Applications of Reinforcement Learning in Logistics and Economics

Larkin Liu [1] [2]

larkin.liu@tum.de

[1]TUM School of Management

[2]Munich Data Science Institute

July 17, 2022

Overview

Larkin Liu

Larkin Liu (born 1992) is a Chinese-Canadian research scientist. He studied first at the University of Toronto, obtaining his Master's degree in Industrial Engineering. Larkin has worked extensively as a Data Scientist in companies across both Germany and Canada. Currently, he is a Doctoral Student at the Technical University of Munich, conducting research at the Chair of Logistics and Supply Chain.

Challenges of Supply Chain Management

- **Big Data** - Large intakes of data, arising from data availability and advancements in big data storage (Hadoop, Apache Spark).
- **Imperfect Information and/or Delays** - Due to complex data tracking and highly stochastic systems.
- **Multi-Scale Uncertainty** - Arising from changes in government policies, unexpected service disruptions, and supply delays etc.

My Current Areas of Active Research

- **Risk Management** - Stochastic Modelling for inventory optimization - multi-sourcing, joint replenishment etc.
- **Competitive Supply Chains** - Market Design, Competitive Strategies, Nash / ϵ-Nash equilibrium multi-agent policies.
- **Methodology** - Fundamental study of mathematical theory in Stochastic Modelling and Machine Learning.

Risk Management in Inventory Policy

- **Robust simulation & Data-Driven Modelling** - non-parametric modelling via Machine Learning. [1]
- **Multi-sourcing policies** - Resilience for dealing with global disruptions in supply chains. [19]
- **Large Scale MDP's** - solutions via Deep Reinforcement Learning (Policy Learning, Q-learning etc.) [16] [18].

Multiplayer Supply Chains

- **Nash and ϵ-Nash Equilibrium Policies** - via Multi-Agent Reinforcement Learning [6]
- **Algorithmic Game Theory** - Efficient Market Design, Optimal Dynamic Pricing etc.[2] [9]
- **Markov Games** - Competitive & Cooperative Multi-Agent Markov Decision Processes [5]

Methods

A blend of approximate and exact methods,

- **Monte Carlo Methods** - in *Approximate Dynamic Programming*, *Monte Carlo Tree Search* [12] can be substituted into a main dynamic programming algorithm to estimate complex value functions. [23]
- **Mixed Integer Programming and/or Piecewise Convex Optimization** - i.e. Bender's Decomposition, Dantzig-Wolfe Decomposition, ADMM [3] [14]
- **Deep Reinforcement Learning** - Modelling complex Q-functions via Deep Neural Networks to yield approximation of the $\mathbb{T} : \{S_t \times A \times S_{t+1}\} \to \{R \in \mathbb{R}\}$. (MDP's, Semi-MDP's, POMDP's).

Selected Works

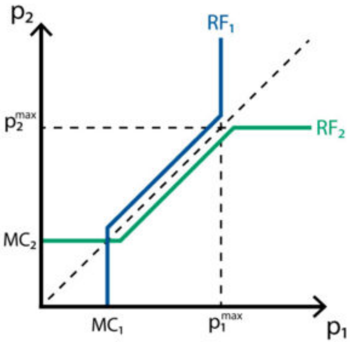
My research papers thus far (includes conferences, journals, and pre-prints).

- Larkin Liu. "Approximate Nash Equilibrium Learning for n-Player Markov Games in Dynamic Pricing". In: *arXiv preprint arXiv:2207.06492* (2022)
- Larkin Liu, Richard Downe, and Joshua Reid. "Multi-armed bandit strategies for non-stationary reward distributions and delayed feedback processes". In: *arXiv preprint arXiv:1902.08593* (2019)
- Larkin Liu and Jun Tao Luo. "mctreesearch4j: A Monte Carlo Tree Search Implementation for the JVM". In: *Journal of Open Source Software* 7.70 (2022), p. 3804

Approximate Nash Equilibrium Learning for n-Player Markov Games in Dynamic Pricing

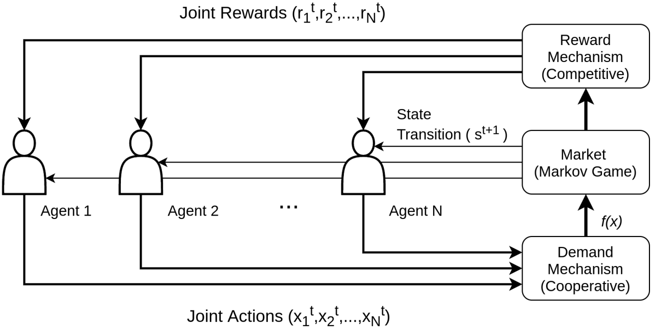
When firms compete to maximize their profit in an oligopoly,

- **Cournot** - Competition on production quantity driven demand.
- **Stackelberg** - Sequential Cournot competition.
- **Bertrand** - Competition on price driven demand.



