STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

Mastering the game of Go with deep neural networks and tree search

Sesha Vadlamudi
10458123
svadlam1@stevens.edu
https://github.com/DeepikaVadlamudi

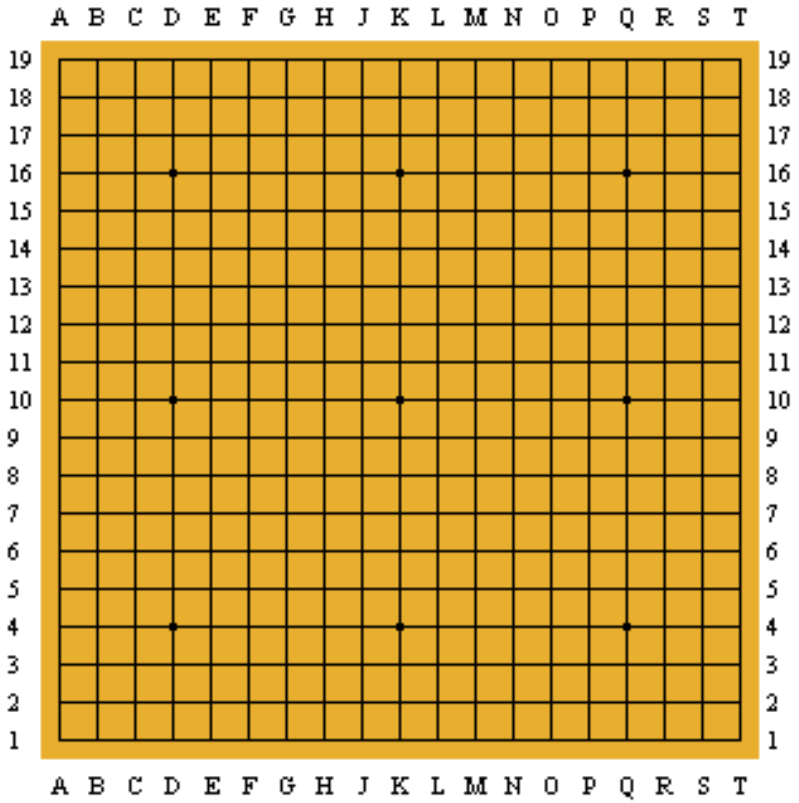
Agenda

- Game of Go
- Training
  - Supervised learning of Policy networks
  - Reinforcement learning of Policy networks
  - Reinforcement learning of Value networks
- Playing the game of GO using AlphaGo
  - Overview of Simulation
  - Monte Carlo Tree Search
  - Decision Making after MCTS
- AlphaGo Zero
- Improvements

This paper provides an efficient method that combines the policy and value networks with Monte Carlo Tree Search (MCTS).

We shall see how in this short and concise presentation.

Game of GO



- The standard GO board has 19x19 grid of lines, containing 361 points.
- State : Arrangement of black, white, and space.
  - AlphaGo uses a 19x19x48 tensor to store other information
- Action : place a stone on a vacant point.
  - Action space : $A \subset \{1, 2, 3, \dots, 361\}$
- Go is very complex.
  - Number of possible sequence of actions is 10^{170}

That is the number of legal moves in the game of Go is way more than the number of atoms in the known universe which is 10^{80} .

Idea – Consolidated :

- Training alone is done in three stages.
  - Supervised learning (SL) policy network P_σ
  - Reinforcement learning (RL) policy network P_ρ
  - Reinforcement learning of value network V_θ

- Execution :
  - Simulation – MCTS + Value network
  - Decision Making

Training

Supervised learning of policy networks P_σ

- First stage of training pipeline
  - build on prior work
  - Classification
- Predicting expert moves
- Trained a 13 layer policy network – SL policy network – on 30M positions from over 160k games
- Input : s – a representation of the board state
- Output : probability distribution over all legal moves a
- Given raw board position, SL policy predicted moves with an accuracy of $\sim 56\%$

