Reinforcement Learning for Automated Robotic Fleets in a Warehouse

Nitin Tangellamudi, Jigar Patel

IE514 Spring 2019

May 10, 2019

Contents

Introduction

- Automated robotic fleets are becoming increasingly prevalent in large scale logistics industries
  - Amazon's Kiva robots
  - Amazon's Drone delivery
  - UPS uses Matternet drones for medical device shipments
  - Waymo's automated taxis

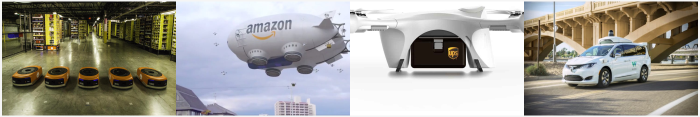


Figure: Rapid adoption of automated fleets

- Field still in its infancy and has tremendous potential

Introduction

- We study automated robotic fleets in a warehouse
- A warehouse continuously receives orders and we must efficiently use our fleet to meet desired level of service
- There are two key aspects:
  1. Dynamic decisions
  2. Fast decisions
- Example: If a robot is assigned to a task, the system should be able to reassign it to another new task if the whole system benefits
- Key Question: How do we devise these optimal policies?

Background - VRP

- Our instance is a sub-branch of Vehicle Routing Problem (VRP), specifically Dynamical Vehicle Routing Problem (DVRP)

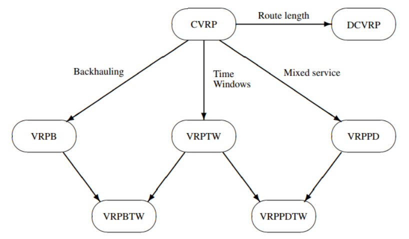


Figure: Problems of VRP and their interconnections[1]

- Classical optimization frameworks have focused on deterministic modeling with "offline" solution algorithms

Background - VRP

- Our scenario combines DVRP and Dynamic-TSP (DTSP) for **multiple** vehicles/salesmen
- For example, the robot (salesman) can only pickup (visit) 1 order (city) but the robot (salesman) can receive new orders (cities) en route
- Generally, DVR problems are intractable due to their combinatorial and stochastic nature [2]

Background - Reinforcement Learning and Q-Learning

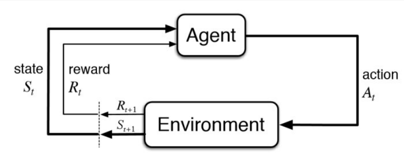


Figure: Agent-Environment Interaction in Reinforcement Learning

- Reinforcement Learning is where an agent learns how to optimally achieve a complex objective in a dynamic environment
- Q-learning is a model-free Reinforcement Learning approach which aims to learn an optimal policy
- In smaller problems, the policy can be stored in a Q-table, the row is the current state, and the columns are the actions that can be taken
- The optimal action given a state using the Q function is defined as:

$$\max_a Q(S_t, a_t)$$

Background- Q-Learning

- An update to the Q function is defined as:

$$Q(S_t, a_t) \leftarrow \mathrm{Q}(S_t, a_t) + \alpha[r_t + \lambda \underset{a}{max} Q(S_{t+1}, a_t) - Q(S_t, a_t)]$$

where,

- $Q(S_t, a_t)$ - the updated Q-value for state and action
- α - learning rate
- r_t - reward
- λ - discount rate for future reward
- $\underset{a}{max} Q(S_{t+1}, a_t)$ - estimated reward for next action

Background - Q-Learning Algorithm

- How do we compute the values in the Q-table?
- We build it iteratively by exploring the space and choosing actions which exploit what we have learned