Bertrand Competition Link

A Simulation of Oligopoly



An Oligopoly Simulation

Solving the ϵ-Nash Equilibrium Conditions

In an ϵ-Nash Equilibrium, no agent can improve its expected policy value by deviating to a different policy by more than a difference of ϵ.

- The solution to ϵ-Nash Equilibrium usually constitute NP-Hard Problems, and are solved via approximation techniques.
- We propose approximation techniques in combination with deep reinforcement learning.
- We demonstrate that approximate Nash Equilibria can be obtained.

ϵ-Nash Equilibrium Conditions

$$\mathbf{v}(\pi^{n}, \pi^{-n*}) \leq \mathbf{v}(\pi^{n*}, \pi^{-n*}) + \epsilon, \quad \forall n \in N \tag{1}$$

Theoretical Market Equilibrium

We propose a hypothetical economic environment, where all agents generate a market price x_n, ϵ is the greatest expected gain when any firm unilaterally undercuts the current market price x_n.

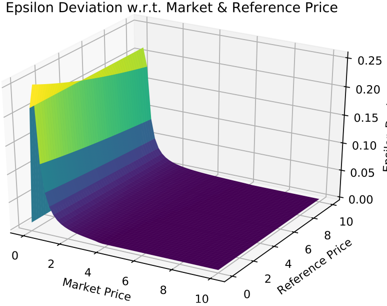
$$\epsilon = \max_{d^* \in \mathbf{R}} \left(\mathbf{E}[\Pi_n(x_n - d^*)] - \mathbf{E}[\Pi_n(x_n)] \right) \tag{2}$$

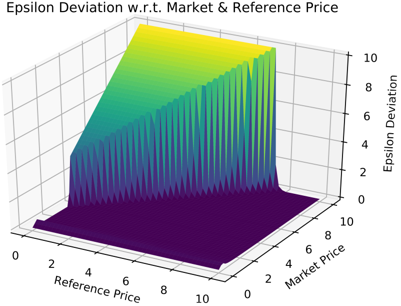
We demonstrate that a theoretical ϵ-Nash Equilibrium, can exist when (Proof in [10]),

$$d^* = \frac{\sqrt{c_1^2 - c_1 + 4(c_2 - 1)c_2 - 2c_2}}{2c_2}$$

where $\quad c_1 = \frac{-(\beta_1+\beta_2)}{f(\tilde{x})} - \frac{1}{\tilde{x}}, \quad c_2 = \frac{-(\beta_1+\beta_2)}{f(\tilde{x})\tilde{x}}$

Theoretical Market Equilibrium Scenarios





Market Scenario 1:
$\beta_0 = 25, \beta_1 = -0.6, \beta_2 = -6.1, a = 0.1$

Market Scenario 2:
$\beta_0 = 15, \beta_1 = -1.05, \beta_2 = -3.1, a = 0.1$

Multi-Agent Nash Q Learning

In a competitive setting the Q function is altered, and is no longer the action which maximizes the Bellman Update, but the option that reaches Nash Equilibrium, $\mathcal{N}(s')$.

$$Q'(s, \bar{x}) \leftarrow (1-\alpha)Q(s, \bar{x}) + \alpha[r + \gamma\mathcal{N}(s')] \tag{3}$$

$$\bar{x}^* = \underset{\bar{x}}{\operatorname{argmax}}\ Q(s', \bar{x}) \prod_{i=1}^{N} \pi_n^*(s', x_n) \tag{4}$$

$$\mathcal{N}(s') = Q(s', \bar{x}^*) \prod_{i=1}^{N} \pi_n^*(s', x_n^*) \tag{5}$$

Multi-Agent Nash Q Learning

The maximum value difference, from deviation is represented as δ.

