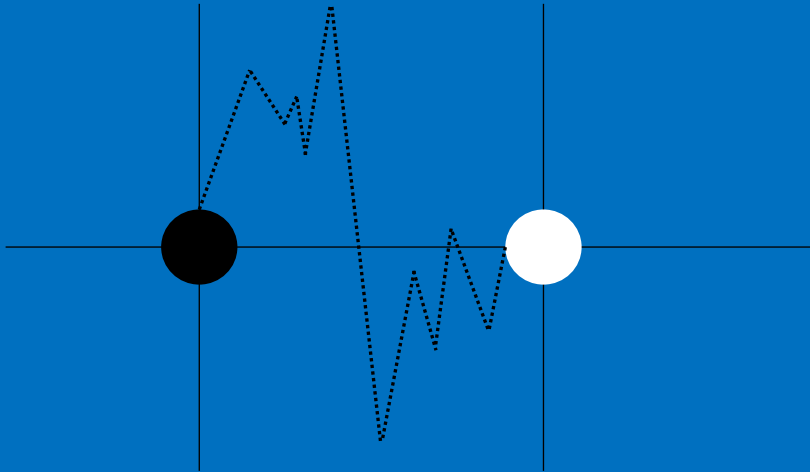


# From AlphaGo to AlphaZero

1. Mastering the Game of Go with Deep Neural Networks and Tree Search (2016)
2. Mastering the Game of Go without Human Knowledge (2017)
3. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm (2017)



- By Shijie Huang (Harvey)
- 11 December 2017

# Content

---

- Deep Reinforcement Learning in Different Games
- Go Game: Basic Rules
- AlphaGo
  - Why Go and Prior Work
  - Structure Breakdown
  - Supervised Learning of Policy Networks
  - Policy and Policy Networks
  - Reinforcement Learning: Policy Networks
  - Reinforcement Learning: Value Networks and Position Evaluation
  - Monte Carlo Tree Search: Search with Policy and Value Networks
- AlphaGo Zero
  - Deep Neural Network – Structure Breakdown
  - Monte Carlo Tree Search: Search with a Single Neural Network
- Humans, AlphaGo and AlphaGo Zero
- Discussions
- AlphaZero

# 1. Deep Reinforcement Learning in Different Games

Assumption: AI can only see/do what humans can see/do

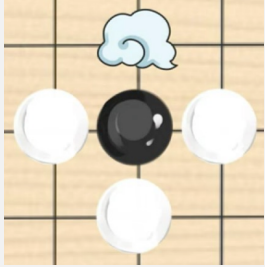


Games	Atari	Board Game	Dota / LOL	Starcraft II
Agents	1	2	$\geq 1$ (Generally 5 vs. 5)	$\geq 2$ (Generally 1 vs. 1)
Information	Perfect information	Perfect Information	Imperfect information (fog of war)	Imperfect information (fog of war)
State space	Limited	Limited but large	Unlimited	Unlimited
Action space	Limited	Limited but large	Limited but large	Unlimited
AI vs. Human	AI dominated	AI dominated	AI won in 1 vs. 1 Completely failed in 5 vs. 5 (OpenAI)	Completely failed in any competitive game but learn to do simple tasks (DeepMind)

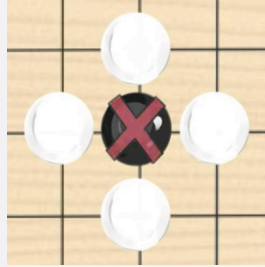
# 1. Go Game: Basic Rules

## How to play

### Liberty

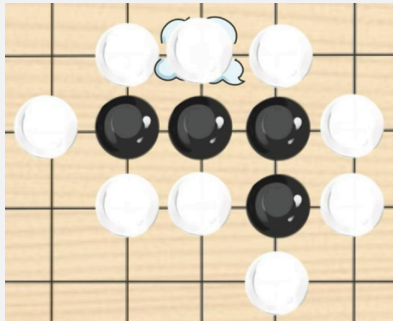


- Liberty is when the area surrounding the go has no other go



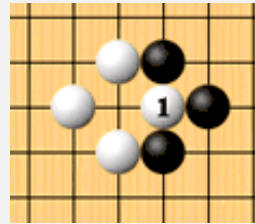
- If a go has no liberty, it is considered "dead" and will be taken out

### Liberty (Multiple Go)



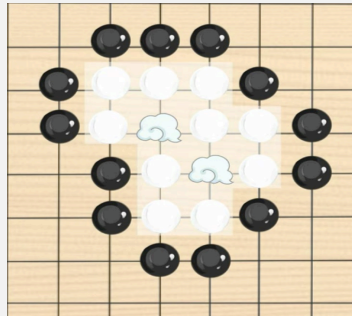
- In this case, black gos are "dead" and will be taken out

