Tutorial: Setting up a Reinforcement Learning pipeline for a Telco Core Network (part 1)

27^{th} Conference on Innovation in Clouds, Internet and Networks
ICIN 2024

Shantanu Verma [*]

[*]Orange Innovation Networks,
Gurgaon, India

Guillaume Fraysse [§]

[§]Orange Innovation Networks,
Châtillon, France

March 11^{th}, 2024

Acknowledging the work of a whole team: current and past members

Current members:

- Guillaume FRAYSSE (Orange Innovation Networks, Châtillon, France)
- Abhishek GUPTA (Orange Innovation Networks, Gurgaon, India)
- Jatinder SINGH (Orange Innovation Networks, Gurgaon, India)
- Gabriel Gomes De SIQUEIRA (Orange Innovation Networks, Châtillon, France)
- Shantanu VERMA (Orange Innovation Networks, Gurgaon, India)

Past members:

- Manu CHAUHAN (now at Tavant)
- Francesca FOSSATI (now at Sorbonne Université)
- Imen GRIDA BEN YAHIA (now at Amazon Web Services, UK)
- Xinqi LONG (MSc student Beihang University, CentralSupelec)
- Yoichi MATSUO (NTT Network Service Systems Laboratories)

Related publications

Publications

- J. Singh, S. Verma, Y. Matsuo, F. Fossati and G. Fraysse, Autoscaling Packet Core Network Functions with Deep Reinforcement Learning NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium, Miami, FL, USA, 2023, pp. 1-6, doi: 10.1109/NOMS56928.2023.10154312
- Y. Matsuo, J. Singh, S. Verma and G. Fraysse, Integrating state prediction into the Deep Reinforcement Learning for the Autoscaling of Core Network Functions NOMS 2023-2023 IEEE/IFIP Network Operations and Management Symposium, Miami, FL, USA, 2023, pp. 1-5, doi: 10.1109/NOMS56928.2023.10154301

Introduction

Assumptions:

- Attendees have background more focused on IP and/or Telco networks and network management
- Attendees have limited knowledge of Reinforcement Learning (RL)

Expectations:

- Do you know about RL or is it a brand new topic ? Do you have experience with it ?
- What are your expectations from this tutorial, if any ?

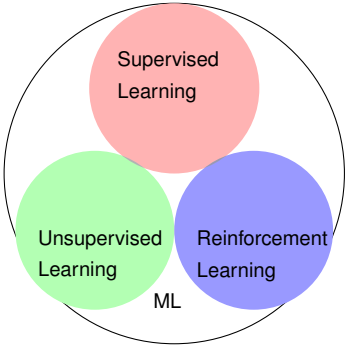
Requirements

Requirements

- Operational Wi-Fi access
- Working Python environment (v3.11+ is optimal)
- A Git client to be able to clone a GitHub repository
- (alternative, but with limitations) A Google account to use Colab
- (optional) A free Weights & Biases account

Definitions: the 3 main paradigms of Machine Learning

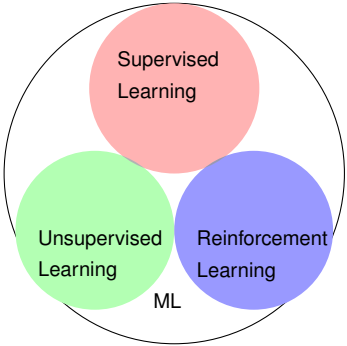
- **Machine Learning (ML)**: A computer program is said to learn from **experience E** with respect to some **class of tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E. (Prof. Tom M. Mitchell [Mitchell, 1997])



A Taxonomy of Machine Learning

Definitions: the 3 main paradigms of Machine Learning

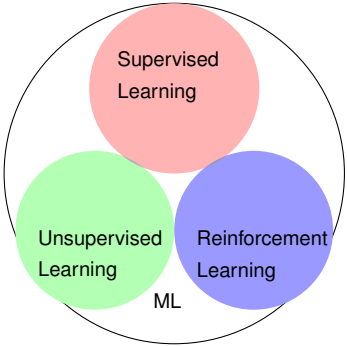
- **Machine Learning (ML)**: A computer program is said to learn from **experience E** with respect to some **class of tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E. (Prof. Tom M. Mitchell [Mitchell, 1997])
  - **Supervised Learning**: Estimating some function $f : X \rightarrow Y$ given a set of labeled training examples (X_i, y_i).



A Taxonomy of Machine Learning

Definitions: the 3 main paradigms of Machine Learning

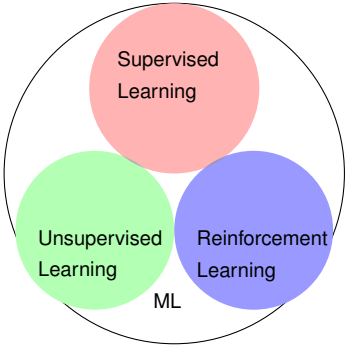
- **Machine Learning (ML)**: A computer program is said to learn from **experience E** with respect to some **class of tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E. (Prof. Tom M. Mitchell [Mitchell, 1997])
  - **Supervised Learning**: Estimating some function $f : X \rightarrow Y$ given a set of labeled training examples (X_i, y_i).
  - **Unsupervised Learning**: Harder to define formally. One common task is the **clustering**: construct a **grouping** of unlabeled data in **homogeneous** classes



A Taxonomy of Machine Learning

Definitions: the 3 main paradigms of Machine Learning

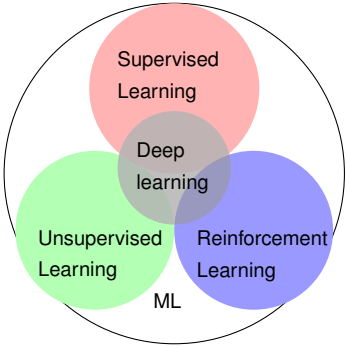
- **Machine Learning (ML)**: A computer program is said to learn from **experience E** with respect to some **class of tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E. (Prof. Tom M. Mitchell [Mitchell, 1997])
  - **Supervised Learning**: Estimating some function $f : X \rightarrow Y$ given a set of labeled training examples (X_i, y_i).
  - **Unsupervised Learning**: Harder to define formally. One common task is the **clustering**: construct a **grouping** of unlabeled data in **homogeneous** classes
  - **RL**: learning what to do - how to map situations to actions - so as to maximize a numerical reward signal [Sutton and Barto, 2018]



A Taxonomy of Machine Learning

Definitions: the 3 main paradigms of Machine Learning

- **Machine Learning (ML)**: A computer program is said to learn from **experience E** with respect to some **class of tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E. (Prof. Tom M. Mitchell [Mitchell, 1997])
  - **Supervised Learning**: Estimating some function $f : X \rightarrow Y$ given a set of labeled training examples (X_i, y_i).
  - **Unsupervised Learning**: Harder to define formally. One common task is the **clustering**: construct a **grouping** of unlabeled data in **homogeneous** classes
  - **RL**: learning what to do - how to map situations to actions - so as to maximize a numerical reward signal [Sutton and Barto, 2018]



A Taxonomy of Machine Learning

Deep Learning: subset of ML methods based on Neural Networks.