Maximum Value Gain δ

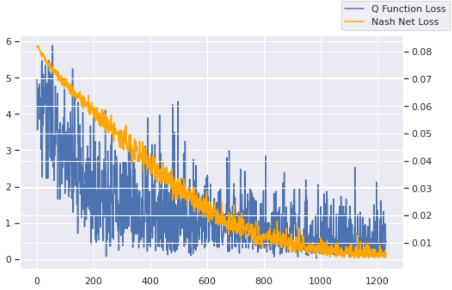
$$V(s, \pi) = \max_{\bar{x}} Q(s, \bar{x}) \prod_{i=1}^{N} \pi_n(s, x_n) \tag{6}$$

$$\delta = \max_{\pi'_n} \left(V(s, \pi'_n, \pi_{-n}) - V(s, \pi_n, \pi_{-n}) \right) \quad \forall s \in \mathbf{S} \tag{7}$$

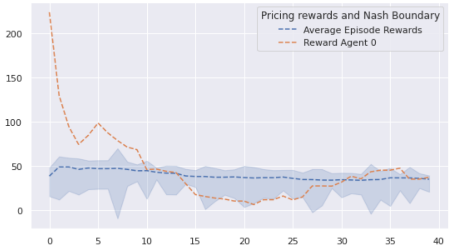
δ can be exhausitive to compute so it can be approximated as a Neural Network Γ_s. Therefore the approximate NE policy is $\hat{\pi}^*(s)$.

$$\hat{\pi}^*(s) = \operatorname*{argmin}_{\pi} \Gamma(\pi)_s \tag{8}$$

Loss Function



Decreasing training loss.



Convergence of agent rewards to a NE bound.

Supplemental Material

Markov Decision Process

Optimal Policy

Provided a policy π, the expected reward, V_t from taking action a_t can be expressed by Eq. 16.

$$V(S_t) = \max_{a \in A} \left[R(S_t, a) + \gamma \sum_{S_{t+1} \in S} P(S_{t+1} | S_t, a) V(S_{t+1}) \right] \tag{9}$$

$$\pi^*(S_t) = \operatorname*{argmax}_{a \in A} V(S_t, a) \tag{10}$$

A discrete $MDP\langle S, A, \mathbb{T}, R \rangle$ designates a set of states S, where the agent traverses from S_t to S_{t+1}, for a horizon of T in t distinct time increments.

$$\mathbb{T} : \{S_t \times A \times S_{t+1}\} \rightarrow \{R \in \mathbb{R}\} \tag{11}$$

Markov Decision Process

Q function

$Q(S_t, a_t)$ provides a measure of the discounted reward provided action a is taken in state S_t

$$Q(S_t, a_t) = R(S_t, a_t) + \gamma \sum_{S_{t+1} \in S} P(S_{t+1}|S_t, a_t)V(S_{t+1}) \tag{12}$$

Key Challenges for real-world MDP's

- Parameters of the underlying process $MDP\langle S, A, \mathbb{T}, R\rangle$ are unknown.
- Imperfect conditions and/or unobservable information.
- High dimensionality of state and action space.

Value Function

The value function of a given policy π^n is represented as $\mathbf{v}_\gamma(\pi^n, \pi^{-n})$,

- π^n represents the policy of agent n,
- π^{-n} represents the policies of the other agents in the system.

A policy π^n stipulates the probability that agent n chooses action $a \in \mathbf{A}(s)$ in state $s \in S$ [5].

$$P_s^t(\pi^n, \pi^{-n}) = [P^t(s'|s, \pi^n, \pi^{-n})]^{s' \in S} \tag{13}$$

Reward function, where $\pi(s, x)$ is the probability that action a is taken in state s under policy π.

$$r(s, \pi^n, \pi^{-n}) = \sum_{a \in \mathbf{A}} r(s, a)\pi(s, a) \tag{14}$$

Markov Decision Process

Optimal Policy

Provided a policy π, the expected reward, V_t from taking action a_t can be expressed by Eq. 16.

$$V(S_t) = \max_{a \in A} \left[R(S_t, a) + \gamma \sum_{S_{t+1} \in S} P(S_{t+1} | S_t, a) V(S_{t+1}) \right] \tag{15}$$

$$\pi^*(S_t) = \operatorname*{argmax}_{a \in A} V(S_t, a) \tag{16}$$

A discrete $MDP\langle S, A, \mathbb{T}, R \rangle$ designates a set of states S, where the agent traverses from S_t to S_{t+1}, for a horizon of T in t distinct time increments.

$$\mathbb{T} : \{S_t \times A \times S_{t+1}\} \rightarrow \{R \in \mathbb{R}\} \tag{17}$$

Competitive Markov Decision Process

A competitive multi-agent MDP can be fundamentally constituted by tuples $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{r}_t, \mathbf{s}_{t+1})$,

- **State** s_t^n - State of each agent n at time t, i.e. vendor inventory level and/or other attributes for the item at time t.
- **Joint Action** $a_t^1, ..., a_t^N$ - The joint actions at time t for all agents.
- **Reward** $r_t^1, ..., r_t^N$ - The immediate reward for all respective agents at time t.