### Ko Rule



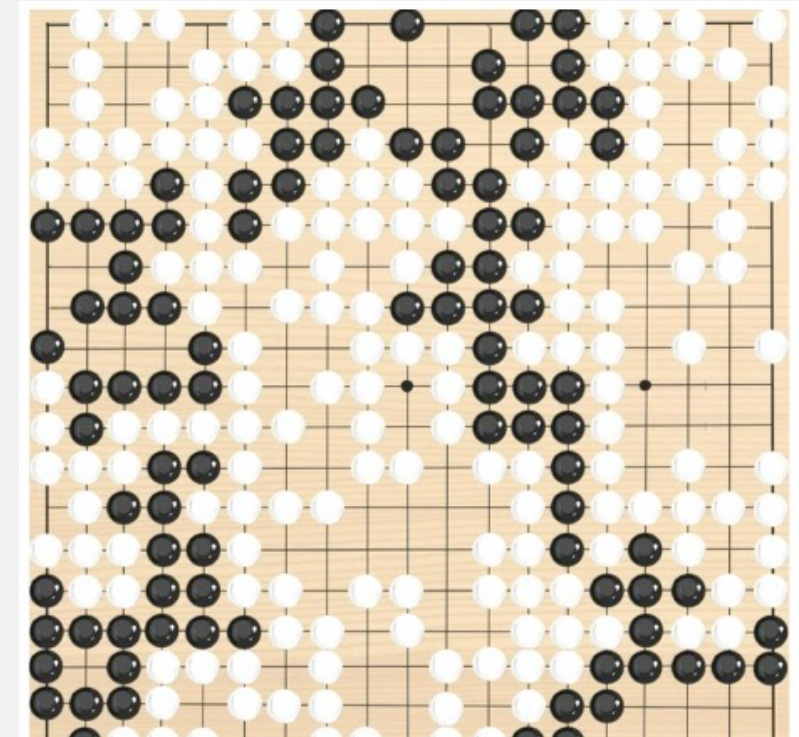
- One cannot place the go in "dead" area unless you can eat them
- Forbid sequential eat in the same position
- Unless you make a move that significantly change the board position

### Life



- If go is surrounded and the area has more than two liberties (i.e. they cannot be taken out with one step), then this area of go is considered to be "alive" (life)
- In the case shown in the left, black gos cannot be placed inside the white gos. Hence, white gos are considered to be "alive"

## How to win



The winner is:

1. whoever occupies more areas of the board
2. whoever has more number of go on the board
3. rules are country dependent but only minor differences exist

# AlphaGo

An “interesting” fact about AlphaGo:

The most powerful version consists of 1920 CPUs and 280 GPUs

Electricity bill is \$3000 per game, training phase: 160,000 game  $\approx$  \$480 million



## 2. Why Go and Prior Work

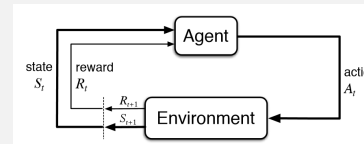
### Why Go Game?



- Most challenging of classic games in computer science
- 19 X 19 board positions
- $250^{150}$  possible sequences of moves (chess:  $35^{80}$ )
- The idea of broad picture: the objective is to occupy more territory.
  - Leads to highly sophisticated evaluation functions
  - No one before AlphaGo has successfully build an effective evaluation function
- Exhaustive search is infeasible.
- Still within the scope of perfect information

### Prior Work

#### Reinforcement Learning



#### Self-play



#### Linear Value Function

$$V_{\theta}(s) = \phi(s)^T \theta$$

#### Tree Search

##### Minimax (Alpha-Beta) Tree Search

- Most game too large for Minimax Tree Search because it requires one to go all the way to the end of the game
- Truncate the tree by using approximated value function  $V_{\theta}(s) \approx v^*(s)$
- Super-human performance in chess
- Not effective in Go