---

Algorithm 1: Q-Learning Algorithm

Result: Finds optimal policy

Define: Hyper-parameters (e.g. discount factor, learning rate), Reward Structure

Initialization: Empty Q-table (states x actions)

for *training iterations* **do**

    Choose an action
    Perform action
    Compute Q-value
    Update Q-table

Background - Q-Learning Algorithm

- Consider one agent example

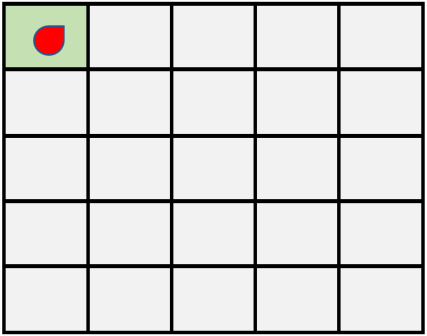


Figure: Environment, t=0

- Rewards $= \left\{ \begin{pmatrix} Order = +1 \\ Drop - off = +1 \\ Blank = 0 \\ Move = -0.1 \end{pmatrix} \right\}$
- $Q(S_{ent}, a_r) \leftarrow \mathrm{Q}(\mathrm{S}_{ent}, a_r) + \alpha[r_t + \lambda \max_a Q(S_{ent}, a) - Q(S_t, a_t)]$

	Actions			
State	Up	Down	Left	Right
Entrance	0	0	0	0
Orders	0	0	0	0
Drop-off	0	0	0	0
Blank	0	0	0	0

Table: Q-table, t=0

Background - Q-Learning Algorithm

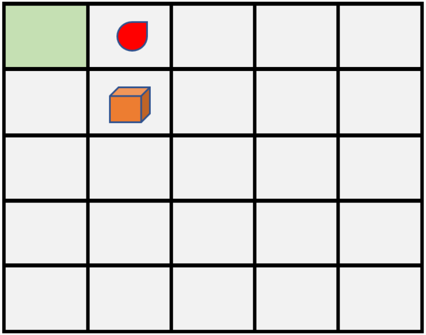


Figure: Environment, t=1

- $Q(S_{ent}, a_r) \leftarrow Q(S_{ent}, a_r) + \alpha[r_{t=1} + \lambda \max_a Q(S_b, a) - Q(S_{ent}, a_r)]$
- $\max_a(Q(S_{ent}, a_u), Q(S_{ent}, a_d), Q(S_{ent}, a_l), Q(S_{ent}, a_r)) = 0$
- $r_t = -0.1$
- $Q(S_{ent}, a_r) \leftarrow 0 + \alpha[-0.1 + \lambda 0 - 0]$

	Actions			
State	Up	Down	Left	Right
Entrance	0	0	0	**-0.1**α
Orders	0	0	0	0
Drop-off	0	0	0	0
Blank	0	0	0	0

Table: Q-table, t=1

Background - Q-Learning Algorithm

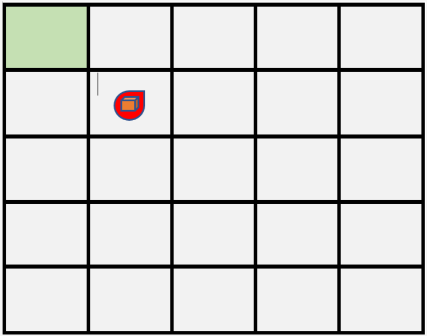


Figure: Environment, t=2

- $Q(S_b, a_d) \leftarrow Q(S_b, a_d) + \alpha[r_{t=2} + \lambda \max_a Q(S_{ord}, a) - Q(S_b, a_d)]$
- $\max_a (Q(S_{ord}, a_u), Q(S_{ord}, a_d), Q(S_{ord}, a_l), Q(S_{ord}, a_r)) = 0$
- $Q(S_b, a_d) \leftarrow 0 + \alpha[1 - 0.1 + \lambda 0 - 0]$

	Actions			
State	Up	Down	Left	Right
Entrance	0	0	0	**-0.1**α
Orders	0	0	0	0
Drop-off	0	0	0	0
Blank	0	**0.9**α	0	0

Table: Q-table, t=2

Background - Double Q-Learning

- Maintain two different Q functions: $Q_{current}$ and Q_{target}
- An update to the Q function is defined as:

$$Q_{current}(S_t, a_t) \leftarrow Q_{current}(S_t, a_t) + \alpha[r_t + \lambda Q_{current}(S_{t+1}, \max_a Q_{target}) - Q_{current}(S_t, a_t)]$$