Depending on the visibility of the system, the representation differs,

- **Fully Observable** $(s_t^1, .., s_t^N, a_t^1, ..., a_t^N, r_t^1, ..., r_t^N, s_{t+1}^1, .., s_{t+1}^N)$ - Attributes are fully observable for all agents at time t to all agents.
- **Censored** $(s_t^n, a_t^1, ..., a_t^N, r_t^n, ..., r_t^N, s_{t+1}^n)$ - Only relevant, or partial data is observable to each respective agent.

Value Function

The value function of a given policy π^n is represented as $\mathbf{v}_\gamma(\pi^n, \pi^{-n})$,

- π^n represents the policy of agent n,
- π^{-n} represents the policies of the other agents in the system.

A policy π^n stipulates the probability that agent n chooses action $a \in \mathbf{A}(s)$ in state $s \in S$ [5].

$$P_s^t(\pi^n, \pi^{-n}) = [P^t(s'|s, \pi^n, \pi^{-n})]^{s' \in S} \tag{18}$$

Reward function, where $\pi(s, x)$ is the probability that action a is taken in state s under policy π.

$$r(s, \pi^n, \pi^{-n}) = \sum_{a \in \mathbf{A}} r(s, a)\pi(s, a) \tag{19}$$

Value Function (cont.)

The reward vector is a 1 x $|S|$ vector,

$$\mathbf{r}(\pi^n, \pi^{-n}) = [r(s'_{s=1}, \pi^n, \pi^{-n}), ..., r(s'_{s=S}, \pi^n, \pi^{-n})]^T \tag{20}$$

$\mathbf{P}^t(\pi^n, \pi^{-n})$ is a $|S|$ x $|S|$ matrix,

$$\mathbf{P}^t(\pi^n, \pi^{-n}) = [P^t_{s=1}(\pi^n, \pi^{-n}), ..., P^t_{s=S}(\pi^n, \pi^{-n})]^T \tag{21}$$

With the definition of $\mathbf{r}(\pi^n, \pi^{-n})$ and $\mathbf{P}^t(\pi^n, \pi^{-n})$, we can define the value function of a policy,

Value Function

$$\mathbf{v}(\pi^n, \pi^{-n}) = \sum_{t=0}^{\infty} \gamma^t \mathbf{P}^t(\pi^n, \pi^{-n}) \mathbf{r}(\pi^n, \pi^{-n}) \tag{22}$$

Value Function (cont. 2)

Assuming $\mathbf{I} - \gamma\mathbf{P}$ is invertible, and for some integer value k, such that $\mathbf{P}^k = \mathbf{0}$ (Nilpotent Matrix Property), we leverage a well known identity,

$$(\mathbf{I} - \gamma\mathbf{P})^{-1} = (\mathbf{I} + \gamma\mathbf{P}^2 + \gamma^2\mathbf{P}^3 + ... + \gamma^{k-1}\mathbf{P}^{k-1}) \tag{23}$$

Therefore, $\mathbf{v}(\pi^n, \pi^{-n})$ can be represented as ,

Value Function [5]

$$\mathbf{v}_\gamma(\pi^n, \pi^{-n}) = [\mathbf{I} - \gamma\mathbf{P}(\pi^n, \pi^{-n})]^{-1}\mathbf{r}(\pi^n, \pi^{-n}) \tag{24}$$

Where $\mathbf{I}$ is the identity matrix, and γ is the discount factor.

Structure of the Nash Equilibrium

In an ϵ-Nash Equilibrium, no agent can improve its expected policy value by deviating to a different policy by more than a difference of ϵ.

ϵ-Nash Equilibrium Conditions

$$\mathbf{v}(\pi^{n}, \pi^{-n*}) \leq \mathbf{v}(\pi^{n*}, \pi^{-n*}) + \epsilon, \quad \forall n \in N \tag{25}$$

Strict Nash Equilibrium when $\epsilon = 0$.

