

Artificial Intelligence

From a View of Structural Estimation

Economic Interpretations of Deep Blue, Bonanza, & AlphaGo

Note: This presentation is based on Igami's. With his permission, I edited his materials and added new slides.

Artificial Intelligence

- ▶ Achieved human-level or **super-human** performances
 - ▶ Image recognition
 - ▶ Natural-language processing
 - ▶ **Classical board games**
 - ▶ Chess
 - ▶ Shogi (Japanese chess)
 - ▶ Go



3

Kasparov vs. Deep Blue (1997)



4

Watanabe vs. Bonanza (2007)



5

Satō vs. Ponanza (2017)

AlphaGo

vs. Lee (2016)



vs. Ke (2017)



“Explainable AI”

- ▶ But it's a “black box” (to many people, at least)
- ▶ US Department of Defense (DoD): “Explainable AI” (XAI) project
 - ▶ *“The effectiveness of these systems is limited by the machine's current inability to explain their decisions and actions to human users.”*
- ▶ End of human intelligence?

AI as Structural Estimation

- ▶ This presentation
 - ▶ Not going to solve all enigmas of AI
 - ▶ But I talk about certain types of AI
- ▶ Game AI
 - ▶ Deep Blue, Bonanza, & AlphaGo
- ▶ Why study game AI?
 - ▶ Archetypical “intelligent” tasks
 - ▶ Well-defined problems
 - ▶ Natural economic interpretations

AI as Structural Estimation

► Main points

- Most elements of game AIs have counterparts in econometrics.
 1. Deep Blue (chess) is a **calibrated** value function.
 2. Bonanza (shogi) is “*an empirical model of human players*” a la Rust ('87).
 3. AlphaGo's “SL policy net” is a DNN version of **CCP** by Hotz-Miller ('93).
 4. AlphaGo's “RL value net” is a DNN version of **CCS** by Hotz-Miller-Sanders-Smith ('94).

- Econometrics can help relax these AI's (implicit) assumptions, which will help:
 - ...make AIs more “**human-like**” (if needed)
 - ...make AI s more “**interpretable**” (in the structural-economic sense).

2. Some Structural Estimations

Value Function Iteration in Finite Time

- Problems where there is a terminal condition
- Nicer to solve? Backward Induction

- We begin with the Bellman operator:

$$\Gamma(V^t)(s) = \max_{a \in A(s)} \left[u(s, a) + \beta \int V^{t'}(s') p(ds'|s, a) \right]$$

- Specify V^T and apply Bellman operator:

$$V^{T-1}(s) = \max_{a \in A(s)} \left[u(s, a) + \beta \int V^T(s') p(ds'|s, a) \right]$$

- Iterate until first period:

$$V^1(s) = \max_{a \in A(s)} \left[u(s, a) + \beta \int V^2(s') p(ds'|s, a) \right]$$

Value Function Iteration in Infinite Time

- Problems where there is no terminal condition. Instead, transversality condition.
- Contraction Mapping

- We begin with the Bellman operator:

$$\Gamma(V)(s) = \max_{a \in A(s)} \left[u(s, a) + \beta \int V(s') p(ds'|s, a) \right]$$

- Specify V^0 and apply Bellman operator:

$$V^1(s) = \max_{a \in A(s)} \left[u(s, a) + \beta \int V^0(s') p(ds'|s, a) \right]$$

- Iterate until convergence:

$$V^T(s) = \max_{a \in A(s)} \left[u(s, a) + \beta \int V^{T-1}(s') p(ds'|s, a) \right]$$

Two Relevant Equations for Discrete Choice Dynamic Programming

1. The Bellman equation

$$V(\cdot; \theta) = f(V(\cdot; \theta); \theta)$$

→ obtain the **value function**

2. Mapping from the value function into a **policy function**: always exists

$$P(\cdot; \theta) = g(V(\cdot; \theta); \theta)$$

Rust ('87): NFXP

► Nested Fixed Point Algorithm (NFXP)

1. **Outer loop:** Parameter (θ) search for value function

- ✓ ...to fit model's **prediction** with **actual** choices in data
- ✓ ...implemented by **Maximum Likelihood** method