SL policy network - continued

- Stochastic gradient ascent is used to maximize the likelihood of the human move a selected in state s

$$\Delta\sigma \propto \frac{\partial log P_\sigma(a|s)}{\partial \sigma}$$

- A small improvement in accuracy led to large improvements in playing strength

Reinforcement learning of policy network P_ρ

- Second stage of training pipeline
  - aims at improving policy network
  - By policy gradient reinforcement learning
- Why?
  - What if the current state s_t has appeared while training?
  - The policy imitates expert action a_t which is a good move
  - But what if s_t has not appeared while training?
  - Then action a_t can be bad.
  - Since number of possible states is too big, there is huge probability that the s_t has not appeared in training

RL Policy Networks - continued

- RL policy P_ρ network is identical in structure to the SL policy network
- The weights ρ are initialized to the same values $\rho = \sigma$
- Games are played between the current policy network and randomly selected previous iteration of policy network
  - Randomizing so as to avoid overfitting
- Weights are updated at each time step t by stochastic gradient ascent so as to maximize expected outcome

$$\Delta\rho \propto \frac{\partial log P_\rho(a_t|s_t)}{\partial\rho} z_t$$

RL policy network - continued

- Reinforcement learning is guided by rewards
- Suppose a game ends at step T
- Rewards:
  - $r_1 = r_2 = r_3 = \cdots = r_{T-1} = 0$
  - $r_T = +1$ (winner)
  - $r_T = -1$ (loser)
- Return is defined by $z_t = \sum_{i=t}^{T} r_i$
- Winner's return: $z_1 = z_2 = z_3 = \cdots = z_T = +1$
- Loser's return: $z_1 = z_2 = z_3 = \cdots = z_T = -1$

RL Policy Networks - continued

- When played against SL policy network, RL policy network won more than 80% of the games
- Won 85% of games against Pachi, an open source Go program
- Only supervised learning program won 11% of games against Pachi

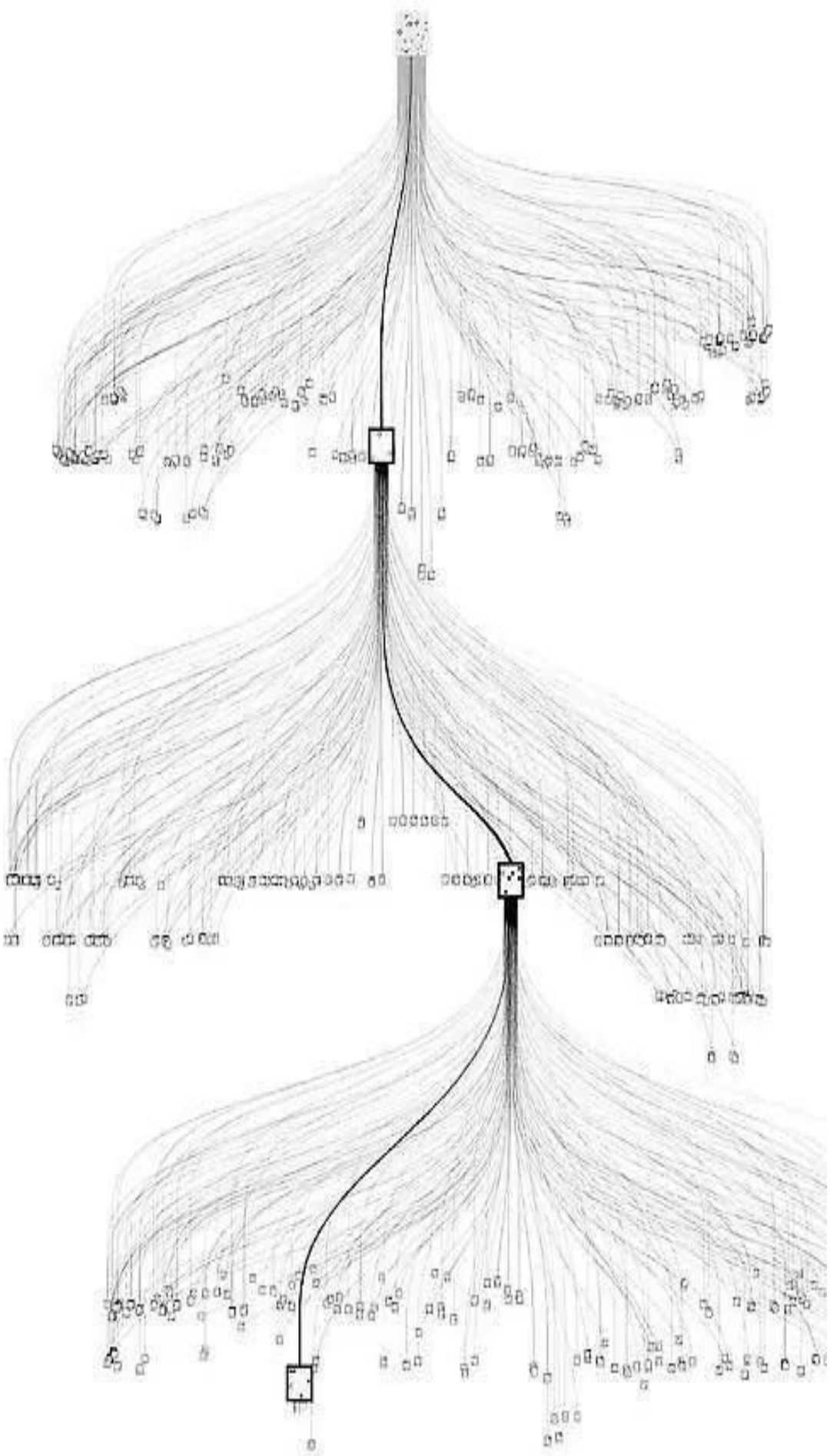
Reinforcement learning of value network V_θ

