Reinforcement Learning for Network Management

MADRE

Guillaume Fraysse

Orange Research, Châtillon,
France

January 12^{th}, 2026

Slides and code samples

slides and code at https://github.com/gfraysse/MADRE

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

Current members from Orange team:

- Guillaume FRAYSSE
- David DELANDE
- Edgar FERNANDES
- Abhishek GUPTA
- Abhishek SAINI
- Shantanu VERMA

Past members:

- Manu CHAUHAN
- Francesca FOSSATI
- Imen GRIDA BEN YAHIA
- Xinqi LONG (intern)
- Yoichi MATSUO (NTT Network Service Systems Laboratories)
- Jose Manuel SANCHEZ VILCHEZ
- Jatinder SINGH
- Gabriel Gomes De SIQUEIRA (intern)

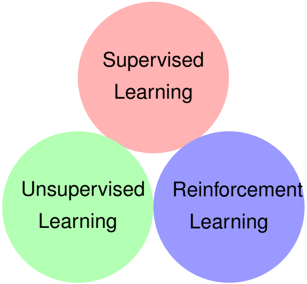
Related publications

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
- S. Verma and G. Fraysse, Setting up a Reinforcement Learning pipeline for a Telco Core Network 27^{th}Conference on Innovation in Clouds Internet and Networks (ICIN) Tutorial, Paris, France, 2024
- X. Long and G. Fraysse, Safe RL for Core Network autoscaling, 2024 20^{th} International Conference on Network and Service Management (CNSM), Prague, Czech Republic, 2024, pp. 1-7, doi: 10.23919/CNSM62983.2024.10814355.
- S. Verma, A. Gupta, J. Singh, J. M. Sanchez Vilchez, G. Fraysse, Adaptive control policies for Core Network autoscaling with Meta-RL 2025 21^{th} International Conference on Network and Service Management (CNSM), Bologna, Italy, 2025.

Definitions: the 3 main paradigms of Machine Learning

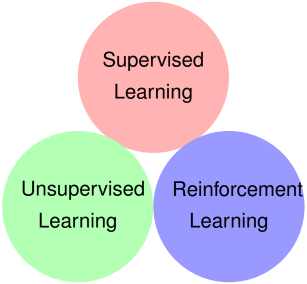
- **Machine Learning (ML)**: A computer program is said to learn from **experience E** with respect to some **class of tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E. (Prof. Tom M. Mitchell [Mitchell, 1997])



A Taxonomy of Machine Learning

Definitions: the 3 main paradigms of Machine Learning

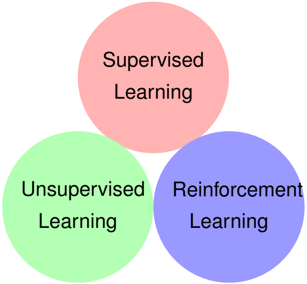
- **Machine Learning (ML)**: A computer program is said to learn from **experience E** with respect to some **class of tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E. (Prof. Tom M. Mitchell [Mitchell, 1997])
    - **Supervised Learning**: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



A Taxonomy of Machine Learning

Definitions: the 3 main paradigms of Machine Learning

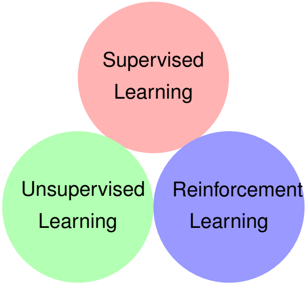
- **Machine Learning (ML)**: A computer program is said to learn from **experience E** with respect to some **class of tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E. (Prof. Tom M. Mitchell [Mitchell, 1997])
  - **Supervised Learning**: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

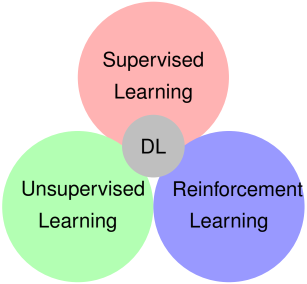
- **Machine Learning (ML)**: A computer program is said to learn from **experience E** with respect to some **class of tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E. (Prof. Tom M. Mitchell [Mitchell, 1997])
  - **Supervised Learning**: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

