Reinforcement Learning

Jonathan Gu *for* Econometrics Field

July 2016

Abstract

This article proves that the current consensus on the update rule used in reinforcement learning does not actually maximize the correct objective. Here we provide the corrected update rule that trains the parameters of a policy function such that the value of every state is maximized.

Contents

1 Motivation

Neural networks are used to solve tasks that are hard to solve by ordinary rule based programming. They are used when analytic functions either do not exist or are infeasible to express. Neural nets are often used to approximate policy functions to determine the best action to take given a current state, or they are used to approximate value functions to determine the expected value of being in a given state.

In most economic models it is common specify either a utility of a welfare function. The goal of agents within the model is to maximize their corresponding function - here we will rename this function to "reward function."

In applied microeconomics researcher commonly use dynamic programming to solve for value functions by back solving from the final time period. In macroeconomics the common practice is to use shooting algorithms. In microeconomic theory researchers tend to specify theoretical models and solve explicitly for some feature of their game. All of the above models specify a reward function, and therefore it is applicable to frame these problems such that neural nets become a viable method for approximating policy functions and value functions.

Given the reward function, the state space, and the possible actions, the next step is to solve for an equilibrium optimal policy function - which tells what the optimum response to a given state is - assuming that all other players are also responding optimally. Here we will refer to equilibrium play and optimal policy as 'perfect play'.

In the application of games, the economist assumes perfect play. Perfect play is the traditional notion of equilibrium where everyone is making the best move possible at every time. In games of immense computational complexity - such as Go or chess - perfect play (or equilibrium play) may not be feasible for the common layman or even for the professional competitor. There are too many possible future states for a human to process all at once.

The extent of the bounded rationality of the human mind is difficult to measure, but there are many games where it may be possible to use reinforcement learning to approximate the optimum value function and the optimum equilibrium moves even when no one knows what such optima are supposed to look like.

When one trains a neural net - one starts off with supervised learning. The neural net observes a dataset labelled with good moves and bad moves. The neural net can accordingly develop a policy function to maximize winning (which will be mathematically formalized below.)

With complex games, and the number of possible future states becomes intractable, the designer lacks labelled good move/ bad move data sets. In order to approximate the unknown optimum, one must turn to reinforcement learning, in which the computer simulates a sequence of moves and discovers a policy function to maximize winning on its own. Reinforcement learning has already been implemented correctly for a computer to handily defeat world Go champion Lee Sedol four games to one in March of 2016.

The failure of the human mind to maximize anything has never stopped economists before. Here we will analyze the how to use reinforcement learning to find optimum equilibrium behavior. This approach will come in most useful when optimum equilibrium behavior is unknown.

2 Goal of Article:

In this article, we will show a lemma of reinforcement learning for finite time games.

The current state of the art in neural networks uses supervised learning to train a beginner policy function, then uses reinforcement learning to further improve the parameters of its approximation. As discussed in the previous section, supervised learning relies on a labelled dataset to update its parameters. We will not go into further detail on supervised learning in this article.

Reinforcement learning requires a reward function to tell the value of each state. In modern implementation of reinforcement learning, parameters are updated according to Ronald J. Williams (1992) [2]:

$$\Delta\rho \propto \frac{\partial log P_\rho(a_t|s_t)}{\partial\rho} z_t \tag{1}$$

Where the goal is to discover parameters corresponding to an optimum value function which evaluates the value of each state under the assumption of perfect play. The notation will be described in more detail later, but here we define ρ to be the parameters, P_ρ as the policy function, a is an action, s is a state, and z is the final outcome. (Conventionally z is positive for a 'good' outcome, and negative for a 'bad' outcome.)

The above rule has all the right ingredients: we update the parameters ρ according to how the parameters affect the policy function, and if the result is a win then we increase the likelihood of a similar move later, if the result is a loss then we decrease the likelihood of a similar move. Although this rule seems to make sense we will show in a lemma below that is is more accurate to use the slightly different update rule:

$$\frac{\partial}{\partial \rho} v_\rho(s_t) = E[z \sum_{l=t}^{T} \frac{\partial log P_\rho(a_l|s_l)}{\partial \rho}] \tag{2}$$

Where the expectation is taken according the policy function P_ρ.

The key difference is the expectation which appears in (2). When updating the parameters of the proposed optimum value function we should look ahead towards the end of the game in a manner ratifying the proposed optimum value function as correct. (1) simply multiplies the final gradient by z for seemingly no reason.

