Artificial Intelligence as Structural Estimation: Economic Interpretations of Deep Blue, Bonanza, and AlphaGo*

Mitsuru Igami†

October 30, 2017

Abstract

Artificial intelligence (AI) has achieved superhuman performance in a growing number of tasks, including the classical games of chess, shogi, and Go, but understanding and explaining AI remain challenging. This paper studies the machine-learning algorithms for developing the game AIs, and provides their structural interpretations. Specifically, chess-playing Deep Blue is a calibrated value function, whereas shogi-playing Bonanza represents an estimated value function via Rust's (1987) nested fixed-point method. AlphaGo's "supervised-learning policy network" is a deep neural network (DNN) version of Hotz and Miller's (1993) conditional choice probability estimates; its "reinforcement-learning value network" is equivalent to Hotz, Miller, Sanders, and Smith's (1994) simulation method for estimating the value function. Their performances suggest DNNs are a useful functional form when the state space is large and data are sparse. Explicitly incorporating strategic interactions and unobserved heterogeneity in the data-generating process would further improve AIs' explicability.

Keywords: Artificial intelligence, Conditional choice probability, Deep neural network, Dynamic game, Dynamic structural model, Simulation estimator.

JEL classifications: A12, C45, C57, C63, C73.

---

*This paper benefited from comments at a seminar at RIKEN Center for Advanced Intelligence Project on August 7, 2017, as well as conversations with Jerry Hausman, Greg Lewis, Robert Miller, Yusuke Narita, John Rust, and Takuo Sugaya.

†Yale Department of Economics and MIT Department of Economics. E-mail: mitsuru.igami@gmail.com.

1 Introduction

Artificial intelligence (AI) has achieved human-like performance in a growing number of tasks, such as visual recognition and natural language processing.[1] The classical games of chess, shogi (Japanese chess), and Go were once thought to be too complicated and intractable for AI, but computer scientists have overcome many of the conceptual as well as practical challenges. In chess, IBM's computer system named Deep Blue defeated Grandmaster Garry Kasparov in 1997. In shogi, a machine learning-based program called Bonanza challenged (and was defeated by) Ryūō champion Akira Watanabe in 2007, but one of its successors (Ponanza) played against Meijin champion Amahiko Satoh and won in 2017. In Go, Google DeepMind developed AlphaGo, a deep learning-based program, which beat the 2-dan European champion Fan Hui in 2015, a 9-dan (highest rank) professional Lee Sedol in 2016, and the world's best player Ke Jie in 2017.

Despite such remarkable achievements, one of the lingering criticisms of AI is its lack of transparency. The internal mechanism seems like a black box to most people, including the human experts of the relevant tasks, which raises concerns about accountability and responsibility when something goes wrong. The desire to understand and explain the functioning of AI is not limited to the scientific community. For example, the US Department of Defense airs its concern that "the effectiveness of these systems is limited by the machine's current inability to explain their decisions and actions to human users," which led it to host the Explainable AI (XAI) program aimed at developing "understandable" and "trustworthy" machine learning.[2] However, the catch for "explainable AI" is that most of the recent progress in AI has relied precisely on the black box-ing of data-analysis procedures. "Deep neural networks (DNNs)" are a flexible model to capture complicated data patterns, with a complicated internal structure of its own. "Reinforcement learning" typically employs a lot of simulations in search for the optimal actions and decisions. Both of these techniques have proved powerful for enhancing AI's performance, but tended to obfuscate its mechanism.

This paper examines three prominent game AIs in recent history, Deep Blue, Bonanza, and AlphaGo. I have chosen to study this category of AIs because board games represent an archetypical task that has required human intelligence, including cognitive skills, decision-making, and problem-solving. They are also well-defined problems for which eco-

[1] The formal definition of AI is contentious, partly because scholars have not agreed on the definition of intelligence in the first place. This paper follows a broad definition of AI as computer systems able to perform tasks that traditionally required human intelligence such as cognitive skills, decision-making, and problem-solving.

[2] See https://www.darpa.mil/program/explainable-artificial-intelligence (accessed on October 17, 2017).

nomic interpretations are more natural than for, say, visual recognition and natural language processing.

A close examination of their development processes and algorithms suggests these AIs' key components are mathematically equivalent to well-known econometric methods to estimate dynamic structural models. Specifically, chess experts and IBM's engineers manually adjusted thousands of parameters in Deep Blue's "evaluation function," which quantifies the probability of eventual winning as a function of the current positions of pieces (i.e., state of the game) and therefore could be interpreted as an approximate value function. *Deep blue is a calibrated value function.* By contrast, Kunihito Hoki, the developer of Bonanza, constructed a dataset of professional shogi games, and used logit regressions with a backward-induction algorithm to determine such parameters. His method for the "supervised learning" of the evaluation function is equivalent to a finite-horizon version of Rust's (1987) nested fixed-point (NFXP) algorithm; hence *Bonanza represents an empirical model of human shogi players' value function that is estimated by a direct (or "full-solution") method.*

Google DeepMind's *AlphaGo embodies an alternative approach to estimating dynamic structural models: two-step estimation.* AlphaGo's first component is called a "policy network" to predict professional Go players' next move as a function of the current board state, which is only a slightly different name for a policy function. The developers used data from the professional Go games and a DNN (i.e., a flexible approximation of an arbitrary function) to construct an empirical mapping from state to choice probabilities, which closely follows Hotz and Miller's (1993) conditional choice probability (CCP) method for estimating dynamic discrete-choice models. Moreover, AlphaGo's second component is a "value network" to evaluate the state of the game, which is constructed by forward simulations using the policy network (i.e., CCP estimates) in the first stage. This procedure is equivalent to the CCP-based conditional choice simulation (CCS) estimator, proposed by Hotz, Miller, Sanders, and Smith (1994) for single-agent dynamic programming (DP) models, and Bajari, Benkard, and Levin (2007) for dynamic games.

Thus, these leading game AIs and the core algorithms for their development turn out to be successful applications of the empirical methods to implement dynamic structural models. These findings have three implications for economics as well as computer science. First, the straight-forward interpretation of these AIs (or their development processes, to be precise) as structural estimation suggests economics has an important role to play in the understanding, explanation, and hence diffusion of this new technology. Economists can open some of the black boxes by providing structural interpretations.

Second, the success of these AIs (in terms of achieving certain performance milestones) should be regarded as a proof of concept for dynamic structural models, with some new insights for future research. The curse of dimensionality has been a major bottleneck for the empirical application of these models, because realistic models typically entail large state spaces, which increase computational costs and/or data requirement. The high predictive power of AlphaGo's policy function suggests DNNs could be a useful "functional form" for flexibly estimating high-dimensional CCPs when only sparse data are available (i.e., only a negligible fraction of the state space, such as $2.56 \times 10^{8}/10^{171} = 2.56/10^{163} \approx 0$, is actually observed in the data).

Third, the equivalence between the structural econometric methods and the algorithms to develop these AIs means well-known econometric issues for the estimation of dynamic games, such as strategic interactions and unobserved heterogeneity, would need to be addressed. Explicitly incorporating these fundamental features of the real data-generating processes could further improve the transparency and the interpretability of AIs. Fortunately, the last decade has witnessed rapid methodological progress on this front, including Kasahara and Shimotsu (2009), Arcidiacono and Miller (2011), and Berry and Compiani (2017).

Literature

This paper establishes the equivalence between some of the algorithms for developing game AI and the aforementioned econometric methods for estimating dynamic models. As such, the most closely related papers are Rust (1987), Hotz and Miller (1993), and Hotz, Miller, Sanders, and Smith (1994). The game AIs I analyze in this paper are probably the most successful (or at least the most popular) empirical applications of these methods.