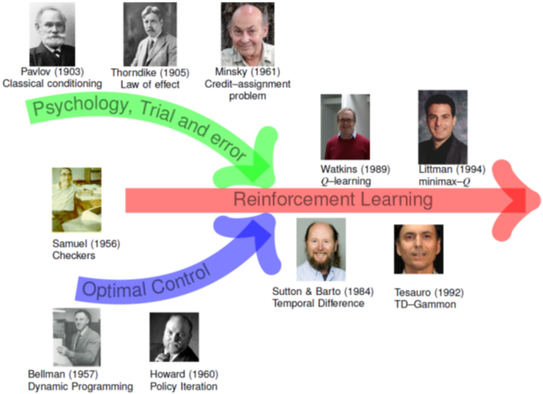
- **Machine Learning (ML)**: A computer program is said to learn from **experience E** with respect to some **class of tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E. (Prof. Tom M. Mitchell [Mitchell, 1997])
  - **Supervised Learning**: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

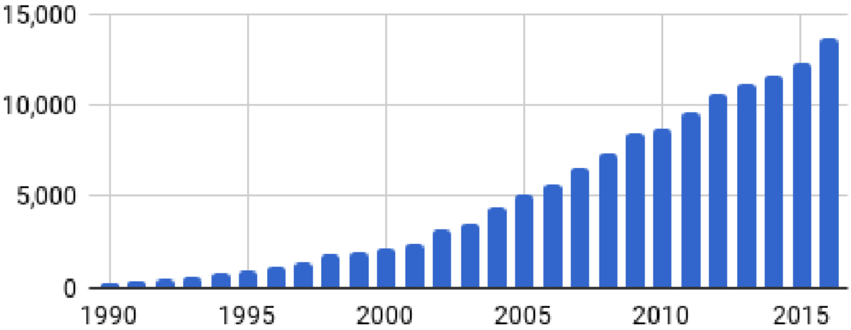
Deep Learning (DL): subset of ML methods based on Neural Networks.

History of Reinforcement Learning



From Prof. Restelli
https://sites.google.com/view/rlan2025/

Growing interest in RL



Evolution of the number of new RL-related publications per year, from [Henderson et al., 2018]

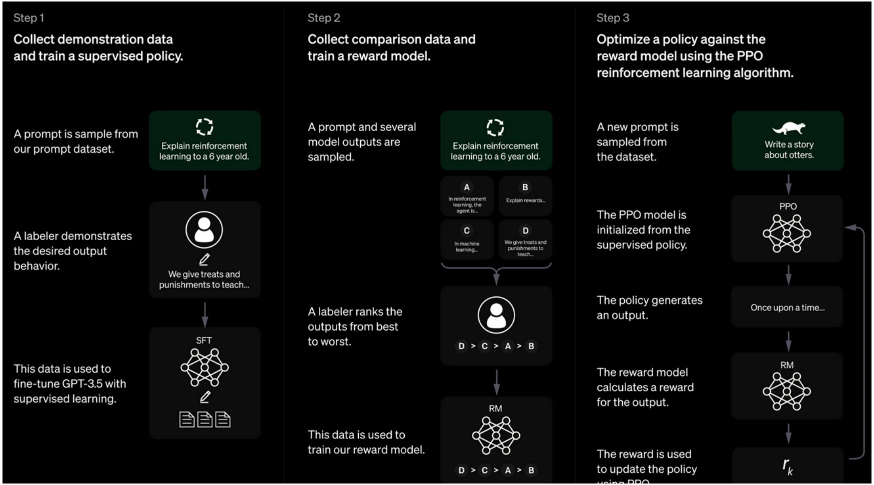
RL Application Areas 1/2

Application Areas	Problem Statement	Environment
Gaming	Chess, Pinball, Atari 2600 Games ([Mnih et al., 2013]), AlphaGo (2015), AlphaStar ([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]), . . .