Example - Value Computation

Given a two player Markov Game, with state space $\mathbf{S} = \{0, 1\}$, and action space $\mathbf{A}_1 = \mathbf{A}_2 = \{0, 1\}$. Provided reward function $r(a_0, a_1, s)$ and transition probability function $p(s'|a_0, a_1, s)$.

$$r(a_0, a_1, s) = \begin{bmatrix} (3,0) & (6,0) \\ (2,0) & (1,0) \end{bmatrix} \tag{26}$$

$$p(s'|a_0, a_1, s = 0) = \begin{bmatrix} (1,0) & (1/3, 2/3) \\ (1,0) & (1,0) \end{bmatrix} \tag{27}$$

$$p(s'|a_0, a_1, s = 1) = \begin{bmatrix} (0,1) & (0,1) \\ (0,1) & (0,1) \end{bmatrix} \tag{28}$$

Example - Value Computation (cont.)

We provide a fixed policy for agent $n = 0$, and two candidate policies for agent $n = 1$.

$$\pi^{n=0} = [(0, 1), (1, 0)] \tag{29}$$

$$\pi^{n=1,0} = [(1, 0), (1, 0)] \tag{30}$$

$$\pi^{n=1,1} = [(0, 1), (1, 0)] \tag{31}$$

For a discount factor $\gamma = 0.75$ compute the value of the joint policy $v(\pi^{n=0}, \pi^{n=1})$ for infinite time horizon. Comment on the Nash Equilibrium property.

Example - Value Computation Solution

Compute the state transition matrix for each joint policy.

$$p(s', s, \pi^{n=0}, \pi^{n=1,0}) = \begin{bmatrix} 1/3 & 2/3 \\ 0 & 1 \end{bmatrix} \tag{32}$$

$$p(s', s, \pi^{n=0}, \pi^{n=1,1}) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{33}$$

$$r(s', s, \pi^{n=0}, \pi^{n=1,0}) = \begin{bmatrix} 6 \\ 0 \end{bmatrix} \tag{34}$$

$$r(s', s, \pi^{n=0}, \pi^{n=1,1}) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{35}$$

The apply Eq. (24) to solve for $\mathbf{v}(\pi^{n=0}, \pi^{n=1})$.

Example - Value Computation Solution (Cont.)

We compute the value for each candidate joint policy.

$$\mathbf{v}(\pi^{n=0}, \pi^{n=1,0}) = \begin{bmatrix} 8 \\ 0 \end{bmatrix} \tag{36}$$

$$\mathbf{v}(\pi^{n=0}, \pi^{n=1,1}) = \begin{bmatrix} 4 \\ 0 \end{bmatrix} \tag{37}$$

The value of the joint policy $\mathbf{v}(\pi^0, \pi^1)$, also serves as a strict Nash Equilibrium point.

$$\mathbf{v}(\pi^0, \pi^1) \leq \begin{bmatrix} 8 \\ 0 \end{bmatrix} \tag{38}$$

Markov Games as a Framework for Multi-Agent Reinforcement Learning [8]

Game Solution via Linear Programming

Consider a single stage game of *rock, paper, scissors* (r, p, s), with the reward matrix $r(x_0, x_1)$ as given (Littman 1994) [8],

	rock	paper	scissor
vs. rock	0	1	-1
vs. paper	-1	0	1
vs. scissor	1	-1	0

Compute the optimal policy for an agent's perspective.

Max Min Approach to Value Computation

For a single stage game, with no state transitions.

$$v = \max_{\pi^n \in \mathbb{P}(X)} \min_{a^{-n} \in X} \sum_{a^n \in X} r(x_0, x_1)\pi^n \tag{39}$$

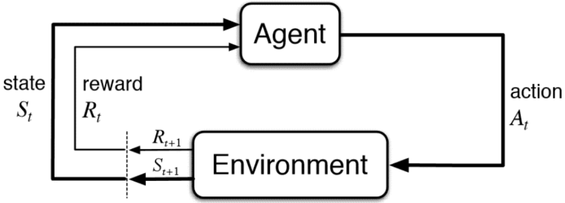
Can be represented by the system of linear inequalities and equations, with respect the the reward matrix,

$$\begin{aligned}
\pi_p - \pi_s &\geq v \quad \textit{vs. rock} && (40)\\
-\pi_r + \pi_s &\geq v \quad \textit{vs. paper} && (41)\\
\pi_r - \pi_p &\geq v \quad \textit{vs. scissor} && (42)\\
\pi_r + \pi_p + \pi_s &= 1 && (43)
\end{aligned}$$

Can be solved using Linear Programming.

