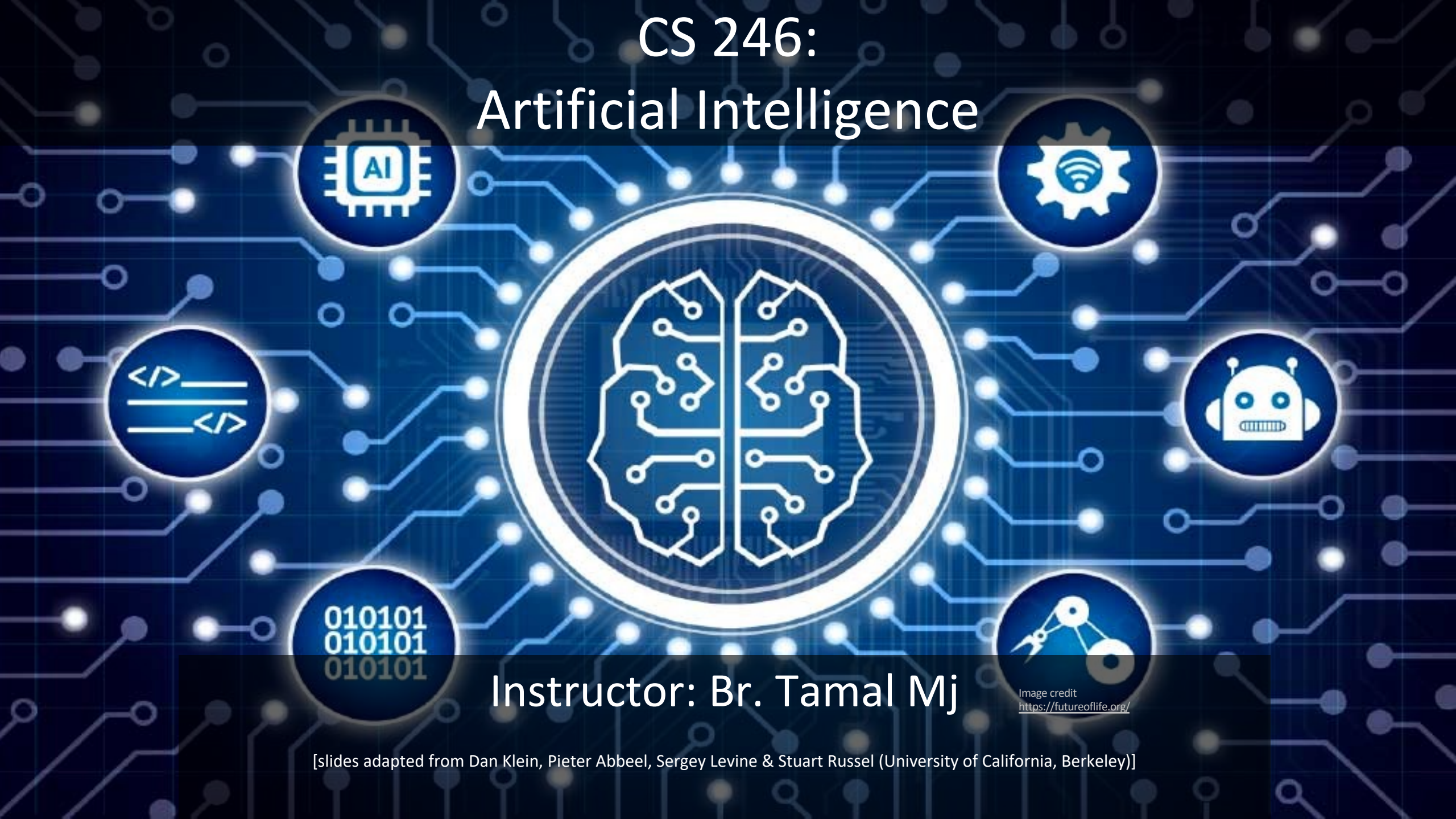
CS 246:
Artificial Intelligence



Instructor: Br. Tamal Mj

Image credit
https://futureoflife.org/

[slides adapted from Dan Klein, Pieter Abbeel, Sergey Levine & Stuart Russel (University of California, Berkeley)]

Om Saha Naav[au]-Avatu
Saha Nau Bhunaktu
Saha Viiryam Karavaavahai
Tejasvi Naav[au]-Adhiitam-
Astu Maa Vidvissaavahai
Om Shaantih Shaantih
Shaantih

Om, May we all be protected
May we all be nourished
May we work together with great energy
May our intelect be sharpened (may our study be effective)
Let there be no Animosity amongst us
Om, peace (in me), peace (in nature), peace (in divine forces)

Alpha Go

- The game of Go has long been viewed as the most challenging of classic games for artificial intelligence
  - owing to its enormous search space and
  - the difficulty of evaluating board positions and moves.
- A new approach to computer Go that uses
  - ‘value networks’ to evaluate board positions and
  - ‘policy networks’ to select moves.