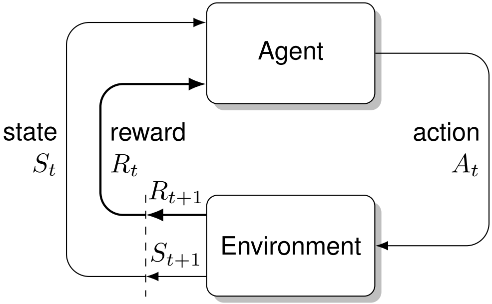
RL Application Areas 2/2

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, ...)

ChatGPT (from https://openai.com/blog/chatgpt/)



Reinforcement Learning



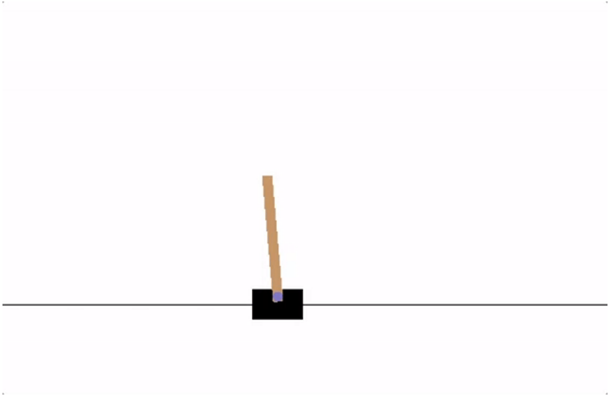
Agent-environment interaction loop in a RL system, from [Sutton and Barto, 2018]

A basic example: moving in a maze

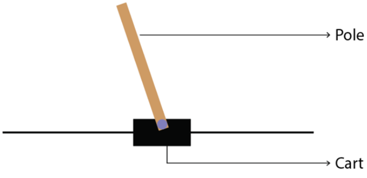
Maze (from Hugging Face tutorial)

Element	**Values set**
Action Space	direction the mouse goes to: $\Uparrow$, $\Downarrow$, $\Leftarrow$ or $\Rightarrow$
Observation Space	the cell the mouse is in
Reward	+0: Going to a state with no cheese in it. +1: Going to a state with a small cheese in it. +10: Going to the state with the big pile of cheese. -10: Going to the state with the poison.

Cart Pole



from https://docs.pytorch.org/tutorials/intermediate/reinforcement_q_learning.html beginframe



Cart Pole

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

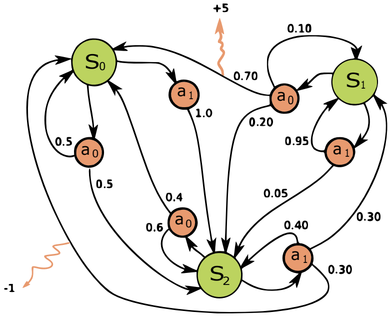
beginframe

Space Invaders

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

RL modeled as Markov Decission Process (MDPs)

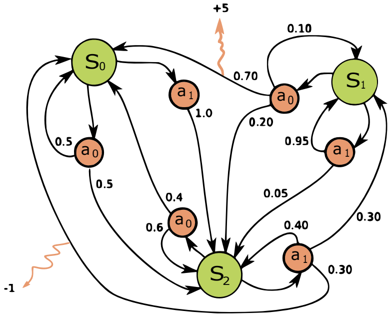
- Markov Decision Processes(MDPs) are the most common models for RL



Example of a Markov Decision Process, (author: waldoalvarez, licence Creative Commons)

RL modeled as Markov Decission Process (MDPs)

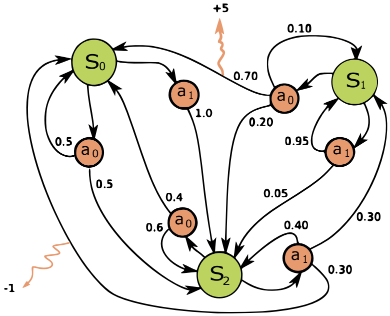
- Markov Decision Processes(MDPs) are the most common models for RL
- Markov assumptions:
  - Information state: sufficient statistic of history
  - State s_t is Markov if and only if: $p(s_{t+1}|s_t, a_t) = p(s_{t+1}|h_t, a_t)$ where History $h_t = (a_1, s_1, r_1, ..., a_t, s_t, r_t)$
  - Future is independent of past given present