Furthermore, here we clearly define what we are maximizing: we want to choose the parameters ρ (which decide the actions) to maximize the value of each state. In other words we want our policy function to tell us to do actions which will give us the highest reward.

3 Suitability of Neural Nets

If we are trying to approximate the continuous function $f : \mathbb{R}^n \to \mathbb{R}$, we can define a single layer feedforward network g as follows:

$$g(x) = \sum_{j=1}^{n} \theta_j h(x' w^j + a_j), a_j, \theta_j \in \mathbb{R}, \text{ and } w^j \in \mathbb{R}^n, w^j \neq 0, n \in \{1, 2, 3, ...\}$$

Where the linear function $x'w^j + a_j$ is the first layer (it is most often a linear function), the functions h is the single 'hidden' layer. We can define h to be any function that is L^p integrable for all $1 \leq p \leq \infty$ and squashes: $\lim_{x\to\infty} h(x) = 1$ and $\lim_{x\to-\infty} h(x) = 0$.

A theorem by Hornik, Stinchcombe, and White (1989)[12] shows that for any triple: ($\varepsilon > 0$, probability measure μ, and compact set $K \subset \mathbb{R}^n$) there exists θ_j, a_j, w_j, n such that:

$\sup\limits_{x \in K} |f(x) - g(x)| \leq \varepsilon$ and $\int_K |f(x) - g(x)| d\mu \leq \varepsilon$.

Thus even a single layer neural net can approximate any continuous function arbitrarily well.

4 Overview of Neural Networks

We analyze games that 1) end in finite time and 2) only rewards at the end of the game. We have the value function $v^*(s)$ which determines the expected value of the outcome of the game from any state s, assuming perfect play by all players.

Examples of real games that have been successfully solved to world championship caliber with neural nets are: chess (Murray Campbell, Joseph Hoane Jr., Feng-hsiung Hsu (2002) [6]), checkers (Jonathan Schaeffer et al. (1992) [7]), othello (Michael Buro (1999) [8]), and Go [5].

In general players do not need to alternate moves, they ccould be moving simultaneously. It is not even necessary for there to be 2 players, there can be any number players in general. (There could even be 1 player as in many macroeconomic optimum planning implementations.)

4.1 Search Approach

The brute force way to solve a game is to search through all the moves into the future. Since it takes too much computation to do an exhaustive search through all possible future states, there are two common methods for finding the optimum policy function:

1. We can simply truncate the search at a certain depth $\hat{t}$ into the future and approximating the value of such a position: $\tilde{v}(\hat{t})$. Truncation is implemented in chess[6], checkers [7], and othello[8]

2. After estimating a simple policy function $P_\rho(a|s)$ which tells which action to take depending on the state, we can use Monte Carlo rollouts to search to the end of the game without branching into different possible actions. Monte Carlo rollouts are important for this article because we have a term $E[z_t|s = s_t]$ which denotes the final result after starting from state s_t. We can estimate $E[z_t|s = s_t]$ by summing over the action sequences according

to the policy $P_\rho(a|s)$. Monte Carlo rollouts are implemented in backgammon[9], and in Scrabble[10],

5 Reinforcement Learning

Reinforcement learning occurs when a software optimizes its parameters on its own. The software is presented with a reward function, and the software then seeks to find a policy function so that it can maximize its cumulative reward.

Reinforcement learning differs from supervised learning because the software is never presented with correct inputs or outputs, and sub-optimal actions are never corrected by an outside source. The software learns on its own. Software is often trained with supervised learning first, then trained with reinforcement learning afterwards.

A Reinforcement learning model looks exactly like a common microeconomic game - which consists of players - types - actions - payoffs - probabilities. Except that types and actions can be grouped into a notion of the state, and we use observability rules to guide what is observable from each state.

The basic reinforcement learning model consists of:

- States: $\mathbb{S} = \{s_0, s_1, s_2, ...\}$
- Possible actions for each state: $\mathbb{A}(s) \subset \{a_0, a_1, a_2, ...\}$
- A reward function $r : \mathbb{S} \to \mathbb{R}$
- Transition rules between states: $f : \mathbb{S} \times \mathbb{A} \to \mathbb{S}$
- Observability rules: $h : \mathbb{S} \to \mathbb{O}$, where $\mathbb{O}$ defines the set of observable attributes of the states $\mathbb{S}$

Here we will add the assumption that the game has a definite end:

Assumption 1: $\exists \mathbb{E} \subset \mathbb{S}$ so that the game ends if it reaches a state $s \in \mathbb{E}$.
And $\forall$ games where T denotes total number of moves to reach the end of the game, $\exists \bar{T}$ such that $T < \bar{T}$. So that all games will definitely end before $\bar{T}$ moves.

