11 Deep Learning, Tree Search: AlphaGo, AlphaZero & MuZero (Advanced Topic)

Table of contents

Model-Based Reinforcement Learning

Last Module: Actor critic (policy gradient)

Previous Modules: Value function approximation using deep learning

This Module: Combining deep learning & tree search

AlphaGo

AlphaGo - Core Idea

First system to *combine deep learning and tree search* for superhuman play.

Domain: *Go* only.

Pipeline integrates:

1. *Supervised learning* from expert games of human players
2. *Reinforcement learning* through self-play
3. *Monte Carlo Tree Search (MCTS)* guided by neural networks

AlphaGo - Neural networks & losses

Neural Network	Description
Policy Network (π_1)	Trained *supervisedly* on human moves; 13-layer CNN (Go board 19×19 × 48 planes)
Policy Network (π_2)	Refined by self-play RL (same architecture)
Value Network (v)	13-layer CNN + 2 fully connected layers; outputs scalar win probability $v(s)$

Training objectives:

$$\mathcal{L}_\pi = -\log \pi_\theta(a^*|s), \qquad \mathcal{L}_v = (v_\phi(s) - z)^2$$

where $z \in \{-1, +1\}$ is the game outcome.

AlphaGo Planning integration

MCTS uses:

- *Policy prior* $\pi_\theta(a|s)$ from policy network π_1 (parameters θ) $\rightarrow$ biases search toward likely moves
- *Value estimate* $v_\phi(s)$ from value network v (parameters ϕ) $\rightarrow$ evaluate leaves

Move selection at root:

$$\pi_{\text{MCTS}}(a|s_0) \propto N(s_0, a)^{1/\tau}$$

Once π_2 is trained through reinforcement learning, AlphaGo uses it to play millions of games against itself.

- Each game produces pairs (s, z): (s_t, z_t) where $z \in \{-1, +1\}$ is the game outcome.

These are used to train the value network $v_\phi(s_t)$ via regression:

$$\min_\phi \left(v_\phi(s_t) - z_t\right)^2$$

- So the value network learns to predict *who* will win from any board position that strong play (i.e., π_2) would reach.

Achieved 4–1 win vs Lee Sedol (2016)

AlphaZero

AlphaZero - Unified Self-Play RL

Extends AlphaGo $\longrightarrow$ *Go*, *Chess*, *Shogi*

- Removes human data and hand-crafted rollout policy
- Fully self-play training loop

AlphaZero - Neural network and objective

Single residual CNN shared by policy + value

- 20 or 40 ResNet blocks, 256 filters, BatchNorm + Rectified Linear Circuit (ReLU)
- Input: stack of board planes (19×19×N)
- Heads:
  - *Policy head:* 1 conv + 1 FC $\rightarrow$ softmax over legal moves
  - *Value head:* 1 conv + 2 FC $\rightarrow$ scalar $v_\theta(s)$

Loss:

$$\mathcal{L}(\theta) = (z - v_\theta(s))^2 - \pi_{\text{MCTS}}^\top \log \pi_\theta + c\|\theta\|^2$$

AlphaZero - Learning & Planning Loop

$$\text{Network} \Rightarrow \text{MCTS} \Rightarrow \text{Self-play games} \Rightarrow \text{Network update}$$

- *MCTS*: ~800 simulations per move
- *Network*: ResNet trained via SGD on MCTS targets
- Unified architecture simplified training → superhuman performance across games

MuZero

MuZero - Learning to Plan Without Rules

- AlphaZero still needed explicit game rules
- *MuZero* learns a *latent model* of dynamics for planning without knowing rules
- Same MCTS framework, but search happens in latent space

MuZero - Neural networks (3)

Neural Network	Function	Notes
Representation h_θ	Observation $\longrightarrow$ latent state s_0 (learns (latent) state representation)	6 ResNet blocks for Atari (pixels $\longrightarrow$ latent)
Dynamics g_θ	Predicts s_{t+1}, r_{t+1} from (s_t, a_t) (learns model)	Small conv stack + reward head
Prediction f_θ	Outputs policy p_t and value v_t from s_t	Two heads (softmax policy, scalar value)

MuZero – TD-Style Learning (Bootstrapped Returns)

Unlike AlphaZero (which uses full-episode Monte Carlo targets), *MuZero* trains its value network using n-step bootstrapped (TD) returns

For each step (t), the target value is:

$$\hat{v}_t = \sum_{i=0}^{n-1} \gamma^i r_{t+i} + \gamma^n v_\theta(s_{t+n})$$

- Combines *observed rewards* and *bootstrapped value* from the predicted future state
- Allows *credit assignment* across long horizons without waiting for episode termination

MuZero minimises a combined loss:

$$\mathcal{L} = \sum_k \Big[(v_k - \hat{v}_k)^2 + (r_k - \hat{r}_k)^2 - \pi_k^\top \log p_k \Big]$$

- *Value loss:* TD-style bootstrapped error
- *Reward loss:* immediate reward prediction
- *Policy loss:* cross-entropy with MCTS visit-count distribution

TD bootstrapping makes MuZero more sample efficient than AlphaZero

Planning (MCTS) provides strong policy/value targets; TD updates keep learning continuous

MuZero - Training and Integration

MCTS operates within the learned model:

$$s_{t+1}, r_t = g_\theta(s_t, a_t)$$

Targets from MCTS train all three nets end-to-end

Loss:

$$\mathcal{L} = \sum_k \left[(v_k - \hat{v}_k)^2 + (r_k - \hat{r}_k)^2 - \pi_k^\top \log p_k \right]$$

- Achieves AlphaZero-level play in Go/Chess/Shogi and strong Atari results from pixels

MuZero - Results and Significance

Domain	*Training time to superhuman level*	*Benchmark / Opponent*	*Notes*
Chess	≈ 4 hours (on 8 TPUv3 pods)	*Stockfish*	Surpassed world-champion chess engine performance
Shogi	≈ 2 hours	*Elmo*	Surpassed leading professional Shogi engine
Go	≈ 9 hours	*AlphaZero / KataGo*	Matched AlphaZero's superhuman play using only learned dynamics
Atari (57 games)	~200M frames	*Rainbow / IMPALA*	Exceeded or matched best model-free RL baselines across games

Key insights

- *No rules given:* MuZero learned *dynamics, value, and policy* purely from experience
- *Unified algorithm:* Same architecture and hyperparameters across all domains
- *Planning efficiency:* Performed *Monte Carlo Tree Search (MCTS)* entirely in *latent space*
- *Sample efficiency:* Achieved AlphaZero-level play within *hours of self-play training*

AlphaGo, AlphaZero & MuZero Comparison

System	# of NNs	Architecture	Uses known rules?	Learns model?	Planning
AlphaGo	2 (policy + value)	13-layer CNNs	✓	✗	MCTS with rules
AlphaZero	1 (shared policy-value ResNet)	20–40 ResNet blocks	✓	✗	MCTS with rules
MuZero	3 (h, g, f modules)	ResNet latent model	✗	✓	MCTS in latent space