  - Initialize $Q_{current}$ and Q_{target} both randomly
  - Set $Q_{target} \leftarrow Q_{current}$
  - Update $Q_{current}$ every step
  - Set $Q_{target} \leftarrow Q_{current}$ every m steps

- This allows for less bias to be propagated through the reward and leads to more accurate estimations, allowing for quicker and better convergence to an optimal policy [6]

Background - Recent Work

- Recent work has been increasingly focused on "online" approaches for dynamically evolving systems
  1. Bertsimas et al. ('19) [2] – explored online vehicle routing for a ride sharing modeled as VRPPDTW
  2. Larsen ('00) [3] – proposes several online heuristics for different flavors of DVRP's and DTSP's
  3. Nazari et. al ('18) [4] – Reinforcement Learning for VRP
- Our work extends [3] by proposing a novel online heuristic and [4] by implementing a Double Deep Q network for multiple agents operating within a warehouse

Problem Definition

- Objective
  - Maximize the number of orders fulfilled (while minimizing fleet travel distance)
- Constraints:
  1. Each robot can only be assigned to one order
  2. Each order can only be assigned to one robot
  3. Each robot has capacity of one
  4. Two robots cannot occupy the same location simultaneously
  5. Two orders cannot occupy the same location simultaneously

Environment Setup

- RL requires special environments which allow an agent to take actions, get rewards and info about state
- Developed a custom Warehouse Environment using OpenAI Gym
- Example of environment dynamics

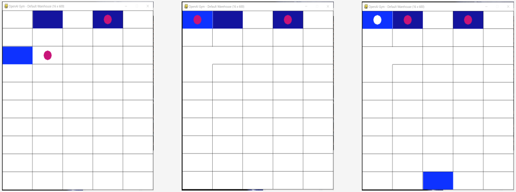


Figure: Robot 1 order pick up (left), going back (center), drop off (right)

- Note Robot 1 does not pick up the second order since it is loaded

Solution Approaches

- Mixed Integer Linear Programs (MILPs) with stochasticity
  - + Algorithms (e.g. branch and bound, cutting plane) with guarantees
  - + Newer software → solve richer models $t_{solve} \approx$ 1-2 minutes
  - − But if the system changes immediately after a solution is computed, we must wait t_{solve} → miss window to make optimal decisions
- Online Heuristic
  - Developed **Adaptive Single Neighbor Assign with Swap (ASNAS)**
  - + Adapt to system regardless of underlying stochasticity
  - + Complexity: $O(rn \log n)$ where $r = \#$ of robots, $n = \#$ of orders
- Double Deep-Q Reinforcement Learning [6]
  - + "Model-free" implementation → system can learn the embedded stochasticity

Adaptive Single Neighbor Assign with Swap (ASNAS)

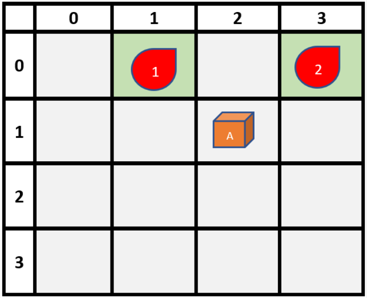


Figure: Environment, t=1

1. Sorted orders for each robot:
   - R1: (A,2)
   - R2: (A,2)
2. Assign each robot to closest unassigned order:
   - R1: (A,2)
   - R2: $\emptyset$
3. Compute Costs:
   - Current: $(R1=2) + (R2=0) = 2$
   - Swap: $(R1=0) + (R2=2) = 2$
4. Check Swap
   - If (Swap $<$ Current) $\rightarrow$ Swap assignments

Adaptive Single Neighbor Assign with Swap (ASNAS)

