An Introduction to Reinforcement Learning and its Application in Supply Chain: Vehicle Routing Problem & Beer Game Problem

Afshin Oroojlooy*

*Reinforcement Learning Reseach Scientist, Artificial Intelligient and Machine Learning Divison, SAS Institute

IIIEC 2021, Feb 2021

Rules!

- Very very brief overview of the topic.
- There is no such thing as a stupid question; Stop me whenever you have any Question!

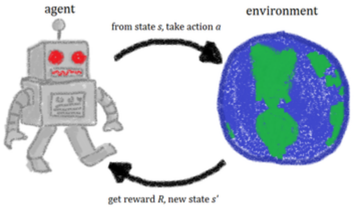
Human Decision Making System

- Interpret rich sensory inputs,

- Uses brain to choose action through utilizing memory through a complex procedures.

What is Reinforcement Learning

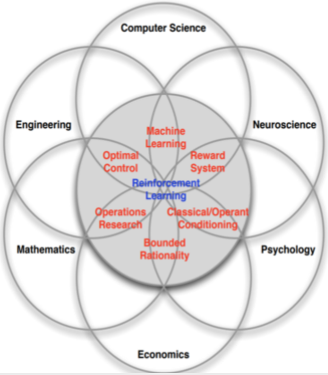
- Reinforcement Learning (RL) is a branch of Machine Learning (ML) concerned with decision making,
    - "Learning to make decisions based on trial and errors". (David Silver)


agent
environment
from state s, take action a
get reward R, new state s'

- Can be formalized as a Markov Decision Process (MDP).

Connection to Other Fields

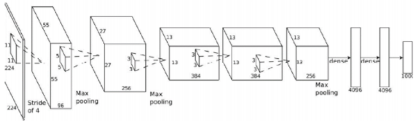
- Computer Science: Designing tools and Algorithms,
- Engineering: Optimal control in a sequential decision making problem,
- Mathematics: The maths of optimal control, mainly in operations research,
- Economics: Decision making in human-involved processes or tasks,
- Neuroscience: Discovering how the human/animal brain makes decisions.

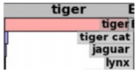


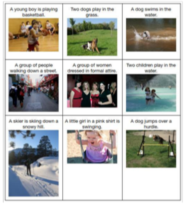
Supervised/Semi-Supervised/Unsupervised Learning vs RL

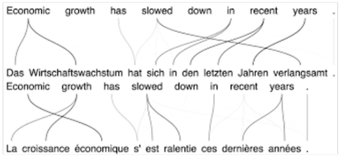
- Supervised learning:
  - There are label for each observation, e.g., image classification, natural language processing, etc.,
  - Not a sequential decision making problem.
- Semi-Supervised learning:
  - There are labeled data, some unlabeled data,
  - Not sequential.
- Unsupervised learning:
  - No label is available,
  - Goal is to find some pattern, clustering data, etc.
- RL training:
  - Sequential decision making,
  - Actions affect the future,
  - No label.

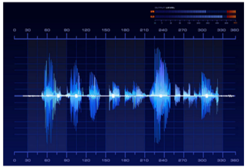
Supervised Example



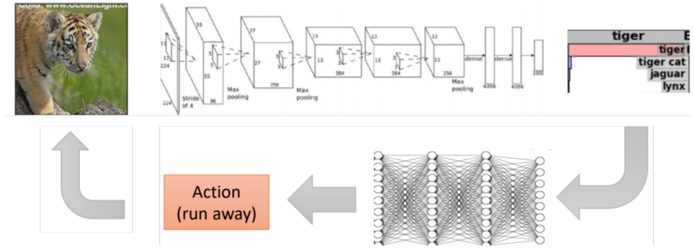



A young boy is playing basketball.
Two dogs play in the grass.
A dog swims in the water.
A group of people walking down a street.
A group of women dressed in formal attire.
Two children play in the water.
A skier is skiing down a snowy hill.
A little girl in a pink shirt is swinging.
A dog jumps over a hurdle.


Economic growth has slowed down in recent years .
Das Wirtschaftswachstum hat sich in den letzten Jahren verlangsamt .
Economic growth has slowed down in recent years .
La croissance économique s' est ralentie ces dernières années .

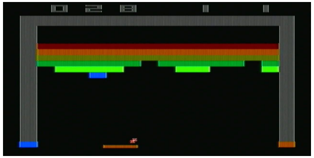
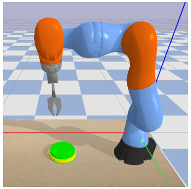


DRL Example


Stride of 4
Max pooling
Max pooling
Max pooling
dense
dense
tiger
tiger
tiger cat
jaguar
lynx
Action
(run away)

Current Applications of DRL

- So popular in Games!,
- Vastly used in Robotics,
- Autonomous driving,
- Ad recommendation,
- Combinatorial problems,

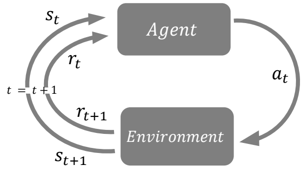



DRL for Autonomous Driving Examples

- Single-lane ring (0 AVs, 22 human-driven vehicles)
- Single-lane ring (1 AV, 21 human-driven vehicles)
- Single Junction Loop (0 AVs, 14 human-driven vehicles)
- Single Junction Loop (1 AVs, 13 human-driven vehicles)
- Single Junction Loop (14 AVs, 0 human-driven vehicles)

Markov Decision Process

- Consider process:



- State $s_t \in \mathcal{S}$
- Action $a_t \in \mathcal{A}(s_t)$
- Reward r_t(used interchangeably $r_{a_t}(s_t, s_{t+1})$ and $r(s_t, a_t)) \in \mathbb{R}$
- Next state $s_{t+1} \in \mathcal{S}$
- (I use s, s‘ interchangeably for state and next state.)
- Assume an initial state distribution $p_1(s_1)$.

Markov Decision Process

- Transition probability matrix $P_a(s, s`) = \Pr(s_{t+1} = s` \mid s_t = s, a_t = a)$
- Reward matrix $R_a(s, s`)$.
- The agent follows policy π_t to take action $a_t = \pi_t(s_t)$.
- This process is a **Markov Decision Process (MDP)** if:
    - A stationary transition distribution with conditional probability $p(s_{t+1}|s_t, a_t)$ satisfies the Markov property, i.e.,

$$p(s_{t+1}|s_1, a_1, \dots, s_t, a_t) = p(s_{t+1}|s_t, a_t)$$

      for any trajectory

$$\tau_{1:T} = s_1, a_1, r_1, \dots, s_T, a_T, r_T$$

      over $\mathcal{S} \times \mathcal{A} \times \mathbb{R}$, such that $r_t : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.
    - The state is fully known for the agent in each time-step.