2. **Inner loop:** Dynamic programming (DP) problem

- i. Takes θ as input
- ii. Solves DP using **value function iteration**
- iii. Returns “optimal” action using a mapping from $V()$ into $P()$

Hotz and Miller ('93): CCP

1. The **alternative** representation

$$\begin{aligned} V(\cdot; \theta) &= h(P(\cdot; \theta); \theta) \\ P(\cdot; \theta) &= g(V(\cdot; \theta); \theta) \end{aligned}$$


2. Substituting the first equation into the second gives:

$$P(\cdot; \theta) = g(h(P(\cdot; \theta); \theta); \theta)$$

3. **The first stage of HM:** corresponding to the inner loop of NFXP

$$\bar{P}(\cdot; \theta) = g\left(h\left(\hat{P}(\cdot); \theta\right); \theta\right)$$

where $\hat{P}(\cdot)$ is the conditional choice probabilities consistently estimated by simply computing the probabilities of various actions conditional on the observed state in the data.

4. **The second stage:** Estimate the parameters by comparing $\bar{P}()$ with the actual data.

Hotz, Miller, Smith and Sanders ('94): CCS

- ▶ For some cases (e.g. large state space), it is difficult to get the alternative representation.

$$V(\cdot; \theta) = h(P(\cdot; \theta); \theta)$$

- ▶ Instead of finding the function h , they use **simulations** to find the value function.
- ▶ **A sketch of the HMSS approach**
 1. Run many forward simulations based on the first-stage CCP of HM.
 2. Record the state (and action) and the result (win/lose) of the game.
 3. Estimate the value function and parameters using the data.

3. Notations

Notations

- ▶ Same class of games
 - ▶ Two players, $i = 1, 2$
 - ▶ Discrete time, $t = 1, 2, 3, \dots$
 - ▶ Alternating moves
 - ▶ Perfect information
 - ▶ Deterministic state transition

$$s_{t+1} = f(s_t, a_t)$$

Notations

- ▶ Same class of games (cont.)
 - ▶ Finite action space ("legal" moves)

$$a_t \in \mathcal{A}(s_t)$$

- ▶ Finite state space

$$s_t \in \mathcal{S} = \mathcal{S}_{cont} \sqcup \mathcal{S}_{win} \sqcup \mathcal{S}_{loss} \sqcup \mathcal{S}_{draw}$$

- ▶ Zero-sum payoffs

$$u_1(s_t) = \begin{cases} 1 & \text{if } s_t \in \mathcal{S}_{win}, \\ -1 & \text{if } s_t \in \mathcal{S}_{loss}, \text{ and} \\ 0 & \text{otherwise,} \end{cases}$$