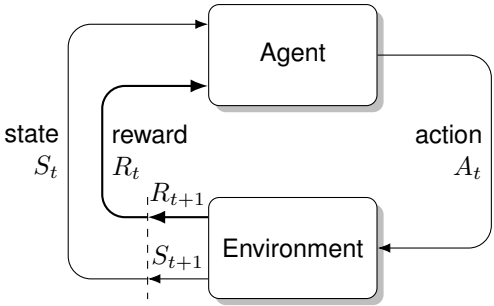
RL Application Areas 1/2

Application Areas	Problem Statement	Environment
Autonomous driving cars	Trajectory optimization, dynamic pathing, Motion planning, route changing, decision position of parking	Carla, DeepTraffic by MIT
Robotics	Object Grasping, Obstacle Avoidance, Navigation & path planning, Simultaneous task execution	CopelliaSim, MuJoCo, PyRobot
Generative AI	Reinforcement Learning with Human Feedback (RLHF) for Large Language Models	Prompts, outputs and rankings by humans (Fine tuning of GPT models for ChatGPT, Llama2, ...)

RL Application Areas 2/2

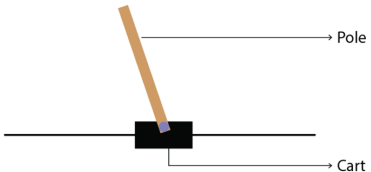
Application Areas	Problem Statement	Environment
Gaming	Chess, Pinball, Atari 2600 Games ([Mnih et al., 2013]), AlphaGo (2015), AlphaCraft ([Vinyals et al., 2019]), DOTA2 ([Berner et al., 2019])	OpenAI/Farama Foundation (Atari, Classic Control), DeepMind Lab, ...
Cloud systems	Job scheduling, Congestion control, Database Optimization (Query optimization, Index structure), Resource Management and **Autoscaling**	Apache OpenWhisk FaaS ([Qiu et al., 2022]), Kubernetes cluster ([Qiu et al., 2020]), ...

Reinforcement Learning



Agent-environment interaction loop in a RL system, from [Sutton and Barto, 2018]

The Cart Pole example



Cart Pole

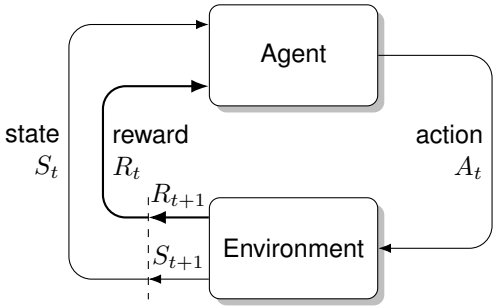
Element	Values set
Action Space	0: Push cart to the left, 1: Push cart to the right
Observation Space	Num. | Observation | Min | Max 0 | Cart Position | -4.8 | 4.8 1 | Cart Velocity | -Inf | Inf 2 | Pole Angle | -24° | 24° 3 | Pole Angular Velocity | -Inf | Inf
Reward	+1 per step taken

The Space Invaders example

orange Space Invaders

Element	Values set			
Action Space	**Value**	**Meaning**	**Value**	**Meaning**
	0	NOOP	1	FIRE
	2	RIGHT	3	LEFT
	4	RIGHTFIRE	5	LEFTFIRE
Observation Space	**Obs. Type**	**Dimensions**	**Range**	
	ram	(128,)	[0-255]	
	rgb	(210, 160, 3)	[0-255]	
	grayscale	(210, 160)	[0-255]	
Reward	You gain points for destroying space invaders. The invaders in the back rows are worth more points.			

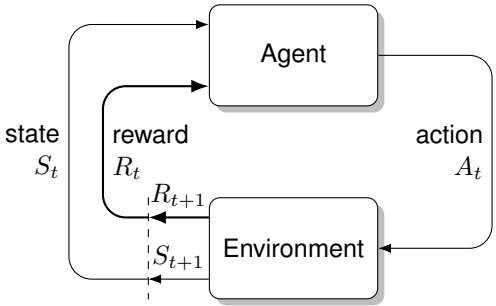
RL Primer



Agent-environment interaction loop in a RL system, from [Sutton and Barto, 2018]

- **State** (S_t): Representation of the environment at any given time. Helps agent to make some decision.
- **Action** (A_t): Decision that an agent makes in a state to move in the environment

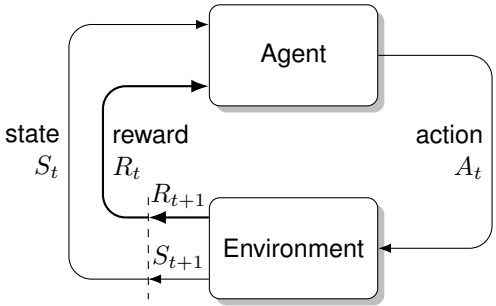
RL Primer



Agent-environment interaction loop in a RL system, from [Sutton and Barto, 2018]

- **Reward** (R_{t+1}): Immediate feedback from the environment, scalar signal for taking some action.
- **Return** (G): Cumulative reward over an episode/terminal condition
  $G_t = R_t + R_{t+1}...R_T$

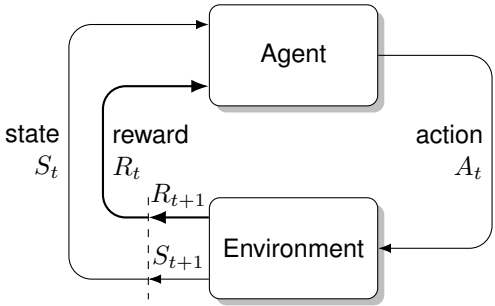
RL Primer



Agent-environment interaction loop in a RL system, from [Sutton and Barto, 2018]

- **Policy** (π): strategy by which agent select actions.
- **Optimal Policy**: What agent has to learn ultimately. i.e. a policy that maximizes the return

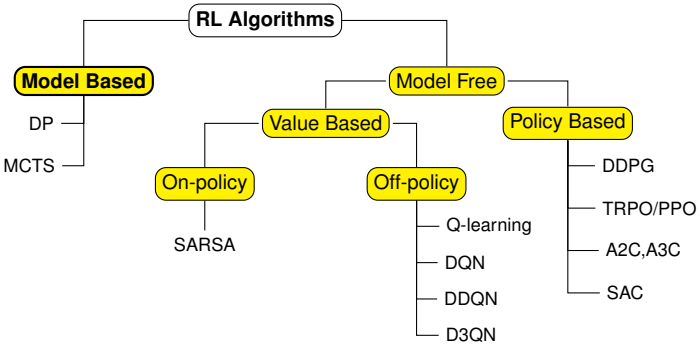
RL Primer



Agent-environment interaction loop in a RL system, from [Sutton and Barto, 2018]