Paper: Mastering the game of Go with deep neural networks and tree search by David Silver et. al.

Main innovation and Key Features

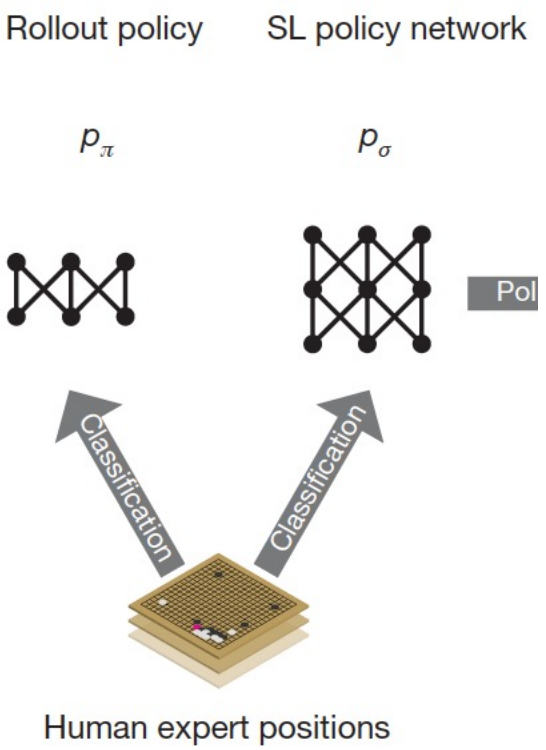
- The main innovation that made AlphaGo such a strong player is that
  - it selected moves by a novel version of MCTS that was guided by both a policy and a value function learned by reinforcement learning with function approximation provided by deep convolutional ANNs.
- Another key feature is that
  - instead of reinforcement learning starting from random network weights, it started from weights that were the result of previous supervised learning from a large collection of human expert moves.

AlphaGo's modification of basic MCTS

- Asynchronous policy and value MCTS (APV-MCTS)
  - It selected actions via basic MCTS but with some twists in how it extended its search tree and how it evaluated action edges.
  - Basic MCTS: expands its current search tree by using stored action values to select an unexplored edge from a leaf node
  - APV-MCTS: expanded its tree by choosing an edge according to probabilities supplied by SL-policy network
  - SL-policy network: a 13-layer deep convolutional ANN trained by supervised learning to predict moves contained in a database of nearly 30 million human expert moves.

- Basic MCTS: evaluates the newly-added state node solely by the return of a rollout initiated from it
- APV-MCTS: evaluates the node in two ways:
  - by this return of the rollout,
  - Also by a value function, v_θ, learned previously by a RL method.
- If s was the newly-added node, its value became
  - $v(s) = (1 - \eta)v_\theta(s) + \eta G$
    - where G = the return of the rollout and
    - η = mixing parameter

- In AlphaGo, $v_\theta(s)$ values were supplied by the value network
  - Another 13-layer deep convolutional ANN
- APV-MCTS's rollouts in AlphaGo were simulated games using a fast rollout policy p_π provided by a simple linear network, also trained by supervised learning before play.

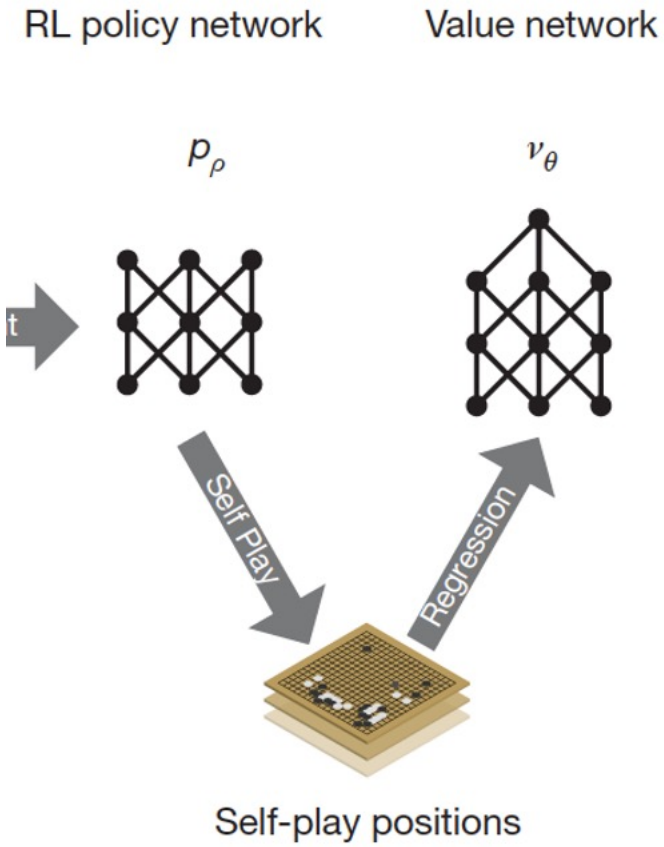


Move Played

- Throughout its execution, APV-MCTS kept track of how many simulations passed through each edge of the search tree, and when its execution completed, the most-visited edge from the root node was selected as the action to take
- This is the move AlphaGo actually made in a game.