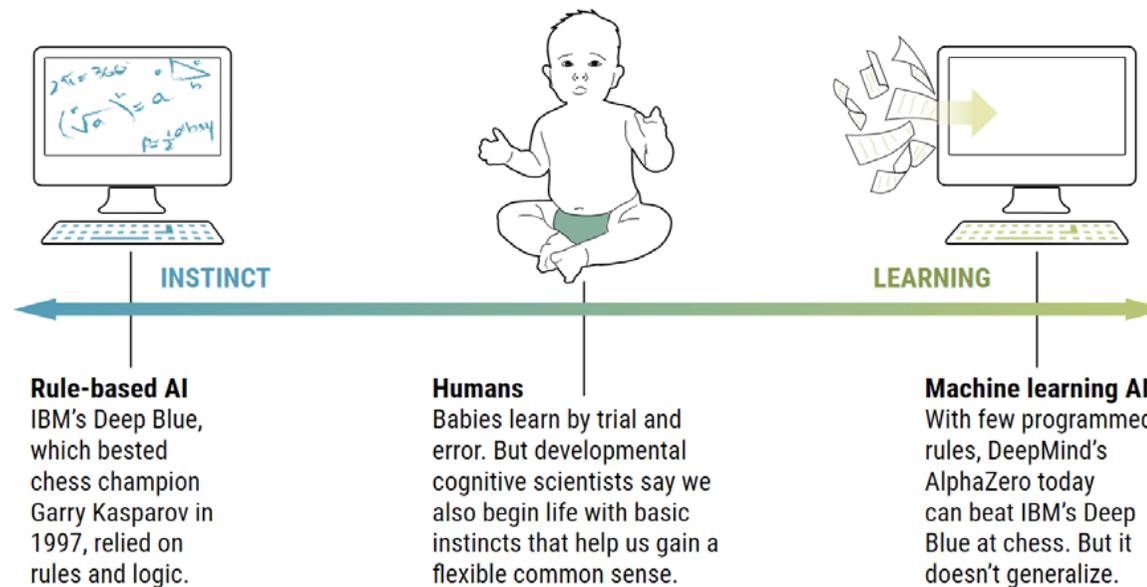
Notations

- ▶ In principle, can be solved exactly & completely
 - ▶ ...for subgame-perfect equilibrium
 - ▶ ...by backward induction from terminal states
- ▶ In practice, we can't.
 - ▶ Size of state space
$$10^{47}, 10^{71}, \text{ and } 10^{171}$$
 - ▶ # atoms in the universe
$$10^{78} \sim 10^{82}$$
 - ▶ Our total HDD capacity (in 2016)
in the order of 10^{20} bytes

Spectrum of AI

Different minds

Over time, artificial intelligence (AI) has shifted from algorithms that rely on programmed rules and logic—instincts—to machine learning, where algorithms contain few rules and ingest training data to learn by trial and error. Human minds sit somewhere in the middle.



4. Deep Blue (Chess)

Case Study #1

Case 1: Deep Blue

- ▶ Three components
 1. Evaluation function
 2. Search algorithms
 3. Databases

Case 1: Deep Blue

- ▶ Component 1: Evaluation function
 - ▶ Evaluates positions
 - ▶ Material value
 - ▶ Pawn (1 point), knight (3), bishop (3), rook (5), queen (9), king (∞)
 - ▶ Other factors
 - ▶ Pawn structure
 - ▶ "A pair of bishops are worth more than their sum", etc.
 - ▶ Protection of kings
 - ▶ Phase of the game: Opening, middle, end

Case 1: Deep Blue

- ▶ Component 1: Evaluation function (cont.)

$$V_{DB}(s_t; \theta) = \theta_1 x_{1,t} + \theta_2 x_{2,t} + \cdots + \theta_K x_{K,t}$$

- ▶ Manually adjusted 8,150 parameters
- ▶ Hard-wired to semiconductor chips

Case 1: Deep Blue

- ▶ Component 2: Search algorithms
 - ▶ “Full-width” search of game tree
 - ▶ Evaluating (almost) all possible positions
 - ▶ ...for a fixed number of moves
 - ▶ ...using “minimax” algorithm
 - ▶ ...and some “pruning” methods (e.g., “alpha-beta cut”)

Case 1: Deep Blue

- ▶ Component 3: Databases
 - ▶ Endgame
 - ▶ Captured pieces never come back (“directional” game)
 - ▶ Ken Thompson’s database
 - ▶ All 5-piece endings (7GB)
 - ▶ Selected 6-piece endings (1.2 TB if included all)
 - ▶ Opening book
 - ▶ Grandmasters Joel Benjamin, Nick Fedorowicz, & Miguel Illescas
 - ▶ 4,000 positions

Deep Blue is Calibrated Value Function

- ▶ “Calibration”: Manual choice & adjustment of parameter values
 - ▶ Not “estimation” from data, by econometric methods
 - ▶ Typically used in macro-economics

- ▶ Deep Blue’s evaluation function

$$V_{DB}(s_t; \theta) = \theta_1 x_{1,t} + \theta_2 x_{2,t} + \dots + \theta_K x_{K,t}$$