Furthermore we add the assumption that the reward function can only yield nonzero values at the end of the game:

Assumption 2: $r(s) = 0 \forall s \notin \mathbb{E}$

5.1 Further Notation

We can define a policy function with regards to the parameters ρ as: $P_\rho(a_t|s_t) : \mathbb{S} \to \triangle\mathbb{A}$. P_ρ yields a distribution over the actions $\mathbb{A}$, conditional on the state: s_t.

ρ are the parameters that determine the policy function. It is the objective of the software to choose ρ such that in the course of the game, the expected reward is maximized.

We can define the optimum value function: $v_\rho(s) = E[z_t|s_t = s, a_{t,\ldots,T} \sim P_\rho]$

The notation $a_{t,\ldots,T} \sim P_\rho$ formalizes the assumption that P_ρ is an approximation of perfect play, and that all future actions will be played according to P_ρ. We can simulate draws from $a_{t,\ldots,T} \sim P_\rho$ with Monte Carlo rollouts - by simulating to the end of the game without branching into other possible decision paths.

To simplify notation, we will also use: z_t corresponding to the final reward of a game that is currently in state: s_t. (In zero sum win/loss games of chess or Go, $z_t \in \{1, -1\}$)

5.2 Sample Problem Settings

Here I will provide two examples of game settings that are applicable to reinforcement learning:

5.3 Signalling

A simple Spence signalling game takes 2 turns [11]. The sends a message after observing his quality. The firm receives the message and then offers a wage.

- The State Space $\mathbb{S} : (q, m, w) \in \mathbb{S}$ is the quality of the worker, the signal of the worker, and a wage offer. We note that the state is a triplet with possibly empty entries.

The quality of the worker is

$$q = \begin{cases} H \in \mathbb{R}^+ & \text{with probability } p \\ L \in \mathbb{R}^+, L < H & \text{with probability } 1-p \end{cases}$$

- The Observability Rule: The Firm does not see the quality of the worker.
- The Action Space: $\mathbb{A}(s) : a \in \mathbb{A}(s)$
  1. Worker: Any nonnegative message $m \in \mathbb{R}^+$
  2. Firm: Any nonnegative wage: $w \in \mathbb{R}^+$
- The State transition: $s_{t+1} = f(s_t, a_t)$:
  Once a worker makes a signal m, the signal m is filled into the state triplet. Once a firm makes an offer w, the offer w is filled into the triplet.
- The reward function at the end of the game is:
  1. Worker: $r^{worker}(q, m, w) = w - \frac{m}{q}$
  2. Firm: $r^{firm}(q, m, w) = q - w$
  3. Both workers and firms do not receive any reward if the game is not over.

5.4 Go

The game of Go is defined by a 19 by 19 grid. Each point on the grid is either filled with a black stone, a white stone, or is empty. There is a player that places white stones, and a player that places black stones. The game is over when the board is filled.

- The State Space $\mathbb{S} : s \in \mathbb{S}$ is any board position. (Any combination of black stones, white stones, or empty spaces on the board). There are less than 3^19 possible board positions.
- The Action Space: $\mathbb{A}(s) : a \in \mathbb{A}(s)$ is any one move possible from the state s. One player places the stone of his color.

- The State transition: $s_{t+1} = f(s_t, a_t)$ determines the new state at time $t+1$ after move a_t at state s_t. Other than alternating moves, there is only one other rule of Go: If a group of white stones is surrounded by black stones without any adjacent blank spots, then the white stones are removed from the board. (And vice versa for black stones).

- The player with the most stones on the board at the end of the game wins. The reward function is:

$$r^i(s) = \begin{cases} 0 & \text{if game is not over at state } s \\ 1 & \text{if player } i \text{ won at state } s \\ -1 & \text{if player } i \text{ lost at state } s \end{cases}$$

  The reward function formalizes the assumption that both players only care about the final win/loss outcome. This follows with **Assumption 2 above**

- Accordingly with **Assumption 1**, the game ends when the entire board is filled. No Go game in history has ever exceeded 500 moves.