- Final stage of training pipeline
  - Position Evaluation
  - Regression
- Estimates a value function $v^p(s)$
- Predicts the outcome from position s of games played by using policy P for both players
- $\Delta\theta \propto \frac{\partial V_\theta(s)}{\partial\theta}(z - v_\theta(s))$

Execution

Overview of Simulation

- How do humans play?
  - We try to look ahead, estimate, the next 2-3 moves
  - Suppose we are in state s_t, we select action a_t
  - This leads to state s_t'
  - What will my opponents action be? His action leads to s_{t+1}
  - Upon observing s_{t+1} what will my action be? a_{t+1}
  - This leads to s_{t+1}'
  - ⋮
- If you can exhaustively foresee all the possible futures, you will win

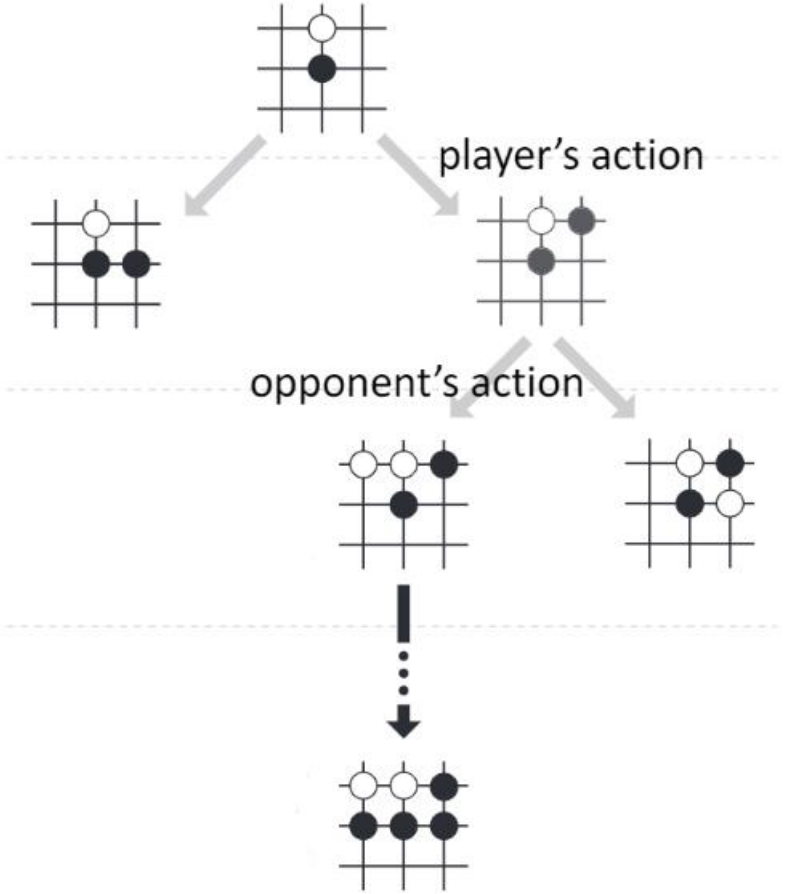


- Strange: I went forward in time… to view alternate futures. To see all the possible outcomes of the coming conflict.
- Quill: How many did you see?
- Strange: Fourteen million six hundred and five.

- Stark: How many did we win?
- Strange: … One.

Select actions by look-ahead search

- Main idea
  - Randomly select an action a
  - Look ahead and see whether a leads to win or lose.
  - Repeat this procedure many times.
  - Choose the action a that has the highest score.



Monte Carlo Tree Search (MCTS)

- Every simulation of MCTS has 4 steps:
  1. Selection: The player makes an action a. (Imaginary, Not actual move)
  2. Expansion: The opponent makes an action; the state updates. (Also imaginary; made by the policy network)
  3. Evaluation: Evaluate the state-value and get score v. Play the game to the end to receive the reward r. Assign score $\frac{v+r}{2}$ to action a.
  4. Backup: Use the score $\frac{v+r}{2}$ to update action-values.

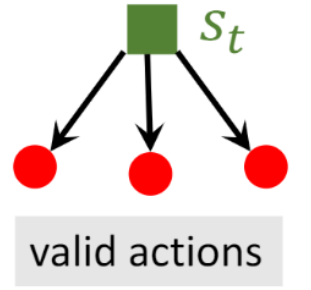
MCTS - Details

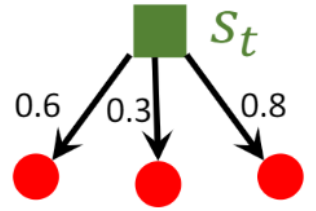
Step 1: Selection

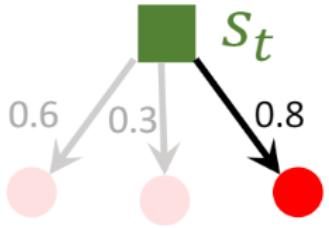
- Observing s_t, which action shall we explore?
- First, for all the valid actions a, calculate the score:

$$a_t = \underset{a}{\operatorname{argmax}}\, Q(s_t, a) + \eta \frac{P(s_t, a)}{1 + N(s_t, a)}$$

- $Q(s_t, a)$ - Action-value computed by MCTS and value network.
- $P(s_t, a)$ - The learned policy network.
- $N(s_t, a)$ – Given s_t, how many times we have selected a so far.
- Second, the action with the largest a_t is selected.