- ▶ Calibrated, approximate terminal-value function
- ▶ ...with L -period forward-looking player

$$V_{DB}(s_{t+L}; \theta)$$

- ▶ ...in a game against its doppelgänger

5. Bonanza (Shogi)

Case Study #2

Case 2: Bonanza

- ▶ Prof. Kunihiro Hoki
 - ▶ Computational chemist at University of Toronto
 - ▶ Interested in computer chess in 2005
 - ▶ Won the world championship in computer shogi in 2006
- ▶ Machine learning to “train” evaluation function
 - ▶ 50 million parameters
 - ▶ Shogi is more complicated:
 - ▶ Bigger board (9 x 9 > 8 x 8) $|\mathcal{S}_{shogi}| \approx 10^{71} > 10^{47} \approx |\mathcal{S}_{chess}|$
 - ▶ More pieces (40 > 32)
 - ▶ More types of pieces (8 > 6)
 - ▶ Pieces never die.

Case 2: Bonanza

- ▶ More flexible evaluation function

$$V_{BO}(s_t; \theta_{BO}) = \theta_1 x_{1t} + \theta_2 x_{2t} + \cdots + \theta_K x_{Kt}$$

- ▶ Factorization of global positions into:
 - ▶ 3 pieces including two kings
 - ▶ 3 pieces including only one king
- ▶ Material values
- ▶ Other factors (Hoki & Watanabe [2007], pp. 119–120)

Case 2: Bonanza

- ▶ Data
 - ▶ 50,000 official games by professionals
 - x 100 (moves/game)
 - = 5 million moves (state-action pairs)
 - ▶ But how do you estimate 50 million parameters
 - ▶ ...from only 5 million data points?
 - ▶ “Regularization” reduces the effective number of free parameters.

Case 2: Bonanza

► How to “train” value function by state-action data?

1. Guess θ to make $V_{BO}(s_t; \theta)$
2. Put $V_{BO}(s_{t+L}; \theta)$ at the end of game tree (truncated after L periods).

3. Solve the tree backward, and find optimal “predicted” move

$$a_t^* = \arg \max_{a \in \mathcal{A}(s_t)} \{V_{BO}(s_{t+L}; \theta)\}$$

4. Keep trying different θ until predicted moves fit actual moves in data.

Case 2: Bonanza

- ▶ Performance
 - ▶ Bonanza lost vs. Ryūō champion Akira Watanabe (2007)
 - ▶ But closer call than expected
 - ▶ Bonanza won vs. Meijin champion Amahiko Satō (2017)
 - ▶ One of Bonanza-children (plus reinforcement learning) by Issei Yamamoto

Bonanza is Harold Zurcher

- ▶ Bonanza is another approximate terminal-value function

$$V_{BO}(s_t; \theta_{BO}) = \theta_1 x_{1t} + \theta_2 x_{2t} + \cdots + \theta_K x_{Kt}$$

- ▶ Backward induction (on a truncated game tree of self-play)

$$a_t^* = \arg \max_{a \in \mathcal{A}(s_t)} \{V_{BO}(s_{t+L}; \theta)\}$$

- ▶ Difference with Deep Blue: **Estimated** from data

- ▶ Just like Rust ('87):

- ▶ “Optimal Replacement of GMC Bus Engines: An Empirical Model of Harold Zurcher”
 - ▶ “Full-solution” method of estimation

Bonanza is Harold Zurcher

- ▶ Literally the same algorithm: Two numerical optimization problems, nested
 1. **Outer loop:** Parameter (θ) search for value function
 - ✓ ...to fit model's **prediction** with **actual** choices in data
 - ✓ ...implemented by Maximum Likelihood method
 2. **Inner loop:** Dynamic programming (DP) problem
 - i. Takes θ as input
 - ii. Solves DP
 - iii. Returns “optimal” actions
 - ✓ ...implemented by value-function iteration

Bonanza is Harold Zurcher

- ▶ Differences between Bonanza & NFXP
 - ▶ **Finite**-horizon DP vs. **infinite**-horizon DP
 - ✓ **Backward induction** vs. **value-function iteration**
 - ✓ But value-function iteration *is* backward induction.
 - ▶ **Two-player** dynamic game vs. **single-agent** DP
 - ✓ But two-player game *becomes effectively* single-agent problem here.
 - ✓ Conversely, NFXP extends to games with alternating moves & finite horizon (c.f., Igami [2017, 2018]; Igami & Uetake [2017])