##### Monte Carlo Tree Search

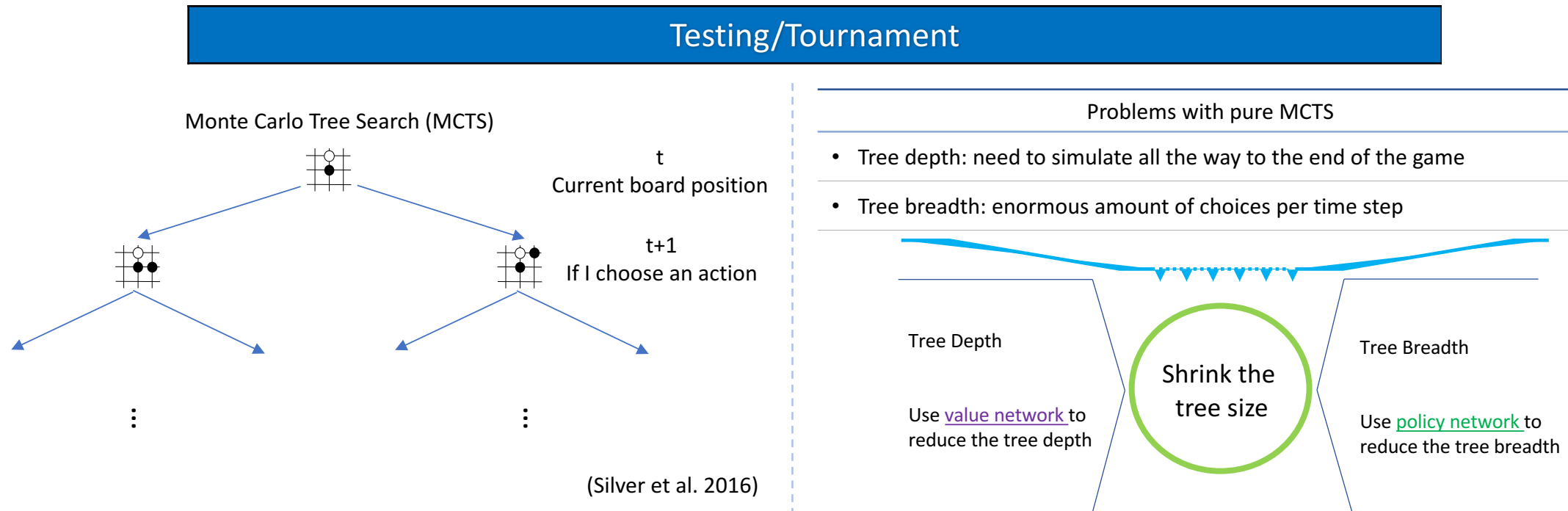
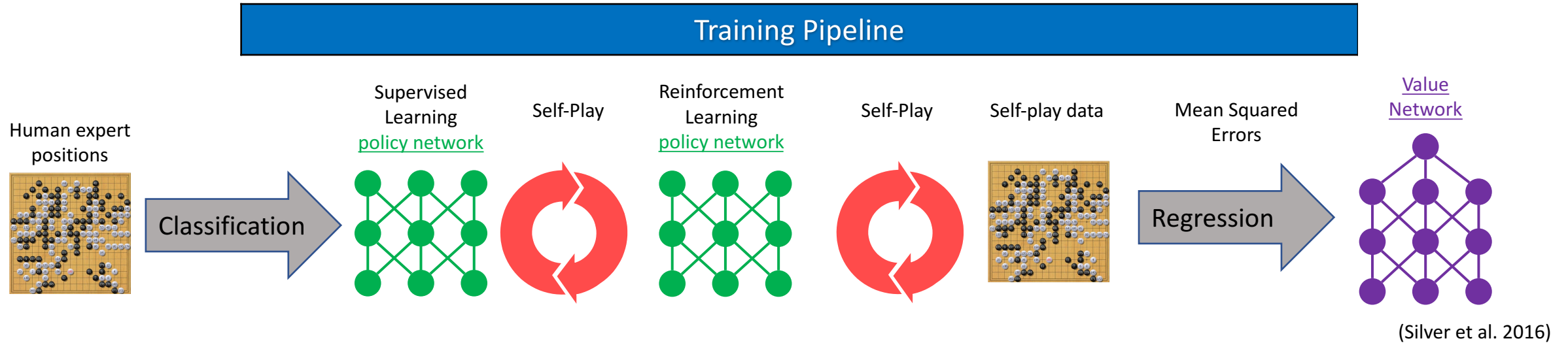
- Double approximation
- $V^n(s) \approx v^{p^n}(s) \approx v^*(s)$
- First use Monte Carlo simulations to estimate the value function of a policy  $p^n$
- Second use the value function to approximate the optimal value function
- Why does it work? In the limit they are equivalent

Why?

Reinforcement Learning: Neurobiology foundations

Tree Search: Players tend to make forecasts (truncated) and predictions

## 2. AlphaGo: Structure Breakdown



## 2. AlphaGo: Supervised Learning of Policy Networks

### Convolutional Neural Network

#### CNN Training and Testing

- To predict human experts move
- 160,000 games (professional players)
- 29.4 Million moves (28.4 million training and 1 million testing)
- Stochastic Gradient Descent update to maximize the action log likelihood

Prediction accuracy: 55-57%

↓

If I feed the neural network with a board status (state), the neural network can predict, on average 55-57% chance, the correct move that a human expert will do under the same board condition

**Better accuracy or higher chances of winning?**  
**After all, human (including experts) make mistakes/their strategy may not be optimal strategy**

↓

**Supervised learning is not enough**

(Smiling) Puppy picture

Pixels



Human expert positions



48 feature planes  
(each 19 X 19)  
19 X 19 X 48

13 layers



+

1 fully connected layer



+

Softmax



↓

In the picture, there is a

- (Smiling) Puppy (80%)
- (Angry) puppy (60%)
  - Kittie (10%)
  - Baby (1%)
  - ...

↓

Human experts will take an move in

- Position 1 (56%)
- Position 2 (30%)
- Position 3 (20%)
- ...