Goal

- The goal is to maximize the expected discounted sum of the rewards r_t, when the systems runs for an infinite horizon:
$$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$$
and we define R_t^γ accordingly:
$$R_t^\gamma = \sum_{i=t}^{\infty} \gamma^{i-t} r(s_i, a_i) (0 < \gamma < 1)$$
- So, need to find policy $\pi : \mathcal{S} \to \mathcal{A}$ that maximizes $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_{a_t}(s_t, s_{t+1})\right]$.
- Given the policy with parameters θ, the goal can be written as:
$$J(\pi_\theta) = \mathbb{E}\left[R_1 | \pi_\theta\right]$$

.

Bellman Equation

- Consider Value of each state as: $V^{\pi}(s) = \mathbb{E}\left[R_1^{\gamma} | S_1 = s; \pi\right]$
- The Bellman Equation through an iterative process obtains the value of each state:

$$V^{\pi}(s) = r(s, \pi(s)) + \gamma \sum_{s'} p(s'|s, \pi(s)) V^{\pi}(s')$$

- Then, the optimal policy can be obtained:

$$V^{\pi^*}(s) = \max_a \left\{ r(s, a) + \gamma \sum_{s'} p(s'|s, a) V^{\pi^*}(s') \right\}$$

- Issues:
  - Need the knowledge of Transition probability matrix $P_a(s, s')$ and the Reward matrix $R_a(s, s')$.
  - Curse of dimensionality.

Other Approaches

Two approaches are common in practice to get a policy:

- Value iteration:
  - Learns value of each state or the value of each action for each state.
  - Value function: the expected sum of discounted rewards,
    - $V^{\pi}(s) = \mathbb{E}\left[R_1^{\gamma} | S_1 = s; \pi\right]$
  - Q-value:
    - $Q(s, a) = \mathbb{E}\left[R_1^{\gamma} | S_1 = a, A_1 = a; \pi\right]$
- Policy Iteration:
  - Learns a stochastic policy $\pi_\theta : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ maps any state $s \in \mathcal{S}$ into $\mathcal{P}(\mathcal{A})$ which measures the probability of taking each action $a \in \mathcal{A}$, and θ determines the policy based on the state input.
  - Basically, $\pi_\theta(a_t | s_t)$ provides the conditional probability of taking action a_t given s_t and $R_t^{\gamma} = \sum_{i=t}^{\infty} \gamma^{i-t} r(s_i, a_i)$ $(0 < \gamma < 1)$.

Approximated Solutions

- Classical value/policy iteration guarantee the optimality.
- Again the curse of dimensionality.

Approximated Solutions

- Classical value/policy iteration guarantee the optimality.
- Again the curse of dimensionality.
- Approximating the value or policy by value function or policy.
- Linear approximator:
    - Proof of convergence to the local optimal.
    - Not powerful enough.
- Non-linear approximator:
    - There is no proof of convergence.
    - Usually powerful enough.
- Same status until recently,
- Emergence of deep learning an cheap computational power!

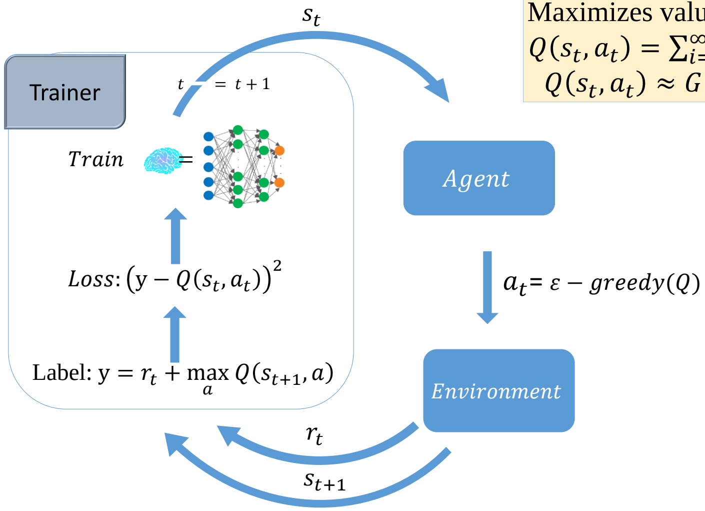
INNOVATIONS IN
The microbiome
nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
LEARNING CURVE
Self-taught AI software attains human-level performance in video games
PAGES 486 & 529
EPIDEMIOLOGY
SHARE DATA IN OUTBREAKS
Forge open access to sequences and more
PAGE 477
COSMOLOGY
A GIANT IN THE EARLY UNIVERSE
A supermassive black hole at a redshift of 6.3
PAGES 490 & 512
QUANTUM PHYSICS
TELEPORTATION FOR TWO
Transferring two properties of a single photon
PAGES 491 & 516
NATURE.COM/NATURE
26 February 2015 £10
Vol. 518, No. 7540

nature
THE INTERNATIONAL WEEKLY JOURNAL OF SCIENCE
At last — a computer program that can beat a champion Go player PAGE 484
ALL SYSTEMS GO
CONSERVATION
SONGBIRDS À LA CARTE
Illegal harvest of millions of Mediterranean birds
PAGE 452
RESEARCH ETHICS
SAFEGUARD TRANSPARENCY
Don't let openness backfire on individuals
PAGE 459
POPULAR SCIENCE
WHEN GENES GOT 'SELFISH'
Dawkins's calling card forty years on
PAGE 462
NATURE.COM/NATURE

Deep Q-Network

- Train a neural network with the Q-values for each action in the output.
- Utilizes the *experience replay* buffer to achieve i.i.d samples.
  - Since the selected mini-batch involves samples from previous episodes, DQN is a off-policy algorithm.
- Introduced the target network to stabilize the training.
- Uses ϵ-greedy algorithm for the exploration.
- Achieved human level control on most of 49 Atari-2600 games.