Q Learning (Watkins 1992) [22] and Deep Q Learning (Mnih et al 2015) [16]

Reinforcement Learning



From (Sutton et al 2018) [20]

- Agent iteratively walks through an undefined MDP, and effective learning the $S \rightarrow Q(S, a)$ mapping, to obtain π^*.
- Useful when $MDP\langle S, A, \mathbb{T}, R\rangle$ are unknown, too complex, or subject to imperfect information.
- RL has many different variations. We will focus on Q Learning and Deep Q Learning.

Q Function

The Q function for an agent n is the sum of the immediate reward of taking action x and the expectation of future value times $\gamma < 1.0$.

$$Q(x, s^n) = r(x, s^n) + \gamma \max_{x'}(Q(x', s^{n\prime})) \tag{44}$$

We define the Q function for multi-agent systems as a vector, where each element represents an agent.

Q Vector Function

$$\mathbf{Q}(\mathbf{x}, \mathbf{s}) = \mathbf{r}(\mathbf{x}, \mathbf{s}) + \gamma \mathbb{E}[\mathbf{v}(\mathbf{s}')] \tag{45}$$

Q Learning

Q learning iteration

$$Q^{i+1}(S_t, a_t) \leftarrow (1-\alpha)Q^i(S_t, a_t) + \alpha \left[R(S_t, a_t) + \gamma \max_a Q^i(S_{t+1}, a) \right] \quad (46)$$

- The Q function, defined in Eq. 12, is iteratively learned via agent exploration of the system, where the $MDP\langle S, A, \mathbb{T}, R\rangle$ are unknown.
- Proof of optimal policy convergence in (Watkins 1992) [22].
- Hyperparameters, learning rate α and reward discount factor γ, require guess work.

Q Learning (cont.)

Initialized

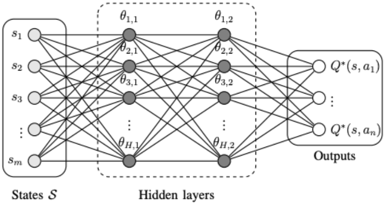
Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	327	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	. . .	. . .	. . .	. . .	. . .	. . .	. . .
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

From Wikipedia Article

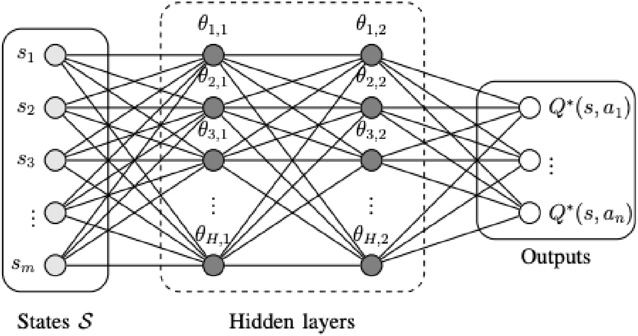
Deep Q Learning



From (Mismar 2019) [15]

- Convolutional Neural Networks can approximate mapping for $S \times A \rightarrow Q(s, a)$.
- (Mnih et al 2015) [16]* adopted deep Q Learning as a state-of-the-art solution for designing AI for single and multiplayer computer games.
- (Wang et al 2018) [21] (Rabe et al 2017) [17] presents examples of recent work in OR showing DQL to be highly effectice in solving MDP's in Logistics and Supply Chain.

Deep Q Learning (cont.)



From (Mismar 2019) [15]

Nash Q Learning for General Sum Stochastic Games (Hu & Wellman 2002) [7]

Nash Q Learning

The update Q value update process is similar to the single agent scenario, in Eq. (46), however the Q update must now consider the joint action, $\mathbf{a_t}$, that is the actions of other competing agents $[a^n, a^{-n}]$.