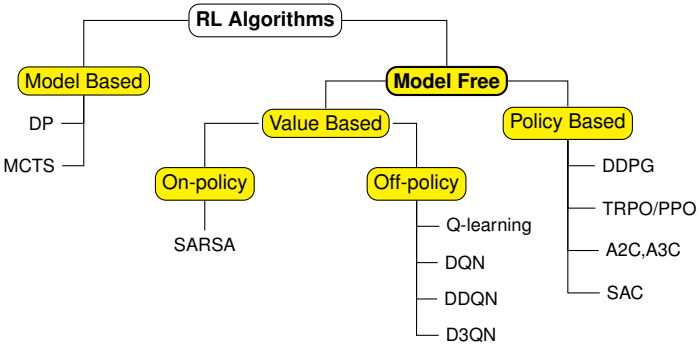
- **Value function** $V_\pi(s)$: Goodness of a state. Expected return from state s when following policy π.
- **Action-Value function or Q-value** $Q_\pi(s, a)$: Value of taking action a in state s following policy π.
- **Advantage value or function** $A_\pi(s, a)$: How much better it is to take one action over another. $A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$

Taxonomy of algorithms



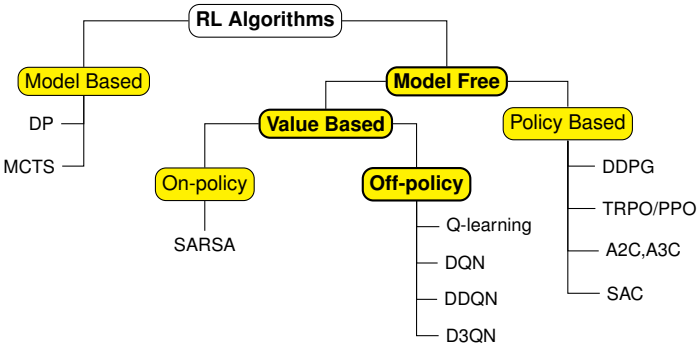
- **Model-based approaches**: Dynamics and transition probabilities of the environment are known. i.e. state in which an action will lead with known reward

Taxonomy of algorithms



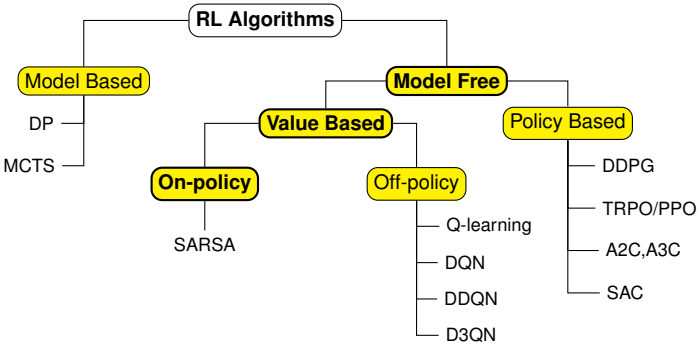
- **Model free approaches**: Most of the real world problems falls into this category. Internal dynamics are unknown. The only way to learn is through experience.

Taxonomy of algorithms



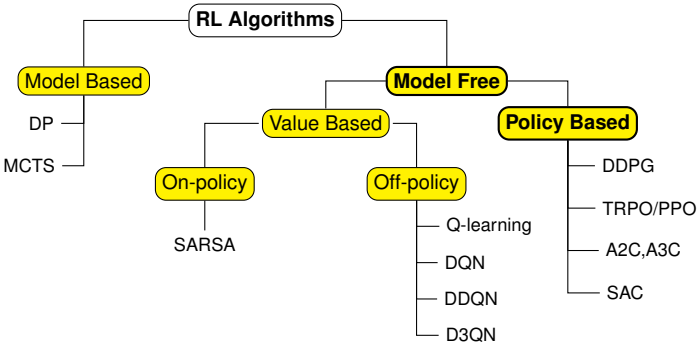
- **Model free approaches**: Most of the real world problems falls into this category. Internal dynamics are unknown. The only way to learn is through experience.
  - **Value Based**: Methods that learns the value function, which in turn derives an optimal policy
    - **Off-Policy**: Policy learned (*Target policy*) is different from policy used for exploring environment (*Behavior policy*).

Taxonomy of algorithms



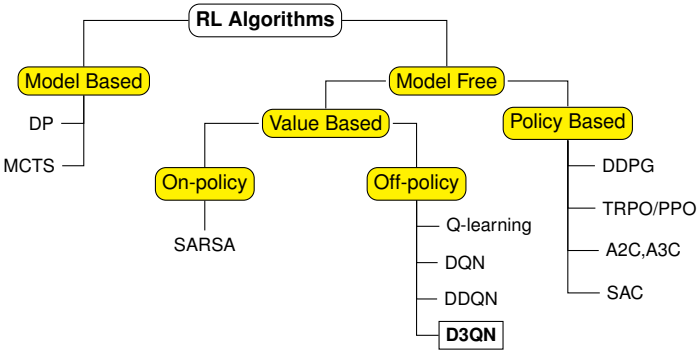
- **Model free approaches**: Most of the real world problems falls into this category. Internal dynamics are unknown. The only way to learn is through experience.
  - **Value Based**: Methods that learns the value function, which in turn derives an optimal policy
    - **On-Policy**: Target policy and behavior policy are same.

Taxonomy of algorithms



- **Policy Based**: Learns the optimal policy directly

Taxonomy of algorithms



RL Primer (cont.)

- **Bellman's Equation**: A way/rule to recursively represent the Q-function or value function. It is an aggregation of two parts:
  - **For value function**: Sum of immediate reward and return from next state
    $V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(s_{t+1})]$
  - **For Q-function:**
    - Immediate reward obtained from taking action a in state s to reach s_{t+1}
    - Discounted return from next state s_{t+1} on executing action a_{t+1} & thereafter following policy π
      $Q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1})]$

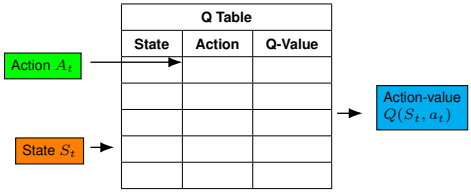
RL Primer (cont.)

- **Bellman's Equation**: A way/rule to recursively represent the Q-function or value function. It is an aggregation of two parts:
  - **For value function**: Sum of immediate reward and return from next state
    $V_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma V_\pi(s_{t+1})]$
  - **For Q-function:**
    - Immediate reward obtained from taking action a in state s to reach s_{t+1}
    - Discounted return from next state s_{t+1} on executing action a_{t+1} & thereafter following policy π
      $Q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1})]$
- **Temporal Difference learning (TD-learning)**: method to learn the value function based on the difference between successive value estimates. As per the TD-learning rule the value function at each time step is updated as:
$$V_\pi(S_t) \leftarrow V_\pi(S_t) + \alpha[R_{t+1} + \gamma V_\pi(S_{t+1}) - V_\pi(S_t)]$$
  - **TD-target**: $R_{t+1} + \gamma V_\pi(S_{t+1})$
  - **TD-error**: $R_{t+1} + \gamma V_\pi(S_{t+1}) - V_\pi(S_t)$

Q-learning

Q-learning [Watkins, 1989], [Watkins and Dayan, 1992]:

- Learning of the Q or action-value function
- **Model-free** & **off-policy** Temporal Difference (TD) control algorithm
- Tabular method. $Q \leftarrow S \bigtimes A$
- The optimal policy is given as:
  $\pi^{'} = \arg\max_{a} Q_{\pi}(s, a)$