Example of a Markov Decision Process, (author: waldoalvarez, licence Creative Commons)

RL modeled as Markov Decission Process (MDPs)

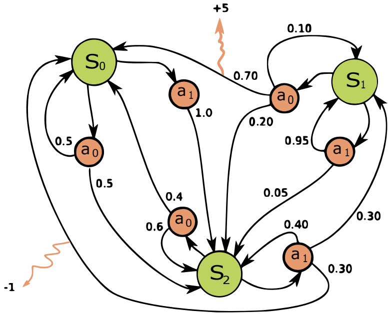
- Markov Decision Processes(MDPs) are the most common models for RL
- Markov assumptions:
  - Information state: sufficient statistic of history
  - State s_t is Markov if and only if: $p(s_{t+1}|s_t, a_t) = p(s_{t+1}|h_t, a_t)$ where History $h_t = (a_1, s_1, r_1, ..., a_t, s_t, r_t)$
  - Future is independent of past given present
- behavior of an agent is defined by a **policy**



Example of a Markov Decision Process, (author: waldoalvarez, licence Creative Commons)

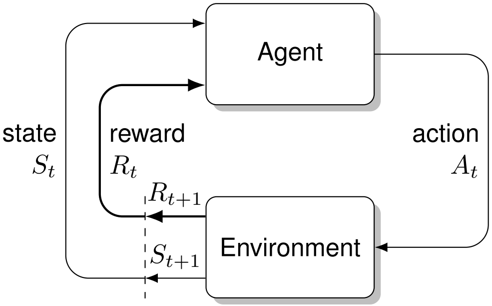
RL modeled as Markov Decission Process (MDPs)

- Markov Decision Processes(MDPs) are the most common models for RL
- Markov assumptions:
  - Information state: sufficient statistic of history
  - State s_t is Markov if and only if: $p(s_{t+1}|s_t, a_t) = p(s_{t+1}|h_t, a_t)$ where History $h_t = (a_1, s_1, r_1, ..., a_t, s_t, r_t)$
  - Future is independent of past given present
- behavior of an agent is defined by a **policy**
- Goal of RL is to find the **optimal** policy



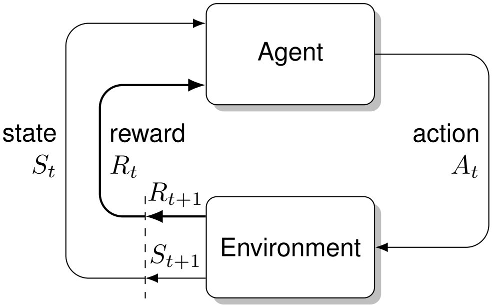
Example of a Markov Decision Process, (author: waldoalvarez, licence Creative Commons)

RL definitions



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

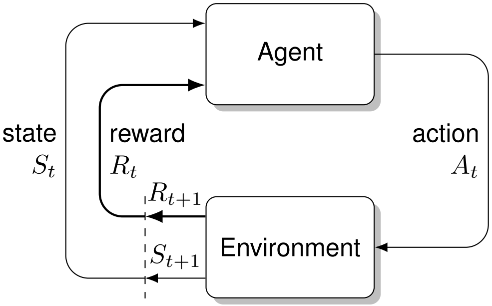
RL definitions



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

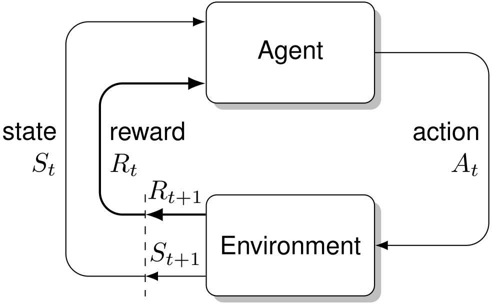
RL definitions



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
  $G_t = R_t + R_{t+1} ... + R_T$