At a higher level, the purpose of this paper is to clarify the connections between some of the recent advance in computer science-based data analysis (i.e., machine learning) and the more traditional econometric models and methods, both of which share probability theory and statistics as the underlying technology. As such, the paper shares the spirit of this rapidly growing literature. For surveys, see Belloni, Chernozhukov, and Hansen (2014); Varian (2014); Athey (2017); and Mullainathan and Spiess (2017), among others.

The paper is also potentially relevant to more recently proposed methods for estimating dynamic structural models. My investigation into the data-generating process (i.e., how human experts prepare for and play these games) suggests the presence of classical econometric problems such as unobserved heterogeneity and identification issues due to multiple equilibria, although the developers of game AIs do not seem to pay much attention to them. Addressing them would require the application of new methods, including Kasahara and

Shimotsu (2009), Arcidiacono and Miller (2011), and Berry and Compiani (2017).

Finally, the successful approximation of unknown functions (policy and value) in the massively high-dimensional state space of Go (with only very small/sparse data) would appear to be the most important achievement of AlphaGo for the applied econometric research on dynamic models. The development team applied a version of multi-layer feed-forward (a.k.a. DNN) models, which have been known to be capable of approximating arbitrary functions (c.f., Hornik, Stinchcombe, and White [1989]) and recently started generating useful applications. For an overview, see Goodfellow, Bengio, and Courville (2016), among others.

2 Model

This section introduces basic notations to describe a dynamic game. The goal is to establish a common mathematical ground for comparing the algorithms (for developing game AIs) and the econometric methods (for estimating dynamic structural models).

Setup

Chess, shogi, and Go belong to the same class of games, with two players ($i = 1, 2$), discrete time ($t = 1, 2, ...$), alternating moves (players 1 and 2 choose their actions, a_t, in odd and even periods, respectively), perfect information, and deterministic state transition,

$$s_{t+1} = f\left(s_t, a_t\right), \tag{1}$$

where both the transition, $f(\cdot)$, and the initial state, s_1, are completely determined by the rule of each game. Action space is finite and is defined by the rule as "legal moves," $a_t \in \mathcal{A}\left(s_t\right)$.

State space is finite as well, and consists of four mutually exclusive subsets:

$$s_t \in \mathcal{S} = \mathcal{S}_{cont} \cup \mathcal{S}_{win} \cup \mathcal{S}_{lose} \cup \mathcal{S}_{draw}. \tag{2}$$

The game continues as long as $s_t \in \mathcal{S}_{cont}$. The payoffs sum to zero:

$$u_1\left(s_t\right) = \begin{cases} 1 & \text{if } s_t \in \mathcal{S}_{win}, \\ -1 & \text{if } s_t \in \mathcal{S}_{lose}, \text{ and} \\ 0 & \text{otherwise,} \end{cases} \tag{3}$$

with $u_2(s_t)$ defined symmetrically. That is, player 1 wins when player 2 loses, and vice versa. For notational simplicity, I use $\mathcal{S}_{win}$ (or $\mathcal{S}_{lose}$) to mean player 1's win (or loss). This setup means chess, shogi, and Go are well-defined finite games. In principle, such games can be solved exactly and completely by backward induction from the terminal states.

In practice, even today's supercomputers and a cloud of servers cannot hope to solve them within our lifetime, because the size of the state space, $|\mathcal{S}|$, is large. The approximate $|\mathcal{S}|$ of chess, shogi, and Go are 10^{47}, 10^{71}, and 10^{171}, respectively, which are comparable to the number of atoms in the observable universe ($10^{78} \sim 10^{82}$) and certainly larger than the total information-storage capacity of humanity (in the order of 10^{20} bytes).[3] Given the size of $\mathcal{S}$ that exceeds our civilization's aggregate capacity, the fact that individual human professionals can play these games seemingly intelligently is remarkable.

How Human Experts Play

Human professionals approach these games with the following heuristics. First, the player forms some expectations about the opponent's strategy, σ_{-i}, which is a probabilistic policy function that maps states to likely moves. Human experts closely study each other's play history to form such beliefs (c.f., Kasparov [2007], Watanabe [2013, 2014]).

Second, the player holds some belieff about the likelihood of eventual winning from each (current) state s_t,

$$\Pr_{win}(s_t, \sigma_{-i}) \equiv \Pr\{s_\infty \in \mathcal{S}_{win} | s_t, \sigma_{-i}\}, \tag{4}$$

where s_∞ represents the eventual state. Of course, even computers cannot calculate or store this information exactly for each specific s_t. Human players develop and refine an approximate, subjective version of such board-state evaluation functions through their own research as well as personal histories of games. Developing these two beliefs, σ_{-i} and $\Pr_{win}(s_t, \sigma_{-i})$, is time-consuming. The player prepares and polish them in advance, prior to the actual match, although some updating may occur during the game.

Third, during the game, the player takes σ_{-i} and $\Pr_{win}(s_t, \sigma_{-i})$ as given, and chooses actions to maximize the probability of winning under the time constraint (typically a few hours of thinking time are allotted to each player per game). Given the large $|\mathcal{S}|$ and the time constraint, the player cannot think through the endgame and instead tries to look forward for a limited number of moves, $L < \infty$, based on σ_{-i} , and chooses his/her move a_t

[3] In 2016, the world's hard disk drive (HDD) industry produced the total of 693 exabytes (EB), or 6.93×10^{20} bytes.

to maximize the expected continuation value, $\Pr_{win}(s_{t+L}, \sigma_{-i})$, by backward induction.

Dynamic Discrete Choice Formulation for Empirical Implementation

Translating such heuristics to the dynamic discrete-choice framework is straight-forward. Let $V_{ijt}(s_t)$ denote player i's expected value function conditional on choosing legal action j (i.e., $a_t = a_j \in \mathcal{A}(s_t)$) in state s_t:

$$V_{ijt}(s_t; \sigma_{-i}) = E\left[u_i(s_\infty) | s_t, a_t = a_j; \sigma_{-i}\right]. \tag{5}$$

The player's optimal strategy, or policy function, is characterized by:

$$a_{it}(s_t; \sigma_{-i}) = \arg \max_{a_j \in \mathcal{A}(s_t)} \left\{V_{ijt}(s_t; \sigma_{-i})\right\}. \tag{6}$$

The game's environment is stationary; hence, the truly optimal value function would not require any further forward-looking/backward-induction calculations (e.g., for L moves ahead). However, the only practically available value function is an approximate and subjective one,

$$V_i(s_t; \sigma_{-i}) \approx V^*(s_t; \sigma_{-i}), \tag{7}$$

and the endgame does require precise backward induction to find checkmates (i.e., the terminal actions). Hence, human experts (as well as computer programs) try to reduce the approximation error and find checkmates by thinking $L < \infty$ moves ahead of the current turn under the time constraint. We may explicitly incorporate this aspect of actual play by adding L as an extra argument in (5) to clarify that the expectations is taken under the time/computational constraint:

$$V_{ijt}(s_t; \sigma_{-i}, L) = E\left[u_i(s_\infty) | s_t, a_t = a_j; \sigma_{-i}, L\right], \tag{8}$$

although I will suppress this notation (for simplicity) in most of the following exposition.[4]

In the data from human players' games, different players choose different moves at different times, despite being in the same state (i.e., $a_{it}(s_t) \neq a_{i't}(s_{t'})$ in general even if $s_t = s_{t'}$). Such heterogeneity could arise due to heterogeneous abilities, beliefs, and constraints, among

[4] If one wants to model the underlying structure of L more precisely, one would need to consider the optimal time allocation problem within each game, which is beyond the scope of this paper (or any of the AIs under investigation).

many other factors that would affect human behavior in reality. Official record of games typically contains only the players' identities and moves. Hence an empirical model would need an error term, ε_{ijt}, to accommodate such unexplained variation in the data:

$$a_{it}\left(s_t, \varepsilon_{it}; \sigma_{-i}\right) = \arg \max_{a_j \in \mathcal{A}(s_t)} \left\{ V_{ijt}\left(s_t; \sigma_{-i}\right) + \varepsilon_{ijt} \right\}, \tag{9}$$

where ε_{it} is a vector of ε_{ijt}'s across all legal moves, and functions as another state variable that represents the collection of unobserved "random" components of payoffs. A conventional approach is to assume ε_{ijt} follows the type-1 extreme value distribution, independently and identically across i, j, and t.[5]

3 Algorithms

This section uses the notations I introduced in the previous section to describe the algorithms that are used either in the development process of the three AIs or in their actual play of the game. This section's exposition frequently uses the terminology from computer chess, shogi, and Go, with only limited translation to the language of economics, because the section's main purpose is description rather than explanation. By contrast, the next section focuses on providing economic translation of the programming jargons as well as structural interpretations of these algorithms, with clear connections to the economics literature.