Deep Q-Network at a Glance


s_t
Trainer
$t = t+1$
Train
Agent
$a_t = \varepsilon - greedy(Q)$
Loss: $(y - Q(s_t, a_t))^2$
Label: $y = r_t + \max_a Q(s_{t+1}, a)$
Environment
r_t
s_{t+1}

Maximizes value:

$$Q(s_t, a_t) = \sum_{i=0}^{\infty} \gamma^i r_{s,a}^{t+i}$$

$$Q(s_t, a_t) \approx G(s_t, a_t)$$

Deep Q-Network

Algorithm 1 DQN Algorithm (nature (2015))

```
Initialize replay memory $D$ to capacity $N$, action-value function $Q$ with random weights $\theta$, and
Target function $\hat{Q}$ with weights $\theta^- = \theta$
for episode=1, ...,M do
    Initialize $s_1$ and preprocessed $\phi_1 = \phi(s_1)$
    for t=1, ...,T do
        With probability $\epsilon$ select $a_t$ randomly
        Otherwise select $a_t \sim \text{argmax}_a Q(\phi(s_t), a; \theta)$
        Execute $a_t$ in emulator, observe $r_t$
        Set $s_{t+1} = s_t$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$in $D$
        Sample random minibatch of $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$from $D$
        Set $y_j = \begin{cases} r_j & \text{If episode terminates at step } j+1 \\ r_j + \gamma \max_u \hat{Q}(\phi_{j+1}, u; \theta^-) & \text{Otherwise} \end{cases}$
        Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters $\theta$.
        Every $C$ steps reset $\hat{Q} = Q$
    end for
end for
```

Policy Gradient

- The goal is to learn the policy directly, i.e., learn a stochastic policy $\pi(a|s)$.
    - This function provides the mapping of the state to the probability of choosing each action.
- This is in nature a on-policy algorithm, resulting in an unbiased estimator of the gradient.
- With Value based approaches, like DQN:
    - One cannot solve continues action control problem.
    - Not possible to learn the optimal policy if it is stochastic, like Scissor/Paper/Stone.
- DQN usually converges fast, although very sensitive to the hyper-parameters.
- Policy gradient algorithms usually need longer time to converge, though not much dependent on the hyper-parameter values.

How to Train?

- Assume state-action trajectory $\tau = s_0, a_0, s_1, \dots$ is available.
- $G_\tau = \sum_{t=0}^{T} \gamma^t r(s_t, a_t)$, $G_t = \sum_{ti=t}^{T} \gamma^{ti} r(s_{ti}, a_{ti})$, and $U(\theta) = E_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) | \pi_\theta \right]$ where to obtain $U(\theta)$ we use

$$
\begin{aligned}
p_\theta(\tau) &= p_\theta(s_0, a_0, \dots, s_T, a_T) && \text{(4.1a)} \\
&= p(s_0) \Pi_{t=0}^{T} \pi_\theta(a_t | s_t) p(s_{t+1} | s_t, a_t) && \text{(4.1b)}
\end{aligned}
$$

Apparently, $G(\tau)$ is equal to $G(0)$. The goal is to get:

$$
\theta^* = \underset{\theta}{\operatorname{argmax}} \, E_{\tau \sim p_\theta(\tau)} \left[\sum_{t=0}^{T} \gamma^t r(s_t, a_t) | \pi_\theta \right]
$$

How to Train?

To get the maximum value of $U(\theta)$, we take gradient with respect to θ:

$$\nabla_\theta U(\theta) = \nabla_\theta E_\tau \sum_{t=0}^{T} \gamma^t r(s_t, a_t) \quad \text{definition of } U(\theta) \tag{4.2a}$$

$$= \nabla_\theta \sum_\tau p(\tau;\theta) G_\tau \quad \text{rewrite the expectation} \tag{4.2b}$$

$$= \sum_\tau \nabla_\theta p(\tau;\theta) G_\tau \quad \text{swap sum and gradient} \tag{4.2c}$$

$$= \sum_\tau p(\tau;\theta) \frac{\nabla_\theta p(\tau;\theta)}{p(\tau;\theta)} G_\tau \quad \text{both multiply and divide by } p(\tau;\theta) \tag{4.2d}$$

$$= \sum_\tau p(\tau;\theta) \nabla_\theta \log p(\tau;\theta) G_\tau \quad \text{use the fact that } \nabla_\theta \log(x) = \frac{1}{x}\nabla_\theta x \tag{4.2e}$$

$$= E_\tau[G_\tau \nabla_\theta \log p(\tau;\theta)] \quad \text{definition of expectation} \tag{4.2f}$$

How to Train?

Now consider $\log p(\tau;\theta)$.

$$
\begin{aligned}
\nabla_\theta \log p(\tau;\theta) &= && (4.3a)\\
&= \nabla_\theta \log \left[\Pi_{t=0}^{T} P(s_{t+1}|s_t,a_t).\pi_\theta(a_t|s_t)\right] && (4.3b)\\
&= \nabla_\theta \left[\sum_{t=0}^{T} \log P(s_{t+1}|s_t,a_t) + \sum_{t=0}^{T} \log \pi_\theta(a_t|s_t)\right] && (4.3c)\\
&= \sum_{t=0}^{T} \nabla_\theta \log P(s_{t+1}|s_t,a_t) + \nabla_\theta \sum_{t=0}^{T} \log \pi_\theta(a_t|s_t) && (4.3d)\\
&= \nabla_\theta \sum_{t=0}^{T} \log \pi_\theta(a_t|s_t) && (4.3e)
\end{aligned}
$$

in which (4.3b) uses the definition of $p(\tau;\theta)$, and (4.3c) uses the fact that $\log xy = \log x + \log y$.

How to Train?

Then, by embedding (4.3e) into (4.2f), one can conclude that:

$$\nabla_\theta U(\theta) = E_\tau \left[\nabla_\theta \log p(\tau;\theta) G_\tau\right] = E_\tau \left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\right)\left(\sum_{t=0}^{T} \gamma^t r_t(s_t, a_t)\right)\right] \tag{4.4}$$

Since the distribution of $\tau = s_0, a_0, r_0, \ldots, a_T, s_T, r_T$ is not known, we cannot obtain the expectation directly. To address this issue, one can use sampling to obtain an estimation of the expectation, which here is an unbiased estimator of the gradient. So, given m trajectories of samples, the gradients can be obtained:

How to Train?

$$E_\tau \left[\nabla_\theta \log p(\tau_i) G_\tau\right] \sim \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log p(\tau_i) G(\tau_i) \tag{4.5a}$$

$$\sim \frac{1}{m}\sum_{i=1}^{m} \left[\left(\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\right)\left(\sum_{t=0}^{T} \gamma^t r_t(s_t, a_t)\right)\right] \tag{4.5b}$$