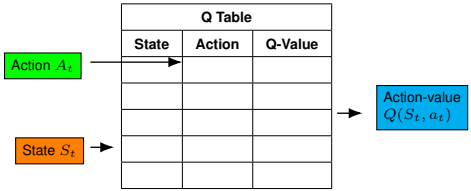
Q-learning

Q-learning

- Q-values are recursively updated as the agent interacts with the environment
- The Q-values are updated as per the TD-learning rule

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_t + \gamma * \max_a Q(s_t, a) - Q(S_t, a_t)]$$

where α is the learning rate and γ the discount factor ($0 \leq \gamma \leq 1$)



Q-learning

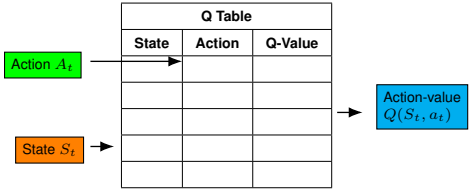
Q-learning

- Q-values are recursively updated as the agent interacts with the environment
- The Q-values are updated as per the TD-learning rule

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_t + \gamma * \max_a Q(s_t, a) - Q(S_t, a_t)]$$

where α is the learning rate and γ the discount factor ($0 \leq \gamma \leq 1$)

- **Does not scale well if the state space is large**

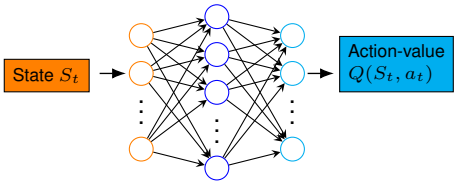


Q-learning

Deep Q-Network

- Deep Q-Network (DQN) [Mnih et al., 2015] makes use of Deep Neural Networks (NNs) to approximate the optimal Q-function
- It makes RL more like supervised learning.
- DQN learns to output the optimal Q-values by minimizing the loss $Q_{predicted} - Q_{actual}$ i.e. TD error $R_{t+1} + \gamma V_\pi(S_{t+1}) - V_\pi(S_t)$.
- It uses two networks.
  - **Policy Network**: To generate the Q-values for actions taken in the current state
  - **Target Network**: To create the Q-values for the next state

  Target network is updated after some epochs with the Policy network



Deep Q-Network (DQN)

Double Deep Q-Networks

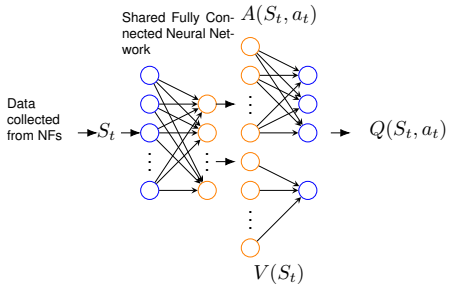
- DQN suffers from overestimation bias & moving target problems.
- DDQN overcomes them by using a different method of **TD target formation**
- Decouples action selection from target Q-value generation. It uses 2 networks similar to DQN.
  - uses Policy network to select the best action for next state (θ)
  - Q-value for the selected action for next state is generated using the target network. (θ')
- In the form of Bellman's equation

$$Q(s, a; \theta) = r + \gamma Q(s', argmax_{a'} Q(s', a'; \theta); \theta')$$

D3QN

Dueling Double Deep Q-networks (D3QN) [Wang et al., 2016] is an architectural enhancement to improve learning efficiency of Q-function:

- It combines the functionality of DQN & DDQN.
- D3QN can learn which states are valuable without considering the effect of each action.
- It decomposes the Q-function as the sum of:
  - **Value stream V(s)**: Estimate of how good it is to be in a state. V(s) is estimated by one separate stream
  - **Advantage stream A(s,a)**: How much extra reward an action can result into. A(s,a) is calculated by another stream

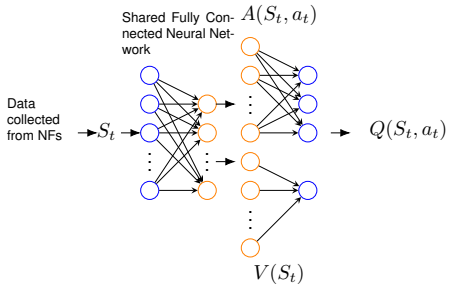


Overview of D3QN

D3QN

D3QN [Wang et al., 2016] is an architectural enhancement to improve learning efficiency of Q-function:

- It combines the functionality of DQN & DDQN.
- D3QN can learn which states are valuable without considering the effect of each action.
- V(s) and A(s,a) are combined through an aggregation layer. $Q(s,a) = V(s) + A(s,a)$
- $Q(S,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + (A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum A(s,a';\theta,\alpha))$



Overview of D3QN

Some further details

Prioritized Experience Replay (PER):

- **Experience Replay Buffer** from [[Lin, 1992]]: stores the experiences (s, a, r, s') collected by agent's interaction with the environment.
  - Common approach in Deep Q-networks.
  - Data is sampled randomly or uniformly from replay buffer to train the DNN
  - Helps in breaking in temporal correlation present in the sequential data
  - Introduces sample efficiency: sampling data from buffer ensures to have independent and identically distributed (i.i.d.) data.
- PER samples the experiences by prioritizing the samples with higher TD-error. By focusing more on challenging situation it helps with:
  - Improving sample efficiency
  - Faster convergence

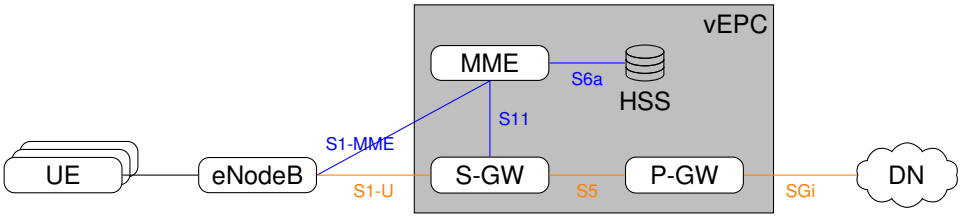
Some further details

Action Masking (AM) from [Huang and Ontañón, 2022] :

- Additional extension used in our experiments
- Some actions that are not allowed are masked during the exploration phase
- Example: turning off the last instance of a Network Function (NF) when one requirement is to always have at least one instance running

Scalability problem

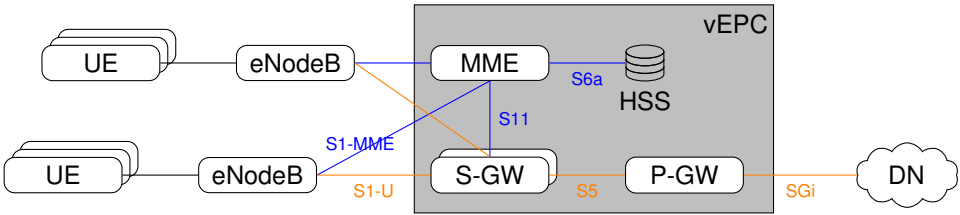
- The scalability problem for NFs is the scaling of cloud resources allocated to NFs according to the workload to guarantee the Quality of Service (QoS).
- This example focus on the horizontal scalability of virtualized Evolved Packet Core (vEPC) NFs
- Objective is to automatically scale the platform depending on the user traffic