3.1 Chess: Deep Blue

IBM's Deep Blue is a whole computer system with custom-built hardware components as well as software components. This paper focuses on the latter, programming-related part. Deep Blue's program consists of three key elements: an evaluation function, a search algorithm, and databases.

Evaluation Function

The evaluation function of Deep Blue is a parametric value function to measure the probability of eventual winning, $\Pr_{win}$, or its monotonic transformation, $g\left(\Pr_{win}\right)$, in each state

[5] The presence of persistent or other systematic unobserved heterogeneity in the data-generating process (DGP) would violate this assumption, such as player-level heterogeneity, match-specific elements, or a secular growth of knowledge about the game in the community of players. I will come back to this point toward the end of the paper.

of the board, s_t:

$$
\begin{aligned}
V_{DB}(s_t) &= g\left(\Pr_{win}(s_t);\theta\right) \\
&= \theta_1 x_{1t} + \theta_2 x_{2t} + \cdots + \theta_K x_{Kt},
\end{aligned} \tag{10}
$$

where $\theta \equiv (\theta_1, \theta_2, \ldots, \theta_K)$ is a vector of K parameters and $x_t \equiv (x_{1t}, x_{2t}, \ldots, x_{Kt})$ is a vector of K observable characteristics of s_t.

A typical evaluation function for computer chess considers "material value" associated with each type of pieces, such as 1 point for a pawn, 3 points for a knight, 3 points for a bishop, 5 points for a rook, 9 points for a queen, and an arbitrarily many points for a king (e.g., 200 or 1 billion), because the game ends when a king is killed. Other factors include the relative positions of these pieces, such as pawn structure, protection of kings, and the fact that a pair of bishops are usually worth more than the sum of their individual material values. Finally, the importance of these factors may change depending on the phase of the game: opening, middle, or endgame.

Reasonable parameterization, $g(\cdot;\theta)$, and the choice of characteristics (variables), x_t, require expert knowledge. Multiple Grandmasters (the highest rank of chess players) advised the Deep Blue development team. Importantly, they did not use any statistical methods or data from the professional games. Each of the several thousand parameters, θ, was *manually adjusted* until the program's performance reached a satisfactory level.

Search Algorithm

The second component of Deep Blue is "search," or a solution algorithm to choose the optimal action at each turn to move. In the language of computer chess programming, the "full-width search" procedure works as follows: the program evaluates every possible position for a fixed number of future moves along the game tree, using the "minimax algorithm" and some "pruning" methods. In the language of economics, this "search" is a version of numerical backward induction. I will provide more precise translation in section 4.

Databases

The two databases of Deep Blue are for the endgame and the opening phases of chess, respectively. The endgame database embodies the cumulative efforts by the computer chess community to solve the game in an exact manner. The currrent state of the art is the Nalimov database, which covers all five-piece endings (i.e., the states with only five pieces, and all possible future states that can be reached from them) for 7.05 GB of hard disk space, and all six-piece endings for 1.2 terabytes (TB). Deep Blue used some earlier version of such endgame databases.

The second database is "opening books," which is a collection of common openings (i.e., move patterns at the beginning of the game) that are considered good play by experts. It also contains good ways to counter the opponent's poor openings, again based on the judgment by experts.

Performance

Deep Blue defeated the top-ranked Grandmaster Garry Kasparov in 1997. Since then, the use of computer programs have become wide-spread in terms of both training and games (e.g., those played by hybrid teams of humans and computers).

3.2 Shogi: Bonanza

In 2005, Kunihito Hoki, an academic chemist then at the University of Toronto, spent his spare time on developing a shogi-playing program named Bonanza, which won the world championship in computer shogi in 2006. Hoki's Bonanza revolutionized the field of game AI by introducing machine learning to "train" (i.e., estimate) a more flexible evaluation function than either those for chess or those designed for the existing shogi programs.

More Complicated Evaluation Function

The developers of chess programs manually adjusted thousands of parameters in the evaluation function, as described in the previous subsection, and could beat the human champions. The same approach had not produced any comparable performance in shogi. The computer programs before Bonanza could compete at amateur players' level at best. This performance gap between chess and shogi AIs is rooted in the fact that shogi is more complicated and intractable than chess, with $|\mathcal{S}_{shogi}| \approx 10^{71} > 10^{47} \approx |\mathcal{S}_{chess}|$.

Several factors contribute to the increased complexity of shogi: a larger board size ($9\times9 > 8 \times 8$), more pieces ($40 > 32$), more types of pieces ($8 > 6$), most of the pieces have limited mobility,[6] and the fact that pieces other than kings never die. This last feature is

[6] Four of the eight types of pieces in shogi can move only one unit distance at a time, whereas only two out of the six types of pieces in chess (pawn and king) have such low mobility. The exact positions of pieces becomes more important in characterizing the state space when mobility is low, whereas high mobility makes pure "material values" relatively more important because pieces can be moved to wherever they are needed within a few turns.

particularly troublesome for any attempt to solve the game exactly. Non-king pieces are simply "captured," not killed, and can then be "dropped," (re-deployed on the capturer's side) almost anywhere on the board, as a legal move at any of the capturer's subsequent turns.

Hoki designed a flexible evaluation function by factorizing the positions of pieces into (i) the positions of any three pieces including the kings and (ii) the positions of any three pieces including only one king. This granular yet tractable characterization turned out to capture important features of the board through the relative positions of three-piece combinations. Bonanza's evaluation function, $V_{BO}\left(s_t;\theta_{BO}\right)$, also incorporated other, more conventional characteristics, such as individual pieces' material values (see Hoki and Watanabe [2007], pp. 119–120, for details). As a result, $V_{BO}\left(s_t;\theta_{BO}\right)$ contains 50 million variables and the same number of parameters (Hoki [2012]).

Machine Learning (Logit Regression)

That the Deep Blue team managed to adjust thousands of parameters for the chess program by human hands is almost incredible. But the task becomes simply impossible with 50 million parameters. Hoki gave up on manually tuning Bonanza's 50 million parameters, θ_{BO}, and chose to rely on statistical methods to automatically adjust them based on the data from the professional shogi players' 50,000 games on official record: supervised machine learning.

Each game takes 100 moves on average. Hence, the 50,000 games contain approximately 5 million pairs of (a_t, s_t). Hoki used additional data from 50,000 unofficial, online game record as well, to cover some rare states such as nyuu-gyoku positions (in which a king enters the opponent's territory and becomes difficult to capture, because the majority of shogi pieces can only move forward, not backward).