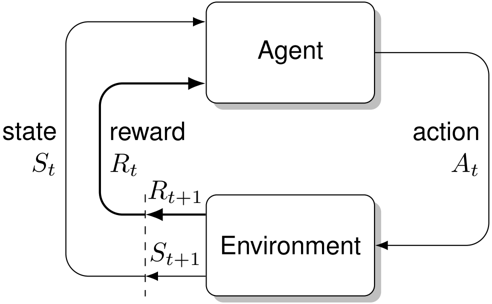
RL definitions



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

- **Policy** (π) determines how the agent chooses actions. $\pi : S \to A$
  - Deterministic policy: $\pi(s) = a$
  - Stochastic policy: $\pi(a|s) = P(a_t = a|s_t = s)$
- **Optimal Policy**: What agent has to learn ultimately. i.e. a policy that maximizes the return

RL definitions



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.
- **Q-value** $Q_\pi(s, a)$ (or Action-Value function): 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. $A_\pi(s, a) = Q_\pi(s, a) - V_\pi(s)$

MDP property

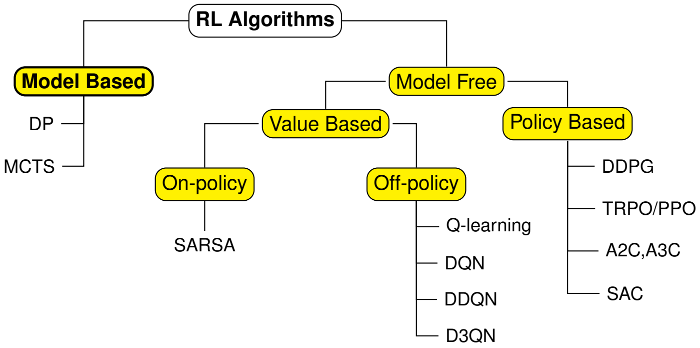
Theorem

For any Markov Decision Process:

- There exists an Optimal Policy π^*, i.e., there exists a Policy π^* such that $V^*_\pi(s) \geq V_\pi(s)$ for all policies π and for all states $s \in S$
- All Optimal Policies achieve the Optimal Value Function, i.e. $V^*_\pi(s) = V^*(s)$ for all $s \in S$, for all Optimal Policies π^*
- All Optimal Policies achieve the Optimal Action-Value Function, i.e. $Q^*_\pi(s,a) = Q^*(s,a)$ for all $s \in S$, for all $a \in A$, for all Optimal Policies π^*

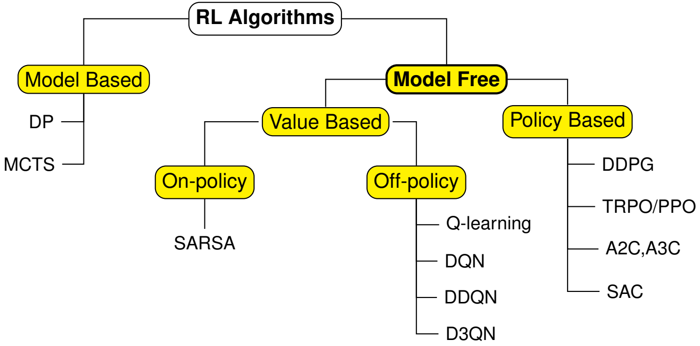
Cf. for example Prof. Ashwin Rao's course
https://web.stanford.edu/class/cme241/lecture_slides/OptimalPolicyExistence.pdf

Taxonomy of algorithms



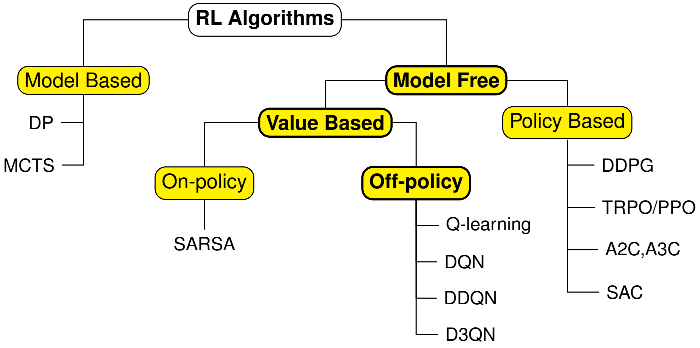
- **Model-based approaches**: Dynamics and transition probabilities of the environment are known. i.e. state in which an action will lead with known reward

