

# QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation

arXiv:1806.10293, Kalashnikov et al, 2018.

*Summarized by* Hyecheol (Jerry) Jang

Department of Computer Sciences  
University of Wisconsin–Madison

RL Paper Study, Jun. 29. 2020



- 1 Motivation
- 2 Goal
- 3 Overview of Model Architecture
- 4 QT-Opt
- 5 Environment Settings
- 6 Experiments
- 7 Discussions
- 8 Bibliography

# Motivation: Why Robotics + Reinforcement Learning

- Usually, Robots are good at **repetitive tasks** (e.g. Assembly Line)

# Motivation: Why Robotics + Reinforcement Learning

- Usually, Robots are good at **repetitive tasks** (e.g. Assembly Line)
- Want to make Robots that **identifies surroundings** and **behave accordingly**, but it is difficult

# Motivation: Why Robotics + Reinforcement Learning

- Usually, Robots are good at **repetitive tasks** (e.g. Assembly Line)
- Want to make Robots that **identifies surroundings** and **behave accordingly**, but it is difficult
  - **Deep Learning**  
Provide ability to handling real-world scenarios
  - **Reinforcement Learning**  
Provide ability to make decision in long-term, using previous experiences in complex and robust scenarios

# Motivation: Why Robotics + Reinforcement Learning

- Usually, Robots are good at **repetitive tasks** (e.g. Assembly Line)
- Want to make Robots that **identifies surroundings** and **behave accordingly**, but it is difficult
  - **Deep Learning**  
Provide ability to handling real-world scenarios
  - **Reinforcement Learning**  
Provide ability to make decision in long-term, using previous experiences in complex and robust scenarios
- Combining two techniques
  - Able to learn policy continuously from their experience
  - No need for manual engineering, use data they collects



- Variance in **visual and physical property of objects**



- Variance in **visual and physical property of objects**
  - Hardness of object (Soft or Hard)
  - Surface Characteristics (Slippery, Sticky, ...)
  - Color Variation
  - Shape Variation
  - ...





- Variance in **visual and physical property of objects**
  - Hardness of object (Soft or Hard)
  - Surface Characteristics (Slippery, Sticky, ...)
  - Color Variation
  - Shape Variation
  - ...
- **Noise** of sensors

- Variance in **visual and physical property of objects**

- Hardness of object (Soft or Hard)
- Surface Characteristics (Slippery, Sticky, ...)
- Color Variation
- Shape Variation
- ...

- **Noise** of sensors

- ⇒ Still hard to handle though we have sufficiently large training set
  - ⇒ Collecting those training set is expensive (real experiments)



- Focused on learning narrow, individual tasks
  - hitting a ball
  - opening door
  - throwing objects
  - ...



- Focused on learning narrow, individual tasks
    - hitting a ball
    - opening door
    - throwing objects
    - ...
- ⇒ Use **Grasping** to achieve *generalization*

- Focused on learning narrow, individual tasks
  - hitting a ball
  - opening door
  - throwing objects
  - ...
- ⇒ Use **Grasping** to achieve *generalization*
- Approached the grasping task as predicting a *grasp pose*
  - 1 Observe the scene (*Normally, using a depth camera*)
  - 2 Choose best location to grasp
  - 3 Reach the location (open-loop setting)

- Focused on learning narrow, individual tasks

- hitting a ball
- opening door
- throwing objects
- ...

⇒ Use **Grasping** to achieve *generalization*

- Approached the grasping task as predicting a *grasp pose*

- 1 Observe the scene (*Normally, using a depth camera*)
  - 2 Choose best location to grasp
  - 3 Reach the location (open-loop setting)
- Different with how humans and animals behave
  - Grasp is a **dynamical process** that sense and control at each stage

- Focused on learning narrow, individual tasks
  - hitting a ball
  - opening door
  - throwing objects
  - ...
- ⇒ Use **Grasping** to achieve *generalization*
- Approached the grasping task as predicting a *grasp pose*
  - 1 Observe the scene (*Normally, using a depth camera*)
  - 2 Choose best location to grasp
  - 3 Reach the location (open-loop setting)
    - Different with how humans and animals behave
    - Grasp is a **dynamical process** that sense and control at each stage
- ⇒ **Where this researches start!!**

- 1 Motivation
- 2 Goal**
- 3 Overview of Model Architecture
- 4 QT-Opt
- 5 Environment Settings
- 6 Experiments
- 7 Discussions
- 8 Bibliography





Use Reinforcement Learning with Deep Neural Network  
to **perform pre-grasp manipulation**,  
**response to dynamic disturbances**,  
and **learn grasping in a generic framework**  
that makes minimal assumptions about the task



