A LOOK AT REINFORCEMENT LEARNING

by

Rahil Sharma

Contents

1. Reinforcement Learning

Reinforcement Learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. Since the learner does not have any information of what actions to take so it discovers the actions which yields the most reward by trying them. The two most important characteristics of reinforcement learning are—trail-and-error search and delayed reward **[SB20]**.

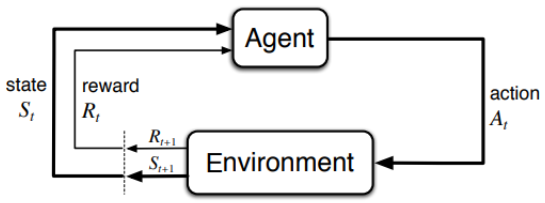


FIGURE 1. A typical RL cycle

1.1. Elements of Reinforcement Learning. — Beyond the agent and the environment, the reinforcement learning system has four main sub-elements: a policy, a value function, a reward signal and a model of the environment.

1. A policy defines the learning agent's way of behaving at a given moment.
2. A reward signal defines the goal of the RL problem.
3. A value function specifies what is beneficial in the long run The value of a state is roughly equal to the total amount of reward an agent can expect to accrue in the future, starting from that state.
4. A model of the environment is something that mimics the environment's behaviour. The model might, for example, predict the next state and reward based on a given state and action. Models are used for planning, which is defined as any method of deciding on a course of action by examining potential future situations before they are actually experienced.

2. Markov Decision Processes

2.1. The Agent-Environment Relationship. — Before diving into the depth of relationship let's first look at the definitions of agent, environment and state:

1. **Agent**: *Software programs that make intelligent decisions and they are the learners in RL. These agents interact with the environment by actions and receive rewards based on there actions.*
2. **Environment**: *It is the demonstration of the problem to be solved.Now, we can have a real-world environment or a simulated environment with which our agent will interact.*
3. **State**: *This is the position of the agents at a specific time-step in the environment. So, whenever an agent performs a action the environment gives the agent reward and a new state where the agent reached by performing the action.*

The agent is considered to be part of the environment if the agent cannot change arbitrarily. So, any decision we want the agent to learn are actions and state can be anything which can be useful in choosing actions.

2.2. The Markov Property. — The Markov Property defines that *future is dependent of the past given the present.* **[Scib]**
Mathematically this statement is:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, ..., S_t]$$

Here S_t is the current state of the agent and S_{t+1} is the next state. So, in simple words the equation means that transition from S_t to S_{t+1} is independent of the past.

2.3. State Transition Property. — For Markov State from S_t to S_{t+1} is given by:

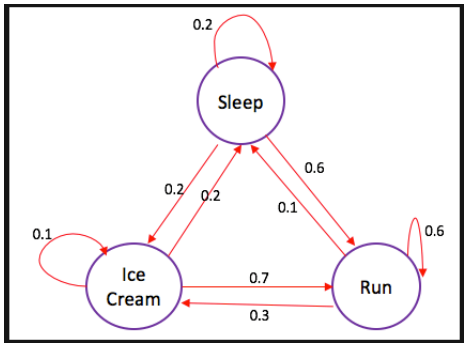
$$P_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s]$$

State Transition Probability can be formulated into a State Transition Probability Matrix by:

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & \cdots & p_{1n} \\ p_{21} & p_{22} & p_{23} & \cdots & p_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{1n} & p_{2n} & p_{3n} & \cdots & p_{nn} \end{bmatrix}$$

In P matrix sum of each row is equal to 1 and each row represents the transition from original/starting state to successor state.

2.4. Markov Process. — Markov Process also known as Markov Chains is the memory less random process. It is a sequence of random states $S_1, S_2, ..., S_n$ with a Markov Property. It is defined using the States S and Transition Probability Matrix P. Let's look at the following example to understand the meaning of Random Process:



Here in this example the edges of the tree are transitions probabilities. Suppose that we are sleeping and there is a 0.6 chance that we will run and 0.2 chance we sleep more and again 0.2 that we will eat ice-cream. Similarly we can see more sequences. Two samples from the chain.