Taxonomy of algorithms



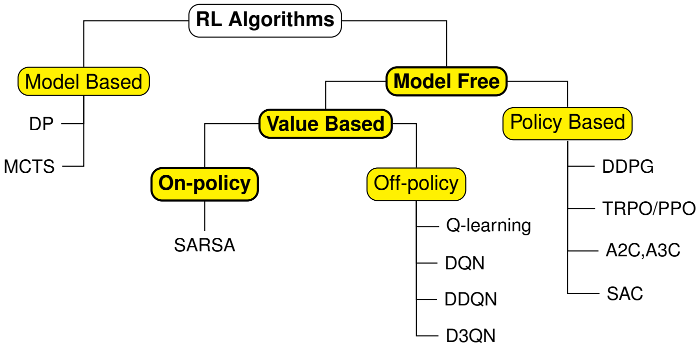
- **Model free approaches**: Most of the real world problems falls into this category. Internal dynamics are unknown. The only way to learn is through experience.

Taxonomy of algorithms



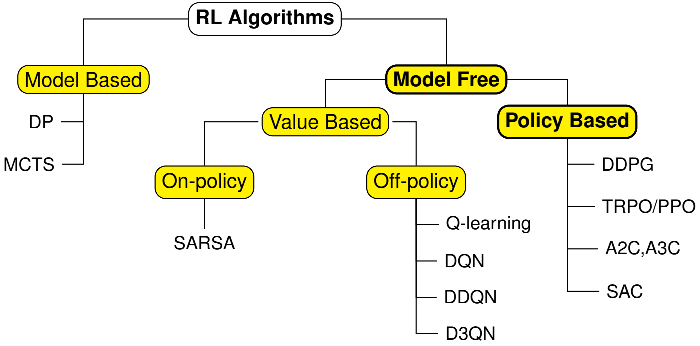
- **Model free approaches**: Most of the real world problems falls into this category. Internal dynamics are unknown. The only way to learn is through experience.
  - **Value Based**: Methods that learns the value function, which in turn derives an optimal policy
    - **Off-Policy**: Policy learned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

Taxonomy of algorithms



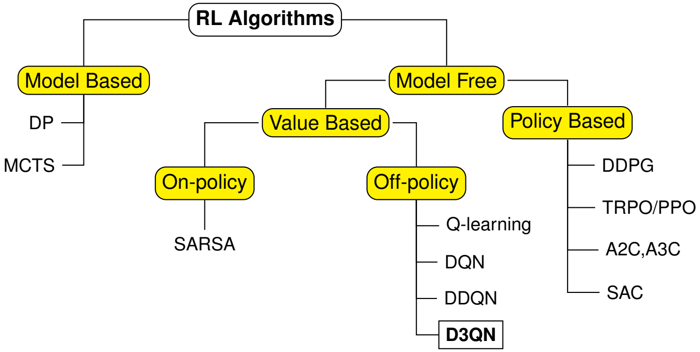
- **Model free approaches**: Most of the real world problems falls into this category. Internal dynamics are unknown. The only way to learn is through experience.
  - **Value Based**: Methods that learns the value function, which in turn derives an optimal policy
    - **On-Policy**: Target policy and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

Taxonomy of algorithms



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

Taxonomy of algorithms



Some fundamental equations in RL

- **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
      $Q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma Q_\pi(s_{t+1}, a_{t+1})]$

Some fundamental equations in RL

- **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:

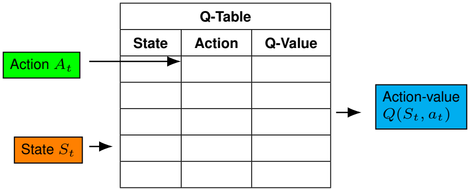
$$V_\pi(S_t) \leftarrow V_\pi(S_t) + \alpha[R_{t+1} + \gamma V_\pi(S_{t+1}) - V_\pi(S_t)]$$

  - **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]:

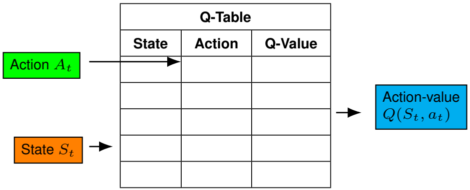
- Learning of the Q (or *action-value*) function
- **Model-free** & **off-policy** Temporal Difference (TD) control algorithm
- Tabular method. $Q \leftarrow S \times A$
- The optimal policy is given as: $\pi^* = \arg\max_a Q_\pi(s, a)$



Q-table for Q-learning

Q-learning

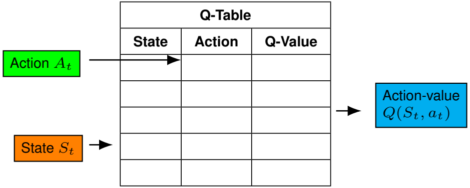
- Q-values are recursively updated as the agent interacts with the environment
- ϵ-greedy policy:
  - **Exploration**: Random actions with probability ϵ.
  - **Exploitation**: as ϵ value decays agent follow the learned policy
- The Q-values are updated as per the TD-learning rule $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_t + \gamma * \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$



Q-table for Q-learning

Q-learning

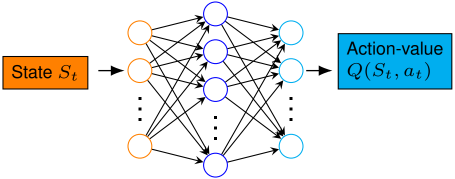
- Q-values are recursively updated as the agent interacts with the environment
- ϵ-greedy policy:
  - **Exploration**: Random actions with probability ϵ.
  - **Exploitation**: as ϵ value decays agent follow the learned policy
- The Q-values are updated as per the TD-learning rule
  $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_t + \gamma * \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$
- **Does not scale well if the state space is large**: *the curse of dimensionality*



Q-table for 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
- 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

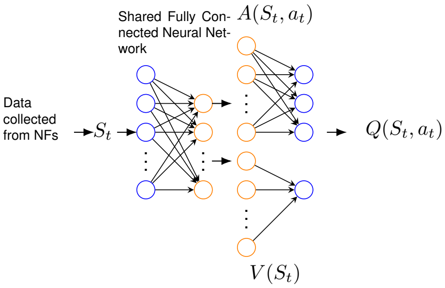
- DQN suffers from overestimation bias & moving target problems.
- DDQN overcomes them by using a different method of **TD target formation**
- Decouples action selection from target Q-value generation. It uses 2 networks similar to DQN.
  - uses Policy network to select the best action for next state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

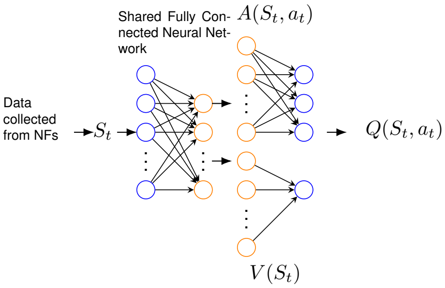


Overview of D3QN

D3QN

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

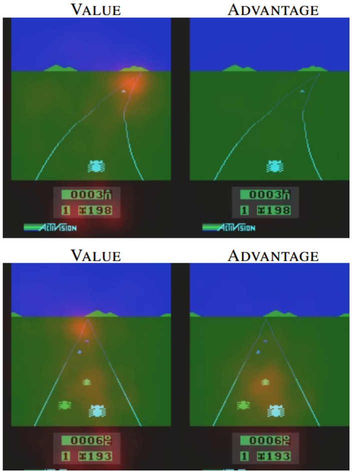
- It combines the functionality of DQN & DDQN.
- D3QN can learn which states are valuable without considering the effect of each action.
- V(s) and A(s,a) are combined through an aggregation layer. $Q(s,a) = V(s) + A(s,a)$
- $Q(S, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + (A(s, a; \theta, \alpha)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



Overview of D3QN

D3QN: Advantage and Value streams behaviors

The advantage stream learns to pay attention only when there are cars immediately in front, so as to avoid collisions. (from [Wang et al., 2016])