REINFORCE Algorithm

Algorithm 3 REINFORCE Algorithm

```
1: initialize the actor network with random weights $\theta$ and critic network with random weights $\phi$
2: for $iteration = 1, 2, \cdots$ do
3:     reset gradients: $d\theta \leftarrow 0$, $d\phi \leftarrow 0$
4:     sample $N$ instances according to $\Phi_{\mathcal{M}}$
5:     for $n = 1, \cdots, N$ do
6:         initialize step counter $t \leftarrow 0$
7:         repeat
8:             choose $y_{t+1}^n$ according to the distribution $P(y_{t+1}^n | Y_t^n, X_t^n)$
9:             observe new state $X_{t+1}^n$
10:            $t \leftarrow t + 1$
11:        until termination condition is satisfied
12:        compute reward $R^n = R(Y^n, X_0^n)$
13:    end for
14:    $d\theta \leftarrow \frac{1}{N} \sum_{n=1}^{N} \left(R^n - V(X_0^n; \phi)\right) \nabla_\theta \log P(Y^n | X_0^n)$
15:    $d\phi \leftarrow \frac{1}{N} \sum_{n=1}^{N} \nabla_\phi \left(R^n - V(X_0^n; \phi)\right)^2$
16:    update $\theta$ using $d\theta$ and $\phi$ using $d\phi$.
17: end for
```

REINFORCE Algorithm

Back to Supply Chain!

Our Goal

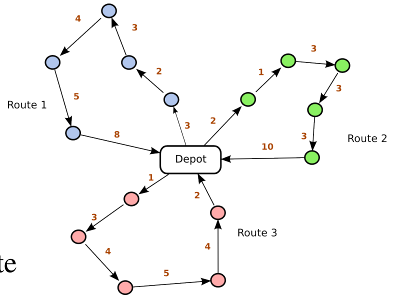
- Use recent advances in Machine Learning (ML) and Reinforcement Learning (RL) to develop a new approach for solving combinatorial optimization problems.
- Approach these problems very differently from classical heuristics.
- Study the and *Beer Game Vehicle Routing Problem* (VRP) as building blocks.
- Extend this infrastructure to more complicated settings.

Reinforcement Learning for Solving the Vehicle Routing Problem

Vehicle Routing Problem (VRP)

- Well-known combinatorial optimization problem
- Multiple customers with different demands
- Find a set of routes, all beginning and ending at a given node (called the depot)
- **Objective**: Minimize the tour length to satisfy all of the demands

- Solving VRP optimally takes too long, even for moderate instances
- Today's “hard” VRP instances involve hundreds of nodes
    - In contrast, hard TSP instances have tens of thousands of nodes

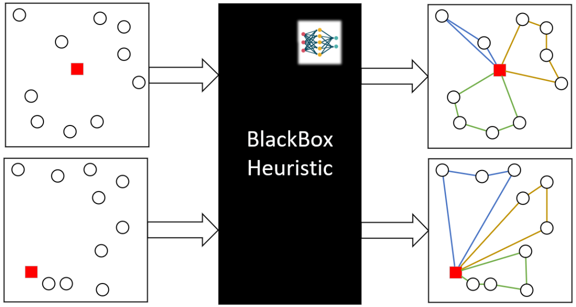


Main Requirements

- Better than existing heuristics and *close to optimality*
- *Fast* (in a few seconds), comparable to heuristics
- *Generalize* well
- No hand-engineering

How to Satisfy Requirements?

- Train a solver (meta-algorithm)
- We train a single model for a given instance type.
- Once trained, our model can solve *any* instance of the same type nearly instantaneously.



Deep RL for combinatorial optimization problems

- Addressing combinatorial optimization problems:
    - Learning optimal TSP from optimal solutions; supervised learning and pointer network [Vinyals et al., 2015]

Deep RL for combinatorial optimization problems

- Addressing combinatorial optimization problems:
  - Learning optimal TSP from optimal solutions; supervised learning and pointer network [Vinyals et al., 2015]
  - Bello et al. [2016] use RL and pointer network. They address two main issues:
    - There are no ground truth labels (i.e., the optimal solution)
    - Cannot generalize to larger size problems

Deep RL for combinatorial optimization problems

- Addressing combinatorial optimization problems:
    - Learning optimal TSP from optimal solutions; supervised learning and pointer network [Vinyals et al., 2015]
    - Bello et al. [2016] use RL and pointer network. They address two main issues:
        - There are no ground truth labels (i.e., the optimal solution)
        - Cannot generalize to larger size problems
    - Solving optimization problems over graphs [Dai et al., 2017]

Deep RL for combinatorial optimization problems

- Addressing combinatorial optimization problems:
    - Learning optimal TSP from optimal solutions; supervised learning and pointer network [Vinyals et al., 2015]
    - Bello et al. [2016] use RL and pointer network. They address two main issues:
        - There are no ground truth labels (i.e., the optimal solution)
        - Cannot generalize to larger size problems
    - Solving optimization problems over graphs [Dai et al., 2017]
- We have generalized their framework to include VRP

Our Algorithm

- Static variables:
  - Locations: both depot and customer nodes
- Dynamic variables:
  - Remaining demand of each node at time t, d_i^t
  - Remaining load if the vehicle visits node i, i.e.

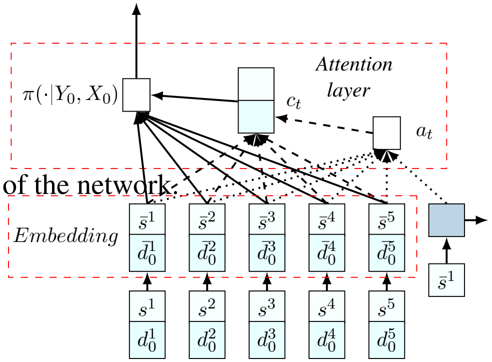
$$\min\{0, l^t - d_i^t\}$$

- *Reward*: tour length
- *Masking*: it acts like constraints
  - Customers nodes with zero demand
  - Customers nodes if the remained load is zero
  - Customers with demands greater than the current vehicle load
- Terminate decoding when there is no more unsatisfied demand

Attention Mechanism

- Similar to Bahdanau et al. [2014]
- *Difference*: Take into account the dynamic elements in the attention
- *Example*: In VRP, demands are only used in the attention
- *Interpretation*: When attending in different nodes, consider the dynamic elements

We used REINFORCE algorithm to train the weights of the network.