1. Sleep — Ice-cream — Ice-cream — Run
2. Sleep — Run — Ice-cream — Sleep

Here Sleep, Ice-cream, Run represents random states. So, Every time we run a chain we get random set of states S.

2.5. Markov Reward Process. — The following are the concepts that will help us understand Markov Reward Process (MRP):

1. **Rewards**: *Rewards are the numerical values that the agent receives on performing some action at some state(s) in the environment. The numerical value can be positive or negative based on the actions of the agent.*
2. **Returns**: *Mathematically, the returns is defined as*

$$G_t = r_{t+1} + r_{t+2} + ... + r_T$$

r_{t+1} is the reward received by the agent at time step $t[0]$ while performing an action to move from one state to another. Similarly, r_{t+2} is the reward received by the agent at time step t_1 by performing an action to move to another state. And, r_T is the reward received by the agent by at the final time step by performing an action to move to another state.

3. **Episodic Tasks**: *These are the tasks that have a terminal state (end state). We can say they have finite states. For example, in racing games, we start the game (start the race) and play it until the game is over (race ends!). This is called an episode. Once we restart the game it will start from an initial state and hence, every episode is independent.*
4. **Continuous Tasks**: *These are the tasks that have no ends i.e. they don't have any terminal state. These types of tasks will never end.*
5. **Discount Factor**: *It determines how much importance is to be given to the immediate reward and future rewards. This basically helps us to avoid infinity as a reward in continuous tasks. It has a value between 0 and 1. A value of 0 means that more importance is given to the immediate reward and a value of 1 means that more importance is given to future rewards. In practice, a discount factor of*

0 will never learn as it only considers immediate reward and a discount factor of 1 will go on for future rewards which may lead to infinity. Therefore, the optimal value for the discount factor lies between 0.2 to 0.8.
So, discount factor we can define the returns as follows.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

MRPs are the Markov chains with values judgement. Basically, we get a value from every state our agent is in. Mathematically, Markov Reward Process can be written as:

$$R_s = \mathbb{E}[R_{t+1}|S_t]$$

This equation expresses how much reward R_s we receive from a particular state S_t.

2.6. Markov Decision Process. — It's a decision-making Markov Reward Process. Everything is the same as it was with MRP, but now we have an actual agency that makes choices and takes actions.
Transition Probability Matrix with respect to Action:

$$P^a_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a]$$

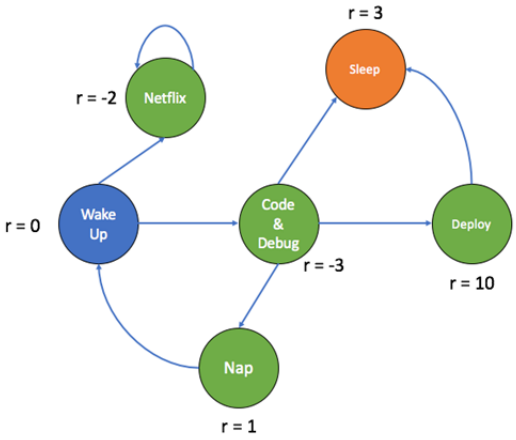
Reward Function with respect to Action:

$$R^a_s = \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$$

2.7. Q-function or State-action value function. — . We define State-action Value function as:

$$q_\pi(s, a) = \mathbb{E}_\pi[S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$$

The equation describes the value of performing a certain action a in a state s with a policy π. Below is an example of MDP:



There are no more probability, as we can see. In fact, our agent now has options, such as watching Netflix or coding and debugging after waking up. Of course, the agent's behaviours are defined in terms of policy, and it will be rewarded accordingly.

3. Bellman Equation and Optimality

3.1. Bellman Expectation Equation. — From the State Value function, we know that the value of a state can be decomposed into a immediate reward $R[t+1]$ and the value of successor state $v[S(t+1)]$ with a discount factor γ. **[Scic]**
Mathematically we define the Bellman Expectation Equation as:

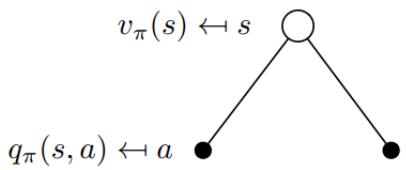
$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s]$$

The meaning of the above equation is that the value of a state is determined by the immediate reward plus the value of successor states when we are following a certain policy π.