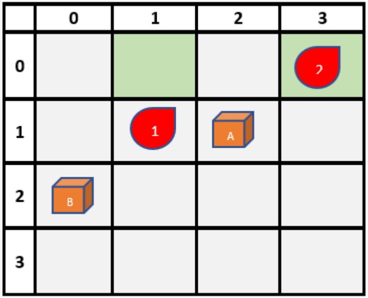


Figure: Environment, t=2

1. Sorted orders for each robot:
   - R1: (A,1), (B,2)
   - R2: (A,2),(B,5)
2. Assign each robot to closest unassigned order:
   - R1: (A,1)
   - R2: (B,5)
3. Compute Costs:
   - Current: (R1=1) + (R2=5) = 6
   - Swap: (R1=2) + (R2=2) = 4
4. Check Swap
   - If (Swap < Current) → Swap assignments
   - R1: (B,2)
   - R2: (A,2)

RL methods

- Flavors of Multi-Agent RL
  1. Super-agent - one centralized decision maker to decide what the fleet does
  2. Many separate agents - Useful when the system is so large that you can restrict an agent's observation space to what the agent can affect and be affected by. E.g. tilting cellular antennas
- We try to diversify our approach by experimenting with the reward structure and the network architecture:
  1. Method 1 - Sparse reward structure - Split output layer
  2. Method 2 - Sparse reward with load balancing -Split output layer
  3. Method 3 - Shaped reward structure - Split output layer
  4. Method 4 - Long output layer size
  5. Method 5 - Hierarchical Learning

Architecture

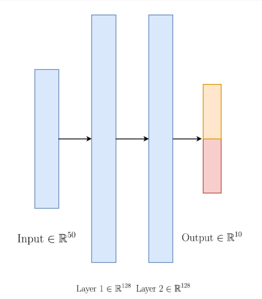


Figure: Method 1 Neural Network Architecture

- Input for DDQN: flattened state vector for a feed-forward network
- Output layer: 10 nodes, 5 for each robot
- Objective: $Y_t^{DQN} \equiv R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t)$
- Update: $\theta_{t+1} = \theta_t + \alpha(Y_t^Q - Q(S_t, A_t; \theta))\nabla_{\theta_t} Q(S_t, A_t; \theta_t)$

Objective and Update Step

$$\text{Objective: } Y_t^{DQN} \equiv R_{t+1}\gamma \max_a Q(S_{t+1}, a; \theta_t)$$

$$\text{Update: } \theta_{t+1} = \theta_t + \alpha(Y_t^Q - Q(S_t, A_t; \theta))\nabla_{\theta_t} Q(S_t, A_t; \theta_t)$$

Method 1 - Sparse Reward Structure

- The reward structure R_1 is:

$$R_1 = \begin{cases} +1, & \text{fulfilling an order} \\ +1, & \text{picking up an order} \\ -0.01, & \text{taking a step} \end{cases}$$

  - Sparse reward structure makes the learning process hard as the agent has to blindly explore the environment before getting a reward
  - In large state spaces, the probability of ever reaching a state space with a reward diminishes exponentially

Method 2 - Sparse Reward with Load Balancing

- Add a regularization term which penalizes an imbalance in number of orders fulfilled among the two robot
- Using the same NN architecture, reward structure R_2 is:

$$R_2 = \begin{cases} +1, & \text{fulfilling an order} \\ +1, & \text{picking up an order} \\ -0.01, & \text{taking a step} \\ -.01|O_1 - O_2|, & \text{difference in orders} \end{cases}$$

- The objective function now becomes:

$$\begin{aligned} Y_t^{DQ} \equiv R_{t+1} & -.01 * |O_1 - O_2| + \\ & \gamma Q_{target}(S_{t+1}, \underset{a}{argmax}\, Q_{current}(S_{t+1}, a; \theta_t); \theta_t') \end{aligned}$$

Method 3 - Shaped Reward Structure

- Give increasing rewards as the agent reaches states closer to the target
- Using the same NN architecture, reward structure R_3 is:

$$R_3 = \begin{cases} +1, & \text{fulfilling order} \\ +1, & \text{loading order} \\ -0.01, & \text{taking a step} \\ +0.05, & \text{at entrance} \\ (10.5 - ||\text{entrance - robot loc.}||_2), & \text{if robot loaded} \end{cases}$$