- **Closed-loop condition** (With feedback, *Morrison, et al.*)
  - For the other papers work on closed-loop grasping, they deals with servoing problems.
  - This paper focuses on making generalized RL algorithm
  - In practice, it makes Kalashnikov et al.'s method (this method) to autonomously acquire complicated grasping strategy



- **Closed-loop condition** (With feedback, *Morrison, et al.*)
  - For the other papers work on closed-loop grasping, they deal with servoing problems.
  - This paper focuses on making a generalized RL algorithm
  - In practice, it makes Kalashnikov et al.'s method (this method) to autonomously acquire a complicated grasping strategy
- **Self-supervised learning task**
  - Compare to previous work (by Zeng et al.), Kalashnikov et al. utilize a more general action space
  - Actions consist of end-effector **Cartesian motion** and **gripper opening/closing**

- **Closed-loop condition** (With feedback, *Morrison, et al.*)
  - For the other papers work on closed-loop grasping, they deals with servoing problems.
  - This paper focuses on making generalized RL algorithm
  - In practice, it makes Kalashnikov et al.'s method (this method) to autonomously acquire complicated grasping strategy
- **Self-supervised** learning task
  - Compare to prevoius work(by Zeng et al.), Kalashnikov et al. utilize more general action space
  - Actions consist of end-effector **Cartesian motion** and **gripper opening/closing**
- Observation comes from **a single RGB camera** over the sholder
  - Many current grasping system utilizes depth sensing
  - Using wrist-mounted cameras

- 1 Motivation
- 2 Goal
- 3 Overview of Model Architecture**
- 4 QT-Opt
- 5 Environment Settings
- 6 Experiments
- 7 Discussions
- 8 Bibliography



- General Formulation of Robotic Manipulation:  
Based on **Markov Decision Process (MDP)**
  - partially observed formulation (POMDP) is more general.
  - However, assuming current observation contains all necessary information for this task, it is sufficient to use MDP.



- General Formulation of Robotic Manipulation:  
Based on **Markov Decision Process (MDP)**
  - partially observed formulation (POMDP) is more general.
  - However, assuming current observation contains all necessary information for this task, it is sufficient to use MDP.
- MDP have a **general and powerful formalism** for decision making problems.  
However, it is **hard to train**

- General Formulation of Robotic Manipulation:  
Based on **Markov Decision Process (MDP)**
  - partially observed formulation (POMDP) is more general.
  - However, assuming current observation contains all necessary information for this task, it is sufficient to use MDP.
- MDP have a **general and powerful formalism** for decision making problems.  
However, it is **hard to train**
- For each step of MDP:
  - 1 Observes Image from robot's camera (see Fig. 1)
  - 2 choose a gripper command, Reward:
    - failed grasp: reward of 0
    - successful grasp: reward of 1Defined *success* when the robot holds the object above a certain height



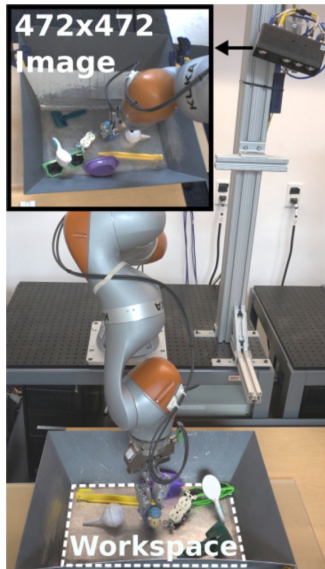


Figure 1: Configuration of robot cell, with a sample observation image on top-right box

# Overview of Model Architecture: Algorithm Selection

- Usually, Generalization needs diverse data
  - However, recollecting experience on numerous objects after every policy update is impractical
  - Reason for **not using on-policy algorithm**

# Overview of Model Architecture: Algorithm Selection

- Usually, Generalization needs diverse data
  - However, recollecting experience on numerous objects after every policy update is impractical
  - Reason for **not using on-policy algorithm**
- Using **scalable off-policy algorithm** based on Q-learning
  - actor-critic algorithm are popular for handling continuous actions
  - However, Kalashnikov et al. found **scalable and more stable ways** to train only Q-function

# Overview of Model Architecture: Algorithm Selection

- Usually, Generalization needs diverse data
  - However, recollecting experience on numerous objects after every policy update is impractical
  - Reason for **not using on-policy algorithm**
- Using **scalable off-policy algorithm** based on Q-learning
  - actor-critic algorithm are popular for handling continuous actions
  - However, Kalashnikov et al. found **scalable and more stable ways** to train only Q-function
- Large Dataset and Network (See Fig. 2)
  - Kalashnikov et al. devised **distributed** training system (with 7 robots)
  - **Asynchronously update** target values, collect **on-policy data**, reloads **off-policy data** from previous experiences, and train network on both data stream.