# 2.1 AlphaGo: Policy and Policy Network


Policy Based Learning

$P(a|s)$  **Formally, a policy maps states to actions**

- In policy gradient framework, it is a probability distribution over all states and actions (Contrast with the value framework)
- Policy will tell the probability of I choose a particular action under current state
- No value function, states and actions can be continuous and infinite
- Often parameterized, a function  $P_{\theta}(a|s)$
- Training becomes an optimization problem

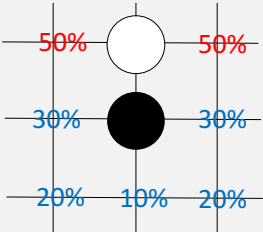
Policy Network in AlphaGo

Take board position **s** as input



$P_{\theta}(a|s)$

Action probability distribution/matrix



50%	n/a	50%
30%	n/a	30%
20%	10%	20%

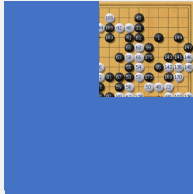

**Training:**  
Stochastic gradient ascent to maximize the log likelihood

Rollout Policy vs. Supervised Learning Policy

Trade off between speed and accuracy

Choose an action	Rollout Policy	SL Policy
Speed	2 $\mu$ s	3ms
Accuracy	24.4%	57%

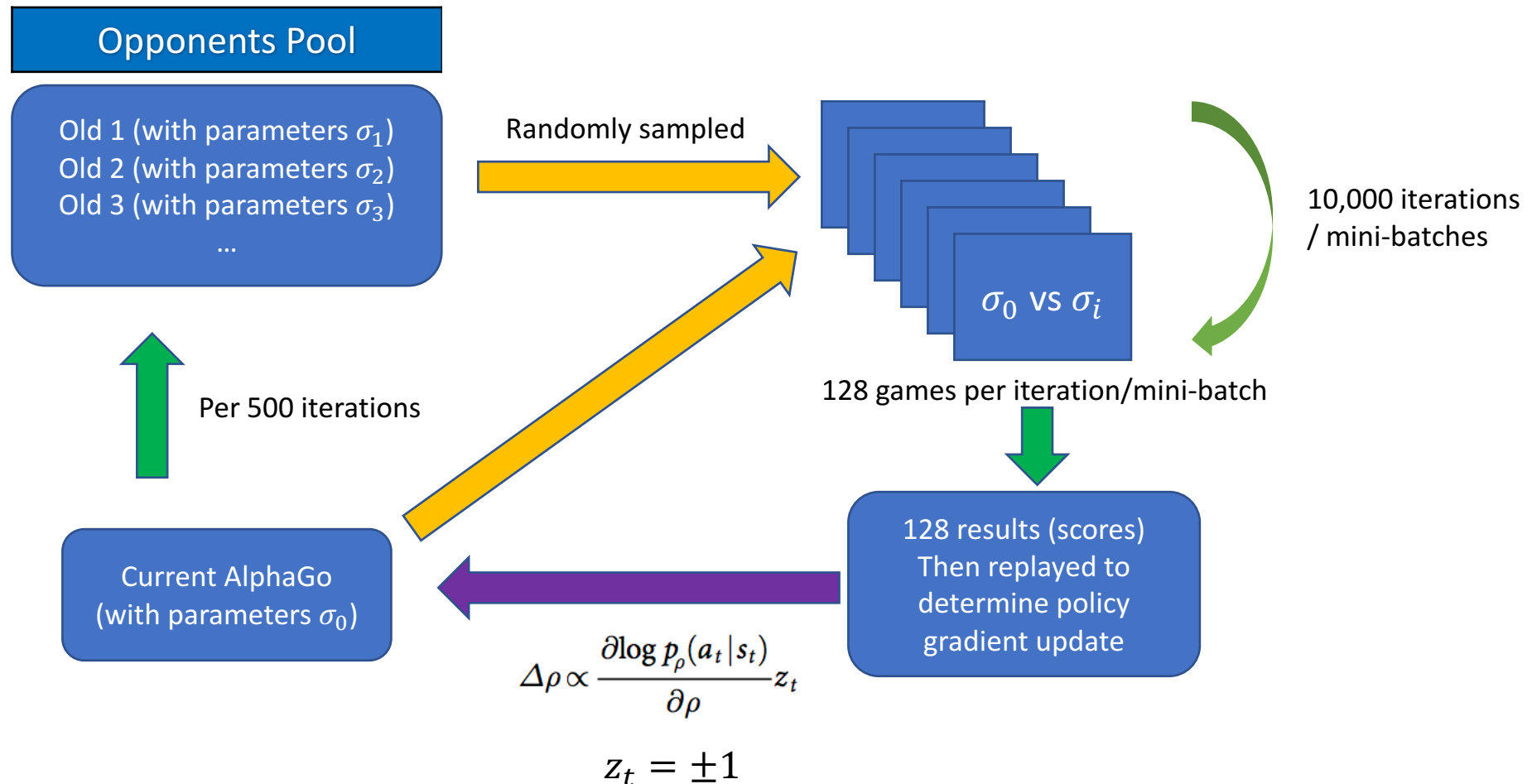
The key differences are

Structure	Rollout Policy	SL Policy
Softmax function	Linear	Non-linear
Feature fed into CNN	<div>Small feature size (A simple demo)</div> 	<div>Full size (19 X 19 X 48)</div> 
Other techniques	<u>Last good reply heuristic</u>	
Notation	$P_{\pi}(a s)$	$P_{\sigma}(a s)$

