Introduction to (Deep) Reinforcement Learning

Sharwin Rezagholi
sharwin.rezagholi[at]technikum-wien.at

Faculty of Computer Science and Applied Mathematics
University of Applied Sciences Technikum Wien

Vienna Deep Learning Meetup @ Robert Bosch AG
May 4, 2023

Reinforcement learning and its applications

Reinforcement learning (RL) is the subfield of machine learning that enables agents to take actions in their environment while maximizing a notion of cumulative reward. In particular, the learning procedure itself solely requires interaction with the environment and the processing of appropriate numerical feedback from the environment.

- Applications:
  - Recommender systems (e.g. online shops, streaming services, news sites, social media)
  - Online advertisement allocation
  - Production line optimization, smart maintenance
  - Robotics (e.g. motion planning)
  - (Video-)Game playing (Chess, Go, AlphaStar)
  - Facility (energy) management systems
  - Automated driving

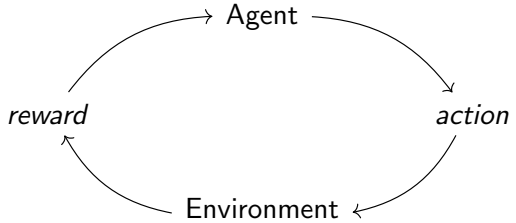
Recall: Supervised learning

Example_input $\longrightarrow$ *Example_output*

Example_input $\longrightarrow$ Supervised learning system $\longrightarrow$ *Model_output*

- Error = distance(Example_output, Model_output)
- Learning from examples.
- Learning with a 'teacher'.

Reinforcement learning



- The basic framework of reinforcement learning, and control theory more generally, is the *feedback loop*.
- The main lever of reinforcement learning is the difference between the obtained reward and the expected reward. Its aim is to use this difference as a signal for learning algorithms.
- Trial-and-error learning. Learning with a 'critic'.
- Critic does not criticize the outcome of the learning system directly but the effect of this outcome!
- The learner produces its own data by interacting with the environment.

Theoretical Ecology: Welt and Umwelt

Jakob Johann von Uexküll (Theoretische Biologie 1920) proposed to study the behavior of animals and their interactions according to the following diagram.

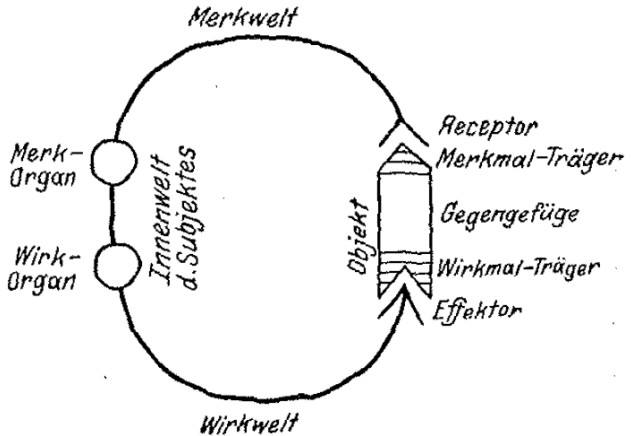
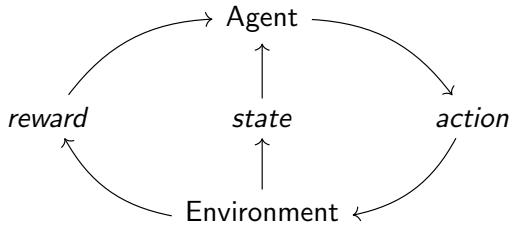


Abb. 3. Schema des Funktionskreises.

Feedback loop (with state change)



- Crucially, the action of the agent may change the environment.
- Dynamic problem: The agent wants to maximize the cumulative reward of its interaction with the environment. How to trade-off instantaneous reward with long-term cumulative reward?
- Mathematical model framework: Markov decision processes.