Similarly we can define the State Action Value Function (Q-Function) as:

$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a]$$

Now let's understand Bellman Expectation Function for a State Value Function with the following backup diagram.

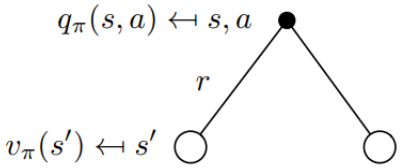


This figure describes the value of being in a certain state.

Mathematically we express this as follows:

$$v_\pi(s) = \sum_{a \in A} \pi(a|s) q_\pi(s,a)$$

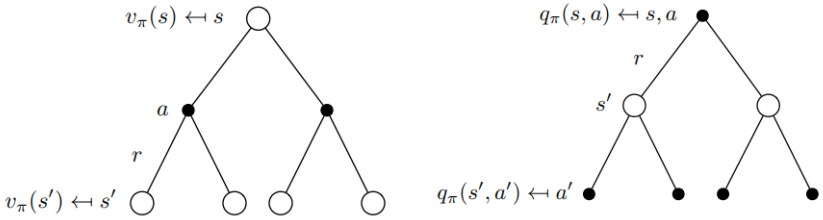
Now let's see backup diagram for State-Action value function.



Now using this we have a equation defining how good it is to take a particular action a in state s:

$$q_\pi(s,a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s')$$

Now, let's stitch these backup diagrams together to define State-Value Function and State-action value function.



State-Value Function from the left Backup Diagram is:

$$v_\pi(s) = \sum_{a \in A} \pi(a|s)(R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_\pi(s'))$$

State-Action Value Function from the right Backup Diagram is:

$$q_\pi(s,a) = R_a^s + \gamma \sum_{s' \in S} P_{ss'}^a \sum_{a' \in A} \pi(a'|s') q_\pi(s',a')$$

So, this is how we can formulate Bellman Expectation Equation for a given MDP to find it's State-Value Function and State-Action Value Function.

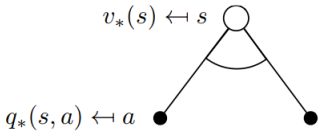
3.2. Optimal Value Function. — *The optimal Value function is one which yields maximum value compared to all other value function.*

1. **Optimal State-Value function**: *It is the maximum value function over all policies.*
$$v_*(s) = \max_\pi v_\pi(s)$$
2. **Optimal State-Action Value function**: *It is the maximum action-value function over all policies.*
$$q_*(s,a) = \max_\pi q_\pi(s,a)$$

3.3. Optimal Policy. — *Optimal Policy is one which results in optimal value function.*
It's worth noting that in an MDP, there could be multiple optimal policies. All optimum policies attain the same optimal value function and optimal state-action value function (Q-function).
By maximizing over $q_*(s,a)$ we can find optimal policy. Mathematically, it can be expressed as:

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in A} q_*(s,a) \\ 0 & \text{otherwise} \end{cases}$$

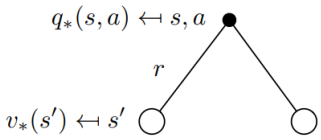
3.4. Bellman Optimality Equation. — The Bellman Optimality Equation is similar to the Bellman Expectation Equation, with the exception that instead of taking the average of the actions our agent is capable of, we take the action with the max value. Let's see the backup diagram of Bellman Optimality Equation for State-value function



The Bellman Optimality Equation for State-value function is expressed as:

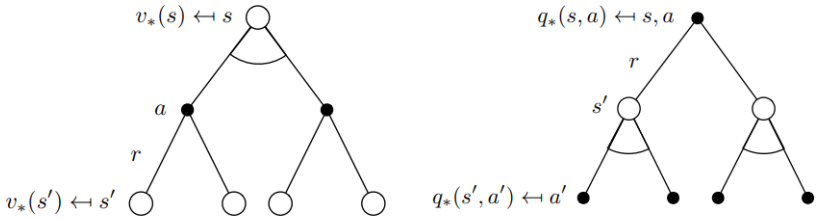
$$v_*(s) = \max_a q_*(s, a)$$

Similarly, The Bellman Optimality backup diagram and Equation for State-Action Value Function (Q-function) are as follows:



$$q_*(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

Now, stitch these backup diagrams for State-value function and State-action value function. Then we have:



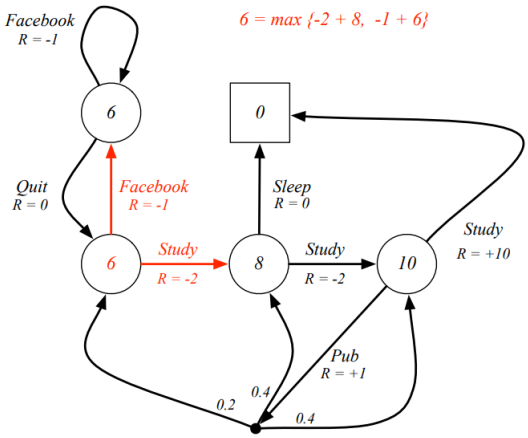
Bellman Optimality Equation for State-Value Function from the left Backup Diagram is:

$$v_*(s) = \max_a R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a v_*(s')$$

Bellman Optimality Equation for State-Action Value Function from the right Backup Diagram is:

$$q_*(s, a) = R_s^a + \gamma \sum_{s' \in S} P_{ss'}^a \max_{a'} q_*(s', a')$$

Below is the example of Bellman Optimality Equation.



Look at the red arrows and assume we want to get the value of state 6 (in red). As you can see, if our agent chooses Facebook, we get a -1 reward, and if our agent chooses to

study, we get a -2 reward. We'll apply the Bellman Optimality Equation for State-Value Function (Since we have the optimal value for other two states we'll take an average and maximise for both the action (choose the one that gives maximum value)) to find the value of state in red. So, as we can see from the diagram, going to Facebook yields a value of 5 for our red state, while going to study yields a value of 6, and we then maximum over the two, yielding 6 as the answer.

4. Iterative Algorithms

4.1. Value Iteration. — After learning the values for all states, we can act based on the gradient. The value of the states is directly learned from the Bellman Update by value iteration. Under certain non-restrictive conditions, the Bellman Update is assured to converge to optimal values. [**Scia**]
Mathematically, it is defined as:

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$$

4.2. Policy Iteration. — Policy learning incrementally looks at the current values and extracts a policy. There two steps in Policy Iteration.

1. **Policy Extraction**: In this we go from value to a policy by using the policy that maximizes over expected values.

$$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi_i}(s')]$$

2. **Policy Evaluation**: The policy evaluation process starts with a policy and then executes value iteration based on a policy.

$$V^{\pi_i}_{k+1}(s) \leftarrow \sum_{s'} T(s,\pi_i(s),s')[R(s,\pi_i(s),s') + \gamma V^{\pi_i}_k(s')]$$

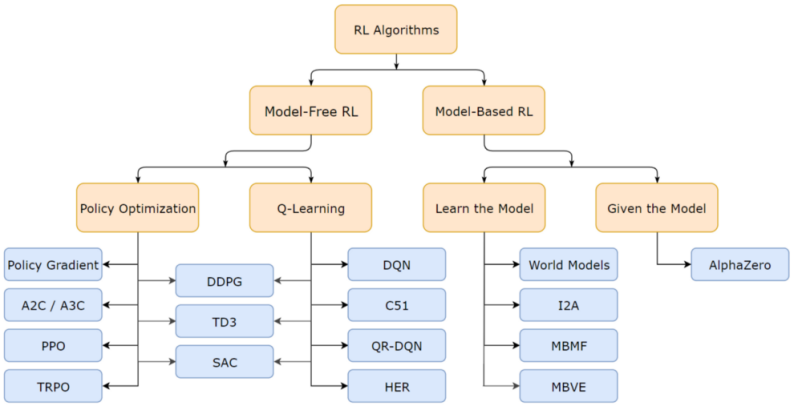
Because of the underlying Bellman Update, policy iteration, like value iteration, is guaranteed to converge for most reasonable MDPs.