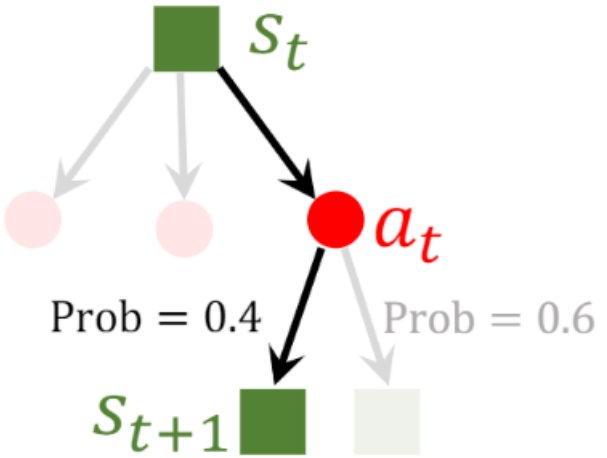


Step 2: Expansion

- What will be the opponent's action?
- Given a_t, the opponents action will be a'_t will lead to new state s_{t+1} .
- The opponent's action is randomly sampled from

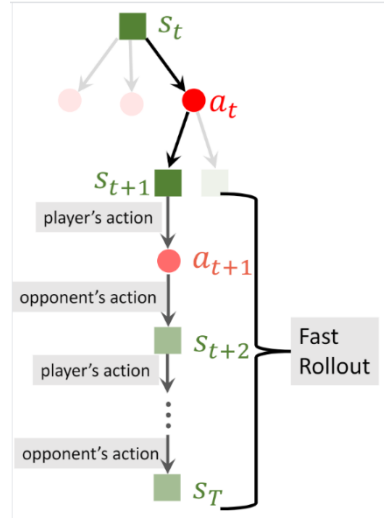
$$a'_t \sim P(\cdot\,|s'_t\,;\theta).$$

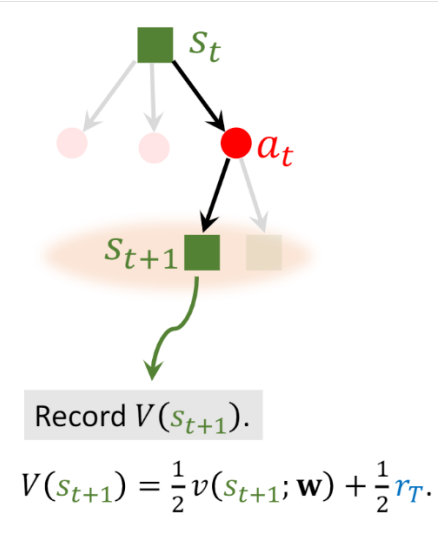
- Here, s'_t is the state observed by the opponent.



Step 3: Evaluation

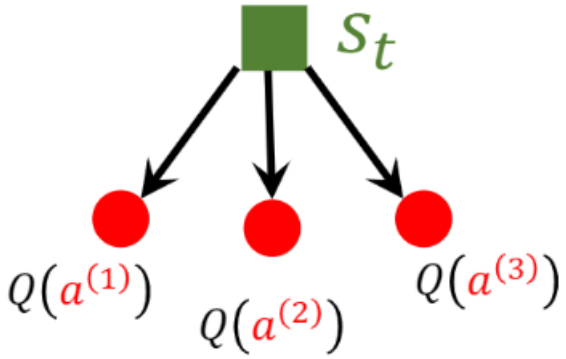
- Run a rollout to the end of the game (step T).
- Player's action: $a_k \sim P(\cdot\,|s_k;\theta)$.
- Opponent's action: $a'_k \sim P(\cdot\,|s'_k;\theta)$.
- Receive reward r_T at the end.
  - Win : $r_T = +1$
  - Lose : $r_T = -1$
- Evaluate the state s_{t+1}.
- $v(s_{t+1};w)$: output of the value network.