Problem description for LTE NFs

Scalability problem

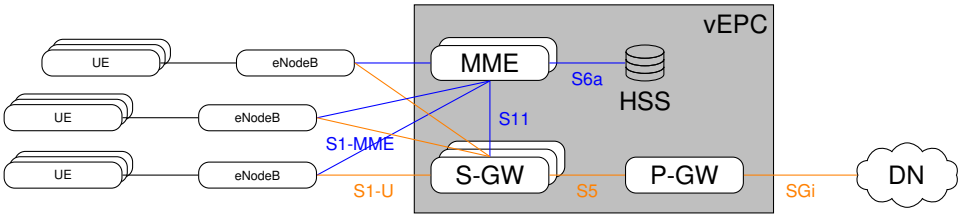
- The scalability problem for NFs is the scaling of cloud resources allocated to NFs according to the workload to guarantee the QoS.
- This example focus on the horizontal scalability of vEPC NFs
- Objective is to automatically scale the platform depending on the user traffic



Problem description for LTE NFs

Scalability problem

- The scalability problem for NFs is the scaling of cloud resources allocated to NFs according to the workload to guarantee the QoS.
- This example focus on the horizontal scalability of vEPC NFs
- Objective is to automatically scale the platform depending on the user traffic

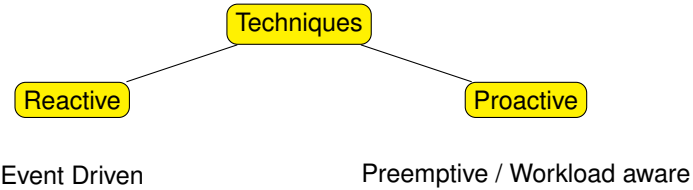


Problem description for LTE NFs

A short introduction of the problem statement

Autoscaling autonomously & dynamically provision (or deprovision) a set of resources in a way that

- caters fluctuant application workloads
- optimize the overall cost
- satisfies application SLA or SLO



Reactive vs Proactive autoscaling

Reactive: where system scales in real-time in response to the workload

- Threshold or event based, where resources as scaled as per pre-defined rules, e.g. scale out when CPU utilization reaches 80%, scale in when it reaches 20%
- Suitable for smooth and gradual workloads
- Pros:
  - Straightforward to implement
  - Scaling on-demand can save some cost & ensure availability
- Cons:
  - Resource flapping: scaling oscillations due to rapidly changing workload
  - longer provisioning time of resources can degrade SLA

Reactive vs Proactive Scaling

Proactive: when a system adjusts itself preemptively to account for upcoming workload variations.

- Generally involves use of control theory, queuing theory, predictive models or machine learning techniques. For example:
- Suitable for systems where SLA degradation and downtime are not acceptable.
  - Time series: forecasts the workload or resource consumption to scale ahead of time.
  - **Reinforcement Learning**: an agent learns a scaling policy by trial and error so that the policy will help achieve optimization objective
- Pros:
  - Scaling ahead of demand ensures application availability and QoS
  - Avoid taking too many actions when the load fluctuates a lot
- Cons:
  - More complex to design
  - Might be difficult to get approval from operational teams

Motivation for using RL for this use case

- Autoscaling can be referred to a classic automation control problem
- Commonly abstracted as MAPE control loop, which repeats itself over time [[Kephart and Chess, 2003]]
  - **M**onitoring: monitor the performance indicators at regular intervals
  - **A**nalysis: whether it is necessary to scale based on monitored KPIs
  - **P**lanning: how many resources to provision/deprovision
  - **E**xecution: execute the plan using cloud provider APIs
- **RL addresses optimal control problems**
  - Capability of learning effect of action over long-term
  - Optimization is a sequential decision making process.

Example of Core Network Cloud environment

The Open-Source vEPC Magma:

- Simulates a LTE network (5G was only in the roadmap when this work started)
- Magma combines Mobility Management Entity (MME), Serving Gateway (SGW), PDN Gateway (PGW) and Home Subscriber Server (HSS) NFs in one software called the Access Gateway (AGW)
- Provides a NMS based on Grafana and Prometheus

Example of Core Network Cloud environment

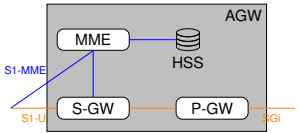
The Open-Source vEPC Magma:

- Simulates a LTE network (5G was only in the roadmap when this work started)
- Magma combines Mobility Management Entity (MME), Serving Gateway (SGW), PDN Gateway (PGW) and Home Subscriber Server (HSS) NFs in one software called the AGW
- Provides a NMS based on Grafana and Prometheus

More details on our testbed:

- Deployed in VMs on the Flexible Engine Cloud
- Tests are done using scripts that use the S1APTester Library of the Magma project

NFs included in Magma AGW



Example of Core Network Cloud environment

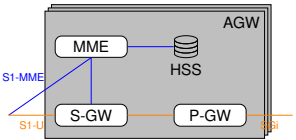
The Open-Source vEPC Magma:

- Simulates a LTE network (5G was only in the roadmap when this work started)
- Magma combines Mobility Management Entity (MME), Serving Gateway (SGW), PDN Gateway (PGW) and Home Subscriber Server (HSS) NFs in one software called the AGW
- Provides a NMS based on Grafana and Prometheus

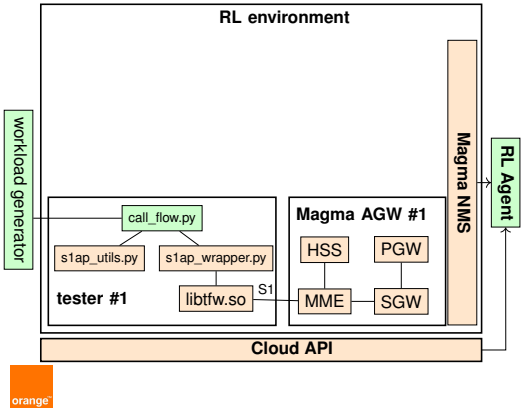
More details on our testbed:

- Deployed in VMs on the Flexible Engine Cloud
- Tests are done using scripts that use the S1APTester Library of the Magma project

