum(exp(x/T))}
> probs(c(1,2))
[1] 0.2689414 0.7310586
> probs(c(1,2), T=5)
[1] 0.450166 0.549834
> probs(c(1,2), T=0.2)
[1] 0.006692851 0.993307149
```

Embedded Representation

- Common use/technique with neural networks
- Reduced representation of players and cards from 15 numbers to 10
- Makes learning models faster and more consistent
- Not usually 100% human-readable

Green: 1
Arms: 1
Wings: 0
Teeth: 0

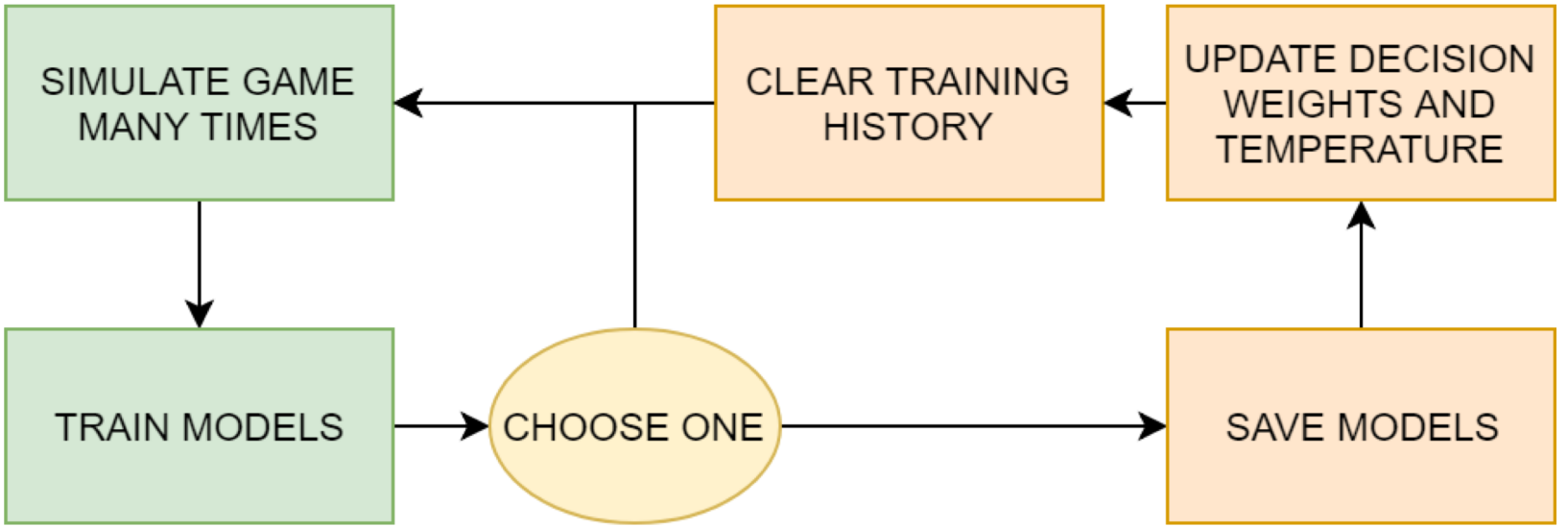
Green: 1
Arms: 1
Wings: 1
Teeth: 1

Green: 0
Arms: 0
Wings: 1
Teeth: 1

Green: 1
Arms: 0
Wings: 0
Teeth: 0