## 2.2 Reinforcement Learning: Policy Network Iterations

The aim here is to allow the policy to be improved by letting the AI play against itself

$$P_{\sigma}(a|s) \Rightarrow P_{\rho}(a|s), \sigma \text{ and } \rho \text{ are weight parameters}$$



## 2.3 Reinforcement Learning: Value Network and Position Evaluation

Now we have a better policy (strategy), we want to predict the outcome of game when we make a move  
i.e. the probability of me winning this game

### Value Function

$$v^p(s) = \mathbb{E}[z_t | s_t = s, a_{t...T} \sim p]$$

- **Interpretation:** the value (chances of win) of the current state after I make a move under the policy  $p$ .
- **Problem:** we do not know the **optimal value function** under perfect play  $v^*(s)$
- **Solution:** use approximation
  - $v_\theta(s) \approx v^{p_\rho}(s) \approx v^*(s)$

### Training

- Parameterized value function  $v_\theta(s)$  with weights  $\theta$
- We know the game results for a given state (board position) under the strongest policy  $p_\rho$ , i.e.  $v_\theta(s)$ .
- **Optimization problem:** optimize  $\theta$  so that  $v_\theta(s)$  is close to  $v^{p_\rho}(s)$
- **Approach:** standard regression (projection) method
- Minimize Mean Squared Errors:
  - $(v_\theta(s) - v^{p_\rho}(s))^2$  or  $(v_\theta(s) - z^k)^2$  where  $z^k = \pm 1$

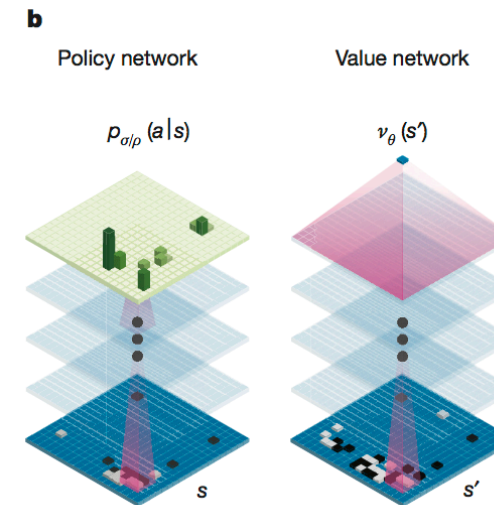
### Overfitting

#### Naïve approach leads to overfitting

- **Training:** Mean Squared Errors of 0.19
- **Testing:** Mean Squared Errors of 0.37

#### Solution and approach

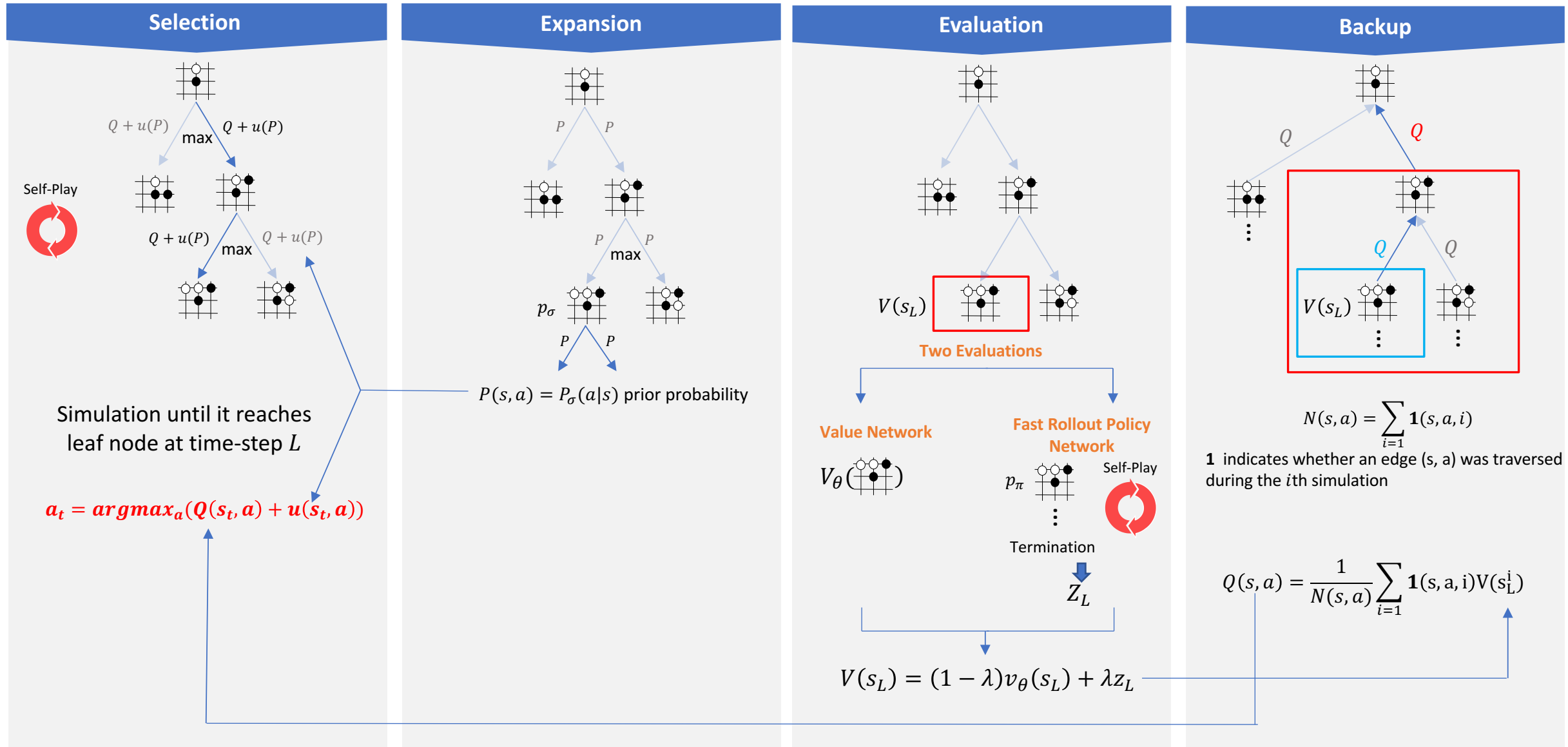
- A new self-play dataset consisting of 30 million distinct positions, each sampled from a separate game
- Each game between the RL policy network and itself
- MSEs: 0.226 (Training) and 0.234 (Testing)