NFs included in Magma AGW

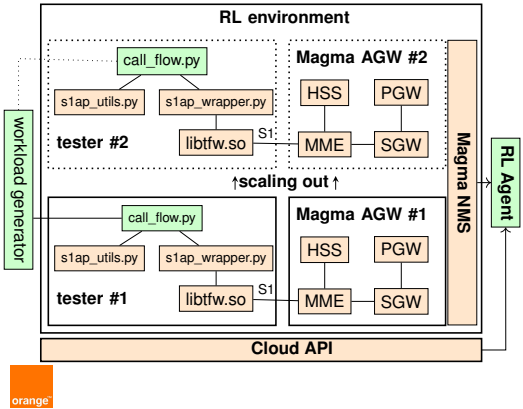


Testbed architecture



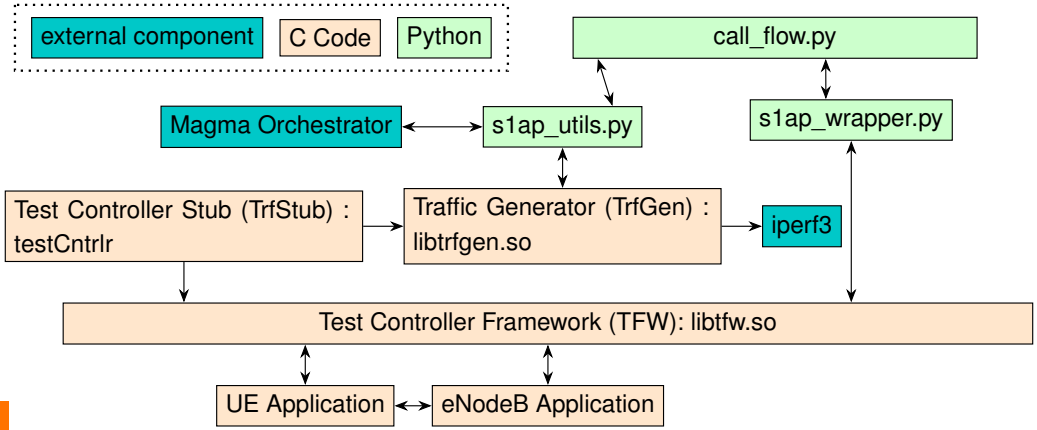
- **Workload generator** reads information about sessions to be simulated and distributes them to available instances of tester
- One-to-One relationship between testers and AGWs
- A tester host the S1APTester Library from Magma and the **call_flow.py** script which simulates User Equipments (UEs) and Radio Access Network (RAN)
- Only Control Plane traffic is created

Testbed architecture



- **Workload generator** reads information about sessions to be simulated and distributes them to available instances of tester
- One-to-One relationship between testers and AGWs
- A tester host the S1APTester Library from Magma and the **call_flow.py** script which simulates User Equipments (UEs) and Radio Access Network (RAN)
- Only Control Plane traffic is created

Tester: integration with S1APTester



Metrics

Metrics are collected on multiple points:

- Magma's Network Management System (NMS) (100+ metrics available): CPU, Memory and aggregated user information
- Directly inside AGW Virtual Machines (VMs): some relevant metrics are not available in Magma, e.g. memory usage of each components of the AGW (a custom script was developed)
- Flexible Engine's Application Programming Interface (API): number of running VMs

From the perspective of RL

In RL terms:

- **Environment** is the testbed
- **State** at time t is computed from data (metrics) collected on the testbed
- Possible **Actions** at time t

$$\boldsymbol{a}_t = \begin{cases} 0 & (\text{keep current NFs}) \\ 1 & (\textbf{scale out}\text{: add one AGW instance.}) \\ -1 & (\textbf{scale in}\text{: remove one AGW instance.}) \end{cases}$$

Specifics of the problem

Delays:

- Action delay: scaling in and out takes time, results of an action cannot be observed right away.
- Observation delay: metrics are not collected in real-time, 15s delay.

Specifics of the problem

Delays:

- Action delay: scaling in and out takes time, results of an action cannot be observed right away.
- Observation delay: metrics are not collected in real-time, 15s delay.

When some action is performed, instances of the AGW can **crash**:

- if a scale in action is performed when load is high,
- if no action is performed when testbed becomes overloaded.

Specifics of the problem

Delays:

- Action delay: scaling in and out takes time, results of an action cannot be observed right away.
- Observation delay: metrics are not collected in real-time, 15s delay.

When some action is performed, instances of the AGW can **crash**:

- if a scale in action is performed when load is high,
- if no action is performed when testbed becomes overloaded.

However:

- There is an exploration phase during the training process, so these actions can be evaluated.
- "*Normal*" behaviour for a software pushed beyond its limits.
- Multiple root causes for crashes: memory exhaustion, number of concurrent processes, ...

Components and Automation done for the real platform

Building Blocks of the RL pipeline

- **Infrastructure provisioning**: automated deployment of testbed
- **Action execution module**: automation for scaling of VMs
- **Data Collection module**: collection of near real-time state of the testbed
- **Monitoring**: tracking & monitoring of the RL experiments
- **Environment definition**: an interface to manage the RL environment as per MDP

Infrastructure provisioning

- **Magma requirement**: A cloud allowing to deploy VMs, e.g. Flexible Engine/GCP/AWS/Azure, ...
- Magma AGW and S1APTester are deployed in separate VMs
- Each AGW-S1APTester pair provisioning takes $\approx$ 12 minutes.
  - On-demand creation during agent training is impractical as it slows down the learning process.
  - Therefore, multiple AGW-S1APTester pairs are pre-deployed and activated or deactivated as needed.
- Automated AGW & S1APTester provisioning and configuration with **Terraform**

Infrastructure provisioning

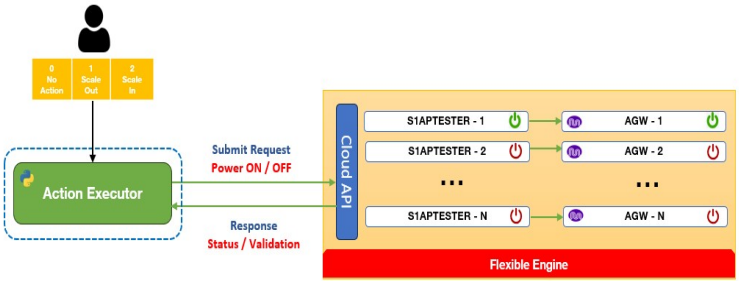
- **Magma requirement**: A cloud allowing to deploy VMs, e.g. Flexible Engine/GCP/AWS/Azure, ...
- Magma AGW and S1APTester are deployed in separate VMs
- Each AGW-S1APTester pair provisioning takes $\approx$ 12 minutes.
  - On-demand creation during agent training is impractical as it slows down the learning process.
  - Therefore, multiple AGW-S1APTester pairs are pre-deployed and activated or deactivated as needed.
- Automated AGW & S1APTester provisioning and configuration with **Terraform**

Why not containers ?

At the time of testing Magma was not container-ready. However the process is similar for a container-based packet core.

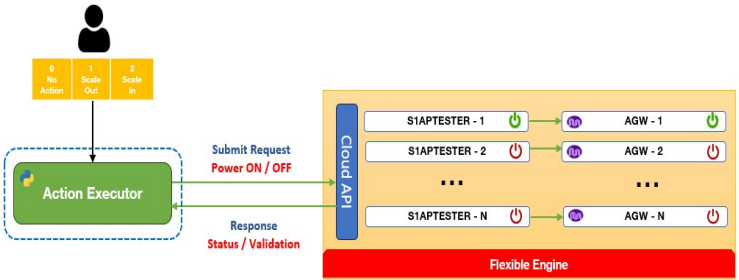
Action Execution module

- **Role**: To execute the decision selected by the RL agent which can be one of:
  - Scale-Out
  - Scale-In
  - No Action
- Leverages the API provided by Flexible engine



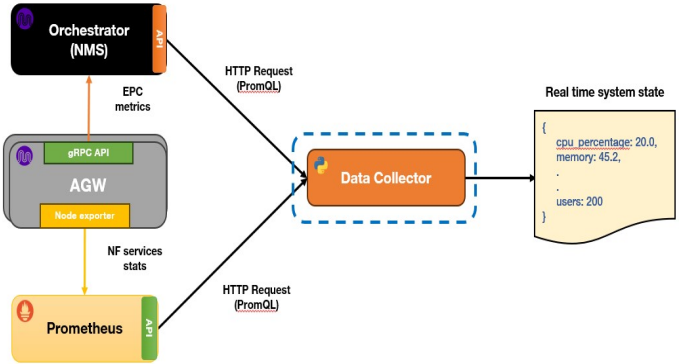
Action Execution module

- Scale-out or Scale-in by single instance based upon the request.
  Scale-Out -> Turn ON
  Scale-In -> Turn OFF
- Always a pair of AGW and S1APTester is scaled.