Like Deep Blue, Bonanza chooses its move at each of its turn t by searching for the action a_t that maximizes V_{BO} in some future turn $t+L$, assuming that the opponent follows the same strategy as itself (i.e., $\sigma_{-i}=\sigma_{BO}$):

$$a_{BO,t}\left(s_t;\theta_{BO},L,\sigma_{-i}=\sigma_{BO}\right)=\arg\max_{a_j\in\mathcal{A}(s_t)}\left\{V_{BO,j,t}\left(s_t;\theta_{BO},L,\sigma_{-i}=\sigma_{BO}\right)\right\}, \tag{11}$$

where I made the dependence of the policy and the value on (L,σ_{-i}) explicit. The conditional values are:

$$V_{BO,j,t}\left(s_t;\theta_{BO},L,\sigma_{-i}=\sigma_{BO}\right)=E\left[u_{BO}\left(s_\infty\right)|s_t,a_{BO,t}=a_j;\theta_{BO},L,\sigma_{-i}=\sigma_{BO}\right]. \tag{12}$$

The eventual probability of winning is approximated by the parametric evaluation function:

$$
\begin{aligned}
V_{BO}\left(s_t;\theta_{BO}\right) &= \theta_1 x_{1t} + \theta_2 x_{2t} + \cdots + \theta_K x_{Kt} \\
&\approx u_{BO}\left(s_\infty\right).
\end{aligned}
\tag{13}
$$

Thus, parameter θ_{BO} uniquely determines the optimal move $a^*_{BO,t}$ (given L) unless there are ties between multiple actions j and j'.

By the same token, the observed action-state pairs (a_t, s_t) are informative about the human players' true, underlying θ_0. This is the implicit identification strategy behind Hoki's data analysis, the immediate goal of which is to make Bonanza predict (fit) the professional players' actual moves in the data.

Hoki used logit regressions to determine the values of θ_{BO} automatically, in combination with the chess-style "full-width search." The use of logit regression implicitly assumes the addition of the error term, ε_{ijt}, that follows i.i.d. type-1 extreme value, as shown in equation (9). This continuous random variable eliminates the possibility of ties between multiple actions j and j', thereby making the mapping between θ_{BO} and σ_{BO} (i.e., the optimal choice probabilities = the policy function) unique.

Performance

Bonanza won the world championship in computer shogi in 2006 and 2013. In 2007, the Ryūō ("dragon king," one of the two most prestigious titles) champion Akira Watanabe agreed to play against Bonanza and won. After the game, however, he said he regretted agreeing to play against it because he felt he could have lost with non-negligible probabilities. Hoki made the source code publicly available. The use of data and machine learning for computer shogi was dubbed the "Bonanza method" and copied by most of the subsequent generations of shogi programs.

Issei Yamamoto, a programmer, named his software Ponanza out of respect for the predecessor, claiming his was a lesser copy of Bonanza. From 2014, Ponanza started playing against itself in an attempt to find "stronger" parameter configurations than those obtained (estimated) from the professional players' data: reinforcement learning (Yamamoto [2017]). Eventually, Ponanza became the first shogi AI to beat the Meijin ("master," the other most prestigious title) champion in 2017, when Amahiko Satoh lost two straight games.

3.3 Go: AlphaGo

The developers of AIs for chess and shogi had successfully parameterized the state spaces of these games and found reasonable parameter values for the respective evaluation functions, $V_{DB}\left(s_t; \hat{\theta}_{DB}\right)$ and $V_{BO}\left(s_t; \hat{\theta}_{BO}\right)$. Meanwhile, the developers of computer Go struggled to find any reasonable parametric representation of the board.

Go is even more complicated than either chess or shogi, with $|\mathcal{S}_{go}| \approx 10^{171} > 10^{71} \approx |\mathcal{S}_{shogi}|$. Go has only one type of piece, a stone, and the goal is simply to occupy larger territories than the opponent when the board is full of black and white stones (for players 1 and 2, respectively). However, the 19×19 board size is much larger, and so is the action space. Practically all open spaces constitute legal moves. The local positions of stones seamlessly interact with the global ones. Even the professional players cannot articulate what distinguishes good positions from bad ones, frequently using phrases that are ambiguous and difficult to codify. The construction of a useful evaluation function was deemed impossible.

Instead, most of the advance since 2006 had been focused on improving the game-tree search algorithms (Yoshizoe and Yamashita [2012], Otsuki [2017]). Even though the board states in the middle of the game is difficult to codify, the terminal states are unambiguous, with either win or loss. Moreover, a "move" in Go does not involve moving pieces that are already present on the board; it comprises of simply dropping a stone on an open space from outside the board. These features of Go make randomized "play-out" easy. That is, the programmer can run Monte Carlo simulations in which black and white stones are alternatingly dropped on random places until the board is filled and the winner is determined. Repeat this forward simulation many times, and one can calculate the probability of winning, $\Pr_{win}$, from any arbitrary state of the board, s_t. This is the basic idea of a method called Monte Carlo tree search (MCTS).

Of course there are 10^{171} states in Go; hence, calculating an approximate $\Pr_{win}(s_t)$ for all s_t's remains impossible. However, a program can use this approach in real time to play Go, because it needs to compare only $|\mathcal{A}(s_t)| < 361 = 19 \times 19$ alternative actions and their resulting states at its turn to move. Forward simulations involve many calculations, but each operation is simple and parallelizable. That is, computing one history of future play does not rely on computing another history. Likewise, simulations that start from a particular state $s_{t+1} = f(s_t, a_j)$ does not have to wait for other simulations that start from $s'_{t+1} = f(s_t, a_{j'})$, where $j \neq j'$. Such computational tasks can be performed simultaneously on multiple computers, processors, cores, or GPU (graphic processing unit). If the developer can use many computers during the game, the MCTS-based program can perform sufficiently many numerical operations to find good moves in a short time.

This was the state of computer Go programming when Demis Hassabis and his team at Google DeepMind proposed a deep learning-based AI, AlphaGo. The four pillars of AlphaGo are (i) a policy network, (ii) a value network, (iii) reinforcement learning, and (iv) MCTS. The first two are the most novel components in the context of existing game AIs.

Supervised Learning (SL) of Policy Network

The first component of AlphaGo is "policy network," which is a deep neural network (DNN) model to predict strong professional players' move a_t as a function of current state s_t. In other words, it is a policy function, $\sigma(s_t; \theta_{AG})$, with a particular specification and 4.6 million "weights" (i.e., parameters θ_{AG}).

Like Hoki did for Bonanza, the AlphaGo team determined θ_{AG} by using the data from an online Go website named Kiseido Go Server (KGS). Specifically, they used the KGS record on 160,000 games played by high-level (6-9 dan) professionals. A game lasts for 200 moves on average, and eight symmetric transformations (i.e., rotations and flipping) of the board generate formally different states. Hence, the effective size of the data is:

$$\begin{aligned} 256 \text{ million (action-state pairs)} &= 160,000 \text{ (games)} \times 200 \text{ (moves/game)} \\ &\quad \times 8 \text{ (symmetric transformations)}. \end{aligned} \tag{14}$$

Note the sample size is still small (negligible) relative to $|\mathcal{S}_{go}| \approx 10^{171}$.

The specification of the model consists of 48 input "channels" (variables), 13 "layers" (stages within a hierarchical architecture), and 192 "kernels" (filters to find local patterns). A complete review of deep neural networks in general (or AlphaGo's model specification in particular) is beyond the scope of this paper, but these objects interact as follows. Each of the 48 channels represents a binary indicator variable that characterizes s_t,

$$x_{kt} = \begin{cases} 1 & \text{if feature } k \text{ is present in } s_t, \text{ and} \\ 0 & \text{otherwise.} \end{cases}, \tag{15}$$

"Features" include the positions of black stones, white stones, and blanks (see Extended Data Table 2 of Silver et al [2016] for the full list).