Value Network

- The value network had the same structure as the deep convolutional SL policy network
  - except that it had a single output unit that gave estimated values of game positions instead of the SL policy network's probability distributions over legal actions



Two Stage Training

- In the first stage, they created the best policy they could by using reinforcement learning to train an RL policy network p_ρ.
  - This was a deep convolutional ANN with the same structure as the SL policy network p_π
  - p_ρ was initialized with the final weights of the SL policy network$p_\pi$that were learned via supervised learning,
  - Then policy-gradient reinforcement learning was used to improve upon the SL policy.
- In the second stage of training the value network, the team used Monte Carlo policy evaluation on data obtained from a large number of simulated self-play games with moves selected by the RL policy network.

- Value networks and Policy networks:
  - These deep neural networks are trained by a novel combination
    - supervised learning from human expert games, and
    - Reinforcement learning from games of self-play.

- Without any lookahead search, the neural networks play Go at the level of state-of-the-art Monte Carlo tree search programs that simulate thousands of random games of self-play

Contribution: A New Search Algorithm

- Combination of Monte Carlo simulation with value and policy networks
  - Using this search algorithm, AlphaGo achieved a 99.8% winning rate against other Go programs, and
  - Defeated the human European Go champion by 5 games to 0

Why the new search algorithm

- All games of perfect information have an optimal value function, $v^*(s)$,
  - determines the outcome of the game, from every board position or state s, under perfect play by all players.
- These games may be solved by recursively computing the optimal value function in a search tree containing approximately b^d possible sequences of moves, where
  - b is the game's breadth (number of legal moves per position) and
  - d is its depth (game length).
  - For chess: $b \approx 35$, $d \approx 80$ and
  - For Go: $b \approx 250$, $d \approx 150$
- In large games, exhaustive search is infeasible

Search Space Reduction

- The search space can be reduced by two general principles.
  - First, the depth of the search may be reduced by position evaluation:
    - truncating the search tree at state s and replacing the subtree below s by an approximate value function v(s) ≈ v*(s) that predicts the outcome from state s.
      - This approach has led to superhuman performance in chess, checkers and othello, but it was believed to be intractable in Go due to the complexity of the game.
  - Second, the breadth of the search may be reduced by sampling actions from a policy $p(a|s)$
    - a probability distribution over possible moves a in position s.

Sampling Actions from a Policy

- Example: Monte Carlo rollouts search to maximum depth without branching at all, by sampling long sequences of actions for both players from a policy p.

- Averaging over such rollouts can provide an effective position evaluation, achieving superhuman performance in backgammon and Scrabble, and weak amateur level play in Go.

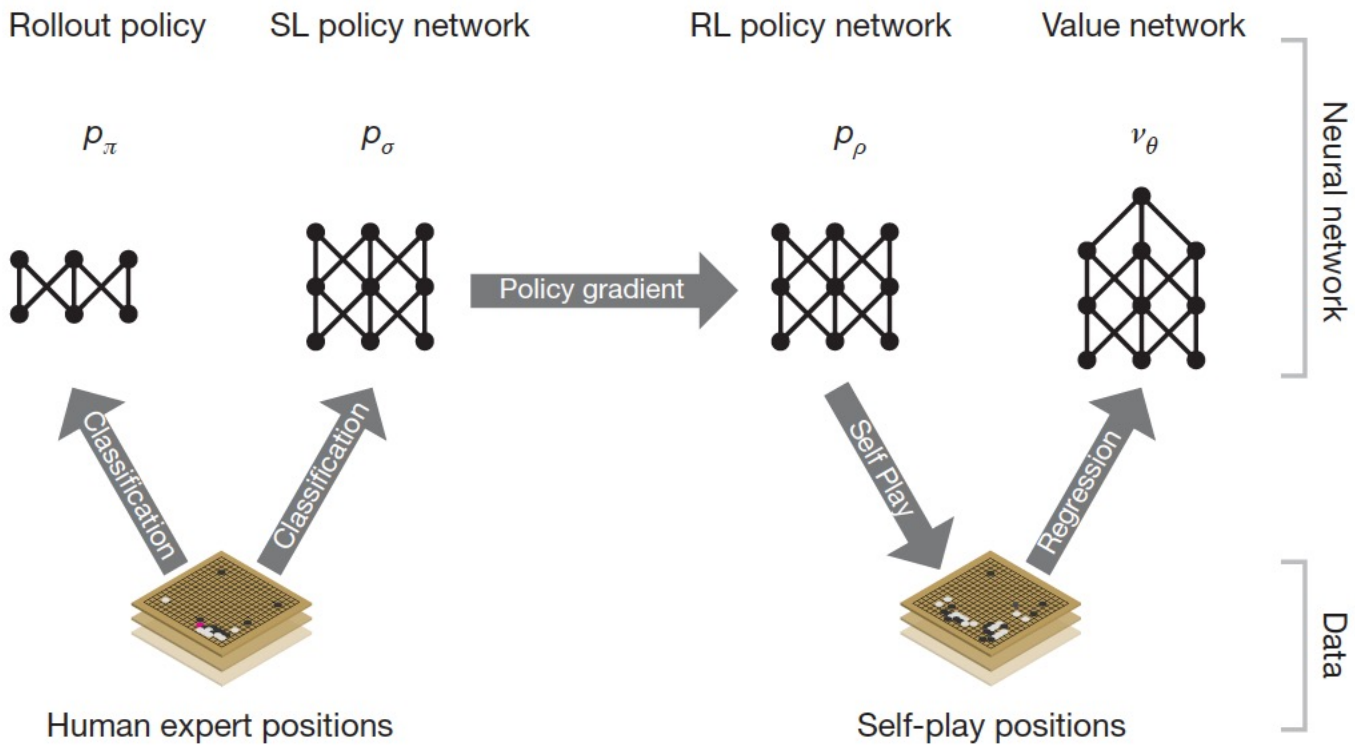
Monte Carlo tree search (MCTS)

