Artificial Intelligence as Structural Estimation

Economic Interpretations of Deep Blue, Bonanza, & AlphaGo

Mitsuru Igami, Yale Department of Economics & Cowles Foundation

The NBER conference on the Economics of Artificial Intelligence at the University of Toronto, September 13–14, 2018

Artificial Intelligence

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



Kasparov vs. Deep Blue (1997)



Watanabe vs. Bonanza (2007)



Satō vs. Ponanza (2017)

AlphaGo

vs. Lee (2016)



vs. Ke (2017)



AI as Structural Estimation

- This paper
  - Not going to solve all enigmas of AI
  - But I do understand 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

- My main points
  - AI-Econometrics mapping
    1. Chess AI = Calibration
    2. Shogi AI = Rust ('87)
    3. Go AI ("policy") = Hotz-Miller ('93)
    4. Go AI ("value") = Hotz-Miller-Sanders-Smith ('94)
  - Econometrics can help:
    - ...clarify AIs' (implicit) assumptions
    - ...make AIs more "human-like"
    - ...make AIs more "interpretable"

2. Notations

Notations

- Same class of games
  - Two players, $i = 1, 2$
  - Discrete time, $t = 1, 2, 3, \ldots$
  - 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 Nash 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

$$\text{in the order of } 10^{20} \text{ bytes}$$



3. Deep Blue (Chess)

Case Study #1

Case 1: Deep Blue

- Three (software) components
  1. Evaluation function
  2. Search algorithms
  3. Databases

Case 1: Deep Blue

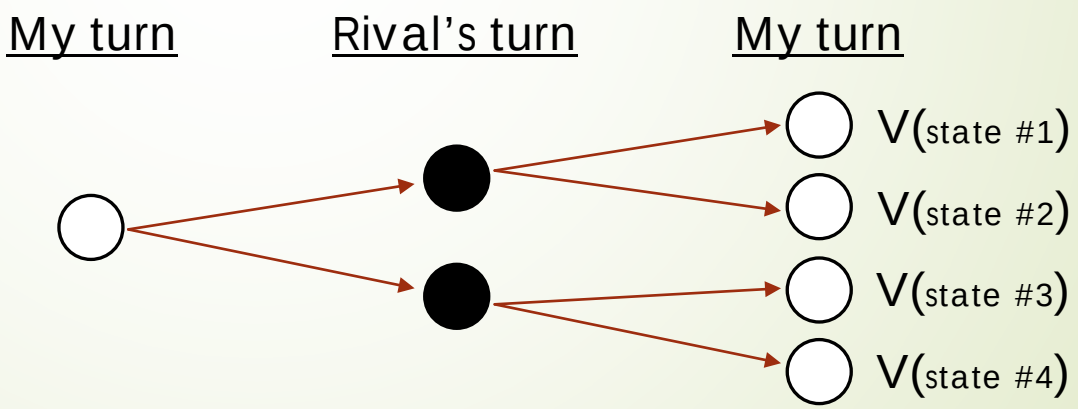
- Component 1: “Evaluation” function

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

  - RHS variables = Material value & other factors
  - *Manually adjusted* 8,150 parameters

Case 1: Deep Blue

- Component 2: “Search” algorithm
  - Game tree, truncated
  - Solve backward
  - Rival minimizes my V(.)



Case 1: Deep Blue

- Component 3: “Databases”
  - Endgame
    - Ken Thompson’s database
      - All 5-piece endings (7GB)
      - Selected 6-piece endings (1.2 TB if included all)
  - Opening book
    - 4,000 positions
      - By Grandmasters Joel Benjamin, Nick Fedorowicz, & Miguel Illescas

Deep Blue = Calibrated Value Function

- "Calibration": Manual choice & adjustment of parameter values
  - Not "estimation" from data, by econometric methods

- Evaluation function

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

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

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

  - ...in a game against its doppelgänger



4. Bonanza (Shogi)

Case Study #2

Case 2: Bonanza

- Prof. Kunihito Hoki
  - Computational chemist at University of Toronto
  - World champion of computer shogi in 2006
- Machine learning to "train" evaluation function
  - 50 million parameters
  - Shogi is more complicated: $|\mathcal{S}_{shogi}| \approx 10^{71} > 10^{47} \approx |\mathcal{S}_{chess}|$
    - Pieces never die (captured → re-deployed)

Case 2: Bonanza

- More flexible evaluation function

$$\begin{aligned} V_{BO}\left(s_t;\theta_{BO}\right) &= \theta_1 x_{1t} + \theta_2 x_{2t} + \cdots + \theta_K x_{Kt} \\ &\approx u_{BO}\left(s_\infty\right). \end{aligned}$$