These x_{kt}'s are not combined linearly (as in V_{DB} or V_{BO}) but processed by many kernels across multiple hierarchical layers. In the first layer, each of the 192 kernels is a 5×5 grid with 25 parameters that responds to a particular pattern within 25 adjacent locations. As the name “kernel” suggests, this 5×5 matrix is applied to perform convolution operations at every one of the 225 ($= 15 \times 15$) locations within the 19×19 board:

$$z_{r,c} = \sum_{l=1}^{192} \sum_{p=1}^{5} \sum_{q=1}^{5} w_{l,p,q} \times x_{l,r+p,c+q} + b, \tag{16}$$

where $z_{r,c}$ is the result of convolution for row r and column c, $w_{l,r,q}$ is the weight for kernel l, row r, and column c, (p, q) denote the row and column of the kernel, $x_{l,r+p,c+q}$ is an input, and b is the intercept term (“bias”). The weights and intercepts constitute the parameters of the model. DNNs of this type is called convolutional neural networks (CNNs) in the machine learning literature, and is primarily used for image-recognition tasks. The results of convolution are subsequently transformed by a function,

$$y_{r,c} = \max \{0, z_{r,c}\}, \tag{17}$$

where $y_{r,c}$ is the transformed output (to be passed on to the next layer as input). This function is called rectified linear unit (or “ReLU”). The resulting 15×15 output is smaller than the 19×19 board; the margins are filled by zeros to preserve the 19×19 dimensionality (“zero padding”).

In the second layer, another set of 192 kernels is used to perform convolution on the outputs from the first layer. The results go through the ReLU transformation again, and proceed to the third layer. The size of the kernels in layers 2 through 12 is 3×3, instead of 5×5 in layer 1. In layer 13, the size of the layer is 1×1, because the goal of the policy network is to put a number on each of the 19×19 board positions. Each of the 19×19 outputs in this last layer goes through a logit-style monotonic (“softmax”) transformation into the $[0, 1]$ interval,

$$CCP_{r,c} = \frac{\exp(y_{r,c})}{\sum_{r'} \sum_{c'} \exp(y_{r',c'})}, \tag{18}$$

so that the final output can be interpreted as the players’ conditional choice probabilities of choosing action j (or board location (r, c)).

The above is the description of the DNN (CNN) specification of the policy function. It is “deep” in the sense that the model contains multiple layers. It is named “neural network” because small units of numerical operations are passing along inputs and outputs in a network-like architecture, with the analogy of computational nodes as biological neurons that transmit electric signals.

In one of the foundational works for deep learning, econometrician Halbert White and his coauthors proved that such a multi-layer model with sufficiently many nodes can approximate any arbitrary functions (Hornik, Stinchcombe, and White [1989]). In practice, such a "nonparametric" model is implemented with a finite number of parameters. The approximate number of parameters in AlphaGo's policy network is:

$$4.6 \text{ million (weights)} = (192 \text{ kernels})^2 \times \left(5^2 + 3^2 \times 11 + 1^2\right), \tag{19}$$

which turns out to be smaller than the 50 million parameters in Bonanza, despite the fact that Go has a larger state space than shogi.

The supervised learning (i.e., estimation) of θ_{AG} uses a standard numerical optimization algorithm to maximize the likelihood function that aggregates the optimal choice probabilities implied by the model, the data, and the parameter values. That is, AlphaGo's policy function is estimated by the classical maximum likelihood method. The team did not add any "regularization" term in the objective function, which is a common practice in machine learning to improve the out-of-sample prediction accuracy at the expense of biased estimates. Nevertheless, the trained (i.e., estimated) policy network, $\sigma\left(s_t; \hat{\theta}_{AG}\right)$, could predict 55.7% of the human players' moves outside the sample, and its top-five move predictions contained the actual human choices in almost 90% of the times.[7]

Reinforcement Learning (RL) of Policy Network

The ultimate goal of the AlphaGo team was the creation of a strong AI, not the prediction of human play *per se* (or the unbiased estimation of θ_0). The second ingredient of AlphaGo is the process of reinforcement learning to make a stronger policy function than the estimated one from the previous step, $\sigma\left(s_t; \hat{\theta}_{AG}\right)$.

Reinforcement learning is a generic term to describe a numerical search for "better" parameter values based on some performance criteria, or "reward," such as the score of the game, $u(s_t)$. The specific task in the current case is to find some $\tilde{\theta}_{AG} \neq \hat{\theta}_{AG}$ such that the

[7] By contrast, a simple parametric (logit) version of the empirical policy function (for the MCTS purposes) achieved only 27% accuracy, which is still remarkable but less impressive than the DNN version's performance.

winning probability is higher under strategy $\sigma\left(s_t;\tilde{\theta}_{AG}\right)$ than $\sigma\left(s_t;\hat{\theta}_{AG}\right)$, or equivalently,

$$\begin{aligned} & V\left(s_t;\sigma_i=\sigma\left(s_t;\tilde{\theta}_{AG}\right),\sigma_{-i}=\sigma\left(s_t;\hat{\theta}_{AG}\right)\right) \\ > \; & V\left(s_t;\sigma_i=\sigma\left(s_t;\hat{\theta}_{AG}\right),\sigma_{-i}=\sigma\left(s_t;\hat{\theta}_{AG}\right)\right) \end{aligned} \tag{20}$$

in the majority of the relevant s_t.

Of course, the outcome of the game depends on both σ_i and σ_{-i}. The condition (20) does not guarantee the superiority of $\sigma\left(s_t;\tilde{\theta}_{AG}\right)$ over any strategy other than $\sigma\left(s_t;\hat{\theta}_{AG}\right)$. The only satisfactory way to address this issue is to solve the game exactly and completely for the optimal strategy $\sigma^*(s_t)$, but that is computationally impossible. The AlphaGo team tries to find "satisficing" $\tilde{\theta}_{AG}$, by making each candidate policy play against many different policies that are randomly sampled from the previous rounds of iteration (i.e., various perturbed versions of $\hat{\theta}_{AG}$ in the numerical search process), and by simulating plays from a wide variety of s_t that are also randomly sampled from those in the data (as well as from perturbed versions of such historical games).

Ultimately, one cannot prove $\sigma\left(s_t;\tilde{\theta}_{AG}\right)=\sigma^*(s_t)$, or assess which of the alternative policies, $\sigma\left(s_t;\tilde{\theta}_{AG}\right)$ and $\sigma\left(s_t;\hat{\theta}_{AG}\right)$, is "closer" to $\sigma^*(s_t)$ in any formal sense, but the development team organized their search procedure carefully. Their approach is consistent with the goal of making a strong AI that can beat the top human players. After all, being "stronger" usually means "winning more often" in natural languages, even if it cannot be proven or defined more formally.

Supervised/Reinforcement Learning (SL/RL) of Value Network

The third ingredient of AlphaGo is the evaluation function, $V_{AG}(s_t)$, to assess the probability of winning from any state s_t. Constructing such an object had been deemed impossible in the computer Go community, but the team managed to estimate the value function from the policy function through simulations.

Their procedure is the following. First, simulate many game plays between the RL policy function, $\sigma\left(s_t;\tilde{\theta}_{AG}\right)$, and itself. Second, pick many (30 million) different states from separate games in the simulation and record their winners, which generates a synthetic dataset of 30 million $(\Pr_{win}, s_t)$ pairs. Third, use this dataset to find the value function that predicts $\Pr_{win}$ from any s_t. In other words, the strategy $\sigma\left(s_t;\tilde{\theta}_{AG}\right)$ implies certain outcomes of the game, and these outcomes become explicit through simulations. Once the outcomes become explicit, the only remaining task is to fit some functional form to predict $\Pr_{win}$ as a function of s_t: plain-vanilla supervised learning.

DeepMind's functional form of choice is another DNN (CNN) with a design that is similar to the policy network: 49 channels, 15 layers, and 192 kernels. The only differences are an additional state variable that reflects the identity of the player (to attribute the win/loss outcome to the correct player) and an additional computational step at the end of the hierarchical architecture that takes all arrays of intermediate results as inputs and returns a scalar ($\Pr_{win}$) as an output.