- MCTS uses Monte Carlo rollouts to estimate the value of each state in a search tree.
- As more simulations are executed, the search tree grows larger and the relevant values become more accurate.
- The policy used to select actions during search is also improved over time, by selecting children with higher values
- Asymptotically, this policy converges to optimal play, and the evaluations converge to the optimal value function.
- The strongest Go programs (before AlphaGo was introduced) were based on MCTS

Deep Convolutional Neural Networks

- Work best with images:
  - image classification, face recognition, and playing Atari games
- Alpha Go: pass in the board position as a 19x19 image and use CNN construct a representation of the position.
- This helps to reduce the effective depth and breadth of the search tree:
  - evaluating positions using a value network, and
  - sampling actions using a policy network.

- The Neural Networks (NNs) were trained using a pipeline consisting of several stages of machine learning:



Models Trained in AlphaGo

- Supervised learning (SL) policy network p_σ
  - Trained directly from expert human moves.
- Fast Rollout policy p_π
  - can rapidly sample actions during rollouts.
- RL Policy network p_ρ
  - Improves the SL policy network by optimizing the final outcome of games of self play. This adjusts the policy towards the correct goal of winning games,
- Value network v_θ
  - rather than maximizing predictive accuracy. predicts the winner of games played by the RL policy network against itself.

AlphaGo efficiently combines the policy and value networks with MCTS.

Training Pipeline and Architecture (a)