- The objective function now becomes:

$$Y_t^{DQ} \equiv R_{t+1} + (10.5 - ||\text{Distance from Order}||) + \gamma Q_{target}(S_{t+1}, \underset{a}{argmax}\, Q_{current}(S_{t+1}, a; \theta_t); \theta_t')$$

Method 4 - NN with Larger Output Layer

- Increase the output layer size from 10 to 25 nodes i.e. every combination of moves for Robot 1 and Robot 2
- For example:
  - Node with (0,0) $\rightarrow$ both robots do nothing
  - Node with (24,24) $\rightarrow$ both robots move right
- Train new network with sparse reward structure R_1

Method 5 - Hierarchical Learning

- Cut the problem into smaller pieces and start at the very bottom of the hierarchical process of order fulfillment i.e. order pick-up
- In this environment we have two robots and only one order
- The episode successfully ends upon a robot moving to an order and becoming loaded
- The reward structure is:

$$R_4 = \begin{cases} +20, & \text{picking up an order} \\ -0.1, & \text{taking a step} \\ +0.25, & \text{sitting at entrance} \\ \frac{1}{||\text{order loc. - robot loc.}||_2}, & \text{if order present} \end{cases}$$

Results and Discussion

- Simulation Parameters:
  - 64,800 time steps simulated for a 18-hr day
  - Each block randomly assigned to one of three demand distributions:
    - Class 1 - constitutes 5% of the grid, receives most orders
    - Class 2 - constitutes 25% of the grid, receives medium order levels
    - Class 3 - constitutes 70% of the grid, receives least orders
- We run the ASNAS heuristic without the swap to get a benchmark
- Run ASNAS with swap to see improvement
- Train RL models for $\sim$ 1M time steps (takes 1-2 days to train)

Results - ASNAS without Swap

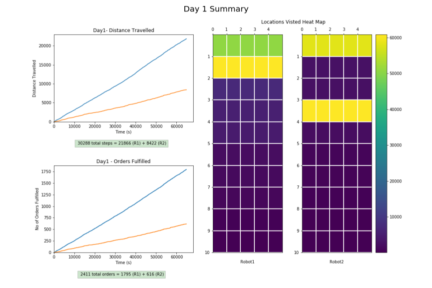


Figure: Summary of Day 1, No Swap

	Robot 1	Robot 2	Total
Day 1	21866	8422	30288
Day 2	23736	10144	33880
Day 3	23174	9791	32965
Day 4	22438	8016	30454
Day 5	15938	5008	20946
Average	21430	8276	29707

Table: Distance Travelled, No Swap

	Robot 1	Robot 2	Total
Day 1	1795	616	2411
Day 2	2049	781	2830
Day 3	2138	811	2950
Day 4	1907	784	2655
Day 5	1574	408	1982
Average	1893	680	2566

Table: Orders Fulfilled, No Swap

Results - ASNAS with Swap

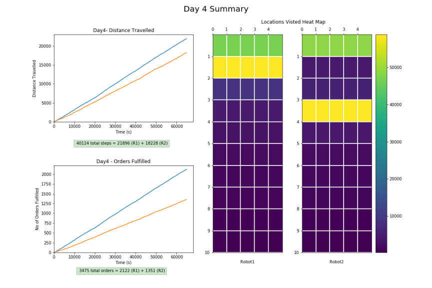


Figure: Summary of Day 4, With swap

	Robot 1	Robot 2	Total
Day 1	20134	20074	40208
Day 2	16834	13142	29976
Day 3	21590	20714	42304
Day 4	21896	18228	40124
Day 5	17202	15208	32410
Average	19531	17473	37004

Table: Distance Travelled, With Swap

	Robot 1	Robot 2	Total
Day 1	1919	1423	3343
Day 2	1738	953	2693
Day 3	2135	1593	3728
Day 4	2122	1351	3475
Day 5	1423	935	2358
Average	1867	1251	3119