Let us denote the estimated value network by

$$V\left(s_t; \hat{\psi}_{AG}, \sigma_i = \sigma_{-i} = \sigma\left(s_t; \tilde{\theta}_{AG}\right)\right), \tag{21}$$

where $\hat{\psi}_{AG}$ represents a vector of estimated parameters (another set of millions of weights inside the CNN). The expression $\sigma_i = \sigma_{-i} = \sigma\left(s_t; \tilde{\theta}_{AG}\right)$ clarifies the dependence of $V(\cdot)$ on the use of the RL strategy $\sigma\left(s_t; \tilde{\theta}_{AG}\right)$ for both the focal player's play and the opponent's during the simulation step to calculate $\Pr_{win}$. These notations are lengthy and cumbersome, but help us keep track of the exact nature of the estimated value network when we proceed to the structural interpretation of these algorithms in the next section.

Combining Policy and Value with MCTS

The fourth component of AlphaGo is MCTS, the stochastic solution method for a large game tree. When AlphaGo plays actual games, it combines the RL policy, $\sigma\left(s_t; \tilde{\theta}_{AG}\right)$, and the RL value (equation 21) within an MCTS algorithm, to achieve the best performance.

Each of these components can be used individually, or in any combinations (Silver et al [2016], Figure 4). The policy function directly proposes the optimal move from any given state. The value function indirectly suggests the optimal move by comparing the winning probabilities across the next states that result from candidate moves. An MCTS can perform a similar state-evaluation task by simulating the outcomes of the candidate moves. They represent more or less the same concept: approximate solutions to an intractable problem.

Nevertheless, the positive ensemble effect of multiple methods is frequently reported, because different types of numerical approximation errors may cancel out each other. This ensemble part involves many implementation details of purely computational tasks, and hence beyond the scope of this paper.

4 Structural Interpretations

This section explains how the concepts and algorithms for the development of the three AIs in the previous section correspond to more familiar ideas and methods in economics. First, Deep Blue is a calibrated value function. Second, the "Bonanza method" is mathematically equivalent to Rust's (1987) nested fixed-point algorithm. Third, AlphaGo's "SL policy network" is a version of Hotz and Miller's (1993) nonparametric CCP estimator, and its "SL/RL value network" is a straight-forward application of Hotz, Miller, Sanders, and Smith's (1994) simulation estimator. Finally, we may interpret AlphaGo's "RL policy network" as a counterfactual experiment scenario in which the professional Go players lived for a long time to accumulate a lot of experience and improved their strategies.

4.1 Deep Blue is Calibration

The fact that IBM manually adjusted the parameter, θ_{DB}, via trials and errors means Deep Blue is a fruit of painstaking efforts to "calibrate" a value function with thousands of parameters: Deep Blue is a calibrated value function.

As I briefly explained in the model section, a truly optimal value function would obviate the need for any forward-looking and backward-induction procedures to solve the game, because the true value function embodies such solution already. However, any computable, parametric value function is merely an attempt to approximate the optimal one, which is why the use of solution algorithms can improve the strength of computer programs.

In the language of economics, the "full-width search" procedure is a brute-force numerical search for the optimal choice (equation 6) by backward induction on a truncated version of the game tree (truncated at length L from the current turn t, with some clever shortcuts to save computational costs), where the terminal values at turn $(t + L)$ is given by the parametric value function (10) and the opponent is assumed to share the same value function as well as the symmetric strategy (i.e., $\sigma_{-i} = \sigma_i = \sigma$). Once we translate the jargons into economics, we may spot certain assumptions that was not so obvious at first glance (such as symmetry) and start approaching AIs as an empirical problem.

Thus, Deep Blue's main component, the "evaluation function," is a parametric (linear) function to approximate the winning probability at the end of a truncated game tree of length L,

$$V_{DB}\left(s_t; \theta_{DB}, \sigma_{-i} = \sigma_{DB}, L\right),$$

in which the opponent plays the same strategy ($\sigma_{-i} = \sigma_{DB}$). In other words, Deep Blue is a calibrated, approximate terminal-value function in a game that the program plays against its doppelgäger.

4.2 Bonanza is Harold Zurcher

Bonanza is similar to Deep Blue. Its main component is an approximate terminal-value function, and the "optimal" action is determined by backward induction on a truncated game tree of self play (equation 11). The only difference is the larger number of parameters (50 million), which reflects the complexity of shogi and precludes any hopes for calibration. Instead, Hoki decided to approach the development task as a data-analysis problem, that is, an empirical analysis of the professional shogi players.

Consequently, Bonanza is an empirical model of Japanese shogi players, in the same sense that Rust (1987) is an empirical model of Harold Zurcher, the Wisconsin-Madison city-bus superintendent whose optimal engine-replacement decision was thoroughly analyzed and whose utility function was structurally estimated. This comparison is not merely an impressionistic analogy. The estimation algorithm for Bonanza is mathematically equivalent to Rust (1987).

Rust's (1987) full-solution estimation method consists of two numerical optimization problems that are nested. First, the overall problem is to find the parameter values that make the model's predictions fit the observed discrete actions in the data. Second, the nested sub-routine takes particular parameter values as inputs and solves the model to predict actions. The first part is implemented by the maximum likelihood method (i.e., the fit is evaluated by the proximity between the observed and predicted choice probabilities). The second part is implemented by the value-function iteration, that is, by numerically solving a contraction-mapping problem to find a fixed point, which is guaranteed to exist and is unique under mild regularity conditions. This is why Rust named this algorithm nested fixed-point (NFXP).

Hoki developed Bonanza in almost exactly the same manner. The overall problem is to find the values of θ_{BO} in the approximate value function (13) that make Bonanza predict the human experts' actions in the data. The nested sub-routine is to take a particular θ_{BO} as inputs and numerically search for the optimal action a_t^* by means of backward induction. The first part is implemented by logit regressions, that is, the maximum-likelihood estimation of the discrete-choice model in which the error term is assumed i.i.d. type-1 extreme value. This specification is exactly the same as Rust's implementation.

The second part, the sub-routine that solves the model by backward induction, proceeds on a truncated game tree, whose "leaves" (i.e., terminal values at $t + L$) are given by the approximate value function and the opponent is assumed to play the same strategy as itself ($\sigma_{-i} = \sigma_{BO}$). At first glance, this model setting and procedure might appear slightly different from Rusts' value-function iteration, but it is essentially the same problem. Rust used value-function iteration because his Harold Zurcher is solving a single-agent infinite-horizon dynamic programming (DP) problem. By contrast, Bonanza is solving an L-period finite-horizon DP with an opponent: a dynamic game.

Nevertheless, this difference does not make the principle of its estimation algorithm any different. The L-period finite-horizon setting represents only a practical computational limit (that the entire game is impossible to solve within finite time), not a conceptual difference. The opponent is assumed to be the same version of Bonanza that shares the same value function, which is a reasonable assumption as long as the developer aims at estimating the average θ of all human players.[8]

Igami (2017, 2018), as well as Igami and Uetake (2017), demonstrate how Rust's NFXP naturally extends to games with alternating moves (see the first two papers for deterministic orders of moves; see the third paper for a stochastic order of moves). Shogi, chess, and Go are games with a deterministic order of alternating moves. Moreover, such games can be solved for a unique equilibrium if it features a finite horizon and discrete choice. Thus, given any assumptions on (σ_{-i}, L), the game becomes an effectively single-agent DP. From this augmented-NFXP perspective, Harold Zurcher's game is a special case with $(\sigma_{-i}, L) = (N, \infty)$, where $\sigma_{-i} = N$ indicates Harold is playing against nature (e.g., heavy snow in Wisconsin) and not humans or robots; Bonanza's game is another special case with $(\sigma_{-i}, L) = (\sigma_{BO}, L < \infty)$.

Thus, Bonanza is to Akira Watanabe what Rust (1987) is to Harold Zurcher, in the strict sense of the words.