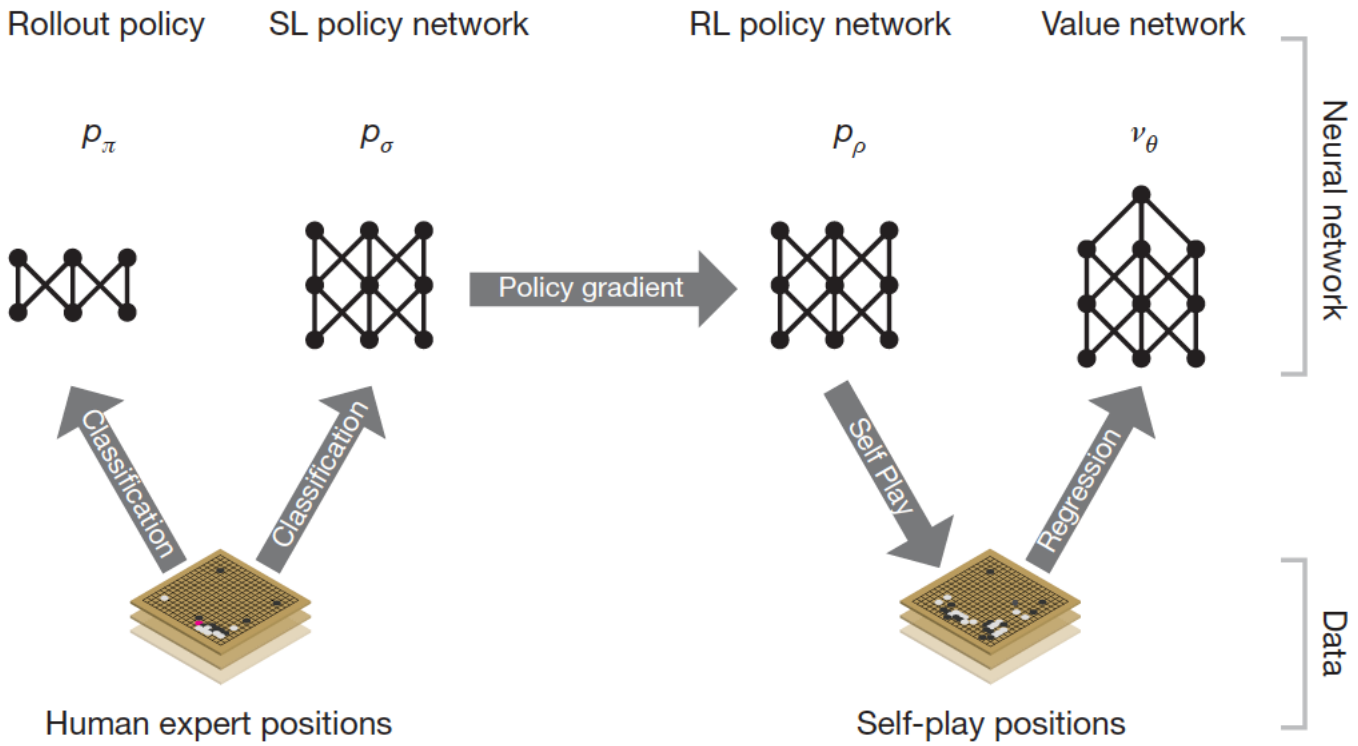
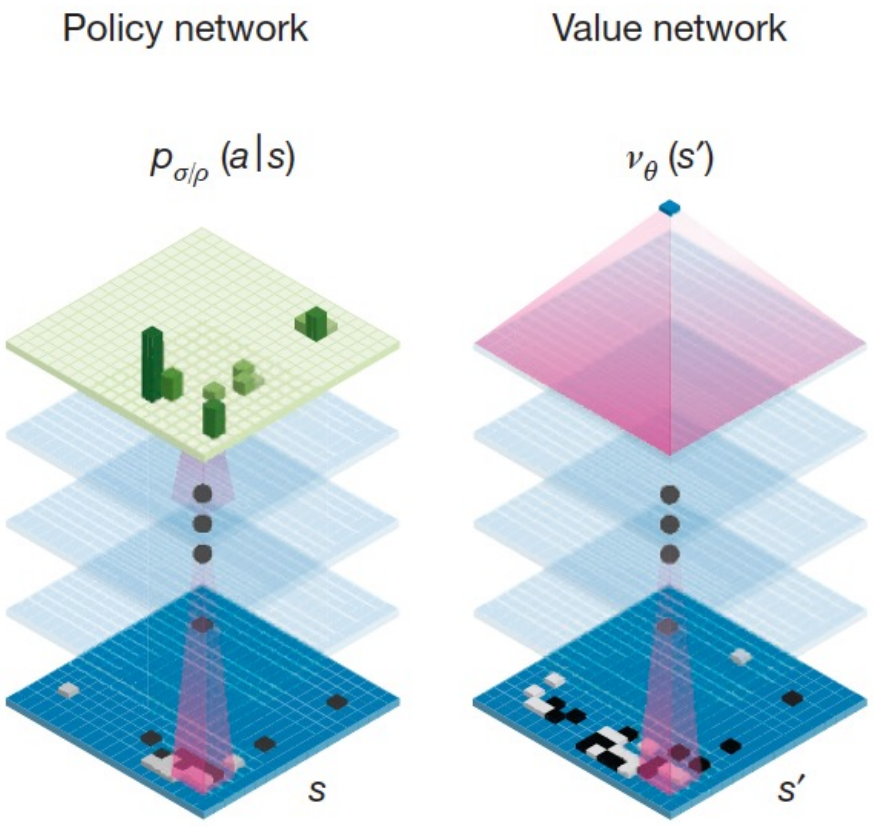


Figure 1 | Neural network training pipeline and architecture. a, A fast rollout policy p_π and supervised learning (SL) policy network p_σ are trained to predict human expert moves in a data set of positions. A reinforcement learning (RL) policy network p_ρ is initialized to the SL policy network, and is then improved by policy gradient learning to maximize the outcome (that is, winning more games) against previous versions of the policy network. A new data set is generated by playing games of self-play with the RL policy network. Finally, a value network v_θ is trained by regression to predict the expected outcome (that is, whether the current player wins) in positions from the self-play data set.

All these networks were trained before any live game play took place, and their weights remained fixed throughout live play.