6. AlphaGo (Go)

Case Study #3

Case 3: AlphaGo

- ▶ Background
 - ▶ Successful parameterization of \mathcal{S} , for chess & shogi
 - ▶ Go is different.
 - ▶ Bigger board (19 x 19): $|\mathcal{S}_{go}| \approx 10^{171}$
 - ▶ Bigger $\mathcal{A}(s_t)$: legal moves = all open spaces
 - ▶ Even experts can't articulate
 - ▶ Developers gave up on **evaluation function...**

Case 3: AlphaGo

- ▶ Background (cont.)
 - ▶ Since 2006, more brute-force method: Monte Carlo Tree Search (MCTS)
 1. Start from current state.
 2. Fill the board randomly ("play-out").
 3. Repeat "play-out" many times.
 4. Calculate $Pr(win)$ from simulations.
 -
 5. Choose the "best" move
 - ▶ Could beat amateurs but not professionals

Case 3: AlphaGo

- ▶ Four components of AlphaGo (original)
 1. Policy network
 2. Value network
 3. Reinforcement learning
 4. Monte Carlo tree search

Case 3: AlphaGo

- ▶ **Component 1:** Supervised learning (SL) of **policy network**
 - ▶ Policy function with 4.6 million parameters,
 - ▶ Specification
 - ▶ **Input:** **Current state** of the board
 - ▶ “Black box”:
 - ▶ 48 input “**channels**” (or variables)
 - ▶ 13 “**layers**” (processing stages)
 - ▶ 192 “**kernels**” (3 x 3 filters for pattern recognition)
 - ▶ Convolution
 - ▶ “Rectified-linear” & “sigmoid” (logit) transformations
 - ▶ **Output:** Predicted choice probabilities of **next move**

$$\Pr(a_t = j | s_t) = \frac{\exp(y_j(s_t))}{\sum_{j' \in \mathcal{A}(s_t)} \exp(y_{j'}(s_t))}$$

Case 3: AlphaGo

- ▶ **Component 1:** Supervised learning (SL) of **policy network** (cont.)
 - ▶ Data
 - ▶ 160,000 games by 6–9 dan professionals on Kiseido Go Server
 - ▶ x 200 (moves/game)
 - ▶ x 8 (symmetric transformations)
 - ▶ = 256 million (action-state pairs)
 - ▶ **“Supervised learning”** (i.e., estimation) of $\psi: \sigma(s_t; \hat{\psi})$
 - ▶ Maximum Likelihood method
 - ▶ No “regularization”

Case 3: AlphaGo

- ▶ **Component 1:** Supervised learning (SL) of **policy network** (cont.)
 - ▶ Performance
 - ▶ Simple parametric (logit) version: 27% accurate prediction
 - ▶ Deep neural network (DNN) version: 56% accurate prediction
 - ▶ Top-5 **predicted** moves cover 90% of **actual** moves.

Case 3: AlphaGo

- ▶ **Component 2:** Reinforcement learning (RL) of **policy network**
 - ▶ Makes **stronger policy function**
 - ▶ Ultimate goal: Beat top humans
 - ▶ Not: “Structurally estimate a model of human players”
 - ▶ Find new parameter values with **higher $Pr(win)$**
 1. Try slightly different $\tilde{\psi} \neq \hat{\psi}$.
 2. Simulate a lot of games between $\sigma(s_t; \tilde{\psi})$ and $\sigma(s_t; \hat{\psi})$.
 3. Pick $\tilde{\psi}$ with highest $Pr(win)$.

$$\Pr_{win} \left(\sigma(s_t; \tilde{\psi}), \sigma(s_t; \hat{\psi}) \right) > \Pr_{win} \left(\sigma(s_t; \hat{\psi}), \sigma(s_t; \hat{\psi}) \right)$$