# Overview of Model Architecture: Algorithm Selection

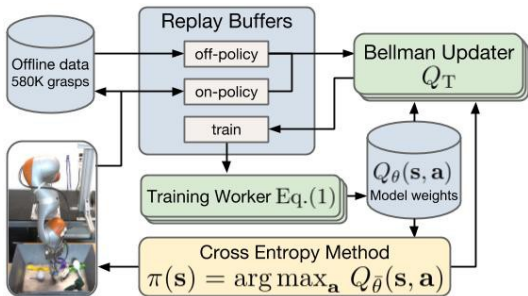


Figure 2: Distributed Reinforcement Learning infrastructure for QT-Opt.



- **On-policy Learning** learns the value of the policy being **carried out by the agent**, including the exploration steps.  
e.g. SARSA(State-Action-Reward-State-Action) (See Fig. 3)



- **On-policy Learning** learns the value of the policy being **carried out by the agent**, including the exploration steps.  
e.g. SARSA(State-Action-Reward-State-Action) (See Fig. 3)
- **Off-policy Learning** learns the value of the optimal policy **independently of the agent's action**  
e.g. Q-Learning (See Fig. 4)

# Optional: On-policy vs Off-policy *Poole et al.*



controller SARSA( $S, A, \gamma, \alpha$ )

inputs:

$S$  is a set of states

$A$  is a set of actions

$\gamma$  the discount

$\alpha$  is the step size

internal state:

real array  $Q[S, A]$

previous state  $s$

previous action  $a$

begin

initialize  $Q[S, A]$  arbitrarily

observe current state  $s$

select action  $a$  using a policy based on  $Q$

repeat forever:

    carry out an action  $a$

    observe reward  $r$  and state  $s'$

    select action  $a'$  using a policy based on  $Q$

$Q[s, a] \leftarrow Q[s, a] + \alpha(r + \gamma Q[s', a'] - Q[s, a])$

$s \leftarrow s'$

$a \leftarrow a'$

end-repeat

end

Figure 3: SARSA Algorithm

controller  $Q$ -learning( $S, A, \gamma, \alpha$ )

2: Inputs

3:  $S$  is a set of states

4:  $A$  is a set of actions

5:  $\gamma$  the discount

6:  $\alpha$  is the step size

7: Local

8: real array  $Q[S, A]$

9: previous state  $s$

10: previous action  $a$

11: initialize  $Q[S, A]$  arbitrarily

12: observe current state  $s$

13: repeat

14: select and carry out an action  $a$

15: observe reward  $r$  and state  $s'$

16:  $Q[s, a] \leftarrow Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$

17:  $s \leftarrow s'$

18: until termination

Figure 4: Q-Learning Algorithm



- 1 Motivation
- 2 Goal
- 3 Overview of Model Architecture
- 4 QT-Opt**
- 5 Environment Settings
- 6 Experiments
- 7 Discussions
- 8 Bibliography

- Continuous action version of Q-Learning
  - For **scalable** learning and optimized for **stability**
  - To handle **large amount of off-policy image data** for complex tasks



- **state:**  $s \in \mathcal{S}$  (*Image Observations*)
- **action:**  $a \in \mathcal{A}$  (*Robot Arm Motions and Gripper commands*)
- at **each time step**  $t$ 
  - 1 Choose an action
  - 2 transition to new state
  - 3 receive reward  $\gamma(s_t, a_t)$

- Need to solve for Optimal Q-function: Minimize Bellman Error

$$\mathcal{E}(\theta) = \mathbb{E}_{(s,a,s') \sim p(s,a,s')} [\mathcal{D}(\mathcal{Q}_\theta(s,a), \mathcal{Q}_\mathcal{T}(s,a,s'))]$$

Where  $\mathcal{Q}_\mathcal{T}(s,a,s') = r(s,a) + \gamma V(s')$  (*target value*)

- $\mathcal{D}$ : divergence metric (squared difference for Q-Learning)
- Expectation is taken under the distribution over all previously observed transition



- Use **two target network** to improve stability
  - Maintaining two lagged version of the parameter vector  $\theta, \bar{\theta}_1, \bar{\theta}_2$
  - $\bar{\theta}_1$ : exponential moving averaged version of  $\theta$ , averaging constant: 0.9999
  - $\bar{\theta}_2$ : lagged version of  $\bar{\theta}_1$ , lagged by 6000 gradient steps