6 The Optimum Parameters

We note that the goal of reinforcement learning is to find the optimum parameters ρ^* such that given any state s, $P^*_\rho(s)$ yields the distribution over actions which maximizes the expected final reward. Finding the optimum ρ^* amounts to finding the optimum policy P^*_ρ and finding the optimum value function v^*_ρ

In our proof, we will derive an analytic expression for the derivative of the optimum value function: v^*_ρ, under the assumption that the current ρ is equal to the optimum ρ^*, and that the policy function and the value function are already the optimum values.

In effect we've found a way to verify that the optimum parameters ρ^* are at a local extrema by checking if the derivative is zero. However we do not have a convexity argument to claim that parameters ρ close to the optimum ρ^* will yield a derivative which brings the parameters even closer to the optimum.

This is why it is common to use supervised learning. First we would use real labelled examples of good moves and bad moves to train the parameters ρ such that they are slightly closer to the optimum ρ^*. If we wish to use reinforcement learning afterwards, we must hope that our current parameters ρ are close enough to ρ^* such that the derivative provided below can bring the parameters the rest of the way to the optimum.

6.1 Main Lemma

Recall the proposed update direction from Williams (1992) [2]:

$\Delta\rho \propto \frac{\partial log P_\rho(a_t|s_t)}{\delta\rho} z_t$ *(1)*

Below we will show that the above update direction is too short sighted - arrives at parameters ρ that maximizes the one-step increase of the value function - in other words it asks: "If I could only take one move - what is the best possible move to take?". The correct equilibrium strategy would instead say: "Given that we're both playing according these two known strategies - what is the best possible action to maximize my reward function?"

Lemma 6.1. *1. The original update direction chooses ρ to maximize the value v in one step:*

$$E[\frac{\partial log P_\rho(a_t|s)}{\partial \rho} z_t] = E[\frac{\partial}{\partial \rho}(v_\rho(s) - v_\rho(f(s,a_t)))|s_t = s, a_{t,...,T} \sim P_\rho]$$

2. *The correct update direction would take into account the full sequence of actions going into the future:*

$$E[\frac{\partial}{\partial \rho} v_\rho(s_t)|a_{t,...,T} \sim P_\rho] = E[z_t \sum_{l=t}^{T} \frac{\partial log P_\rho(a_l|s_l)}{\partial \rho} |a_{t,...,T} \sim P_\rho]$$

Proof. We take the expectation of the updating direction referred in (1):

$E[\frac{\partial log P_\rho(a_t|s)}{\partial \rho} z_t|s = s_t, a_{t,...,T} \sim P_\rho]$
$= \sum_{a_t \in A(s)} \frac{\partial log P_\rho(a_t|s)}{\partial \rho} E[z_{t+1}|s = s_{t+1} \equiv f(s,a_t), a_{t+1,...,T} \sim P_\rho] P_\rho(a_t|s)$
$= \sum_{a_t \in A(s)} \frac{\partial P_\rho(a_t|s)}{\partial \rho} E[z_{t+1}|s = s_{t+1} \equiv f(s,a_t), a_{t+1,...,T} \sim P_\rho]$
$= \sum_{a_t \in A(s)} \frac{\partial P_\rho(a_t|s)}{\partial \rho} v_\rho(f(s,a_t))$ **(3)**

On examining the value function:

$v_\rho(s) = E[z_t|s_t = s, a_{t,...,T} \sim P_\rho]$
$= \sum_{a_t \in A(s)} E[z_{t+1}|s_{t+1} = f(s,a_t), a_{t+1,...,T} \sim P_\rho] P_\rho(a_t|s)$
$= \sum_{a_t \in A(s)} v_\rho(f(s,a_t)) P_\rho(f(s,a_t)|s)$

We take the derivative of the value function:

$\frac{\partial}{\partial \rho} v_\rho(s) = \sum_{a_t \in A(s)} (\frac{\partial}{\partial \rho} v_\rho(f(s,a_t))) P_\rho(f(s,a_t)|s) + \sum_{a_t \in A(s)} v_\rho(f(s,a_t))(\frac{\partial}{\partial \rho} P_\rho(f(s,a_t)|s))$

If we combine the above result with **(3)**:

$E[\frac{\partial log P_\rho(a_t|s)}{\partial \rho} z_t|s = s_t, a_{t,...,T} \sim P_\rho] =$
$\sum_{a_t \in A(s)} \frac{\partial}{\partial \rho}(v_\rho(s) - v_\rho(f(s,a_t))) P_\rho(f(s,a_t)|s)$

Finally, the above result shows that at $s = s_t$,
$\frac{\partial log P_\rho(a_t|s)}{\partial \rho} z_t$ is an unbiased estimate of $E[\frac{\partial}{\partial \rho}(v_\rho(s) - v_\rho(f(s,a_t)))|s_t = s, a_{t,...,T} \sim P_\rho]$

Now we are close to the final result: We can now sum the values $\frac{\partial log P_\rho(a_t|s)}{\partial \rho} z_t$ going into the future to achieve an unbiased estimate of the derivative.