Case 3: AlphaGo

- ▶ **Component 2:** Reinforcement learning (RL) of **policy network** (cont.)
 - ▶ In principle, this “**RL policy net**” should be enough to beat humans.
 - ▶ In practice, Google DeepMind prepared **two more** components
 - ▶ ...in search of “ensemble effects”

Case 3: AlphaGo

► Component 3: SL/RL of value network

► How to construct evaluation function from policy function

1. Simulate many games (of RL policy vs. RL policy)

$$\Pr_{win} \left(\sigma \left(s_t; \tilde{\psi} \right), \sigma \left(s_t; \tilde{\psi} \right) \right)$$

2. Collect synthetic “data” (on state-winner pairs)
3. Use “data” to “train” a new model, to predict $\Pr(\text{win})$ from state.

► This “new model” is another DNN of similar design:

- 49 variables
- 15 layers
- 192 kernels

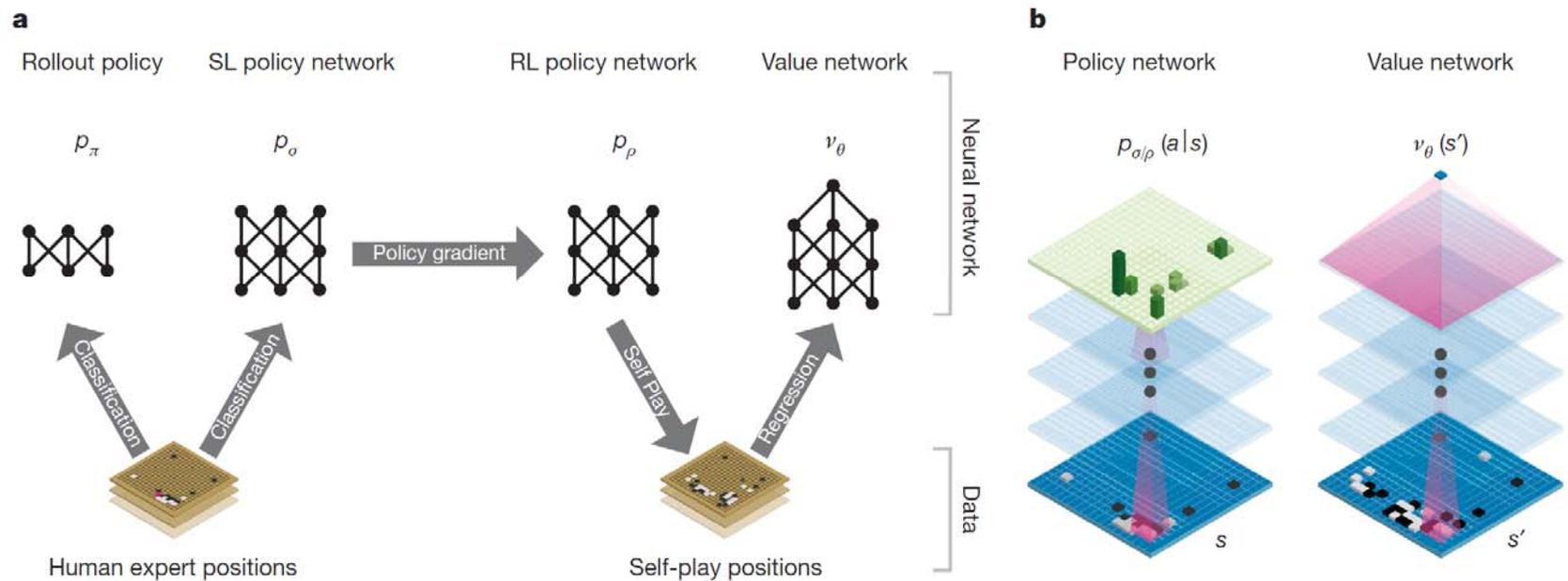
Case 3: AlphaGo

- ▶ **Component 3: SL/RL of value network**
 - ▶ **Value function**, with millions of parameters in θ

$$V\left(s_t; \hat{\theta}, \sigma_i = \sigma_{-i} = \sigma\left(s_t; \tilde{\psi}\right)\right)$$