Example: Playing Atari Breakout

State-Action values

- We need to know the value of taking an action in a certain state. The RL task is the estimation of state-action values.
- The value of taking action a in state s is

$$q^*(s,a) = \mathbb{E}\Big(r + \gamma \cdot \max_{a'} q^*(s',a')\Big)$$

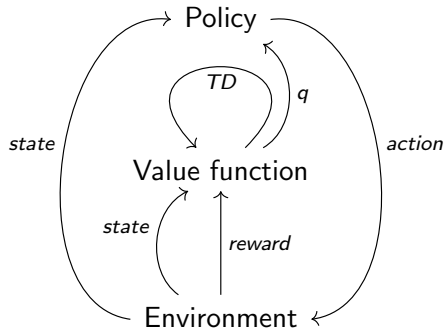
  where r is today's reward, s' is the next state, a' is the next action, and $\gamma \in (0,1)$ is a discount factor.
- Minimize the temporal difference error (TD):

$$TD = \Big(r + \gamma \cdot \max_{a'} q(s',a')\Big) - q(s,a)$$

- Bootstrapping: We use estimates to make estimates better!

Main idea: Temporal difference learning

- TD learning started in the 80s (Sutton, Barto, ...)
- The q-learning algorithm stems from the early 90s (Dayan, ...)



Neurobiological artifacts of reinforcement learning?

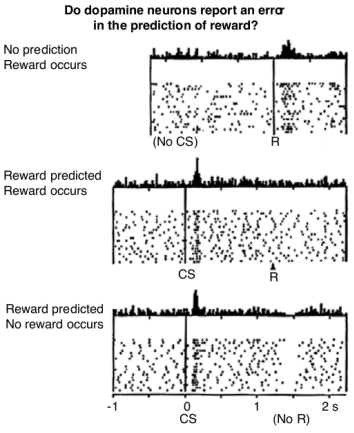


Fig. 1. Changes in dopamine neurons' output code for an error in the prediction of appetitive events. (**Top**) Before learning, a drop of appetitive fruit juice occurs in the absence of prediction—hence a positive error in the prediction of reward. The dopamine neuron is activated by this unpredicted occurrence of juice. (**Middle**) After learning, the conditioned stimulus predicts reward, and the reward occurs according to the prediction—hence no error in the prediction of reward. The dopamine neuron is activated by the reward-predicting stimulus but fails to be activated by the predicted reward (right). (**Bottom**) After learning, the conditioned stimulus predicts a reward, but the reward fails to occur because of a mistake in the behavioral response of the monkey. The activity of the dopamine neuron is depressed exactly at the time when the reward would have occurred. The depression occurs more than 1 s after the conditioned stimulus without any intervening stimuli, revealing an internal representation of the time of the predicted reward. Neuronal activity is aligned on the electronic pulse that drives the solenoid valve delivering the reward liquid (top) or the onset of the conditioned visual stimulus (middle and bottom). Each panel shows the peri-event time histogram and raster of impulses from the same neuron. Horizontal distances of dots correspond to real-time intervals. Each line of dots shows one trial. Original sequence of trials is plotted from top to bottom. CS, conditioned, reward-predicting stimulus; R, primary reward.

Schultz et al. 1997, A neural substrate of prediction and reward, *Science* 275.

The need for function approximation

- On small problems (small state sets, small action sets) tabular learning of state-action values through temporal differences works. On large (practically relevant) problems it is impossible.
- The number of atoms in the observable universe is estimated to be roughly 10^{82}. The cardinality of the set of 17×17 binary arrays is larger than 10^{86}.
- To use reinforcement learning methods in domains with large state or action spaces, we must approximate value functions instead of explicitly assigning values to every state-action-pair.

Deep reinforcement learning: Approximation via ANNs

Deep RL took of around 2015 (Deep Q learning by Mnih et al.). Great results have been obtained by using artificial neural networks, fitted by stochastic gradient descent, as function approximators in RL algorithms.

Theorem (Universal approximation (Cybenko, Hornik 1989))

Feedforward artificial neural networks can approximate any compactly-supported continuous function to any desired degree with respect to the max-norm.

Paraphrase:
"For any nice function and any desired precision, there exists a sufficiently large neural network that approximates the function to the desired precision."