Attention Mechanism

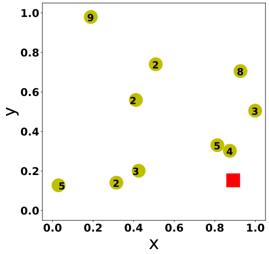
- $\bar{s}^i \in \mathbb{R}^D$: Embedded static input i.
- $\bar{d}^i \in \mathbb{R}^D$: Embedded dynamic input i.
- $h^t \in \mathbb{R}^D$: memory state of the recurrent neural network (RNN) cell at decoding step t.
- At decode step i,

$$score(s^i, d^i, h^t) = v_a^T \; tanh \left(W_a [\bar{s}^i ; \bar{d}^i ; h^t] \right) \qquad (\text{ concat})$$

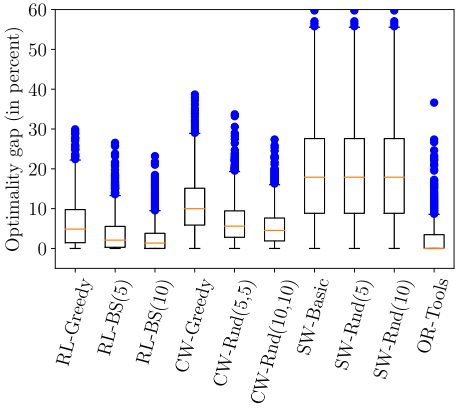
$$a_i(s^i, d^i, h^t) = \frac{exp(score(s^i, d^i, h^t))}{\sum_k exp(score(s^k, d^k, h^t))}$$

Experiment 1: Capacitated VRP

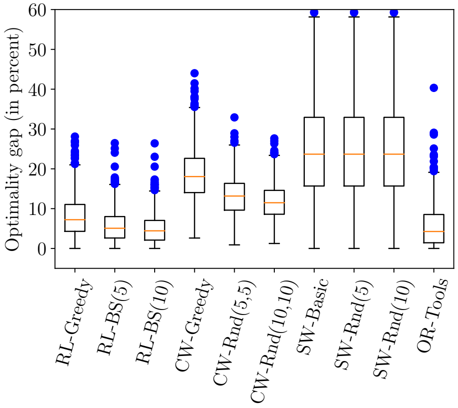
- A single capacitated vehicle is responsible for delivering items to multiple customer nodes
- It must return to the depot to refill when it runs out
- Points are randomly generated in $[0, 1] \times [0, 1]$
- Demand of each node is randomly chosen from $\{1..9\}$



Experiment 1: Tour Length Comparison



(a) Comparison for VRP10



(b) Comparison for VRP20

Experiment 1: Tour Length Comparison - cnt'd

	RL-Greedy	RL-BS(5)	RL-BS(10)	CW-Greedy	CW-Rnd(5,5)	CW-Rnd(10,10)	SW-Basic	SW-Rnd(5)	SW-Rnd(10)	OR-Tools
RL-Greedy		12.2	7.2	99.4	97.2	96.3	97.9	97.9	97.9	41.5
RL-BS(5)	85.8		12.5	99.7	99.0	98.7	99.1	99.1	99.1	54.6
RL-BS(10)	91.9	57.7		99.8	99.4	99.2	99.3	99.3	99.3	60.2
CW-Greedy	0.6	0.3	0.2		0.0	0.0	68.9	68.9	68.9	1.0
CW-Rnd(5,5)	2.8	1.0	0.6	92.2		30.4	84.5	84.5	84.5	3.5
CW-Rnd(10,10)	3.7	1.3	0.8	97.5	68.0		86.8	86.8	86.8	4.7
SW-Basic	2.1	0.9	0.7	31.1	15.5	13.2		0.0	0.0	1.4
SW-Rnd(5)	2.1	0.9	0.7	31.1	15.5	13.2	0.0		0.0	1.4
SW-Rnd(10)	2.1	0.9	0.7	31.1	15.5	13.2	0.0	0.0		1.4
OR-Tools	58.5	45.4	39.8	99.0	96.5	95.3	98.6	98.6	98.6	

(a) Comparison for VRP50

	RL-Greedy	RL-BS(5)	RL-BS(10)	CW-Greedy	CW-Rnd(5,5)	CW-Rnd(10,10)	SW-Basic	SW-Rnd(5)	SW-Rnd(10)	OR-Tools
RL-Greedy		25.4	20.8	99.9	99.8	99.7	99.5	99.5	99.5	44.4
RL-BS(5)	74.4		35.3	100.0	100.0	99.9	100.0	100.0	100.0	56.6
RL-BS(10)	79.2	61.6		100.0	100.0	100.0	99.8	99.8	99.8	62.2
CW-Greedy	0.1	0.0	0.0		0.0	0.0	65.2	65.2	65.2	0.0
CW-Rnd(5,5)	0.2	0.0	0.0	92.6		32.7	82.0	82.0	82.0	0.7
CW-Rnd(10,10)	0.3	0.1	0.0	97.2	65.8		85.4	85.4	85.4	0.8
SW-Basic	0.5	0.0	0.2	34.8	18.0	14.6		0.0	0.0	0.0
SW-Rnd(5)	0.5	0.0	0.2	34.8	18.0	14.6	0.0		0.0	0.0
SW-Rnd(10)	0.5	0.0	0.2	34.8	18.0	14.6	0.0	0.0		0.0
OR-Tools	55.6	43.4	37.8	100.0	99.3	99.2	100.0	100.0	100.0	

(b) Comparison for VRP100

Experiment 1: Solution Time

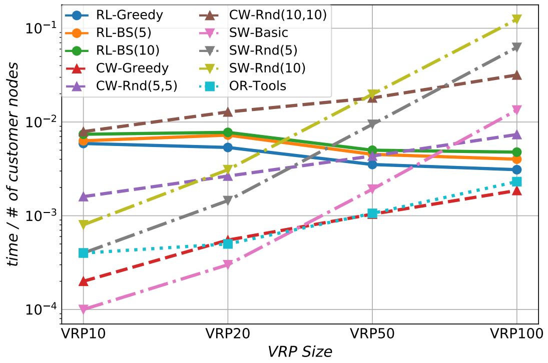


Figure: Log of ratio of solution time to the number of customer nodes using different algorithms.

Scalability: 100 $\times$ faster when you process in batch

Experiment 1: How the Decoding Works?

Other Experiments

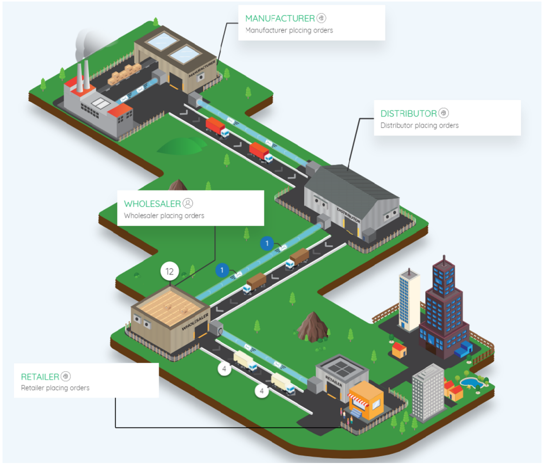
- VRPs with Split Demands: Demand of a customer can be satisfied at different routes
- Stochastic demand.
- See the paper for more details!