4.3 AlphaGo is Two-step Estimation

Whereas Deep Blue and Bonanza are linear value functions, AlphaGo's biggest innovation is the use of a non-linear universal approximator to estimate the professional players' policy function, without imposing parametric restrictions ("SL policy network"). This first step is equivalent to the estimation of conditional choice probabilities (CCPs) in Hotz and Miller's

[8] In section 5, I will discuss the possibility and desirability of relaxing this assumption, for a more serious empirical analysis.

(1993) two-step method.

Subsequently, Google DeepMind uses these CCP estimates (or the stronger version, "RL policy network") to simulate many plays and estimate the probability of winning as a function of state, using another universal approximator ("SL/RL value network"). This step is equivalent to the estimation of structural parameters by conditional choice simulations (CCSs) in Hotz, Miller, Sanders, and Smith's (1994, henceforth "HMSS") extension of the original CCP method, to avoid costly matrix inversion.

Because Go is a dynamic game, this algorithm to estimate AlphaGo's value function is also exactly the same as the second-stage forward simulation in Bajari, Benkard, and Levin (2007), which is an application of HMSS (1994) to dynamic games, with an additional assumption that the players' strategies are symmetric. Monte Carlo tree search embodies the same idea, of using forward simulations to calculate the value function.

In the context of estimation and data analysis, reinforcement learning ("RL policy network") would be interpreted as a counterfactual experiment in which the players (initially embodied by "SL policy network") are long-lived and accumulate experiences without a time limit.[9]

SL Policy Network is First-stage CCP Estimates

Just like computer scientists have dealt with the high-dimensional problems of chess, shogi, and Go, the economics and econometrics of dynamic structural models have always faced the "curse of dimensionality" problem in computation. The NFXP method requires the solution of the fully dynamic model, which becomes computationally expensive as the size of the state space increases.

Hotz and Miller (1993) proposed a solution. To the extent that the actual choices in the data reflect the optimal choice probabilities that are conditional on the observed state in the data, we can estimate the policy function directly from the data. This procedure is the estimation of CCPs in their first stage.

They also proved the existence of a one-to-one mapping between the policy function and the value function, so that we can invert the former to estimate the latter. This procedure is implemented by means of matrix inversion in Hotz and Miller's (1993) original method.

In the context of structural econometrics, typical parameters of interest are preferences (e.g., utility and revenue) and technologies (e.g., cost and investment efficiency), which are

[9] In the context of pure reinforcement learning without any initialization based on the data from human experts, econometric interpretations do not exist because no data are involved. This seems to be the case for the latest version of the program, AlphaGo Zero (Silver et al [2017]).

constituents of the value function, rather than the policy function. Hence, the two-step method is an indirect approach to estimate the structural parameters in the second stage, after achieving an intermediate objective of estimating the policy function.

The benefit of this approach is that the procedure does not require solving a fully dynamic model, which is computationally expensive. The cost of this approach is that the requirement for data becomes more demanding. One should avoid imposing parametric assumptions on the first-stage policy function, because the goal is to find the parameters of (the components of) the value function that are implied by the raw data patterns, not by their parametric surrogates. A priori restrictions on the policy function could potentially contradict the optimal solution of the underlying DP.

Thus, being nonparametric and preserving flexible functional forms are crucial for an adequate implementation of the two-step method. In practice, no data are perfect, and certain states never appear in reality, which necessitates some extrapolation and parametric assumptions. Nevertheless, nonparametric estimation is always desirable for first-stage CCPs. This is the sense in which the CCP method is demanding of data. It trades the *computational* curse of dimensionality for the *data* curse of dimensionality.

Given this econometric context, the use of DNN seems a sensible choice of functional form, because Hornik, Stinchcombe, and White (1989) show a multi-layer feed-forward network model is a universal approximator for arbitrary functions, as long as the network is sufficiently large and deep (i.e., has a sufficient degree of flexibility) to capture complicated data patterns. The sole purpose and requirement of Hotz and Miller's first-stage is to capture the actual choice patterns in the data as flexibly as possible. Silver et al (2016) report SL policy network's out-of-sample move-prediction accuracy is 55% (and close to 90% with top-five predictions in Maddison et al [2015]), whereas that of a simple parametric (logit) version is 27%. This level of fit is a remarkable achievement, because the sample size is small (practically zero) relative to the size of the state space.

RL Policy Network is a Counterfactual with Long-lived Players

Whereas SL policy network has a clear connection to econometrics, RL policy network does not, because this procedure involves changing parameter values from the unbiased and consistent CCP estimates, $\hat{\theta}_{AG}$, to something else, $\tilde{\theta}_{AG} \neq \hat{\theta}_{AG}$. Nevertheless, we may still interpret $\tilde{\theta}_{AG}$ as an outcome of some specific counterfactual experiment in which the human players (as represented by $\hat{\theta}_{AG}$) lived long careers and learned better strategies by accumulating additional experiences form many games. This interpretation is possible as long as the initial values are set at $\hat{\theta}_{AG}$ and the parameter updating rules are simple and intuitive.

SL/RL Value Network is Second-stage Simulation Estimation of Value Function

The SL/RL value network is the first successful evaluation function for Go, according to Silver et al (2016), which is a remarkable achievement. The procedure to obtain this value function is a straight-forward application of Hotz, Miller, Sanders, and Smith's (1994) conditional choice simulation (CCS) estimator, combined with another DNN to approximate the complicated relationship between $\Pr_{win}$ and s_t in the high-dimensional state space.

HMSS (1994) proposed an alternative approach to the second step of the Hotz-Miller method. Instead of analytical inversion of a large matrix to calculate an implied value function from the policy function (CCPs), they suggest running many forward simulations by using the CCPs. With sufficiently many simulations, the implied value function and its underlying structural parameters can be estimated, because Hotz and Miller (1993) already proved that the policy-value mapping is one-to-one and the parameters are identified.

Bajari, Benkard, and Levin (2007, henceforth "BBL") extended HMSS (1994) to dynamic games and proposed moment inequality-based estimation approach under the maintained hypothesis that the same equilibrium is played throughout the sample period and across geographical markets. Although AlphaGo is developed for the game of Go and hence obviously related to BBL (2007), the developer team has not been interested in or explicitly incorporated the strategic interactions among multiple human players in the data.

Consequently, the forward simulations to estimate AlphaGo's SL/RL value network, ψ_{AG}, simply assumes symmetry (i.e., $\sigma_{-i} = \sigma_{AG}$), and hence effectively treats the environment as a single-agent problem. In this sense, AlphaGo's value function estimation is an empirical application of HMSS (1994) rather than BBL (2007). If economists were to write a serious empirical paper about Go, we could (and should) incorporate the inherent strategic interactions and player heterogeneity in the data.

Finally, the AlphaGo team estimates the value function implied by the RL policy, $\sigma\left(s_t; \tilde{\theta}_{AG}\right)$, as shown in (21), rather than the one based on SL policy, $\sigma\left(s_t; \hat{\theta}_{AG}\right)$. Hence, the economic interpretation of AlphaGo's (RL) value network is that it represents the board-state evaluation by the hypothetical player who has been trained over centuries or millennia of game play ($\tilde{\theta}_{AG}$), rather than the human players of our time in the data ($\hat{\theta}_{AG}$). In other words, AlphaGo's policy and value functions are those of an (intellectual) great-grandchild of today's human players.

MCTS Blends Harold Zurcher with CCP Estimates

The actual play of AlphaGo is generated by an ensemble of RL policy network, RL value network, and MCTS. Hence, it is a hybrid of the human players' great-grandchild and (the stochastic version of) Harold Zurcher's full-solution machine.

5 Discussions

I conclude this paper by discussing three implications and directions for future research.