- Use **two target network** to improve stability
  - Maintaining two lagged version of the parameter vector  $\theta, \bar{\theta}_1, \bar{\theta}_2$
  - $\bar{\theta}_1$ : exponential moving averaged version of  $\theta$ , averaging constant: 0.9999
  - $\bar{\theta}_2$ : lagged version of  $\bar{\theta}_1$ , lagged by 6000 gradient steps
- Compute target value by
$$V(s') = \min_{i=1,2} Q_{\bar{\theta}_i}(s', \arg \max_{a'} Q_{\bar{\theta}_1}(s', a'))$$

- Use **two target network** to improve stability
  - Maintaining two lagged version of the parameter vector  $\theta, \bar{\theta}_1, \bar{\theta}_2$
  - $\bar{\theta}_1$ : exponential moving averaged version of  $\theta$ , averaging constant: 0.9999
  - $\bar{\theta}_2$ : lagged version of  $\bar{\theta}_1$ , lagged by 6000 gradient steps
- Compute target value by
$$V(s') = \min_{i=1,2} Q_{\bar{\theta}_i}(s', \arg \max_{a'} Q_{\bar{\theta}_i}(s', a'))$$
- the policy is recovered by  $\pi(s) = \arg \max_a Q_{\bar{\theta}_1}(s, a)$

- Use **two target network** to improve stability
  - Maintaining two lagged version of the parameter vector  $\theta, \bar{\theta}_1, \bar{\theta}_2$
  - $\bar{\theta}_1$ : exponential moving averaged version of  $\theta$ , averaging constant: 0.9999
  - $\bar{\theta}_2$ : lagged version of  $\bar{\theta}_1$ , lagged by 6000 gradient steps
- Compute target value by
$$V(s') = \min_{i=1,2} Q_{\bar{\theta}_i}(s', \arg \max_{a'} Q_{\bar{\theta}_i}(s', a'))$$
- the policy is recovered by  $\pi(s) = \arg \max_a Q_{\bar{\theta}_1}(s, a)$
- After collects samples from environment interaction, then perform off-policy training on all samples collected
  - Problem: Large-scale learning (Technically Difficult)
  - Solution: Use **Parallel Asynchronous** version (which leads ability to scale up the process)





- Difficult to deal with continuous actions, e.g. continuous gripper motion



- Difficult to deal with continuous actions, e.g. continuous gripper motion
- *Previous Solutions:*
  - using a second network that "amortize" the maximization
  - constraining the Q-function to be convex in  $a$ , so that makes the function to easily find maximum analytically

- Difficult to deal with continuous actions, e.g. continuous gripper motion
- *Previous Solutions:*
  - using a second network that "amortize" the maximization
  - constraining the Q-function to be convex in  $a$ , so that makes the function to easily find maximum analytically
- problem of previous solutions
  - Unstable: problematic for large-scale RL tasks where running hyperparameter sweeps is very expensive
  - Action-convex value functions are poor fit for complex manipulation tasks



- To **maintain the generality of non-convex Q-function** while **not using a second maximizer network**
- the Bellman equation is evaluated with stochastic optimization
  - handles non-convex and multimodal optimization landscapes



- To **maintain the generality of non-convex Q-function** while **not using a second maximizer network**
- the Bellman equation is evaluated with stochastic optimization
  - handles non-convex and multimodal optimization landscapes
- QT-Opt
  - $\pi_{\bar{\theta}_1}(s)$  is evaluated by running stochastic optimization over  $a$ , using  $\mathcal{Q}_{\bar{\theta}_1}(s, a)$  as the objective value



- To **maintain the generality of non-convex Q-function** while **not using a second maximizer network**
- the Bellman equation is evaluated with stochastic optimization
  - handles non-convex and multimodal optimization landscapes
- QT-Opt
  - $\pi_{\bar{\theta}_1}(s)$  is evaluated by running stochastic optimization over  $a$ , using  $\mathcal{Q}_{\bar{\theta}_1}(s, a)$  as the objective value
  - Use Cross-Entropy Method
    - ① samples a batch of  $N$  at each iteration
    - ② fits a Gaussian distribution to the best  $M < N$  samples
    - ③ Samples next batch of  $N$  from the Gaussian



- To **maintain the generality of non-convex Q-function** while **not using a second maximizer network**
- the Bellman equation is evaluated with stochastic optimization
  - handles non-convex and multimodal optimization landscapes
- QT-Opt
  - $\pi_{\bar{\theta}_1}(s)$  is evaluated by running stochastic optimization over  $a$ , using  $Q_{\bar{\theta}_1}(s, a)$  as the objective value
  - Use Cross-Entropy Method
    - ① samples a batch of  $N$  at each iteration
    - ② fits a Gaussian distribution to the best  $M < N$  samples
    - ③ Samples next batch of  $N$  from the Gaussian
      - easy to parallelize
      - moderately robust to local optima