(Silver et al. 2016)

# 2.4 Monte Carlo Tree Search: Search with Policy and Value Networks

## Asynchronous Policy and Value Monte Carlo Tree Search (APV-MCTS)



# AlphaGo Zero

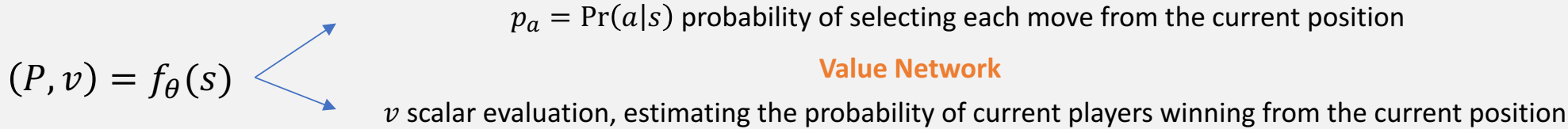
Simple Neural Network Structure  
Completely tabula Rasa

An “interesting” fact about AlphaGo Zero:  
The system consists of only 4 TPUs,  
each of which is 15-30 times more efficient than GPUs in performance per-watt  
And 64 GPUs, 19 CPUs.

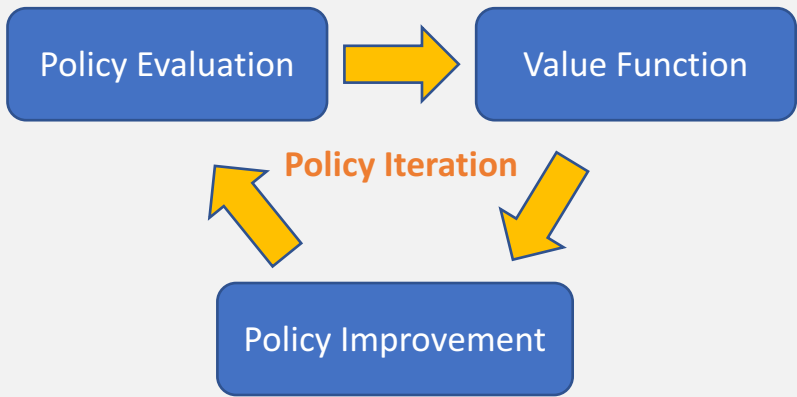


# 3.1 Deep Neural Network- Structure Breakdown

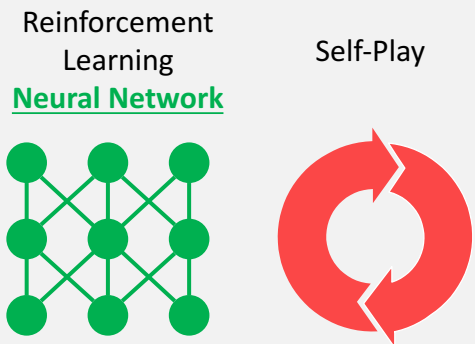
## Single Neural Network Architecture



## Core RL Concepts

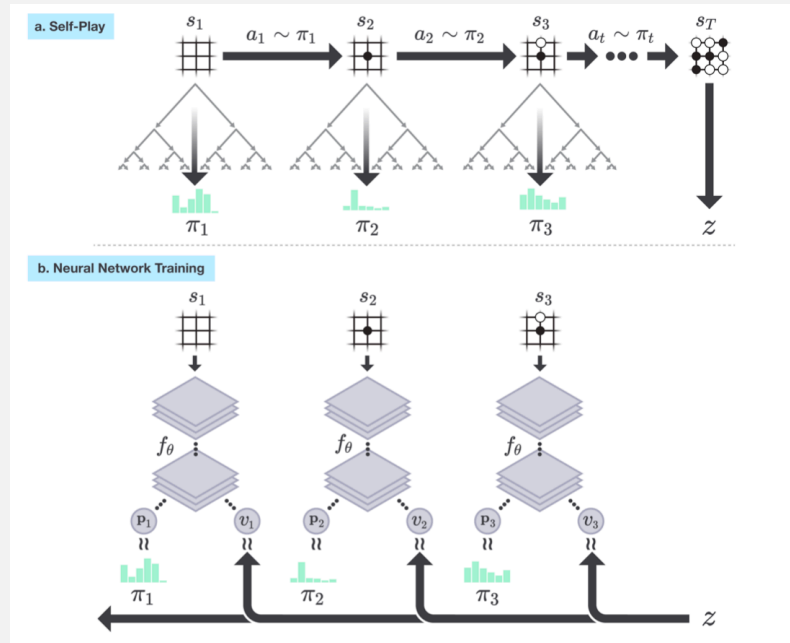