Training Pipeline and Architecture (b)



b, Schematic representation of the neural network architecture used in AlphaGo. The policy network takes a representation of the board position s as its input, passes it through many convolutional layers with parameters σ (SL policy network) or ρ (RL policy network), and outputs a probability distribution $p_\sigma(a|s)$ or $p_\rho(a|s)$ over legal moves a, represented by a probability map over the board. The value network similarly uses many convolutional layers with parameters θ, but outputs a scalar value $v_\theta(s')$ that predicts the expected outcome in position s'.

Supervised learning of policy networks

- Build on prior work on predicting expert moves in the game of Go using supervised learning
- The final softmax layers of the SL policy network outputs a probability distribution over all legal moves a, i.e., $p_\sigma(a|s)$.
- The input s to the policy network is a simple representation of the board state
- The network is trained using stochastic gradient ascent to maximize the likelihood of the human move a selected in state s

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

- Depth: 13-layer Data: 30 million positions faced by experts.
- Accuracy: 57.0%

Reinforcement Learning of Policy Networks

- Policy network p_ρ is identical in structure to the SL policy network.
- Trained using policy gradient reinforcement learning (RL)
- Weights ρ are initialized to the same values, $\rho = \sigma$.
- Games were played between the current policy network p_ρ and a randomly selected previous iteration of the policy network

Reinforcement Learning of Policy Networks

- Reward function r(s):
  - 0 for all non-terminal time steps t < T.
  - +1 for winning and -1 for losing
- Outcome $z_t = \pm 1$ from the perspective of the current player at time step t
- Weights are updated at each time step t by stochastic gradient ascent in the direction that maximizes expected outcome

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

Reinforcement Learning of Value Networks

- Purpose: position evaluation, i.e., estimating a value function $v^p(s)$ that predicts the outcome from position s of games played by using policy p for both players

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

- Approximates the value function using a value network $v_\theta(s)$ with weights θ

$$v_\theta(s) \approx v^{p_\rho}(s) \approx v^*(s)$$

- Has a similar architecture to the policy network, but outputs a single prediction instead of a probability distribution.

Training of Value Networks

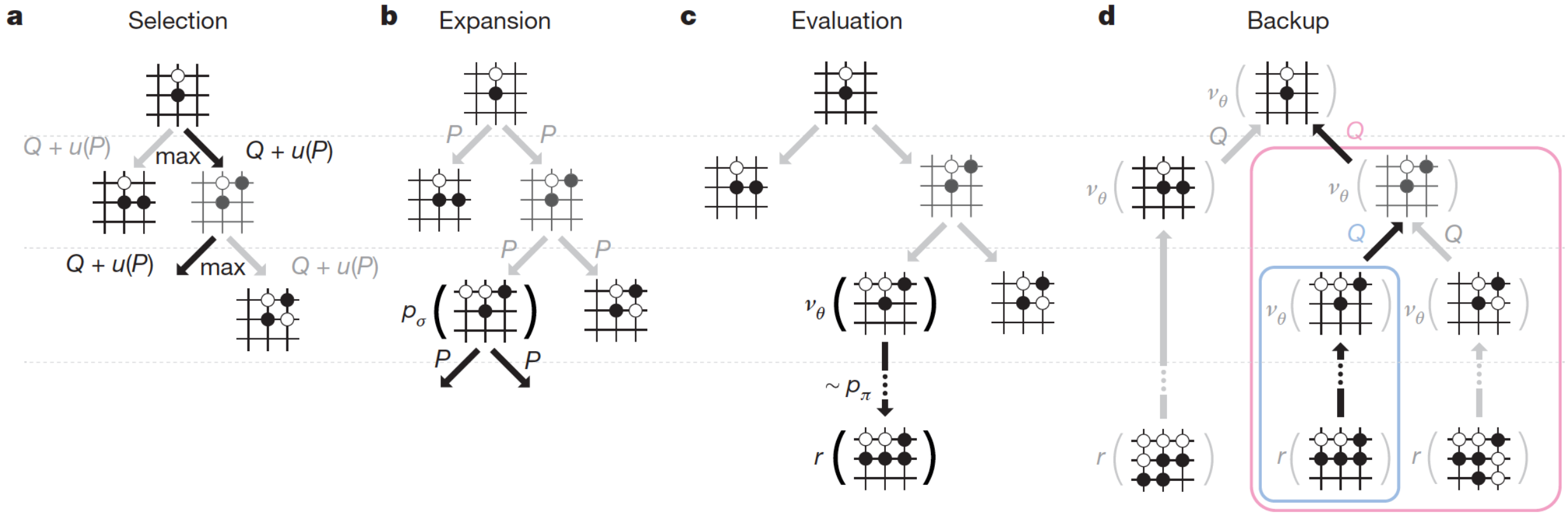
- Trains the weights of the value network by regression on state-outcome pairs (s, z), using stochastic gradient descent to MSE between the predicted value $v_\theta(s)$, and the corresponding outcome z

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial\theta}(z - v_\theta(s))$$

- Compared to Monte Carlo rollouts using the fast rollout policy p_π; the value function was consistently more accurate.
- A single evaluation of $v_\theta(s)$ approached the accuracy of Monte Carlo rollouts using the RL policy network p_ρ, but using 15,000 times less computation.