4.3. Q-Value Iteration. — The challenge with knowing optimal values is that formulating a policy from them might be difficult. Because the "argmax" operator is nonlinear and difficult to optimise, Q-value Iteration is a step closer to direct policy extraction. The optimal policy at each state is simply the state's maximum q-value. [**AIb**]

$$Q'(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q(s',a')]$$

5. Learning Approaches

Below is the Reinforcement Learning taxonomy as defined by OpenAI. [**AIa**]

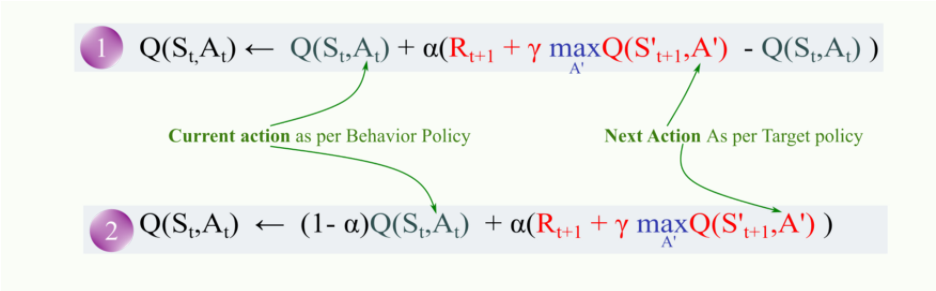


5.1. Model-Free RL. — *Model-free RL, on the other hand, uses experience to learn directly one or both of two simpler quantities (state/ action values or policies) which can achieve the same optimal behavior but without estimation or use of a world model. Given a policy, a state has a value, defined in terms of the future utility that is expected to accrue starting from that state.*[**Roy**]

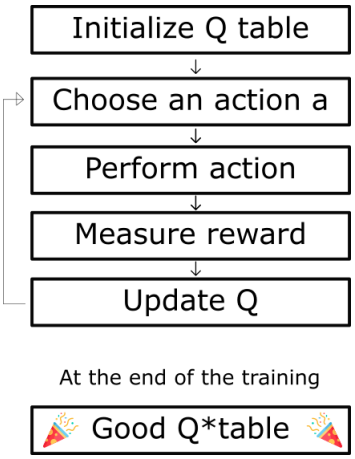
The two main methods to represent agents in Model-Free RL are:

1. **Policy Optimization**: In policy optimization methods the agent learns directly the policy function that maps state to action. The policy is determined without using a value function. Important to mention that there are two types of policies: deterministic and stochastic. Deterministic policy maps state to action without uncertainty. It happens when you have a deterministic environment like a chess table. Stochastic policy outputs a probability distribution over actions in a given state. This process is called Partially Observable Markov Decision Process (POMDP).

2. **Q-Learning**: It is an off policy algorithm. In it we evaluate target policy π while following behaviour policy μ. Consider the diagram below for Q-learning Algorithm written in two ways:



The chart below describes a representation of the algorithm.



5.2. Model-Based RL. — *Model-based RL uses experience to construct an internal model of the transitions and immediate outcomes in the environment. Appropriate actions are then chosen by searching or planning in this world model.*

Learn the model: Below steps describes the procedure:

1. run base policy $\pi_0(\mathbf{a}_t|\mathbf{s}_t)$ (e.g., random policy) to collect $\mathcal{D} = \{(\mathbf{s}, \mathbf{a}, \mathbf{s}')_i\}$
2. learn dynamics model $f(\mathbf{s}, \mathbf{a})$ to minimize $\sum_i \|f(\mathbf{s}_i, \mathbf{a}_i) - \mathbf{s}'_i\|^2$
3. plan through $f(\mathbf{s}, \mathbf{a})$ to choose actions

For the control function, supervised learning is used to train a model to minimise the least square error using sampled data. In step three, the model and a cost function are used to create an optimal trajectory. The cost function may calculate the distance between us and the target place, as well as the amount of effort spent.