Data Collection module

Role: Collect the data from the monitoring sources for the agent to use as the current state

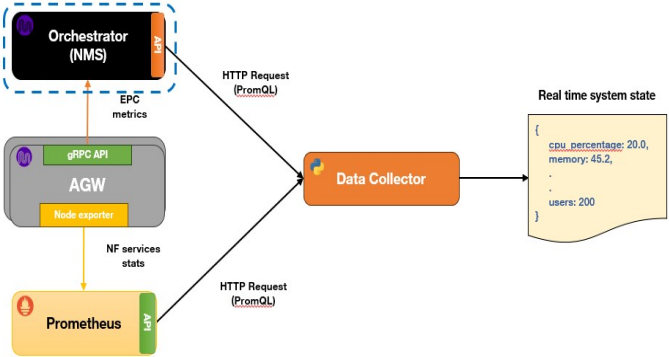


Data Collecting utility

Data Collection module

Role: Collect the data from the monitoring sources for the agent to use as the current state

- **Orchestrator**: A component of Magma which coordinates the services within the platform and manages the lifecycle of Network functions.
- It implements Prometheus as the monitoring stack which collects data related to gateways, subscribers, reliability, throughput etc.

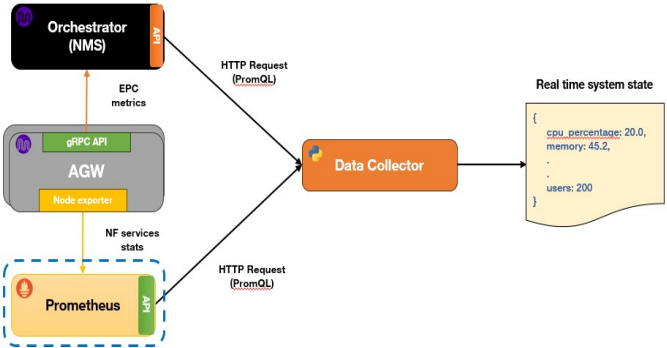


Data Collecting utility

Data Collection module

Role: Collect the data from the monitoring sources for the agent to use as the current state

- **Prometheus**: A standalone Prometheus is deployed to collect custom metrics which are ***not*** provided by Orchestrator.
- Every AGW is configured with the Prometheus Node exporter's textile collector.

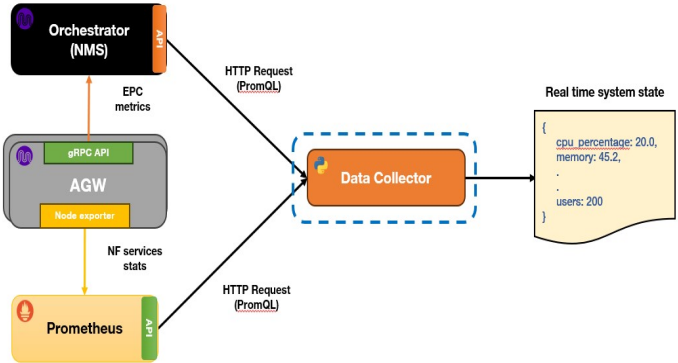


Data Collecting utility

Data Collection module

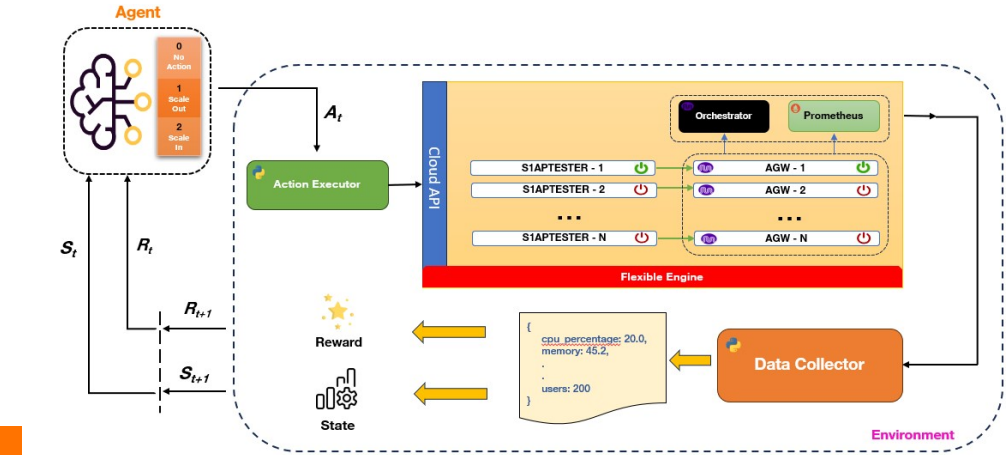
Collects data via APIs exposed by Magma NMS and Prometheus.

- **From Orchestrator**:
  - Connected Users (UEs)
  - UE attach rate/5 minute
  - % CPU usage of AGW
  - Memory consumption of AGW
- **From Prometheus**:
  - Memory consumption of NFs inside AGW like MME.
  - Number of dropped calls