  - "Factorizing" positions into 3-piece combinations

    = basis function

Case 2: Bonanza

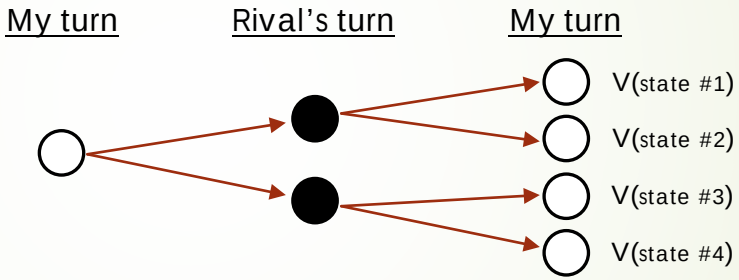
- Data
  - 50,000 official games by professionals

    x 100 (moves/game)

    = 5 million moves (state-action pairs)

Case 2: Bonanza

- How to "train" value function by 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).



  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_{BO} until predicted moves fit actual moves in data.

Bonanza = Harold Zurcher

- Same as Rust ('87)
  - "Optimal Replacement of GMC Bus Engines: An Empirical Model of Harold Zurcher"
- Two numerical optimization problems, nested
  1. **Outer loop**: Parameter (θ) search for value function
     - ✓ ...to match model's prediction & actual choices in data
     - ✓ ...implemented by MLE
  2. **Inner loop**: Dynamic programming (DP) problem
     - i. Takes θ as input
     - ii. Solves DP
     - iii. Returns "optimal" actions
     - ✓ ...implemented by value-function iteration
- "Nested Fixed-Point" algorithm





5. 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. Reinforcement learning
  3. Value network
  4. Monte Carlo tree search

AlphaGo (1 of 4)

- 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

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

AlphaGo (1 of 4)

- 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 ψ: $\sigma\left(s_t;\hat{\psi}\right)$
    - Maximum Likelihood method
    - No "regularization"

SL Policy = Hotz-Miller ('93)

- Hotz & Miller ('93):
  - "**C**onditional **C**hoice **P**robabilities 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 (2 of 4)

- Component 2: Reinforcement learning (RL) of policy network
  - Make a stronger policy = Find parameter values with higher *Pr(win)*

1. Try slightly different $\tilde{\psi} \neq \hat{\psi}$.
2. Simulate a lot of games between $\sigma\left(s_t; \tilde{\psi}\right)$ and $\sigma\left(s_t; \hat{\psi}\right)$.
3. Pick $\tilde{\psi}$ with highest *Pr(win)*.

“RL Policy” = Policy-function Iteration

- Component 2: Reinforcement learning (RL) of policy network
  - Like a “counterfactual experiment” with long-lived players
  - Same as “policy-function iteration”
    - Approximating the optimal strategy
      - (guaranteed to exist here)

AlphaGo (3 of 4)

- 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(win) from state.
  - This "new model" is another DNN of similar design:
    - 49 variables
    - 15 layers
    - 192 kernels

AlphaGo (3 of 4)

- 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: $\sigma_i = \sigma_{-i} = \sigma\left(s_t; \tilde{\psi}\right)$

"Value Network" = HMSS ('94)

- 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

AlphaGo (4 of 4)

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

Summary

A) Deep Blue (chess) = value function, calibrated

B) Bonanza (shogi) = value function, estimated by Rust ('87)

C) AlphaGo = two-step estimation

1. "SL policy net" = policy function, estimated by HM ('93)
2. "RL policy net" = "policy-function iteration" / counterfactual simulations
3. "value net" = value function, estimated by HMSS ('94)

6. Discussions

Making implicit assumptions explicit

Issue #1: Implicit Assumptions

- Error term (random, transient utility component)
  - Associated with each discrete alternative, $a_t \in \mathcal{A}(s_t)$

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

  - Assumed i.i.d. across everything

Issue #2: Data-Generating Process

- True DGP:
  1. Consideration sets & selective search
  2. Cross-sectional heterogeneity
  3. Inter-temporal heterogeneity
  4. Strategic interactions
  5. Time & physical constraints

Issue #3: “Explainable” AI

- Clear modeling

  → clear interpretations

- Make implicit assumptions explicit

  → Understand what AI does (not)

- Relaxing these assumptions

  → More “reality”, more “human-like”

AI as Structural Estimation

- Economics brings discipline to
  - Ad-hoc data-analysis works
  - Ad-hoc decision-making algorithms
  - Ad-hoc "interpretations" & "explanations"

- "Economics of AI", and "Economics for AI"
  - Think of each AI as someone else's structural-empirical paper...
  - ...and you'll come up with lots of "referee comments".