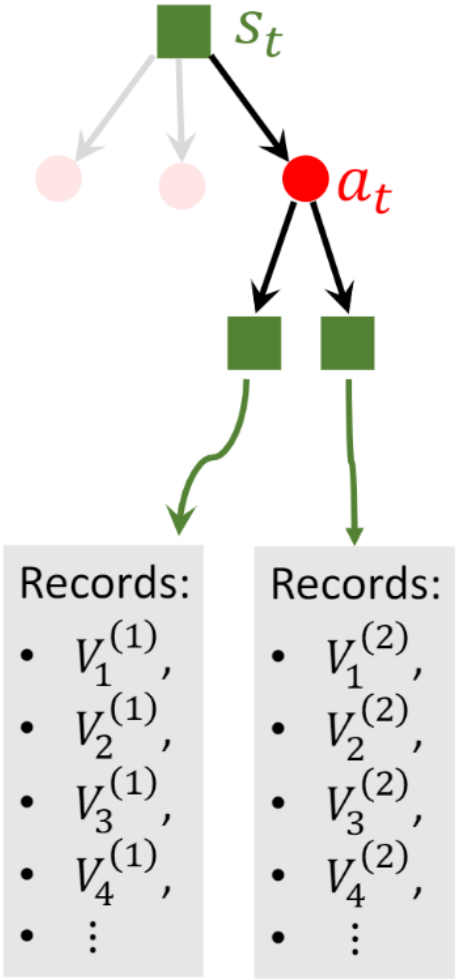




Step 4: Backup

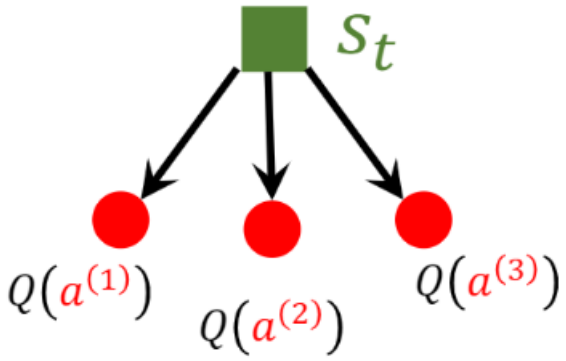
- MCTS repeats such simulation many times.
- Each child of a_t has multiple recorded $V(s_{t+1})$.
- Update action-value:
  - $Q(s,a) = mean(the\ recorded\ V's)$
- These Q values will be used in Step 1 (selection).





Revisit Step 1 selection

- First, for all valid actions a, calculate the score:
  - $a_t = \underset{a}{\text{argmax}}\, Q(s_t, a) + \eta \frac{P(s_t,a)}{1+N(s_t,a)}$
- Second, the action with the largest a_t is selected.



Decision making after MCTS

- $N(s, a)$: How many times a has been selected so far.
- After MCTS, the player makes the actual decision:

$$a_t = \underset{a}{\operatorname{argmax}}\, Q(s_t, a) + \eta \frac{P(s_t, a)}{1 + N(s_t, a)}$$

- To perform the next action, AlphaGo does MCTS all over again.
- Initialize $Q(s_t, a)$ and $N(s_t, a)$ to zeros.

AlphaGo Zero

- AlphaGo Zero is stronger than AlphaGo. (100-0)
- It was released in 2017
- Differences:
  - AlphaGo Zero does not use expert human experience.
    - For the Go game human experience is harmful.
  - MCTS is used to train the policy network.

Improvements:

- **Make the value network adaptive.**
  - As of now Value Network is being trained by regression
  - This may be modified so as to reduce the training time plus to increase accuracy
    - Initially train by regression
    - Afterwards update the weights of the value network
      - Whenever a game is played
      - Use Gradient ascent to maximize the likelihood of the outcome of the game
- Increase the previous history by augmenting more number of planes to the state
  - This may improve the accuracy though the computation cost may go up

STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

Acknowledgements

Followed Shusen Wang's work on GitHub and a good amount of content is borrowed from his lecture notes.

STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

References

- *AlphaGO – Silver and others – Mastering the game of GO with deep neural networks and tree search, Nature 2016*
- *https://github.com/wangshusen/DeepLearning/blob/master/Slides/13_RL_5.pdf*
- *https://nikcheerla.github.io/deeplearningschool/2018/01/01/AlphaZero-Explained/*

Questions?

stevens.edu

Sesha Phani Deepika Vadlamudi
svadlam1@stevens.edu