## Training Pipeline



## Policy Evaluation

$f_{\theta}(s) \Rightarrow \text{MCTS} \Rightarrow \pi_t$ , a move is selected according to policy  $\pi_t$   
Terminal state  $s_T$  will give us a game winner  $Z$

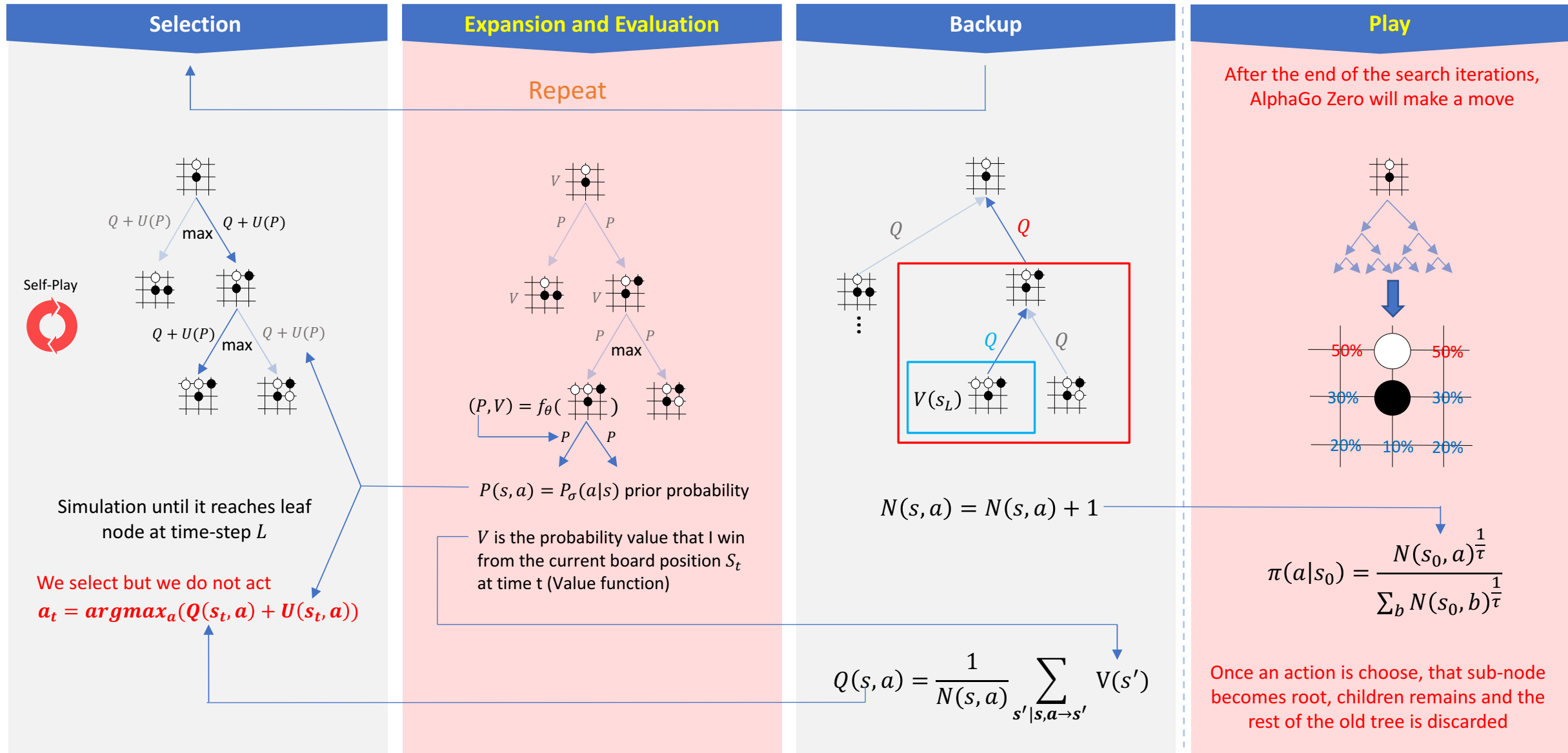


## Policy Improvement

At time  $t$ , feed the board position  $s_t$  into policy  $f_{\theta}(s) = f_{\theta}(s_t)$ , which gives us  $p_t$  and  $v_t$ .  
Optimization problem: optimize weight parameters  $\theta$  to minimize prediction errors between  $p_t - \pi_t$  and  $v_t - Z_t$   
**Minimize Loss Function:**  $l = (z - v)^2 - \pi^T \log p + c||\theta||^2$