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

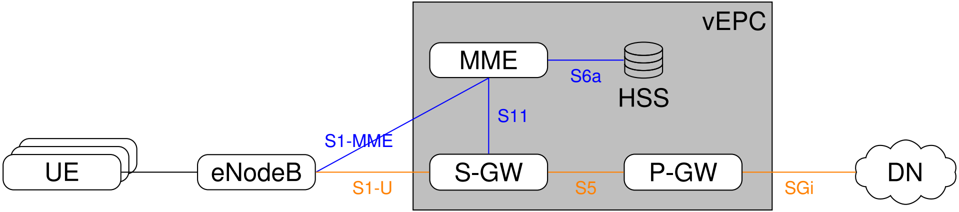
Some further details

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

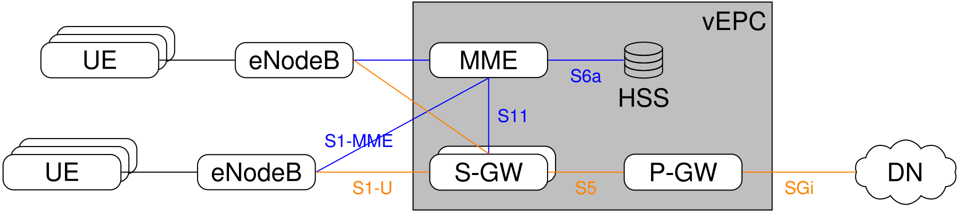
- The scalability problem for NFs is the scaling of cloud resources allocated to NFs according to the workload to guarantee the Quality of Service (QoS).
- This example focus on the horizontal scalability of virtualized Evolved Packet Core (vEPC) NFs
- Objective is to automatically scale the platform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



Problem description for LTE NFs

Scalability problem

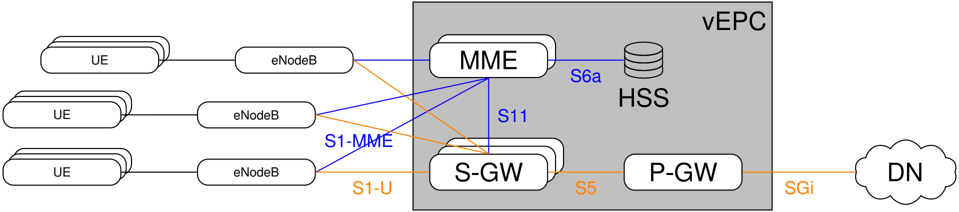
- The scalability problem for NFs is the scaling of cloud resources allocated to NFs according to the workload to guarantee the QoS.
- This example focus on the horizontal scalability of vEPC NFs
- Objective is to automatically scale the platform depending on the user traffic



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

Techniques

Reactive — Event Driven

Proactive — Preemptive / Workload aware

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.

Reward function

Metric	Definition
U	Number of User Equipments (UEs) connected
M	Memory usage, in MB
D	Number of dropped sessions

- Maximum reward value is 1
- Encourages the NFs to use resources optimally around 70%
- Resource usage above 80% or crashes are penalized with lowest reward value -1.

$$r = \begin{cases} 1 - (0.7 - \max(M, U) - D) & \text{if M, U} \in [0, 80] \\ \max(-\max(M, U) - 10 * D, -1) \text{otherwise.} \end{cases}$$

Lessons learned

RL is complex:

- Not your typical network or software engineer skill
- Very active area of research

Lessons learned

RL is complex:

- Not your typical network or software engineer skill
- Very active area of research

Automation of Network Core scaling:

- Development is required
- Probably better to use Infrastructure as Code framework to be IaaS-independent

Lessons learned

RL is complex:

- Not your typical network or software engineer skill
- Very active area of research

Automation of Network Core scaling:

- Development is required
- Probably better to use Infrastructure as Code framework to be IaaS-independent

Load generation is always tricky:

- Commercial products exist
- Lack of open source tools to generate traffic in a consistent way for a long time
- Traffic needs to be balanced across all instances

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 | Spring 2024 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*.

Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D. (2018). Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32.

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.

Bibliography (cont.)

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.

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.