Deep Q learning

- We use a neural network $\text{qNet}_w(s, \cdot) \in \mathbb{R}^{|A|}$ to approximate q-values. Here $w \in \mathbb{R}^d$ denotes the parameters of the neural network (weights and biases).
- Temporal difference error:

$$TD = \Big(r + \gamma \cdot \max_{a'} \text{qNet}_w(s', a')\Big) - \text{qNet}_w(s, a)$$

- In principle we would like to use this 'error' as a loss function for gradient descent as in supervised learning.
- 'Fake' a supervised learning problem with feature vector x and target y:

$$\begin{aligned} x &= s \\ y &= r + \gamma \cdot \max_{a'} \text{qNet}_w(s', a') \end{aligned}$$

- Gradient descent:

$$\nabla_w \text{distance}\Big(y, \text{qNet}_w(x, \cdot)\Big)$$

Naive neural Q learning: Problems

- 'The deadly triad':
  - *Bootstrapping*: Estimates for a state-action tuple are used to update estimates of other state-value tuples.
  - If there is also *function approximation*, then, whenever one q-value is changed, others change also, possibly even the value of the state that is bootstrapped upon ('detrimental generalization').
  - This is further exacerbated by *off-policy* learning where the state being bootstrapped upon is usually not reached in the next timestep.
  - Result: Divergence of state-action value estimates.
- Not a true gradient procedure: The target variable y depends on w itself. Circularity!
- Targets change very quickly since target values themselves are updated after every step. ('Moving target')
- Samples are not iid. (The correlation is due to the sequential dependence in the generation of data in RL.)

Solution to the deadly triad

- *Experience replay*: Decorrelate observations by bootstrapping from a replay memory.
  - We store the data that we generate.
  - We draw batches from this memory to enact minibatch gradient descent.
  - Additional benefit: The convergence properties are generally improved by minibatch learning.
- *Use a slow target network*: Update target network periodically or 'softly'. If the targets are more stable, learning works better.
  - If the target network lags behind the policy network, the situation is more like classical ML. (The target is still moving, but it moves slowly.)
- And several other tricks:
  - Use multi-step returns before bootstrapping.
  - Double- (or Triple-) Q-learning.
  - ...

Challenges in applications: Create the 'environment', model the state

Discovering faster matrix multiplication algorithms with reinforcement learning

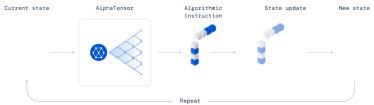
https://doi.org/10.1038/s41586-022-05172-4

Received: 2 October 2021

Accepted: 2 August 2022

Alhussein Fawzi[1,2]✉, Matej Balog[1,2], Aja Huang[1,2], Thomas Hubert[1,2], Bernardino Romera-Paredes[1,2], Mohammadamin Barekatain[1], Alexander Novikov[1], Francisco J. R. Ruiz[1], Julian Schrittwieser[1], Grzegorz Swirszcz[1], David Silver[1], Demis Hassabis[1] & Pushmeet Kohli[1]

We cast the problem of finding efficient matrix multiplication algorithms as a reinforcement learning problem, modelling the environment as a single-player game, TensorGame. The game state after step t is described by a tensor $\mathcal{S}_t$, which is initially set to the target tensor we wish to decompose: $\mathcal{S}_0 = \mathcal{T}_n$. In each step t of the game, the player selects a triplet $(\mathbf{u}^{(t)}, \mathbf{v}^{(t)}, \mathbf{w}^{(t)})$, and the tensor $\mathcal{S}_t$ is updated by subtracting the resulting rank-one tensor: $\mathcal{S}_t \leftarrow \mathcal{S}_{t-1} - \mathbf{u}^{(t)} \otimes \mathbf{v}^{(t)} \otimes \mathbf{w}^{(t)}$. The goal of the player is to reach the zero tensor $\mathcal{S}_t = \mathbf{0}$ by applying the smallest number of moves. When the player reaches the zero tensor, the sequence of selected factors satisfies $\mathcal{T}_n = \sum_{t=1}^{R} \mathbf{u}^{(t)} \otimes \mathbf{v}^{(t)} \otimes \mathbf{w}^{(t)}$ (where R denotes the number of moves), which guarantees the correctness of the resulting matrix multiplication algorithm. To avoid playing unnecessarily long games, we limit the number of steps to a maximum value, R_{limit}.

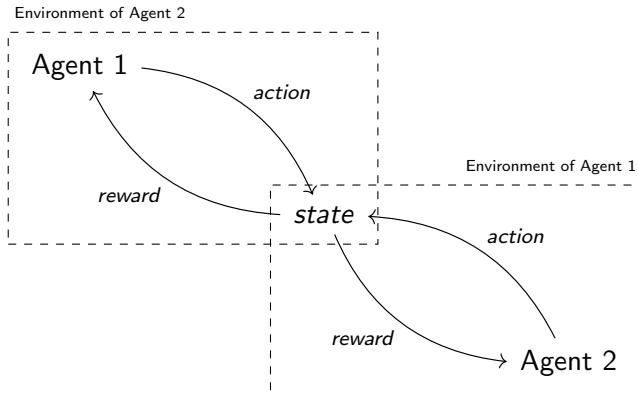


Single-player game played by AlphaTensor, where the goal is to find a correct matrix multiplication algorithm. The state of the game is a cubic array of numbers (shown as grey for 0, blue for 1, and green for -1), representing the remaining work to be done.