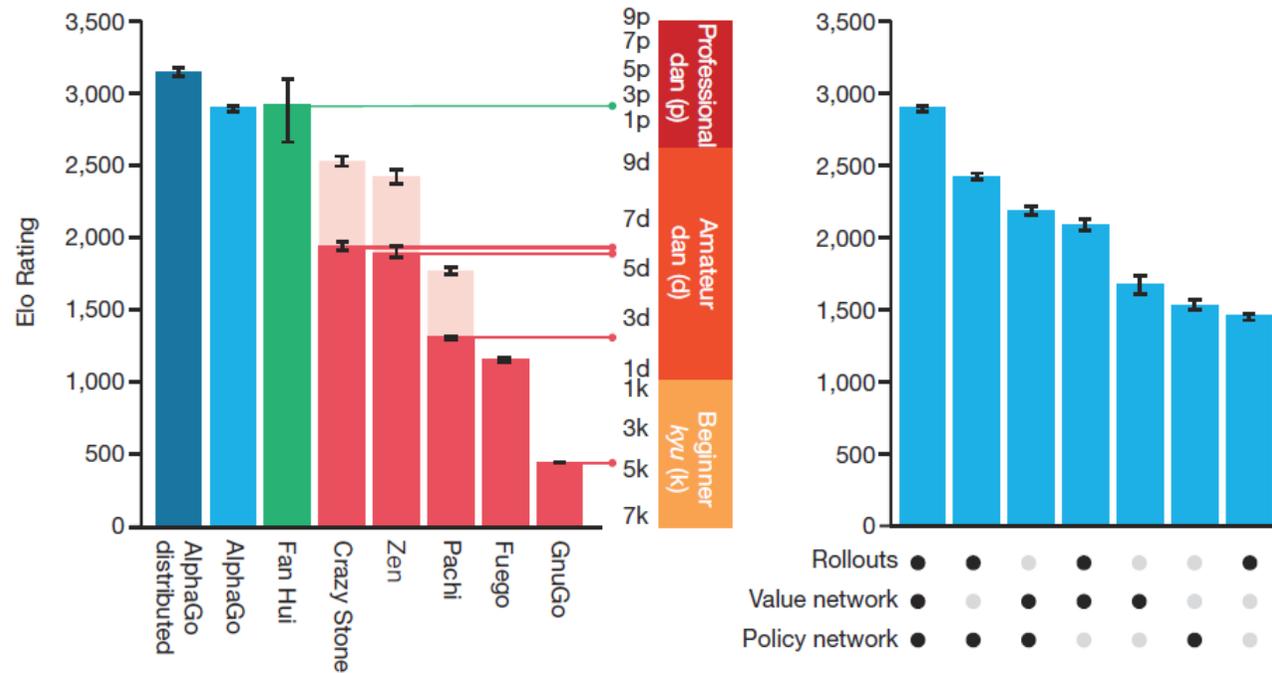
- ▶ Input: current state
- ▶ Output: Pr(win)
- ▶ Given: the DNN functional form & parameter values θ
- ▶ ...and under assumption: $\sigma_i = \sigma_{-i} = \sigma\left(s_t; \tilde{\psi}\right)$

Neural Network Training Pipeline and Architecture



Case 3: AlphaGo

- ▶ **Component 4:** Monte Carlo tree search (MCTS)
 - ▶ MCTS: stochastic numerical search of optimal policy
 - ▶ ...because exact & complete backward induction is impossible



“Ensemble effects” of combining multiple components

Figure 4: Tournament Evaluation of AlphaGo (panels a & b)

Silver et al (2016), “Mastering the game of Go with deep neural networks and tree search”, *Nature*, 529: 484–489.