Images collected from *Sketching and Guessing with Keras* Application:
http://maxcandocia.com/app/sketching-and-guessing-with-keras/

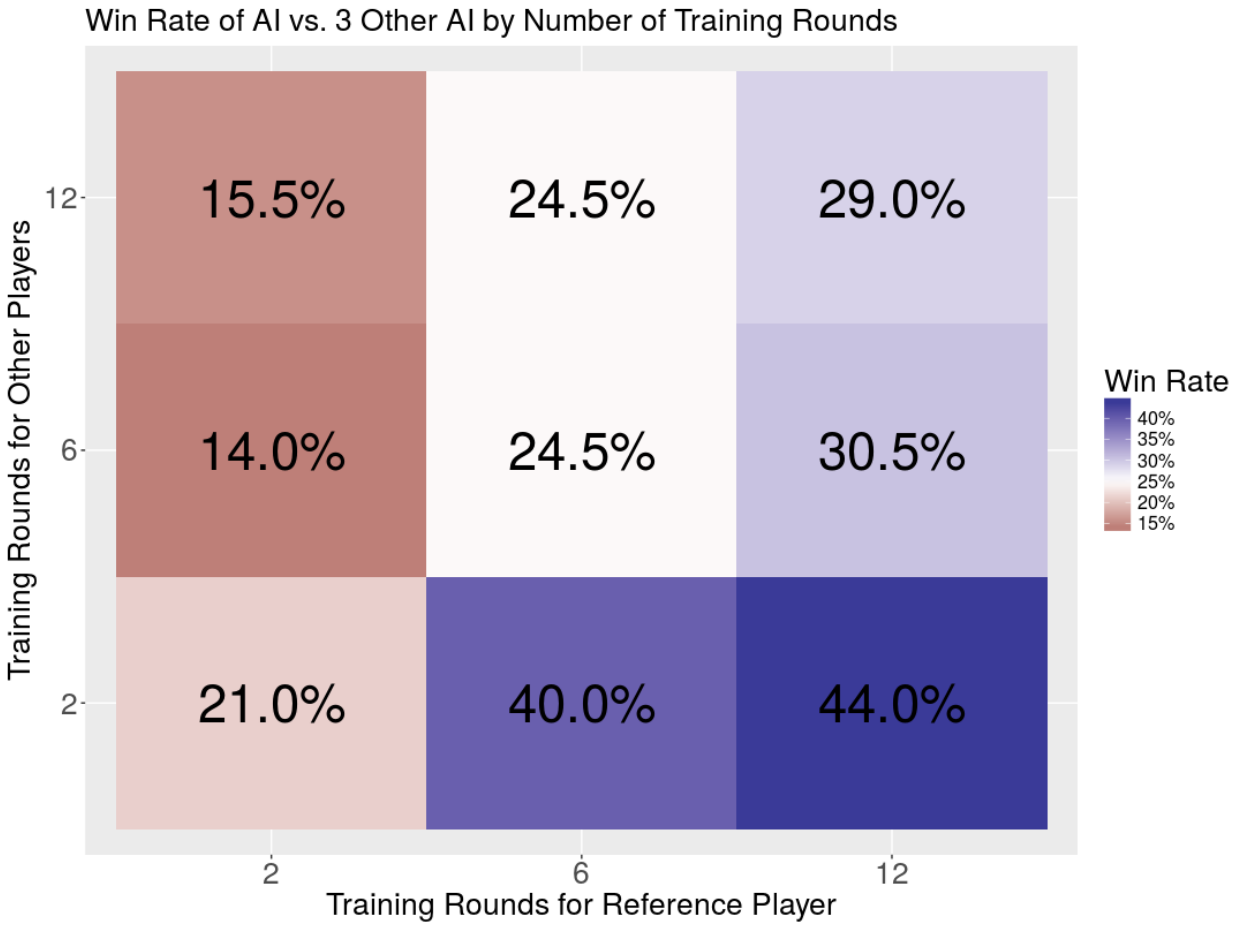
General Training Strategy



Splendor Reinforcement Learning Results Method

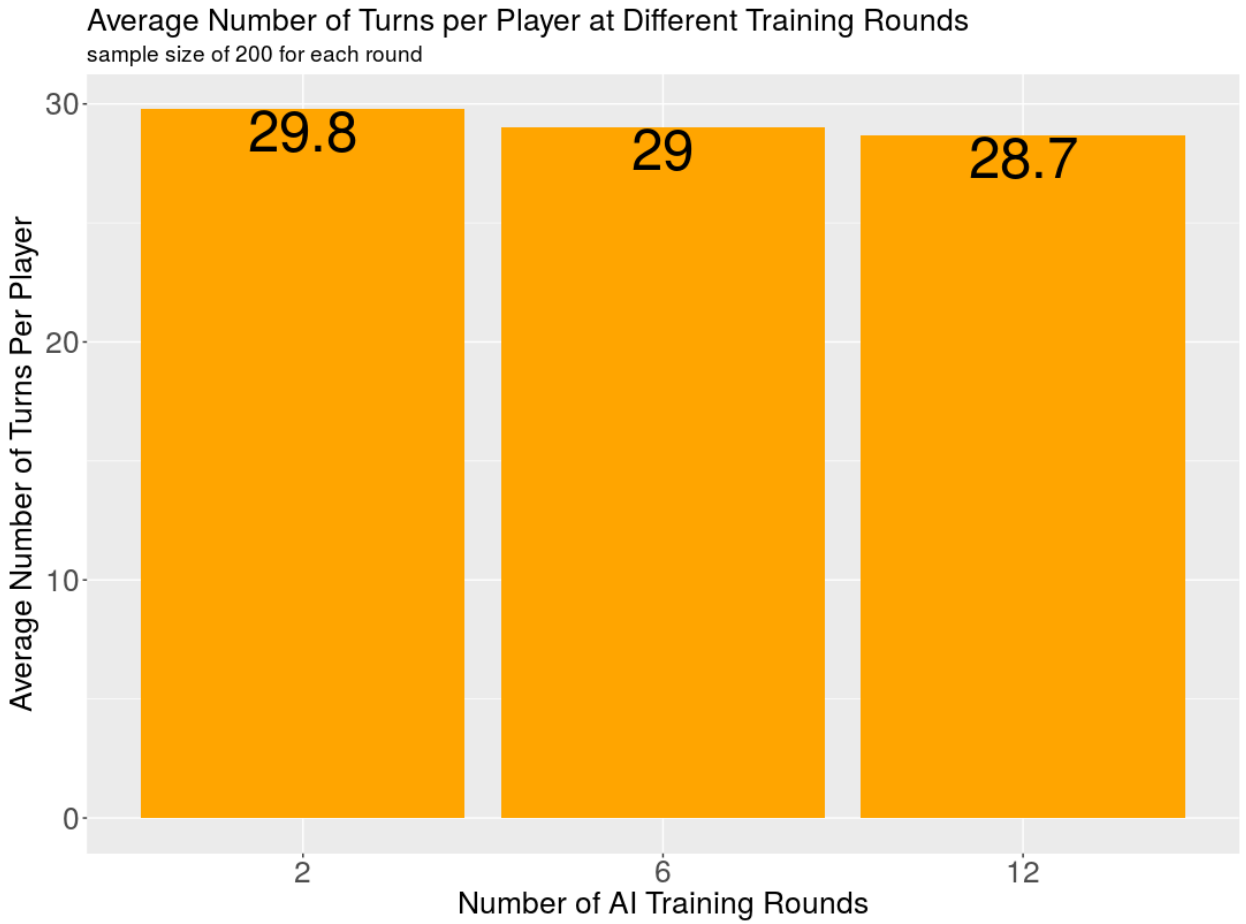
- Compared AI networks trained after 2, 6, and 12 rounds
- Pairwise competition (e.g., 2 vs 6 vs 6 vs 6)
  - Demonstrates that players improve after training
- Same-round competition (e.g., 2 vs 2 vs 2 vs 2)
  - Using winners of those rounds lets us see “goal”/strategy of AI