Conclusion & Discussion

- A simple framework: only requires the reward and feasibility checks at each step.
- Robust to problem changes
- Significantly better than well-known classical heuristics
- Competitive with OR-Tools
- A fast framework

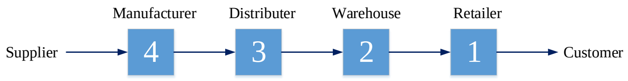
A Deep Q-Network for the Beer Game: Reinforcement Learning for Inventory Optimization

Beer Game


MANUFACTURER
Manufacturer placing orders
DISTRIBUTOR
Distributor placing orders
WHOLESALER
Wholesaler placing orders
RETAILER
Retailer placing orders
12
1
1
4
4

Supply Chain Point of View

- Serial network.
- Stochastic demand.
- Deterministic lead times.
- Total lead time is stochastic due to stockouts upstream.



Supply Chain Point of View

- Choose order quantities q_i to minimize:

$$z = \sum_{t=1}^{T} \sum_{i=1}^{4} c_h^i (IL_t^i)^+ + c_p^i (IL_t^i)^-$$

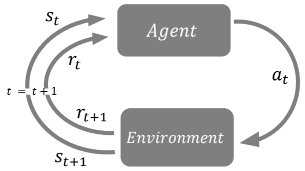
  where:
  - i is the agent index,
  - $t = 1, \ldots, T$ is the index of the time periods,
  - T is the (random) time horizon of the game,
  - IL_t^i is the inventory level of agent i in period t.
  - c_p^i and c_h^i are shortage and holding costs for agent i.
- For a given **known** demand distribution, and when $c_p^i = 0, \forall i = 2, 3, 4$ base-stock policy (order to up level) is optimal [Clark and Scarf, 1960].
- There is no algorithm for general shortage costs.

Current Approaches

- *Base stock policy [Clark and Scarf, 1960]*
  - Does not provide solution when other players play irrationally.
- *Anchoring and adjustment formulas [Sterman, 1989, Croson and Donohue, 2006].*
  - Based on state variables.
  - Aims to model human behavior, not minimize cost.
- *Meta-heuristic algorithms [Kimbrough et al., 2002].*
  - The agent are able to communicate.

Markov Decision Process

- Markov Decision Process (MDP).



- Multi-agent process.
- Infinite size of state and action spaces.
- Partial observation, POMDP.

Problem Category

- Beer game is a multi agent cooperative Dec-POMDP.
- NEXP Complete [Bernstein et al., 2002].

Deep Q-Network

- We propose an extension of Deep Q-Network (DQN) algorithm [Mnih et al., 2015] to efficiently play the beer game.
- DQN is a Reinforcement Learning (RL) algorithm to solve general MDPs.
- Effective in solving large-state-space problems.

Our Approach: Challenges

- DQN cannot be applied directly to the beer game.
  - Applied to solve single agent games and extended to two agents zero-sum games.
  - Beer game is a cooperative non-zero-sum game.
  - Applying DQN to beer game results in competitive game.
  - The obtained policy fails to minimize total cost.
- Infinite size of action space.

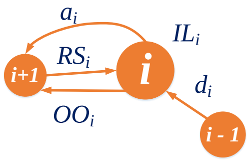
DQN Requirements

Any RL needs a well defined MDP.

- Definition of state,
- Definition of action,
- Reward function.

DQN Algorithm: State Space

- Agent i in time step t has observation:



$$o_t^i = \left[(IL_j^i, OO_j^i, AO_j^i, AS_j^i, a_j^i)\right]_{j=1}^{t}.$$

  - IL_t^i: the inventory level
  - OO_t^i: the on-order items at agent i, i.e., the items that have been ordered from agent $i+1$ but not received yet
  - AO_t^i: the demand/order received from agent $i-1$
  - AS_t^i: the items received from agent $i+1$
  - a_t^i: the action agent i takes
- Capture only the last m periods, i.e.:

$$s_t^i = \left[(IL_j^i, OO_j^i, AO_j^i, AS_j^i, a_j^i)\right]_{j=t-m+1}^{t}.$$

DQN Algorithm: Action Space and Reward Function

- Action Space
  - Order can be any value in $[0, \infty)$
  - DNN provides the Q-value of all possible actions.
  - Infinite size is not practical.
  - Use a $d + x$ rule to select the order quantity.
    - $x \in [a_l, a_u], a_l, a_u \in \mathbb{Z}$
- Reward Function
  - Observe s_t^i and take action a_t^i.
  - Need to know r_t^i to measure quality a_t^i.
  - Use IL_{t+1}^i to obtain r_t^i (shortage or holding costs).

DQN Algorithm: Feedback Scheme

- Game ends: Agents are made aware of the total reward, z.
- Provide feedback about how they played.
- Update observed reward in all $e_t^i = (s_t^i, a_t^i, r_t^i, s_{t+1}^i)$,

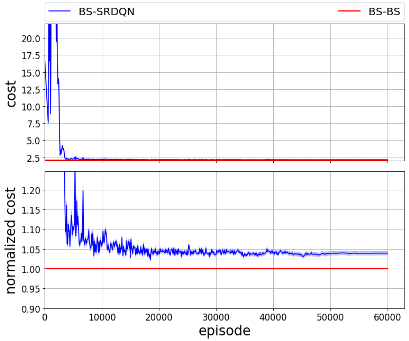
$$r_t^i = r_t^i + \beta_i f(z), \qquad \forall t \in \{1, \dots, T\},$$

  where β_i is a regularization coefficient for agent i.
- Learn the minimum total cost.

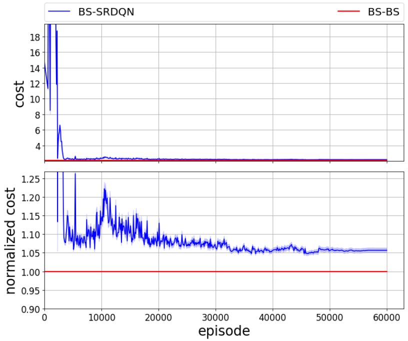
Numerical Experiment

- Basic Case,
  - $d_0^t \in \mathbb{U}[0,2]$, $A(s_t) = \{-2,-1,0,1,2\}$.
  - $c_p = [2,0,0,0]$ and $c_h = [2,2,2,2]$.
  - Two types of co-players:
    - Rational players, who follow base-stock policy,
    - Irrational players, who follow Sterman formula.
- Three cases from the Literature.
- A real world dataset.