5.3. Temporal Difference (TD) Learning. — TD Learning is a blend of the Dynamic Programming (DP) method and Monte Carlo (MC) method. The following are the key characteristics of a TD Learning:

1. There is no model (the agent does not know state MDP transitions)
2. The agent learns from sampled experience (Similar to MC)
3. Like DP, TD methods update estimates based in part on other learned estimates, without waiting for a final outcome (they bootstrap like DP).
4. It can learn from incomplete episode thus this method can be used in continuous problems as well
5. TD updates a guess towards a guess and revise the guess based on real experience

Consider the following analogy: Monte Carlo learning is similar to an annual examination in which students complete their episode at the end of the year. TD learning, on the other hand, is similar to a weekly or monthly examination (students can alter their performance based on this score (reward received) after each little interval, and the ultimate score is the sum of all weekly examinations (total rewards)).

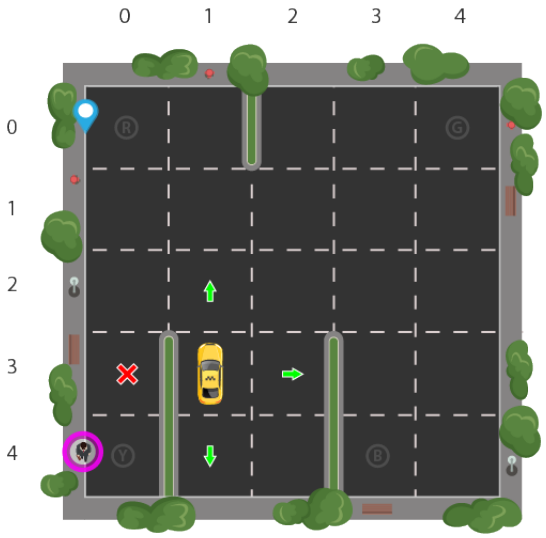
6. Reinforcement Q-Learning with OpenAI Gym

Here we will learn how to teach a taxi to pick up and drop off the passengers with RL. [**Kan**]

6.1. Rewards. — Our agent is imaginary driver and it will learn how to control the taxi by trial experiences in the environment. The following are the points to consider:

1. For a successful drop off the agent should receive high reward.
2. If the passenger is dropped off in wrong direction then the agent would be penalized.
3. After each time-step, the agent should receive a slight negative reward for failing to reach the destination. "Slight" negative because we would prefer our agent to arrive late rather than make wrong decisions in order to get to the location as soon as possible.

6.2. State Space. — The state space is the all the possible situations our cab could face.



The parking lot is 5×5 grid which gives a 25 possible cab locations. Our state space includes these 25 places. Our taxi's present location is coordinated, as you can see (3,

1).There are 4 locations where the cab can pick up and drop off: R, G, Y, B or [(0,0), (0,4), (4,0), (4,3)] in (row, col) coordinates. The passenger at Y wish to go to R.
We can consider all combinations of passenger locations and destination locations to come up with a total number of states for our taxi environment; there are four (4) destinations and five (4 + 1) passenger locations when we account for one (1) additional passenger state of being inside the taxi.
So, total possible states $5 \times 5 \times 5 \times 4 = 500$ in the environment of the Taxi.

6.3. Action Space. — The agent would encounter one of the 500 states. The six possible actions are:

1. North
2. South
3. East
4. West
5. pickup
6. drop-off

As you can see in the example above, the cab is unable to do some operations in certain situations due to the presence of walls. We will just give a -1 penalty for every wall hit in the environment's code, and the taxi will not move. This will just result in more penalties, prompting the cab to consider getting over the blockade.

6.4. Implementation in Python. — In our code we will use OpenAI Gym environemnt called "Taxi-V3". Open the attached code in Google Collab to see the implementation and run the code for the output. Notice that first it runs the program without RL and then it runs program with Q-Learning.

6.5. Comparing Q-learning to No Reinforcement Learning. — The agent is evaluated in the following metrics:

1. **Average rewards per move**: The higher the reward, the better the agent's performance. As a result, selecting on rewards is an important aspect of Reinforcement Learning. Because both time-steps and penalties are negatively rewarded in our example, a larger average reward would imply that the agent arrives at the target as quickly as possible while incurring the fewest penalties.
2. **Average number of penalties per episode**: The lower the number, the better our agent's performance. We'd like this metric to be 0 or very close to zero.
3. **Average number of time steps per trip**: We want a small number of time-steps per episode since we want our agent to take minimum steps(i.e. the shortest path) to reach the destination.