Table: Orders Fulfilled, With Swap

Results - ASNAS

- No swap
  - Number of steps taken by Robot 1 is, on average, 250% more than Robot 2
  - Number of orders fulfilled by Robot 1 is, on average, 270% more than Robot 2
  - As expected as Robot 1 gets preference in order assignments
- Swap
  - Number of steps taken by Robot 1 is, on average, 10% more than Robot 2
  - Number of orders fulfilled by Robot 1 is, on average, 50% more than Robot 2
  - Swap results in 20% more orders fulfilled by the system as compared to the no swap case

Results - Method 1: Sparse Reward

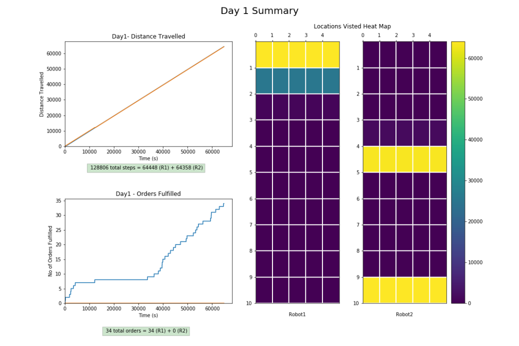


Figure: RL Model 1, Summary of Day 1

- Poor results compared to ASNAS with only 34 orders fulfilled
- Robot 1 fulfills all orders while Robot 2 moves randomly along the back

Results - Method 2: Sparse Reward with Load Balancing

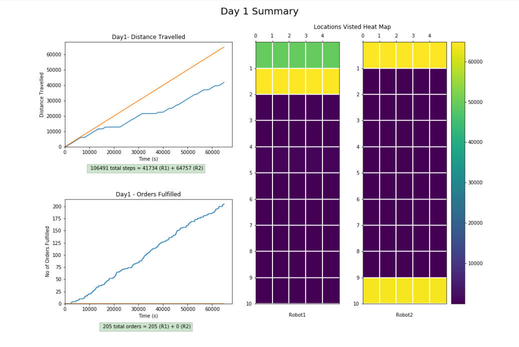


Figure: RL Model 2, Summary of Day 1

- Improvement over Method 1 as 205 orders fulfilled
- Robot 1 fulfills all orders while Robot 2 still moves randomly along the back

Results - Method 3: Shaped Reward Structure

- We tried to tailor the reward for more direct behavior such as:
  - Staying close to the base
  - Going back to the base if loaded
- However, the robots learned to just pick up an order and stagnate close to the entrance, accumulating rewards but not fulfilling an order

Results - Method 4: NN with Larger Output Layer

- The purpose was to see if the NN can structure a better relationship given "more room" to differentiate actions between two robots

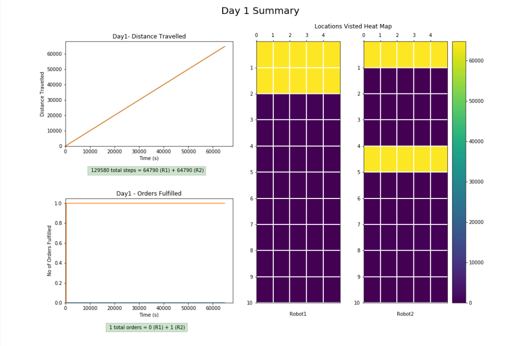


Figure: RL Model 4, Summary of Day 1

- Poor results, most likely due to missetting hyper-parameters

Results - Method 5: Hierarchical Learning

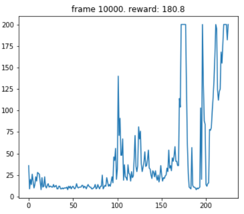
- Robot 2 learned to sit at the entrance minimizing travel costs, which is an improvement from previous methods
- However, Robot 1 has not learned to navigate to an order even after training for 1M iterations
- Since orders can appear anywhere, it makes it harder to learn to move towards the order
- Initially thought it would be a challenge for one of the robots to learn to 'stay back' but this has been achieved