Nash Q learning iteration (Hu & Wellman [7])

$$Q^{i+1}(S_t, \mathbf{a_t}) \leftarrow (1-\alpha)Q^i(S_t, \mathbf{a_t}) + \alpha \left[R(S_t, \mathbf{a_t}) + \gamma \mathbb{N}(S_{t+1}, \mathbf{a}_{t+1})\right] \quad (47)$$

Where, for N agents, the Nash Operator $\mathbb{N}$ is defined as,

$$\mathbb{N}(S_t, a_t^1, a_t^2, ..., a_t^N) = Q(S_t, a_t^1, a_t^2, ..., a_t^N) \prod_{n=1}^{N} \pi^n(S_t, a^n) \quad (48)$$

Nash Q Learning (cont.)

At each Q update iteration, the Nash Equilibrium must be solved using the estimated Q functions for all agents.

$$v^n(s) = \max_{a \in \mathbf{A}} Q^n(s, a^n, a^{-n}) \tag{49}$$

Where, for N agents, the Nash Operator $\mathbb{N}$ is defined as,

$$\prod_{n=1}^{N} \pi^n(s, a^n) Q^n(s, a_t^n, a_t^{-n^*}) \leq \prod_{n=1}^{N} \pi^n(s, a^n) Q^n(s, a_t^{n^*}, a_t^{-n^*}) \tag{50}$$

$$v(s, \pi^n, \pi^{-n^*}) \leq v(s, \pi^{n^*}, \pi^{-n^*}) \quad \forall s \in \mathbf{S} \tag{51}$$

References I

[1] Anna-Lena Beutel and Stefan Minner. "Safety stock planning under causal demand forecasting". In: *International Journal of Production Economics* 140.2 (2012), pp. 637–645. URL: https://EconPapers.repec.org/RePEc:eee:proeco:v:140:y:2012:i:2:p:637-645.

[2] Martin Bichler, Simon Field, and H. Werthner. "Introduction: Theory and Application of Electronic Market Design". In: *Electronic Commerce Research* 1 (July 2001), pp. 215–220. DOI: 10.1023/A:1011512919970.

[3] Stephen Boyd et al. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers". In: *Foundations and Trends in Machine Learning* 3 (Jan. 2011), pp. 1–122. DOI: 10.1561/2200000016.

[4] Lihua Chen, Yi Lu, and Rui Zhao. "Analysis and application of modern supply chain system in China". In: *Modern Supply Chain Research and Applications* 1 (June 2019). DOI: 10.1108/MSCRA-01-2019-0004.

[5] J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer, 1997.

[6] Junling Hu and Michael P. Wellman. "Nash Q-Learning for General-Sum Stochastic Games". In: *J. Mach. Learn. Res.* 4.null (Dec. 2003), pp. 1039–1069. ISSN: 1532-4435.

[7] Junling Hu and Michael P. Wellman. "Nash Q-Learning for General-Sum Stochastic Games". In: *J. Mach. Learn. Res.* 4.null (Dec. 2003), pp. 1039–1069. ISSN: 1532-4435.

[8] Michael L. Littman. "Markov Games as a Framework for Multi-Agent Reinforcement Learning". In: *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*. ICML'94. New Brunswick, NJ, USA: Morgan Kaufmann Publishers Inc., 1994, pp. 157–163. ISBN: 1558603352.

[9] Jue Liu, Zhan Pang, and Linggang Qi. "Dynamic pricing and inventory management with demand learning: A bayesian approach". In: *Computers Operations Research* 124 (2020), p. 105078. ISSN: 0305-0548. DOI: https://doi.org/10.1016/j.cor.2020.105078. URL: https://www.sciencedirect.com/science/article/pii/S0305054820301957.

[10] Larkin Liu. "Approximate Nash Equilibrium Learning for n-Player Markov Games in Dynamic Pricing". In: *arXiv preprint arXiv:2207.06492* (2022).

[11] Larkin Liu, Richard Downe, and Joshua Reid. "Multi-armed bandit strategies for non-stationary reward distributions and delayed feedback processes". In: *arXiv preprint arXiv:1902.08593* (2019).

[12] Larkin Liu and Jun Tao Luo. *An Extensible and Modular Design and Implementation of Monte Carlo Tree Search for the JVM*. 2021. arXiv: 2108.10061 [cs.LG].

[13] Larkin Liu and Jun Tao Luo. "mctreesearch4j: A Monte Carlo Tree Search Implementation for the JVM". In: *Journal of Open Source Software* 7.70 (2022), p. 3804.

References II

[14] Robert Mattila et al. "Computing monotone policies for Markov decision processes: a nearly-isotonic penalty approach **This work was partially supported by the Swedish Research Council under contract 2016-06079 and the Linnaeus Center ACCESS at KTH.". In: *IFAC-PapersOnLine* 50.1 (2017). 20th IFAC World Congress, pp. 8429–8434. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2017.08.1575. URL: https://www.sciencedirect.com/science/article/pii/S2405896317321705.