AlphaGo is Two-Step Estimation

- ▶ Two approaches to estimating dynamic structural models
 - ▶ Rust ('87): Full-solution
 - ▶ Solve DP to estimate policy/value
 - ▶ Hotz & Miller ('93): Two steps
 1. Estimate policy function
 2. Estimate value function (& its underlying components)

- ▶ Context
 - ▶ Computational cost of solving DP (at each iteration of parameter search)
 - ▶ Want to avoid “curse of dimensionality” by side-stepping DP

AlphaGo is Two-Step Estimation

► Hotz & Miller ('93):

► “Conditional Choice Probabilities and the Estimation of Dynamic Models”

Step 1: Use data (on states & actions) to directly estimate policy function (CCPs)

► This is exactly what “SL policy network” is.

AlphaGo is Two-Step Estimation

► Hotz & Miller ('93):

Step 2: Policy & value are “duals” ⇒ Can invert **policy** for **value**!

- Hotz & Miller ('93) propose matrix inversion.
- Hotz, Miller, Sanders, & Smith ('94) propose numerical “inversion” via simulations.
 - “A Simulation Estimator for Dynamic Models of Discrete Choice”
- This is exactly what “SL/RL **value network**” is.
- ✓ Bajari, Benkard, & Levin ('07): Extension to **dynamic games**

Reinforcement Learning & MCTS

- ▶ **Econometrics**: Dirty data from human behavior
- ▶ **RL & MCTS**: No (human) data
 - ▶ RL of policy network: “**counterfactual experiment**” with long-lived players
 - ▶ MCTS: Pure **numerical approximation** to the exact solution

Reinforcement Learning & MCTS

- ▶ **AlphaGo Zero** (October 2017): No human data or inputs
 - ▶ Combines (i) **policy net**, (ii) **value net**, & (iii) **MCTS** into one
 - ▶ Reasonable:
 - ▶ Collapse (**redundant**) **duals** into single model.
 - ▶ Why did (**original**) **AlphaGo** bother with human data?
 - ▶ Exact, complete solution is not available.
 - ▶ Self-trained strategy could grow "**exotic**".
 - ▶ **Not** guaranteed to be "**stronger**" against human strategies
 - ▶ Reasonable:
 - ▶ First, "**initialize**" with human play.
 - ▶ Then, make it stronger **iteratively**.

Reinforcement Learning & MCTS

▶ AlphaGo Zero

▶ Once you beat all top humans, make it:

▶ “Cleaner” & “self-made”

▶ No human input

▶ “Simpler”

▶ Single object

▶ “Bigger”

▶ Bigger/deeper DNN architecture

= more flexible functional form

Summary

- A) **Deep Blue** (chess) is a value function
...that is **calibrated**.
- B) **Bonanza** (shogi) is a value function
...that is **estimated** by Rust's ('87) **nested fixed-point** method.
- C) AlphaGo's "**SL policy net**" is a policy function
...that is estimated by Hotz & Miller's ('93) **conditional choice probability** method.
- D) AlphaGo's "**RL policy net**" is like a **counterfactual experiment**
...in which Go players are forced to beat each other forever.
- E) AlphaGo's "**SL/RL value net**" is a value function
...that is estimated by Hotz-Miller-Sanders-Smith's ('94) **conditional choice simulations**.
- F) No obvious link between **AlphaGo Zero** & econometrics
...but its certain **design features** & **course of development** make sense.

7. Implicit Assumptions

...underlying the machine-learning algorithms for these game AIs

Implicit Assumptions

- ▶ Presence of an error term (random, transient utility component)
 - ▶ Associated with each of the discrete alternatives: $a_t \in \mathcal{A}(s_t)$

$\varepsilon(a_t) \sim$ type-1 extreme value

- ▶ Assumed i.i.d. across alternatives, players, time, and games

Implicit Assumptions

- ▶ The true data-generating process contains:
 1. Consideration sets & selective search
 2. Cross-sectional heterogeneity
 3. Inter-temporal heterogeneity
 4. Strategic interactions
 5. Time & physical constraints

Opportunities for Future Research

- ▶ Relaxing these implicit assumptions...
 - ▶ ...allows AIs to become more “human-like”
 - ▶ ...allows AIs to become more “interpretable” (in the structural-economic sense)

- ▶ DNN for nonparametric CCP estimation

- ▶ Gains from trade
 - ▶ AI as structural estimation
 - ▶ AI for structural estimation