# 3.2 Monte Carlo Tree Search: Search with a Single Neural Network

## Asynchronous Policy and Value Monte Carlo Tree Search (APV-MCTS)



# 4. Humans, AlphaGo and AlphaGo Zero

## Humans

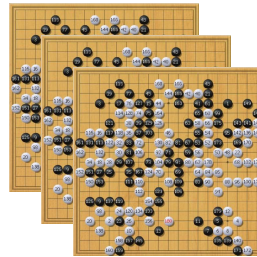
They develop new policies only when they become masters in Go

**Ancient Master / Experts**

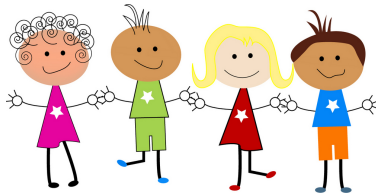


**Dozens of manuals/books**

Heritage from ancient knowledge in thousands of years



**Kids**



The problem is that there is **limited evaluation process**  
children will not (often are not allowed) to question ancient knowledge  
(mostly because of the Asian culture)

## AlphaGo

**Supervised Learning**



**Self-play**



**Policy Evaluation**

Humans are right

Humans are wrong

**Policy Improvement**

## AlphaGo Zero

1. Only basic rules
2. Start from completely random play

**Self-play**



**Policy Evaluation**

I am right

I am wrong

**Policy Improvement**

**Self-developed manuals**



- Concepts of shapes, territory, influence
- Joseki
- Fuseki
- Tesuji
- Sente
- Shicho

# 5. Discussions

## Human vs. Artificial Intelligence

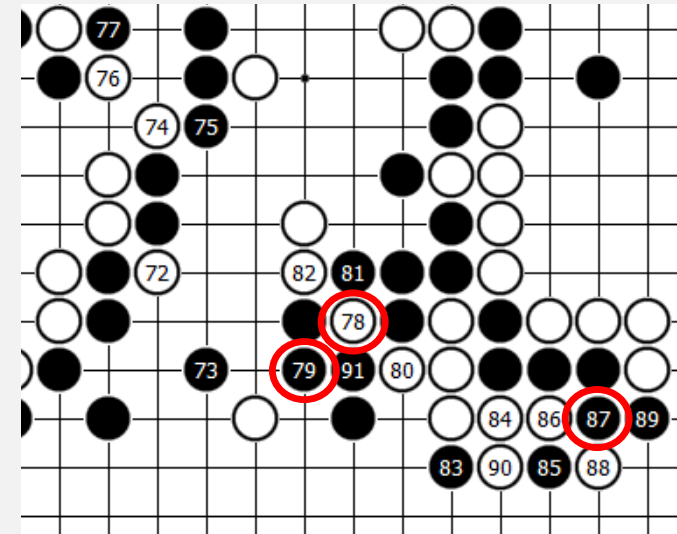
### Humans in Go game

- Humans are often wrong
- Master manuals are more like guidance rather than mandatory knowledge
- Self-evaluation and improvement are key to learning
- The concept of tabula rasa is crucial in learning theories

### AI in Go game

- Algorithms are **much more important** than data
- People tend to assume that machine learning is all about big data and massive computations, THAT IS WRONG!
- AlphaGo Zero does not use any human data whatsoever, it always has the opponent at just right level (self-play), and improves itself from self-learning
- Yet it performs much better, and significantly less computational requirement

### The only game that AlphaGo resigned



- AlphaGo (Black) calculates that Lee Sedol has 0.01% chances to make move 78
- Policy informed AlphaGo that move 79 had a winning probability of 70%
- It realised that the winning probability was actually only 55% after move 87
- Often considered as a “bug” in MCTS

# 6. AlphaZero: General Version

## Generalize AlphaGo Zero in board games

### AlphaGo Zero vs. AlphaZero

	AlphaGo Zero	AlphaZero
Value Function	Binary win/loss Probability of Winning	Expected outcome (Chess has win, loss, draw)
Rotation and Reflection	Invariance (faster training)	Variance (Asymmetric rules)
Policy Improvement	Current best player vs. New player after each iteration If win by 55% then update the policy	Updated continually, even during the iterations

### Neural network architecture in different games

	Chess	Shogi	Go
Input Feature	119	362	17
Policy Plane	8 X 8 X 73	9 X 9 X 139	19 X 19 + 1
Training Time	9h	12h	34h
Training Games	44 million	24 million	21 million