Results: Basic Cases, BS co-player, R, W

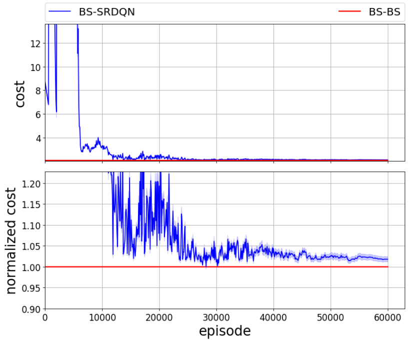


(a) DQN plays retailer

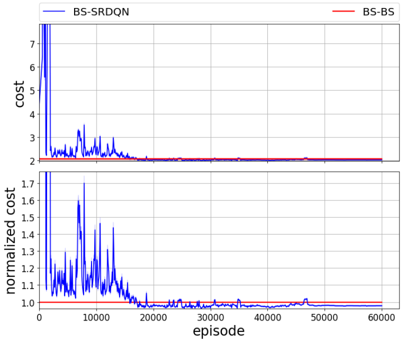


(b) DQN plays warehouse

Results: Basic Cases, BS co-player, D, M

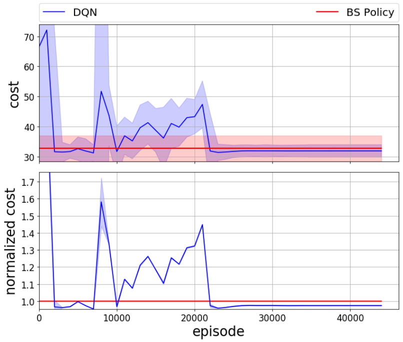


(a) DQN plays distributor

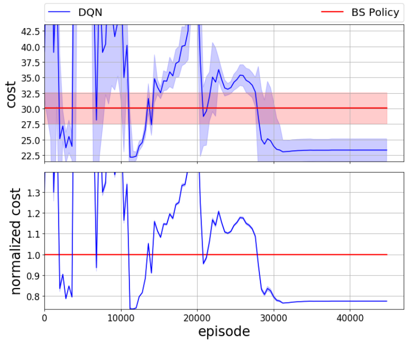


(b) DQN plays manufacturer

Results: DQN Plays with Sterman Formula, R, W

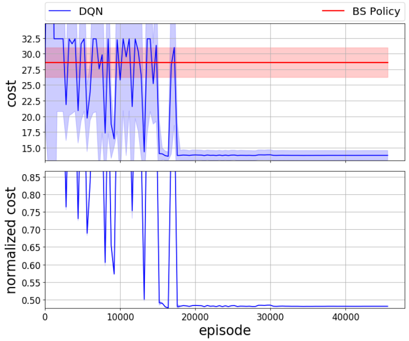


(a) DQN plays retailer

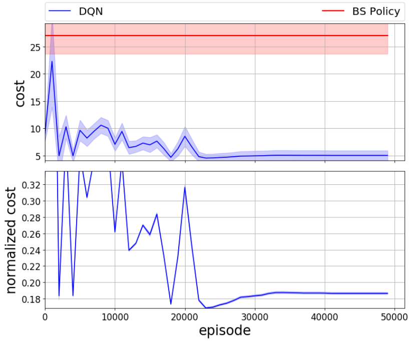


(b) DQN plays warehouse

Results: DQN Plays with Sterman Formula, D, M



(a) DQN plays distributor



(b) DQN plays manufacturer

Results: Literature Cases

Three cases from the Literature.

- Demand distribution and action space:
    - $d_0^t \in \mathbb{U}[0,8]$, $A(s_t) = \{-8,\ldots,8\}$ [Croson and Donohue, 2006]
    - $d_0^t \in \mathbb{N}(10,2^2)$, $A(s_t) = \{-5,\ldots,5\}$ adapted from [Chen and Samroengraja, 2000]
    - $d_0^t \in C(4,8)$, $A(s_t) = \{-8,\ldots,8\}$ [Sterman, 1989].
- Three types of co-players:
    - base-stock policy,
    - Sterman formula,
    - random policy.

Results: Literature Cases

Figure: Results of DQN playing with co-players who follow Sterman policy.

	Uniform			Normal			Classic		
	DQN	`Strm-BS`	Gap (%)	DQN	`Strm-BS`	Gap (%)	DQN	`Strm-BS`	Gap (%)
R	6.88	8.99	-23.45	9.98	10.67	-6.44	3.80	13.28	-71.41
W	5.90	9.53	-38.10	7.11	10.03	-29.06	2.85	8.17	-65.08
D	8.35	10.99	-23.98	8.49	13.83	-38.65	3.82	20.07	-80.96
M	12.36	13.90	-11.07	13.86	15.37	-9.82	15.80	19.96	-20.82
Average			-24.15			-20.99			-59.57

RL vs. BS

- Our algorithm works much better than BS when playing with Sterman.
- Obtains more than 30% improvement over BS in average.
- The importance is that Sterman is more or less like the way that a real human plays.
- We anticipate same performance with human playing.

Other Experiments

- Two real-world datasets.
- A very small dataset (100 observations).
- Transfer learning for speed up the training.

Results: New Real World Data-set

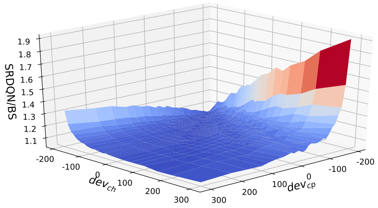
Figure: Results of real-world basket dataset.