[15] Faris Mismar, Jinseok Choi, and Brian Evans. "A Framework for Automated Cellular Network Tuning With Reinforcement Learning". In: *IEEE Transactions on Communications* 67 (Oct. 2019), pp. 7152–7167. DOI: 10.1109/TCOMM.2019.2926715.

[16] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning". In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 00280836. URL: http://dx.doi.org/10.1038/nature14236.

[17] Markus Rabe. "Combining a discrete-event simulation model of a logistics network with deep reinforcement learning.". In: *MIC and MAEB 2017 Conference* (2017), pp. 765–774.

[18] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. arXiv: 1707.06347 [cs.LG].

[19] L. Silbermayr and S. Minner. "Dual Sourcing under Disruption Risk and Cost Improvement through Learning". In: *European Journal of Operational Research* 250.1 (2012), pp. 226–238. DOI: 10.1016/j.ejor.2015.09.017.

[20] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018. URL: http://incompleteideas.net/book/the-book-2nd.html.

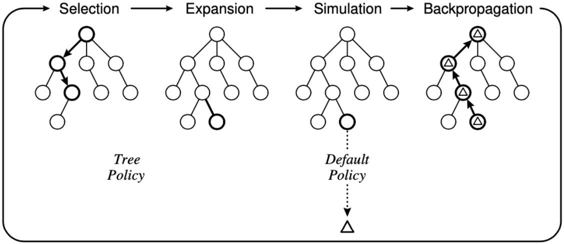
[21] Michael Wang. "Impacts of supply chain uncertainty and risk on the logistics performance". In: *Asia Pacific Journal of Marketing and Logistics* 30 (Apr. 2018), pp. 00–00. DOI: 10.1108/APJML-04-2017-0065.

[22] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning". In: *Machine Learning* 8.3 (May 1992), pp. 279–292. ISSN: 1573-0565. DOI: 10.1007/BF00992698. URL: https://doi.org/10.1007/BF00992698.

[23] David Wozabal, Nils Löhndorf, and Stefan Minner. "Optimizing Trading Decisions for Hydro Storage Systems Using Approximate Dual Dynamic Programming". In: *Operations Research* 61 (July 2013), pp. 810–823. DOI: 10.2307/23481798.

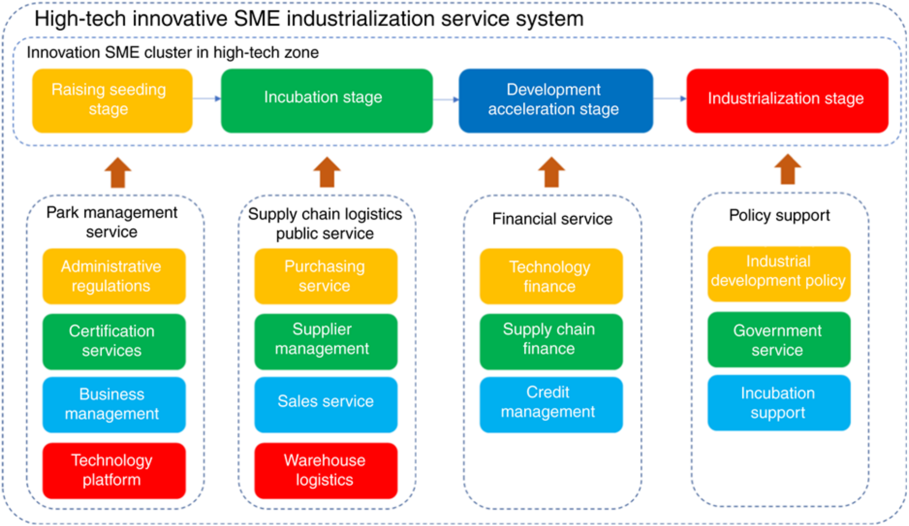
Monte Carlo Tree Search

- **Performance** how can we improve the performance of MCTS?
- **Application** in what ways can we apply performant implementations of MCTS (ie. [12]) on real world logistical problems?



Outline of MCTS - from Browne:2012.

Background - Supply Chain Complexity



Example of a modern Small-Midsize Enterprise service supply chain. From (Chen et al 2019) [4]