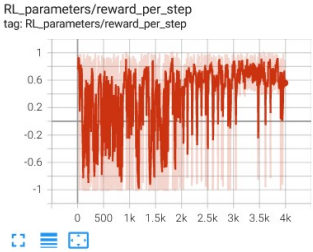
Data Collecting utility

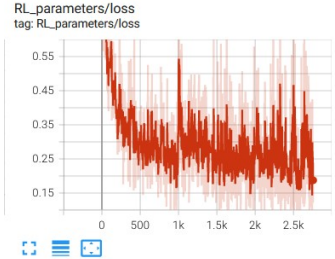
Overall pipeline



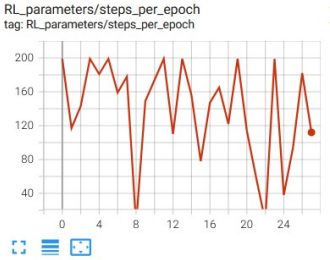
S. Verma, G. Fraysse

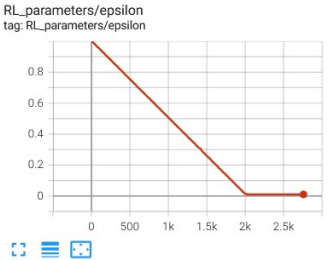
Logging and monitoring

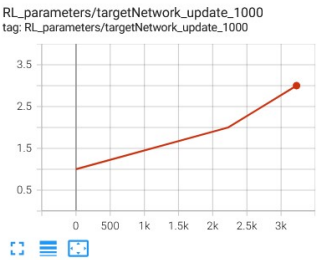




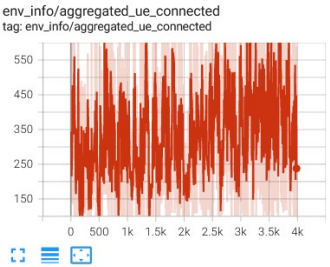
Logging and monitoring

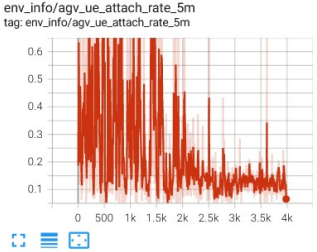


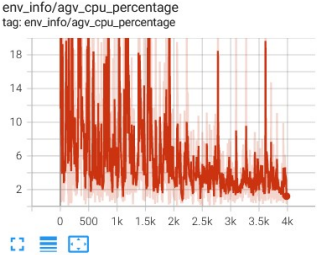




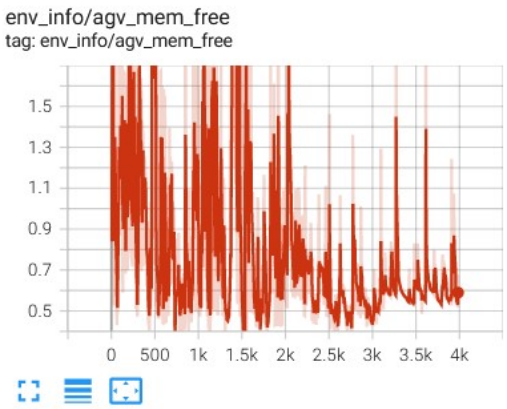
Logging and monitoring

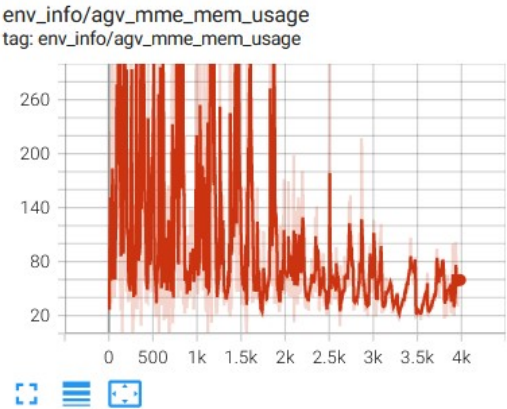




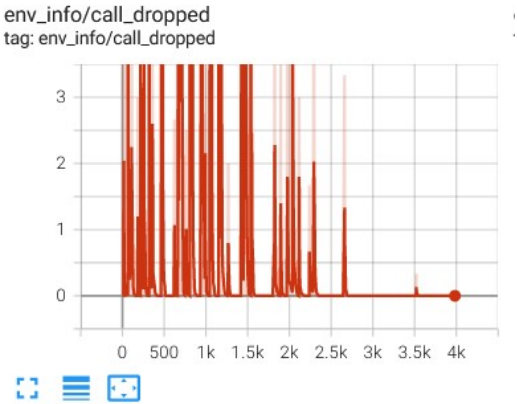


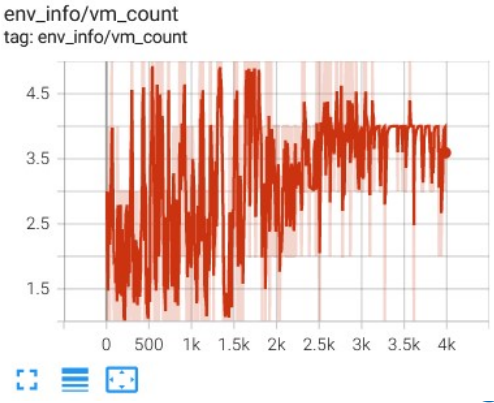
Logging and monitoring



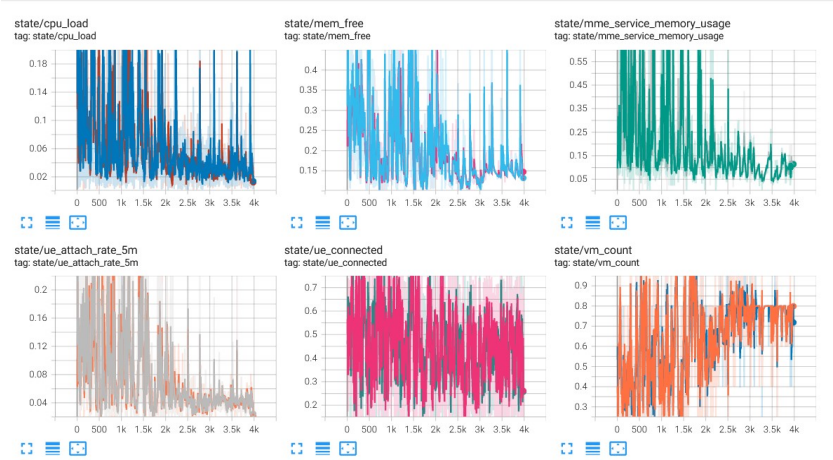


Logging and monitoring





Logging and monitoring



Resources on RL

- Reinforcement Learning: An Introduction (second edition) by Richard S. Sutton and Andrew G. Barto. Licensed under the Creative Commons Attribution-NonCommercial-NoDerivs 2.0 Generic License
- Stanford CS234: Reinforcement Learning | Winter 2019 the complete CS234: Reinforcement Learning course of Prof. Emma Brunskill at Stanford captured on video
- David Silver's course it includes a link to the video captures
- Reinforcement Learning Specialization by Martha White and Adam White at the University of Alberta, available on Coursera.

Bibliography

Berner, C., Brockman, G., Chan, B., Cheung, V., Dębiak, P., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., et al. (2019). Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*.

Huang, S. and Ontañón, S. (2022). A closer look at invalid action masking in policy gradient algorithms. *The International FLAIRS Conference Proceedings*, 35.

Kephart, J. O. and Chess, D. M. (2003). The vision of autonomic computing. *Computer*, 36(1):41–50.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8:293–321.

Mitchell, T. (1997). Learning, machine. *McGraw Hill*.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Qiu, H., Banerjee, S. S., Jha, S., Kalbarczyk, Z. T., and Iyer, R. K. (2020). FIRM: An intelligent fine-grained resource management framework for SLO-Oriented microservices. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 805–825. USENIX Association.

Qiu, H., Mao, W., Patke, A., Wang, C., Franke, H., Kalbarczyk, Z. T., Başar, T., and Iyer, R. K. (2022). SIMPPO: A scalable and incremental online learning framework for serverless resource management. In *Proceedings of the 13th Symposium on Cloud Computing*, SoCC '22, page 306–322, New York, NY, USA. Association for Computing Machinery.

Bibliography (cont.)

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT Press.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in Starcraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016). Dueling Network Architectures for Deep Reinforcement Learning. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1995–2003. PMLR.

Watkins, C. J. C. H. (1989). Learning from delayed rewards. *PhD dissertation*.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3–4):279–292.