Then for any $j \geq i$, $z_i \sum\limits_{l=i}^{j} \frac{\partial log P_\rho(a_l|s_l)}{\partial \rho}$ is an unbiased estimate of: $E[\frac{\partial}{\partial \rho}(v_\rho(s_i) - v_\rho(s_j))|s_t = s_i, a_{t,\ldots,T} \sim P_\rho]$

We arrive at the correct update direction for the parameters ρ:
$E[z \sum\limits_{l=t}^{T} \frac{\partial log P_\rho(a_l|s_l)}{\partial \rho}]$
$= \frac{\partial}{\partial \rho}(v_\rho(s_t) - v_\rho(s_T) = \frac{\partial}{\partial \rho} v_\rho(s_t)$

Where the last equality is because s_T is the final state of the game, then $v_\rho(s_T) = r(s_T)$ which does not depend on ρ.

$\square$

7 Conclusion

Neural Nets can use reinforcement learning to approximate optimal policy functions that are intractable to compute and impossible to derive analytically. Reinforcement learning can be applied in the vast majority of economic models. We've shown that the conventional update direction for the parameters in reinforcement learning is too shortsighted. We derived a corrected update direction that maximizes the reward in any state.

It may be the case that when the policy function $P(a|s)$ is still relatively 'untrained' and far from perfect play, looking only one step into the future instead of simulating all the way towards the end of the game would be better due to the lack of precision with predicting future play. Here we've shown exactly what the original update direction measures. If the modeler has qualms about the accuracy of predicting future play, then the modeler may choose to explicitly discount the corresponding terms in the correct update direction:

$$\frac{\partial}{\partial \rho} v_\rho(s_t) = E[z \sum_{l=t}^{T} \frac{\partial log P_\rho(a_l|s_l)}{\partial \rho}]$$

One should use the full analytic expression we've derived above to check if we've arrived at a proper local extrema.

Here we have used a general setting to show that the update direction of a reinforcement learning algorithm should not be as (1) - Williams (1992) [2]. The original update direction maximized a one step increase in value function. Instead we show that it is better to look towards the end of the game to maximize the entire value function itself.

References

[1] Richard Bellman (1954): "The Theory of Dynamic Programming"

[2] Ronald J. Williams (1992): "Simple statistical gradient-following algorithms for connectionist reinforcement learning." - Machine Learning, Volume 8, Page 229 - 256: 1992

[3] David J.C. MacKay (2003): "Information Theory, Inference, and Learning Algorithms" Cambridge University Press

[4] David Kriesel (2005): "A Brief Introduction to Neural Networks"

[5] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, Demis Hassabis (2016) - Nature: "Mastering the game of Go with deep neural networks and tree search"

[6] Murray Campbell, Joseph Hoane Jr., Feng-hsiung Hsu (2002): "Deep Blue" - Artificial Intelligence. Vol. 134, Issues 1-2, Page 57-83. Amsterdam, Elsevier: 2002.

[7] Jonathan Schaeffer et al. (1992): "A World Championship Caliber Checkers Program" - Artificial Intelligence. Vol. 53, Page 273-289. Amsterdam, Elsevier: 1992.

[8] Michael Buro (1999): "From simple features to sophisticated evaluation functions" - 1st International Conference on Computers and Games.

[9] Gerald Tesauro and Gregory (1996): "On-line policy improvement using Monte-Carlo tree search." - Advances in Neural Information Processing, 1068-1074

[10] Brian Sheppard (2002) - "World Championship Caliber Scrabble" - Artifical Intelligence 134, 241 - 275.

[11] Michael Spence (1973) - "Job Market Signalling" - Quarterly Journal of Economics

[12] Kur Hornik, Maxwell Stinchcombe, Halber White (1989): "Multilayer Feeddforward Networks are Universal Approximators"