Incorporating Strategic Interactions and Unobserved Heterogeneity

First, the empirical methods for dynamic structural models have advanced since the time of Rust (1987), Hotz and Miller (1993), and HMSS (1994) to address fundamental economic and econometric problems of strategic interactions, multiple equilibria, and unobserved heterogeneity.[10] Because the AI developers' immediate goal has been to produce a program that can beat the human champions, such econometric considerations have never surfaced on their research agenda. Now that AlphaGo has achieved their long-time dream of super-human performance, however, they might as well find new research agenda related to those classical econometric problems. We can help.

DNN for Nonparametric CCP Estimation

Second, the use of DNN-style specifications for the first-stage nonparametric estimation of CCPs seems a good idea. Given the sheer size of the state space (10^{171}) and only a small dataset (the effective number of observations is 2.56×10^8, which is virtually zero relative to 10^{171}). This class of model specification ("multi-layer feed-forward network") has long been known to be capable of approximating arbitrary functions (e.g., Hornik, Stinchcombe, and White [1989]). AlphaGo (Maddison et al [2015] and Silver et al [2016] finally offered a proof of concept in the dynamic-game context, which is sufficiently interesting and potentially

[10] BBL (2007); Aguirregabiria and Mira (2007); Pakes, Ostrovsky, and Berry (2007); and Pesendorfer and Schmidt-Dengler (2008) proposed methods for analyzing dynamic games along the lines of the two-step estimation method, whereas recent empirical applications, such as Igami (2017, 2018), Zheng (2016), Yang (2017), and Igami and Uetake (2017), build on the full-solution method.

Kasahara and Shimotsu (2009) propose a method (based on rank conditions of the state transition dynamics) to identify the lower bound of the number of unobserved types that is required to rationalize data patterns. Arcidiacono and Miller (2011) use an expectation-maximization algorithm to estimate CCPs in the presence of such unobserved types. Berry and Compiani (2017) advance an instrumental-variables approach to address unobserved heterogeneity in dynamic games.

relevant for economic research. I would appreciate any econometric research to clarify the properties of the relevant classes of DNN.

Structural Econometrics for "Explainable AI"

Third, the fact that this paper could provide a clear mapping between some of the computer-science algorithms to develop game AIs and the well-known econometric methods to analyze dynamic structural models suggests economics could potentially be helpful in understanding and explaining AIs. The US Department of Justice's "Explainable AI" project and other similar news seem to indicate our society's general interest in unpacking the black box with logic and intuition. Economics is a discipline with the philosophy and technology to explain the complicated black box of reality, and hence is well-positioned to supply what the society seems to demand.

References

[1] Arcidiacono, P., and R. A. Miller. 2011. "Conditional Choice Probability Estimation of Dynamic Discrete Choice Models With Unobserved Heterogeneity." *Econometrica*, 79: 1823–1867.

[2] Athey, Susan. 2017. "Beyond prediction: Using big data for policy problems." *Science*, 355: 483–485.

[3] Bajari, P., C. L. Benkard, and J. Levin. 2007 "Estimating Dynamic Models of Imperfect Competition." *Econometrica*, 75: 1331–1370.

[4] Belloni, A., V. Chernozhukov, and C. Hansen. 2014. "High-Dimensional Methods and Inference on Structural and Treatment Effects." *Journal of Economic Perspectives*, 28: 29–50.

[5] Berry, S. T., and G. Compiani. 2017. "An Instrumental Variable Approach to Dynamic Models." Manuscript, Yale University.

[6] Campbell, M., A. Hoane, and F. Hsu. 2002. "Deep Blue." *Artificial Intelligence*, 134: 57–83.

[7] Goodfellow, I., Y. Bengio, and A. Courville. 2016. *Deep Learning*. Cambridge, MA: The MIT Press.

[8] Hoki, K. 2012. "Kazuno bouryoku de ningen ni chousen! Bonanza no tanjou," in Computer Shogi Association, ed., *Ningen ni katsu computer shogi no tsukuri kata*. Tokyo: Gijutsu hyouron sha.

[9] Hoki, K., and A. Watanabe. 2007. *Bonanza Vs Shoubunou: Saikyou shogi sohuto wa ningen wo koeruka*. Tokyo: Kadokawa (in Japanese).

[10] Hornik, K., M. Stinchcombe, and H. White. 1989. "Multilayer Feedforward Networks are Universal Approximators," *Neural Networks*, 2: 359–366.

[11] Hotz, V. J., and R. A. Miller. 1993. "Conditional Choice Probabilities and the Estimation of Dynamic Models." *Review of Economic Studies*, 60: 497–529.

[12] Hotz, V. J., R. A. Miller, S. Sanders, and J. Smith. 1994. "A Simulation Estimator for Dynamic Models of Discrete Choice." *Review of Economic Studies*, 61: 265–289.

[13] Igami, M.. 2017. "Estimating the Innovator's Dilemma: Structural Analysis of Creative Destruction in the Hard Disk Drive Industry, 1981–1998," *Journal of Political Economy*, 125: 798–847.

[14] Igami, M.. 2018 "Industry Dynamics of Offshoring: The Case of Hard Disk Drives." *American Economic Journal: Microeconomics*, forthcoming.

[15] Igami, M., and K. Uetake. 2017. "Mergers, Innovation, and Entry-Exit Dynamics: Consolidation of the Hard Disk Drive Industry, 1996–2016." Manuscript, Yale University.

[16] Kasahara, H., and K. Shimotsu. 2009. "Nonparametric Identification of Finite Mixture Models of Dynamic Discrete Choices." *Econometrica*, 77: 135–175.

[17] Kasparov, G.. 2007. *How Life Imitates Chess: Making the Right Moves, from the Board to the Boardroom*. London: Bloomsbury.

[18] Maddison, C. J., A. Huang, I. Sutskever, and D. Silver. 2015. "Move Evaluation in Go Using Deep Convolutional Neural Networks." *ICLR*.

[19] Mullainathan, S., and J. Spiess. 2017. "Machine learning: an applied econometric approach." *Journal of Economic Perspectives*, 31: 87–106.

[20] Otsuki, T.. *Saikyou igo AI AlphaGo kaitai shinsho*. Tokyo: Shoeisha (in Japanese).

[21] Pakes, A., Ostrovsky, M., and Berry, S. 2007. "Simple estimators for the parameters of discrete dynamic games (with entry/exit examples)." *RAND Journal of Economics*, 38: 373–399.

[22] Pesendorfer, M., and P. Schmidt-Dengler. 2008. "Asymptotic Least Squares Estimators for Dynamic Games." *Review of Economic Studies*, 75: 901–928.

[23] Rust, J.. 1987. "Optimal Replacement of GMC Bus Engines: An Empirical Model of Harold Zurcher." *Econometrica*, 55: 999–1033.

[24] Silver, D., A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. 2016. "Mastering the game of Go with deep neural networks and tree search." *Nature*, 529: 484–489.

[25] Silver, D., J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis. 2017. "Mastering the game of Go without human knowledge." *Nature*, 550: 354–359.

[26] Varian, Hal. 2014. "Big Data: New Tricks for Econometrics." *Journal of Economic Perspectives*, 28: 3–28.

[27] Watanabe, A.. 2013. *Shōbushin*. Tokyo: Bungei shunju (in Japanese).

[28] Watanabe, A.. 2014. *Watanabe Akira no shikou: Banjou bangai mondou*. Tokyo: Kawade shobou shinsha (in Japanese).

[29] Yamamoto, I.. 2017. *Jinkou chinou wa donoyouni shite "Meijin" wo koetanoka?* Tokyo: Diamond sha (in Japanese).

[30] Yang, Chenyu. 2017. "Could Vertical Integration Increase Innovation?" Manuscript, University of Rochester.

[31] Yoshizoe, K., and H. Yamashita. 2012. *Computer Go: Theory and Practice of Monte Carlo Method* (ed. by H. Matsubara). Tokyo: Kyouritsu shuppan (in Japanese).

[32] Zheng, Fanyin. 2016. "Spatial Competition and Preemptive Entry in the Discount Retail Industry." Manuscript, Columbia University.