Measure	Random agent's performance	Q-learning agent's performance
Average rewards per move	-3.9012092102214075	0.6962843295638126
Average number of penalties per episode	920.45	0.0
Average number of timesteps per trip	2848.14	12.38

In our experiment more than 100 episodes were computed and the results shows that Q-Learning is the clear winner.

7. Practical Applications

Reinforcement Learning is most applicable where there is a lot of data. [**Mwi**]

1. **Gaming**: RL is extensively used in building the Artificial Intelligence (AI) models for playing the video games. For example, Using reinforcement learning, AlphaGo Zero was able to learn the game of Go from scratch. It learned by playing against itself. After 40 days of self-training, Alpha Go Zero was able to outperform the version of Alpha Go known as Master that has defeated world number one Ke Jie.
2. **Autonomous Driving**: Autonomous driving tasks where RL is applied: *Trajectory Optimization, Dynamic Pathing, Motion Planning, Controller Optimization, and scenario based learning policies for highways.*
3. **Trading and Finance**: Supervised time-series models can be used for forecasting the stock prices and future sales. However, these models lack the ability to take action at a particular stock price. Here, RL agent can decide what action to take; whether to hold, buy or sell the stock. For example, IBM has a sophisticated reinforcement learning based platform that has the ability to make financial trades. It computes the reward function based on the loss or profit of every financial transaction.
4. **Natural Language Processing**: To mention a few RL is used in text summarization, question answering, and machine translation. Take for example, authors from the University of Colorado and the University of Maryland, propose a reinforcement learning based approach to simultaneous machine translation. The interesting thing about this work is that it has the ability to learn when to trust the predicted words and uses RL to determine when to wait for more input.

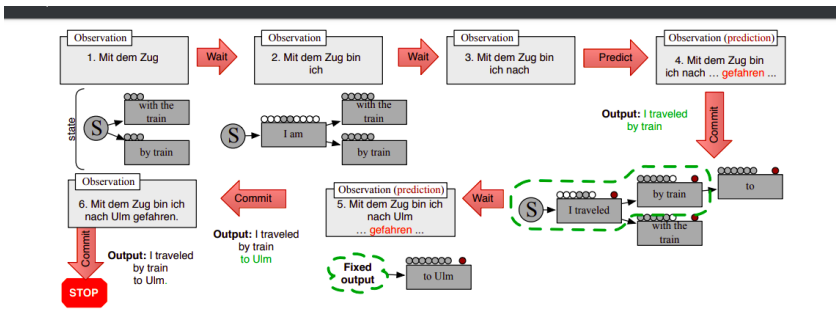


Figure 2: A simultaneous translation from source (German) to target (English). The agent chooses to wait until after (3). At this point, it is sufficiently confident to predict the final verb of the sentence (4). Given this additional information, it can now begin translating the sentence into English, constraining future translations (5). As the rest of the sentence is revealed, the system can translate the remainder of the sentence.

5. **Robotics Manipulation**: In Robots, RL can give the ability to grasp various objects including unseen objects during training, For example, Google AI applied this approach to robotics grasping where 7 real-world robots ran for 800 robot hours in a 4-month period. In this experiment, the QT-Opt approach succeeds in 96

References

[AIa] O. AI – "Reinforcement learning taxonomy".

[AIb] S. L. AI – "Reinforcement learning algorithms — an intuitive overview".

[Kan] S. Kansal – "Reinforcement q-learning from scratch in python".

[Mwi] D. Mwiti – "10 real-life applications of reinforcement learning".

[Roy] B. Roy – "Temporal-difference (td) learning".

[SB20] R. S. Sutton & A. G. Barto – *Reinforcement learning: An introduction*, The MIT Press, 2020.

[Scia] T.-D. Science – "Fundamental iterative methods of reinforcement learning".

[Scib] T. D. Science – "Reinforcement learning : Markov-decision process (part 1)".

[Scic] T. Science – "Reinforcement learning: Bellman equation and optimality (part 2)".

March 24, 2022

RS, Simon Fraser University