Searching with Policy and Value Networks

- AlphaGo combines the policy and value networks in an MCTS algorithm



MCTS in Alpha Go

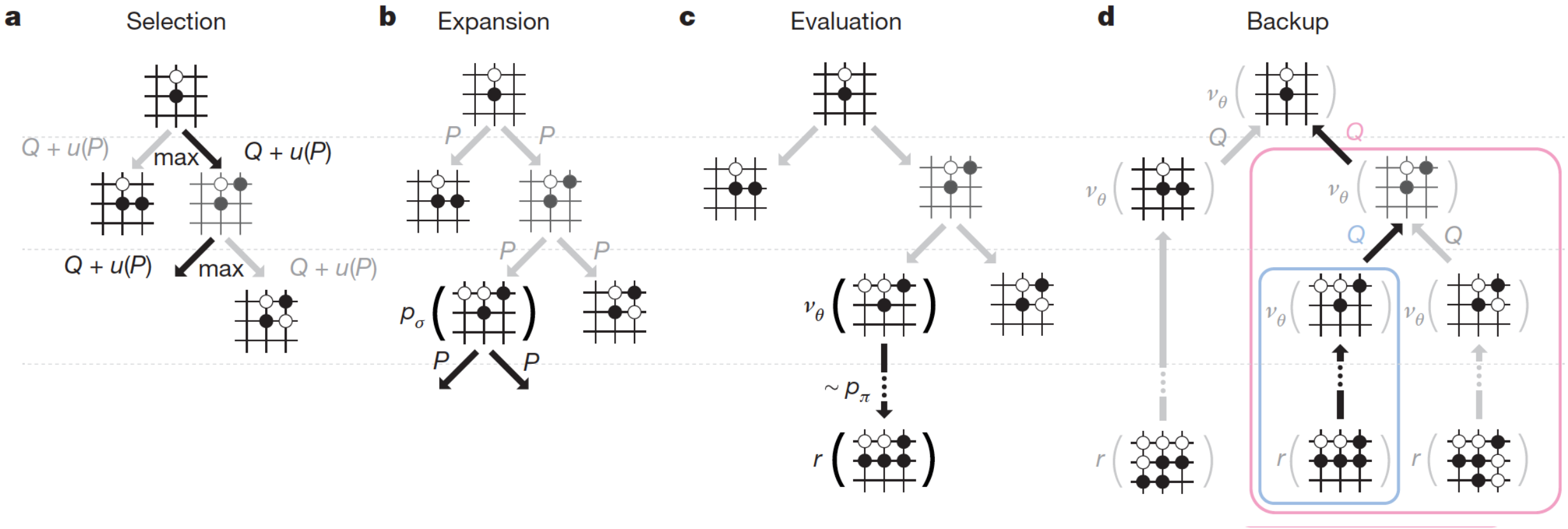


Figure 3 | Monte Carlo tree search in AlphaGo. a, Each simulation traverses the tree by selecting the edge with maximum action value Q, plus a bonus $u(P)$ that depends on a stored prior probability P for that edge. **b**, The leaf node may be expanded; the new node is processed once by the policy network p_σ and the output probabilities are stored as prior probabilities P for each action. **c**, At the end of a simulation, the leaf node is evaluated in two ways: using the value network v_θ; and by running a rollout to the end of the game with the fast rollout policy p_π, then computing the winner with function r. **d**, Action values Q are updated to track the mean value of all evaluations $r(\cdot)$ and $v_\theta(\cdot)$ in the subtree below that action.

MCTS in Alpha Go: Selection

- Each edge (s, a) of the search tree stores
  - an action value $Q(s, a)$,
  - visit count $N(s, a)$, and
  - prior probability $P(s, a)$
- The tree is traversed by simulation, starting from the root state.
- At each time step t of each simulation, an action a_t is selected from state s_t

$$a_t = \underset{a}{\operatorname{argmax}}(Q(s_t, a) + u(s_t, a)) \qquad \text{where} \qquad u(s, a) \propto \frac{P(s, a)}{1 + N(s, a)}$$

MCTS in Alpha Go: Selection

Regular MCTS

$$\mathrm{UCT}(s,a) = \frac{Q(s,a)}{N(s,a)} + c \cdot \sqrt{\frac{\ln N(s)}{N(s,a)}}$$

- $Q(s,a)$: Cumulative reward for action a at state s.
- $N(s,a)$: Visit count for action a at state s.
- $N(s)$: Total visit count for state s.
- c: Exploration parameter.