Discussion

- The effective of Double Deep Q Learning is well understood and perhaps best appreciated in the case of playing Atari games above human levels
- However, applying it to Multi-Agent RL in a cooperative environment presented many challenges [9]:
    1. Game-Theoretic Effects
    2. Credit Assignment and Lazy Agent Problem
    3. Join Action space
- Addressing these issues are new-born fields of research
- For this project, we delve deeper to explore the delicate nature of training and learning in these networks

Discussion - Sensitivity of Q-Learning

- Consider training a Cartpole agent whose goal is to keep a pole upright with its actions being moving the cart left or right



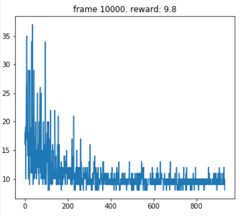


Figure: Good Training (Left) vs Poor Training (Right) of Cartpole

- In the two instances, we slightly change two hyper-parameters, the learning rate and the epsilon decay yet it has a huge impact in training

Extensions and Future Work

- **Key challenge**: How can we overcome the need to do complicated reward engineering?
- Andrychowicz et. al (18) [10] propose a novel technique, Hindsight Experience Replay (HER) which allows us to learn from binary and sparse rewards
- Example: Consider a Binary Bit Flipping Environment
  - The agent's goal is to learn a target sequence of 2^n bits
  - State space size $= 2^n$
  - Q-Learning and Policy Gradients fail for $n > 14$
- Compare to our environment with 50 cells and 2 robots $\rightarrow$ size of state space $= (5^2)(2^{48})$

Extensions and Future Work - HER

- Consider a state sequence $S_1, ..., S_T$, a goal $g \neq S_1, ..., S_T$ i.e. the agent is not at the goal and receives a reward of -1 at every time step
- **Idea:** Retrain the agent and use terminating states as pseudo-goals i.e. learn something about how to achieve S_T
- Andrychowicz et. al demonstrate DDQN + HER can solve the Bit Flipping environment for $n \geq 50$

Extensions and Future Work - HER

- We replicate the experiment with $n = 11$

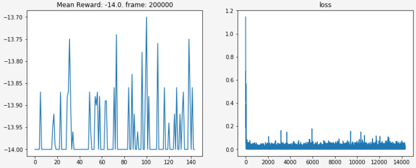


Figure: DDQN without HER in Bit Flipping Environment

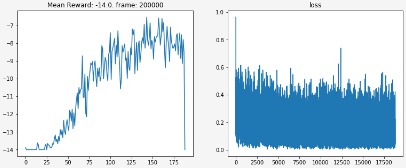


Figure: DDQN + HER in Bit Flipping Environment

Extensions and Future Work - HER

- As we saw, HER significantly improves the learning ability in large state spaces
- Another benefit of HER is it is independent of the initial state
- In our case, this can be extended to teaching a single robot how to react when an order arrives i.e. learning the shortest path
- Extending this to 2 robots is non-trivial and requires an appropriate goal formulation for the DDQN + HER

Extensions and Future Work: Method 6 - DDQN using CNN

- DDQN has proven quite successful in playing single-agent Atari games using a Convolutional Neural Network (CNN) with the game screen as input [7]
- Using a CNN will allow for the state space to be smaller than using the current input matrix ... can anyone tell me why?

References

- Paolo Toth , Daniele Vigo. Overview of Vehicle Routing Problems. Discrete Mathematics and Application: The Vehicle Routing Problem. 2002.
- Emilio Frazzoli, Marco Pavone. Multi-Vehicle Routing. Encyclopedia of Systems and Control. 2015.
- Dimitris Bertsimas, Patrick Jaillet, Sebastien Martin. Online Vehicle Routing: The Edge of Optimization in Large-Scale Applications.
- Nazari M., Oroojlooy A., Takac M., Snyder L. Reinforcement Learning for Solving the Vehicle Routing Problem. NIPS. 2018.
- Allan Larsen. The Dynamic Vehicle Routing Problem. Denmark: Technical University of Denmark (DTU). 2000. IMM-PHD, No. 2000-73
- Hado van Hasselt , Arthur Guez, and David Silver. Deep Reinforcement Learning with Double Q-learning. Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16). 2015