		Category-6			Category-13			Category-22		
co-player	Agent	DQN	BS	Gap (%)	DQN	BS	Gap (%)	DQN	BS	Gap (%)
BS	R	40.69	39.52	2.96	63.66	58.06	9.65	15.12	14.52	4.10
	W	46.29	39.52	17.14	74.31	58.06	27.99	17.08	14.52	17.61
	D	45.51	39.52	15.16	59.80	58.06	3.00	15.03	14.52	3.47
	M	45.16	39.52	14.26	59.34	58.06	2.21	16.21	14.52	11.63
	Average			12.38			10.71			9.20
Strm	R	82.32	79.68	3.31	133.27	153.56	-13.22	21.65	22.54	-3.94
	W	148.25	179.69	-17.50	247.99	311.90	-20.49	30.96	37.76	-18.02
	D	179.53	205.31	-12.55	312.13	351.61	-11.23	42.38	41.86	1.25
	M	218.70	230.87	-5.27	375.98	384.47	-2.21	48.96	49.40	-0.89
	Average			-8.00			-11.79			-5.40

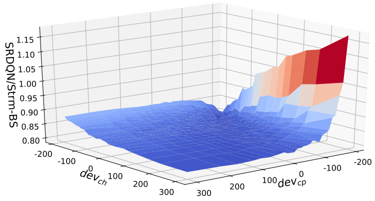
Sensitivity Analysis

- Check the robustness of the trained model to the changes of c_p and c_h.

Figure: Results of sensitivity analysis of cost, by perturbing c_p and c_h. Each sub-figure shows the result of the mentioned co-players policy when demand distribution is $N(10, 2)$.



(a) Base-stock



(b) Sterman

Conclusion

- Introduced an extension of DQN algorithm to solve the beer game problem.
- A feedback scheme is proposed.
- A Transfer learning model is proposed to reduce the training time.
- Numerical experiments show:
    - DQN performs well regardless of the way other agents play.
    - Learns to play close to optimal when others agents play with BS policy.
    - Provides smaller cost than BS policy when plays with irrational players.
    - Transfer learning makes the training $\sim$15 times faster.

Other Application

Other Applications

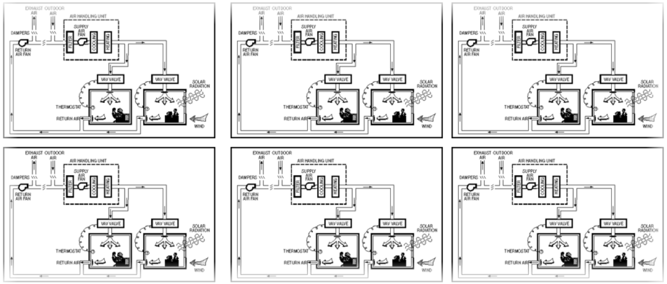
Application in Marketing

- Consider long-term interactions with customers
- 20-30% higher revenue compared to greedy methods

Application in Traffic Signal Control Problem

- Non linear mix-integer programming model.
- Not robust to the changes of the traffic fellow.
- One single model which works for any intersection with any traffic.

Application in Air Conditioning



- Better than periodic-review models.
- Considering long-term objectives results in smoother policies, e.g. less on-off

Application in Internet of Things

- All devices are connected in a connected distributed network.
- The goal is to minimize the long-term maintenance cost or maximize the availability.
- Applications in gas stations, production lines, or any system with many similar devices in a connected network.

Other Applications

- Scheduling (Jobshop, Flowshop)
- Automatic Maintenance Agent
- Max cut and graph coloring problem
- Knapsack Problem
- ...

Other Algorithms

- Discrete action space:
    - Double DQN, Dueling DQN, Double Dueling DQN,
    - Apex-DQN,
    - IMPALA,
- Both continues and discrete action spaces
    - Actor-Critic
    - Asynchronous Advantage Actor-critic (A3C)
    - Synchronous Advantage Actor-critic (A2C)
    - Trust Region Policy Optimization (TRPO)
    - Proximal Policy Optimization (PPO)
    - Soft Actor-Critic (SAC)
- Only continues control
    - Deep Deterministic Policy Gradient (DDPG)
    - Twin Delayed DDPG (TD3)

Extensions

- Batch-RL
    - No environment,
    - No exploration.
- Partially observed MDP (POMDP)
- Multi-Agent RL
    - Several Agents attend in the system,
    - Non-stationary environment!,
    - Communication, bandwidth limit,
    - Learn to communicate.

Top Conferences

- Not much of **journals** in this field.
- Mostly conference, quite competitive.
- NeurIPS, ICML, ICLR, AAAI, KDD,
    - Acceptance rate about 10-20%,
    - NeurIPS 2020 had around 12000 submissions,
    - You get at least 3 reviews,
    - Usually you 10 days to answer the questions and submit back a single page answers,
    - Double blind.
    - See the list at All-Ranking and CS-Ranking

Open Source Python Packages

Open Source Libraries

Open Source Python Packages for RL

- Environment
  - OpenAI gym environment.
- Algorithm
  - Ray (rise lab, University of California, Berkeley)
  - Horizon (Facebook)
  - TF-Agents(Google)
  - Acme (Deep Mind, Google)
  - Tensorforce
  - KerasRL
  - ...

Link to our Paper and Code

Reinforcement learning for solving the vehicle routing problem, NIPS 2018
github code: `https://github.com/OptMLGroup/VRP-RL`

A Deep Q-Network for the Beer Game ... (MSOM 2020) https://arxiv.org/abs/1708.05924
github code: `https://github.com/OptMLGroup/DeepBeerInventory-RL`
Opex Analytics Beer Game

AttendLight: Universal Attention-Based Reinforcement Learning Model for Traffic Signal Control, NeurIPS 2020

Online Reinforcement Learning with Applications in Customer Journey Optimization, NeurIPS RL Symposium 2017

References I

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of markov decision processes. *Mathematics of operations research*, 27(4):819–840, 2002.

Fangruo Chen and Rungson Samroengraja. The stationary beer game. *Production and Operations Management*, 9(1):19, 2000.

Andrew J Clark and Herbert Scarf. Optimal policies for a multi-echelon inventory problem. *Management science*, 6(4):475–490, 1960.

Rachel Croson and Karen Donohue. Behavioral causes of the bullwhip effect and the observed value of inventory information. *Management science*, 52(3):323–336, 2006.

References II

Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. *arXiv preprint arXiv:1704.01665*, 2017.

Steven O Kimbrough, Dong-Jun Wu, and Fang Zhong. Computers play the beer game: Can artificial agents manage supply chains? *Decision support systems*, 33(3):323–333, 2002.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Afshin Oroojlooyjadid, MohammadReza Nazari, Lawrence Snyder, and Martin Takáč. A deep q-network for the beer game with partial information. *arXiv preprint arXiv:1708.05924*, 2017.

Pentaho. Foodmart's database tables. `http://pentaho.dlpage.phi-integration.com/mondrian/mysql-foodmart-database`, 2008. Accessed: 2015-09-30.

John D Sterman. Modeling managerial behavior: Misperceptions of feedback in a dynamic decision making experiment. *Management Science*, 35(3):321–339, 1989.

Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.