Alpha Go MCTS

$$\mathrm{PUCT}(s,a) = Q(s,a) + c_{puct} \cdot P(s,a) \cdot \frac{\sqrt{N(s)}}{1 + N(s,a)}$$

- $Q(s,a)$: Mean action value.
- $N(s,a)$: Visit count for action a at state s.
- $N(s)$: Total visit count for state s.
- $P(s,a)$: Prior probability from the policy network.
- c_{puct}: A constant determining the level of exploration.

Guided Exploration: The policy network's prior probabilities guide the tree search towards more promising moves, reducing the reliance on random exploration.

MCTS in Alpha Go: Expansion

- **Regular Expansion**:
  - If the leaf node is not terminal, one or more child nodes are added to expand the search tree.
- **Alpha Go Expansion:**
  - When a new node is expanded, AlphaGo initializes its child nodes with prior probabilities from the policy network.
  - For each possible action aaa from state s , store $P(s, a)$.

Informed Node Expansion: Instead of adding nodes arbitrarily or uniformly, AlphaGo uses the policy network to prioritize which nodes to expand.

Efficient Tree Growth: Focuses computational resources on more promising parts of the tree.

MTCS in Alpha Go: Simulation

- When the traversal reaches a leaf node s_L at step L, the leaf node may be expanded.
- The leaf position s_L is processed just once by the SL policy network p_σ. The output probabilities are stored as prior probabilities P for each legal action a, $P(s,a) = p_\sigma(a|s)$
- s_L is evaluated in two very different ways:
  - first, by the value network $v_\theta(s_L)$; and
  - second, by the outcome z_L of a random rollout played out until terminal step T using the fast rollout policy p_π;
  - The two evaluations are combined, using a mixing parameter λ, into a leaf evaluation $V(s_L)$

$$V(s_L) = (1-\lambda)v_\theta(s_L) + \lambda z_L$$

MTCS in Alpha Go: Backup

- At the end of simulation, the action values and visit counts of all traversed edges are updated.

$$N(s,a) \leftarrow N(s,a) + 1$$

$$W(s,a) \leftarrow W(s,a) + V(s')$$

$$Q(s,a) \leftarrow \frac{W(s,a)}{N(s,a)}$$

- $W(s,a)$: Total value of action a at state s.
- $V(s')$: Value estimate from the value network for the new state s'.

Value-Based Backpropagation: Uses neural network evaluations instead of simply binary win/loss outcomes.

MTCS in Alpha Go: Backup

- Each edge accumulates the visit count and mean evaluation of all simulations passing through that edge

$$N(s,a) = \sum_{i=1}^{n} 1(s,a,i)$$

$$Q(s,a) = \frac{1}{N(s,a)} \sum_{i=1}^{n} 1(s,a,i) V(s_L^i)$$

s_L^i is the leaf node from the ith simulation, and
$1(s,a,i)$ indicates whether an edge (s, a) was traversed during the ith simulation

Once the search is complete, the algorithm chooses the most visited move from the root position.

Additional Enhancements in AlphaGo

- **Combined Policy and Value Networks**
  - **Policy Network Training**: Trained on expert human games to predict strong moves.
  - **Value Network Training**: Trained through self-play to predict the probability of winning from a state.
- **Supervised and Reinforcement Learning**
  - AlphaGo's networks are trained using a combination of supervised learning (from human expert games) and reinforcement learning (through self-play), improving their predictive accuracy over time.
- **Dynamic Exploration**: Adjusts c_{puct} over time to balance exploration and exploitation dynamically.

Clarification: Do the value and policy RL networks get trained during MCTS?

- No
- In AlphaGo and its successors like AlphaGo Zero, the value and policy neural networks are not trained during the Monte Carlo Tree Search (MCTS) process itself.
- Instead, these networks are trained **offline** using data generated from previous games, including those guided by MCTS.
- During MCTS, the networks are used to evaluate positions and suggest promising moves, but they are not updated or trained in real-time as the search unfolds.

Training Phases vs. Inference Phase

- Training Phase (Offline):
  - **Data Generation:** AlphaGo generates training data by playing numerous games against itself (self-play), using MCTS guided by the current versions of the value and policy networks.
  - **Network Updates:** After a game is completed, the move sequences and outcomes are used to update the neural networks through supervised learning (for the policy network) and reinforcement learning (for both networks).
  - **Iterative Improvement:** This process repeats iteratively, gradually improving the networks as they learn from more data.

Training Phases vs. Inference Phase

- Inference Phase (During MCTS):
  - **Fixed Networks:** During the actual MCTS process used in gameplay or evaluation, the networks are fixed—they are not updated or trained.
  - **Guiding the Search:** The policy network provides prior probabilities for action selection, and the value network estimates the value of positions. These outputs guide the MCTS to explore more promising lines of play.
  - **Efficiency Considerations:** Updating Neural Nets during MCTS would be impractical and would significantly slow down the search process.