- Requires large amount of diverse data to generalize over new scenes and objects for the image-based policy





- Requires large amount of diverse data to generalize over new scenes and objects for the image-based policy
- Needs of Distributed System (See Fig. 2)



- Requires large amount of diverse data to generalize over new scenes and objects for the image-based policy
- Needs of Distributed System (See Fig. 2)
  - ① Replay buffer stores both **off-line data** (from disk) and on-going experiments' data (**on-line data**)



- Requires large amount of diverse data to generalize over new scenes and objects for the image-based policy
- Needs of Distributed System (See Fig. 2)
  - ① Replay buffer stores both **off-line data** (from disk) and on-going experiments' data (**on-line data**)
  - ② Data in buffer labeled with target Q-value using a set of 1000 "bellman updater" jobs



- Requires large amount of diverse data to generalize over new scenes and objects for the image-based policy
- Needs of Distributed System (See Fig. 2)
  - ① Replay buffer stores both **off-line data** (from disk) and on-going experiments' data (**on-line data**)
  - ② Data in buffer labeled with target Q-value using a set of 1000 "bellman updater" jobs
  - ③ Store the labeled sample in the training buffer
    - Some samples in the training buffer are **labeled with lagged version** of the Q-network

- Requires large amount of diverse data to generalize over new scenes and objects for the image-based policy
- Needs of Distributed System (See Fig. 2)
  - ① Replay buffer stores both **off-line data** (from disk) and on-going experiments' data (**on-line data**)
  - ② Data in buffer labeled with target Q-value using a set of 1000 "bellman updater" jobs
  - ③ Store the labeled sample in the training buffer
    - Some samples in the training buffer are **labeled with lagged version** of the Q-network
  - ④ Training workers pull the labeled sample from the training buffer **randomly** to update the Q-function
    - each training workers compute gradient that sent to parameter server asynchronously

- Requires large amount of diverse data to generalize over new scenes and objects for the image-based policy
- Needs of Distributed System (See Fig. 2)
  - ① Replay buffer stores both **off-line data** (from disk) and on-going experiments' data (**on-line data**)
  - ② Data in buffer labeled with target Q-value using a set of 1000 "bellman updater" jobs
  - ③ Store the labeled sample in the training buffer
    - Some samples in the training buffer are **labeled with lagged version** of the Q-network
  - ④ Training workers pull the labeled sample from the training buffer **randomly** to update the Q-function
    - each training workers compute gradient that sent to parameter server asynchronously
- Empirically, it requires up to **15M gradient steps** to train effective Q-function due to **complexity of the task** and **the size of dataset and model**

- 1 Motivation
- 2 Goal
- 3 Overview of Model Architecture
- 4 QT-Opt
- 5 Environment Settings**
- 6 Experiments
- 7 Discussions
- 8 Bibliography

- 1 Motivation
- 2 Goal
- 3 Overview of Model Architecture
- 4 QT-Opt
- 5 Environment Settings
- 6 Experiments**
- 7 Discussions
- 8 Bibliography



- 1 Motivation
- 2 Goal
- 3 Overview of Model Architecture
- 4 QT-Opt
- 5 Environment Settings
- 6 Experiments
- 7 Discussions**
- 8 Bibliography

- ① Motivation
- ② Goal
- ③ Overview of Model Architecture
- ④ QT-Opt
- ⑤ Environment Settings
- ⑥ Experiments
- ⑦ Discussions
- ⑧ Bibliography



- **Kalashnikov, Dmitry, et al. QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. 28 Nov. 2018, [arxiv.org/abs/1806.10293](https://arxiv.org/abs/1806.10293).**
- Irpan, Alex, and Peter Pastor. Scalable Deep Reinforcement Learning for Robotic Manipulation. 28 June 2018, [ai.googleblog.com/2018/06/scalable-deep-reinforcement-learning.html](https://ai.googleblog.com/2018/06/scalable-deep-reinforcement-learning.html).
- Morrison, Douglas, et al. "Closing the Loop for Robotic Grasping: A Real-Time, Generative Grasp Synthesis Approach." Robotics: Science and Systems XIV, 2018, doi:10.15607/rss.2018.xiv.021.
- Poole, David Lynton, and Alan K. Mackworth. Artificial Intelligence: Foundations of Computational Agents. Cambridge University Press, 2018.