Win Rate Comparison



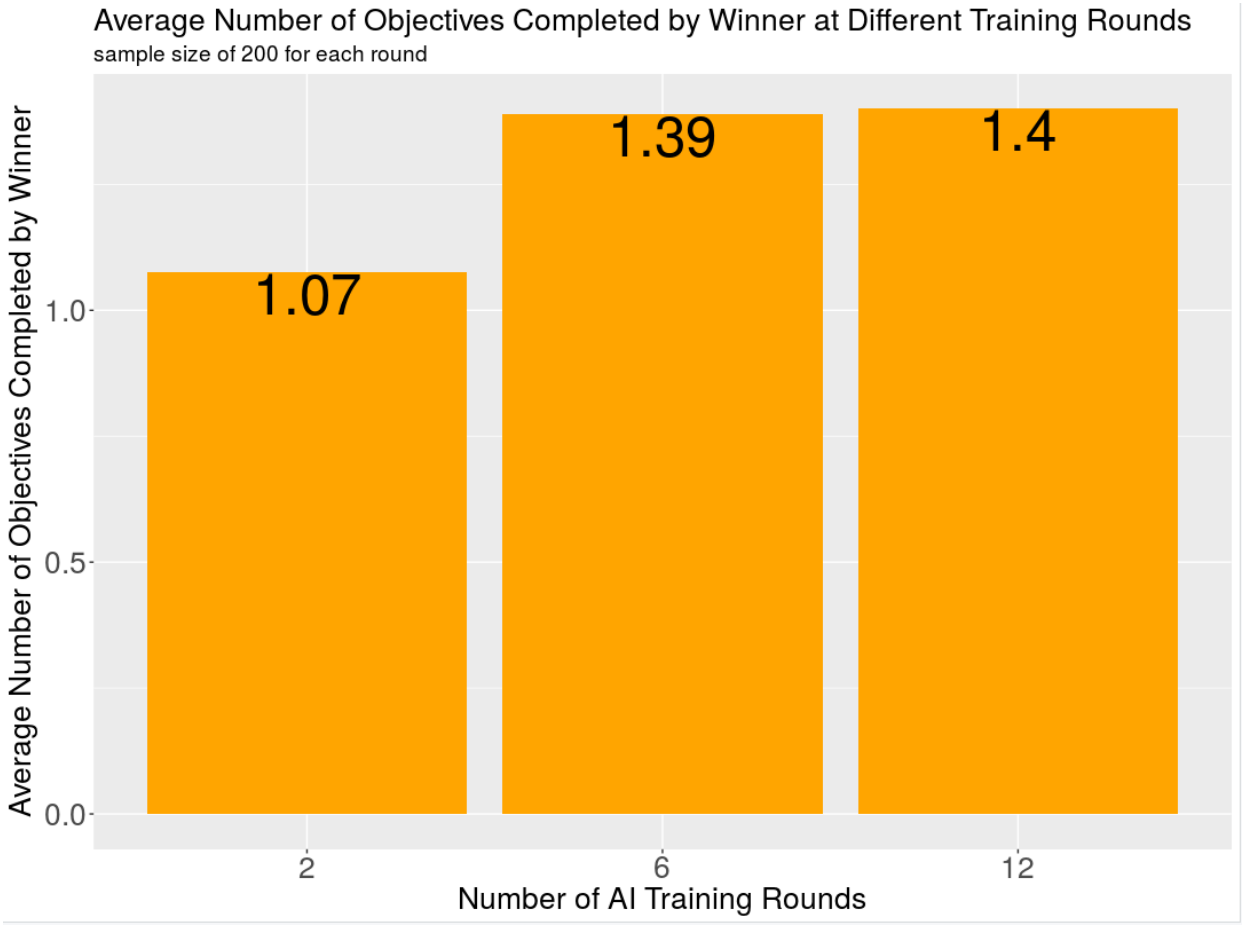
Obvious improvements of AI networks trained 6 or 12 rounds over 2 rounds, but no noticeable difference between 6 and 12 rounds of training

Differences in Winning Strategies (1)



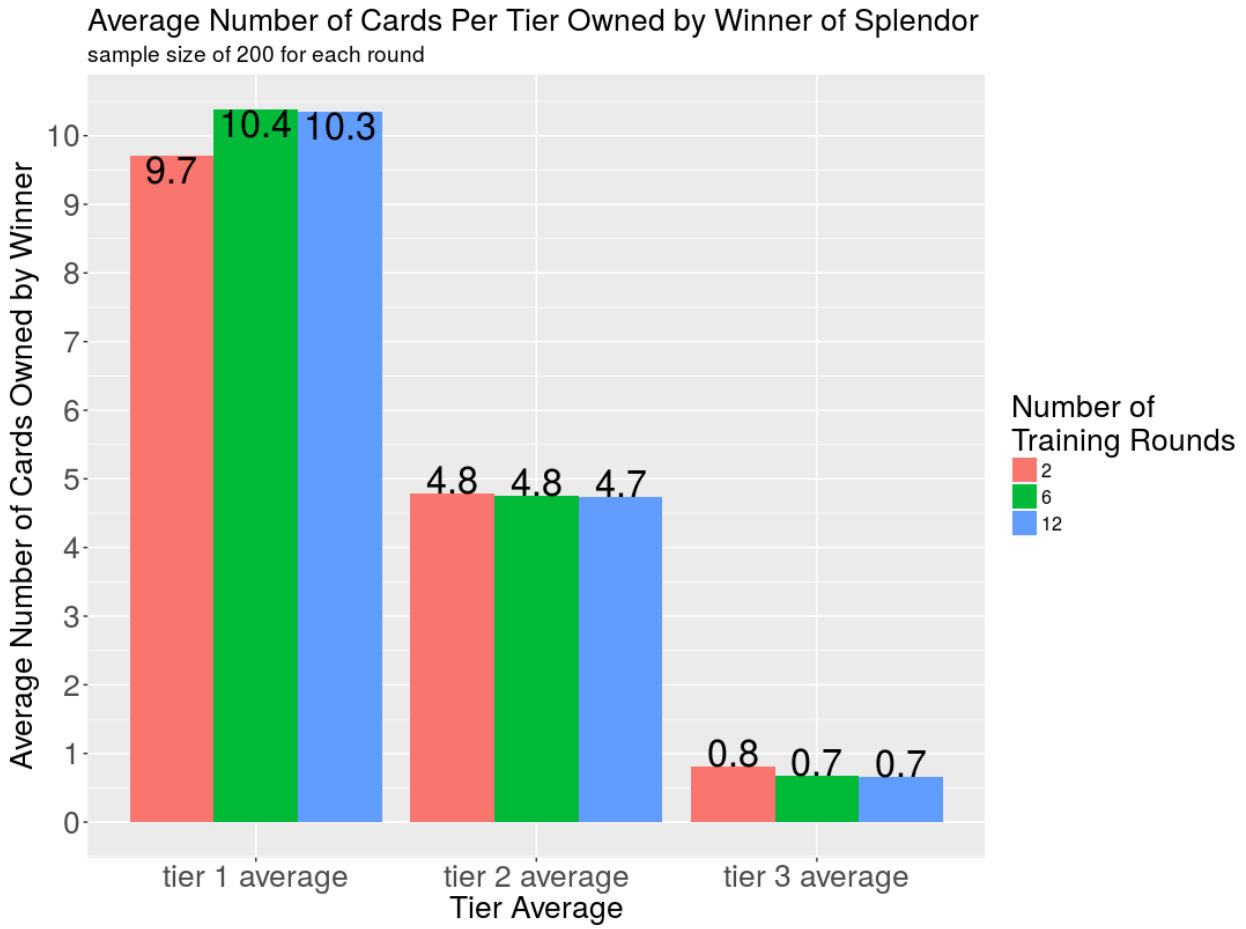
Number of turns taken is slightly improved, but not huge

Differences in Winning Strategies (2)



Getting a second objective is a high priority for better-trained networks

Differences in Winning Strategies (3)



Better-trained networks have a slight preference for favoring an additional tier 1 card over a tier 3 card

Obstacles and Possible Solutions

- Training is slow
  - Computer memory (RAM) needed for faster training
    - Larger batch size
    - Paralell computing
  - More time needed
- AI improvements taper off after many rounds of training
  - Short-term metrics to compare a player to other players possibly useful for training in a more adversarial way
  - Refactor code, allow more flexibility in how models can work

Code and Personal Materials

- GitHub Repositories:
  - https://github.com/mcandocia/machi_ai
  - https://github.com/mcandocia/splendor_ai
- My blog articles:
  - http://maxcandocia.com/article/2016/Apr/06/how-computers-recognize-images/
  - http://maxcandocia.com/article/2017/Jul/22/using-neural-networks-to-play-board-games/
  - http://maxcandocia.com/article/2017/Jul/30/using-ai-for-machi-koro-strategy

Other Resources

- Learn Python: http://www.diveintopython3.net/
  - Learn numpy: https://hackernoon.com/introduction-to-numpy-1-an-absolute-beginners-guide-to-machine-learning-and-data-science-5d87f13f0d51
  - Learn Keras: https://elitedatascience.com/keras-tutorial-deep-learning-in-python