Challenges in applications: Natural rewards are insufficient

- Reward sparsity:
  - In many applications rewards are sparse. In Chess the natural reward for a move is always zero unless its a checkmating move.
  - The reward signal may be too weak to use RL efficiently.
- 'Credit assignment problem':
  - The agent has won a chess game after 50 moves? Which moves were crucial?
  - Reward shaping may be necessary: Define surrogate rewards (E.g.: define interim goals). This is conceptually similar to feature engineering.
  - This can be very tricky and even make matters worse.
  - Sometimes the agent finds an unexpected way to obtain the surrogate rewards without obtaining the actual rewards.

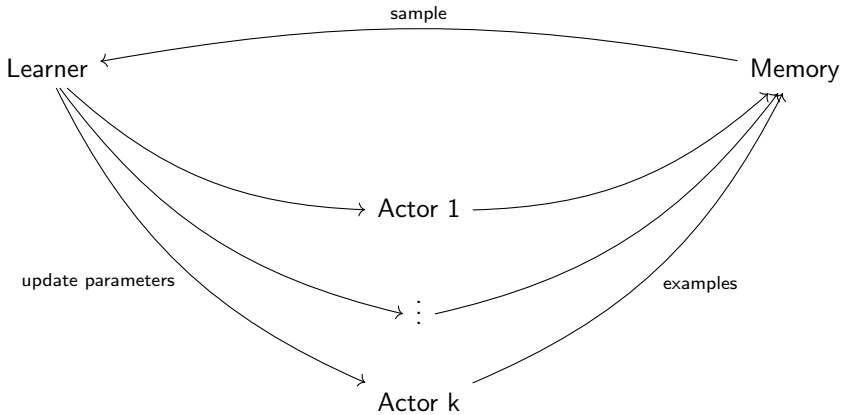
Challenges in applications: Multi-agent settings



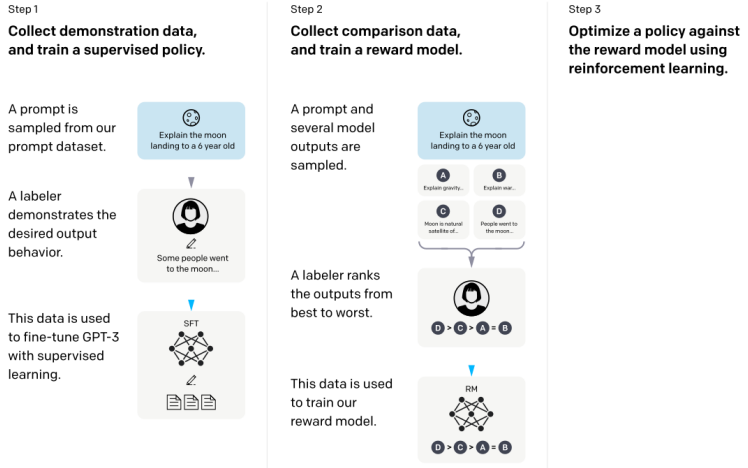
In multi-agent settings (e.g. autonomous vehicles, factory floor), an agent's environment includes other agents. This complex-system structure makes multi-agent learning a delicate issue. Goals of the individual agents and goals for the agent population may not be naturally compatible: Coordination problems.

Challenges in applications: Scaling

- RL is sample-inefficient. A large amount of data is necessary.
- The majority of computation time is usually not spend with deep neural network training, but with the data-generation in the simulation environment.
- This must usually be somehow parallelized.



Application: RL to tune a LLM



The instruct-gpt method (Ouyang et al. 2022) forms the core of ChatGPT. A reward function is constructed from human labeling to be used in an RL-based tuning of an LLM.

Contact

Sharwin Rezagholi
sharwin.rezagholi[at]technikum-wien.at

University of Applied Sciences Technikum Wien
Department Computer Science
https://www.technikum-wien.at/departments/computer-science/