



ABSOLUTDATA

An Infogain Company

BRAIN WAVE

DATA SCIENCE DIGEST

— 6TH EDITION —



Abhimanyu Saraf

Subject Matter Expert (Data Sciences), Products & Innovation Labs, Absolutdata

**"If intelligence was a cake,
unsupervised learning would be the cake,
supervised learning would be the icing on the cake, and
reinforcement learning would be the cherry on the cake."**

– Yann LeCun, VP & Chief AI Scientist at Facebook

Warm greetings from the Data Sciences team!

As most of you know, Reinforcement Learning (RL) is a subfield of machine learning that teaches an agent how to choose an action from its action space within a particular environment, in order to maximize the cumulative rewards (or minimize cumulative loss) over time. In layman terms, RL enables an algorithm to make decisions through repeated trial-and-error actions within a given environment and learn for ensuring appropriate decisioning further. Reinforcement Learning is not a novel phenomenon in the realm of data sciences; in fact, it has evolved over seven decades of academic rigor.

The missing link in evolution

The erstwhile approaches of Machine Learning were designed for the cases of supervised learning specifically, wherein the overall algorithm remains more straightforward. But in RL, there are more components of the algorithm that could be the potential targets for design automation (e.g., neural network architectures for agent networks, strategies for sampling from the replay buffer, overall formulation of the loss function etc.), and the best model update procedure (for integrating these components) is not always obvious to many. Prior efforts for the automation of RL algorithm have focused primarily on model-update rules.

These approaches tend to learn the efficiently optimize the RL update procedure by itself, with the aid of gradient-based methods. However, these learned rules are not interpretable or generalizable, because the learned weights are opaque and specific to domains.

The second wind

Barring the applications in the behavioral sciences, RL has been in practice since the 80's at least. The academic discourse for Reinforcement Learning pursued three concurrent 'threads' of research (trial and error, optimal control, and temporal difference), before being united in the research in the 1990's.

However, it has gained prominence due to the very reason that it by-passes the need to have training data upfront. Consider the A/B test, for example. An RL practitioner would say “Why wait for training data? Let’s just start showing options (probably more than just A and B) to customers and continue to show the ones that get the best results.”

To the nines and back...

RL straddles the two-fold goal of exploration and exploitation: the classic optimization problem. It has led to various, often stochastic, amendments to the original method without a convincing general solution that works well in most applications. If we allow the agent to settle in too quickly on what appears to be the winning solution, it may never explore all the other options (the full extent of the State space). This may result in overfitting or an early hang up on a local optimum.

Cleverer RL algorithm or more data?

In recent years, AutoML has won our confidence in automating the design of machine learning components, such as neural networks architectures and model-update rules. One example is Neural Architecture Search (NAS), which has been used to develop better neural network architectures for image classification and efficient architectures for running on phones and hardware accelerators. In addition to NAS, AutoML-Zero shows that it’s even possible to learn the entire algorithm from scratch using basic mathematical operations.

We represent the loss function, which is used to optimize an agent’s parameters over its experience, as a computational graph, and use Regularized Evolution to evolve a population of the computational graphs over a set of simple training environments. This results in increasingly better RL algorithms, and the discovered algorithms generalize to more complex environments, even those with visual observations.

RL in key with challenges

We have come across numerous use cases, wherein the recent developments in RL helped us to overcome either the limitations of its own characteristics or the extant challenges of other approaches. RL for Cold-Start Problem in Recommendation Systems, Reinforcement Learning with Augmented Data, Distributed Reinforcement Learning, Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value, Visualization of Weaknesses of Deep Reinforcement Learning Agents etc. have been significant add to the possibilities.

Two cents worth

The future of Reinforcement Learning is both exciting and fascinating, as the recent research efforts aim to improve the interpretability, accountability and trustworthiness of RL algorithms.

- Businesses should seek to implement Reinforcement Learning into their AI strategies, when they face one or more of the following scenarios:
- Data for learning currently does not exist, and they don’t want to wait to accumulate it (because delay might be costly)
- The data is changing rapidly, causing the outcome to change more quickly, which a typical model refresh cycle cannot accommodate
- There is a large state space
- Business is using simulations, because the system or process is too complex for teaching machines through trial and error
- Business is seeking to augment human analysis, by optimizing operational efficiency and providing decision support.

Hope this edition of BrainWave gives you a lateral perspective and drives you to inculcate this paradigm in your areas of expertise too!

References



[Reinforcement Learning – A Review of the Historic, Modern, and Future Applications of this Special Form of Machine Learning](#)



[Artificial Development by Reinforcement Learning Can Benefit from Multiple Motivations](#)



[Google AI Blog – Evolving Reinforcement Learning Algorithms](#)

Contents

1	Statisticionary Mathematical Formulation of Optimal Policy	Page 05
2	Coder's Cauldron Reinforcement Learning – An Example Using the OpenAI Cart Pole Problem	Page 08
3	Vivid Visualization Reinforcement Learning	Page 13
4	Thriving Traction Stochastic Ensemble Value Expansion (STEVE) Model for Low Sample Datasets	Page 16
5	Folk Wisdom's Fallacy Busting Some Common Reinforcement Learning Myths	Page 19
6	Experience Extended Decision Option Generator – A Meta Learning Approach in NAVIK	Page 21
7	Food for Thought Experiment Boost Productivity Through Multi Agent Reinforcement Learning	Page 23
8	Data Science Competitions/Seminars/Fora/Courses	Page 25

Mathematical Formulation of Optimal Policy

Statistictionary

Reinforcement Learning:

Reinforcement learning is another branch of machine learning after supervised and unsupervised learning. It is used to make sequential decisions while training machine learning models. As per the following flow diagram, In an uncertain and complex environment, an agent focuses on maximizing the reward and learn to achieve goal. It is basically controlling the machine to do what a programmer wants by either rewarding or penalizing the actions it performs.^[1]

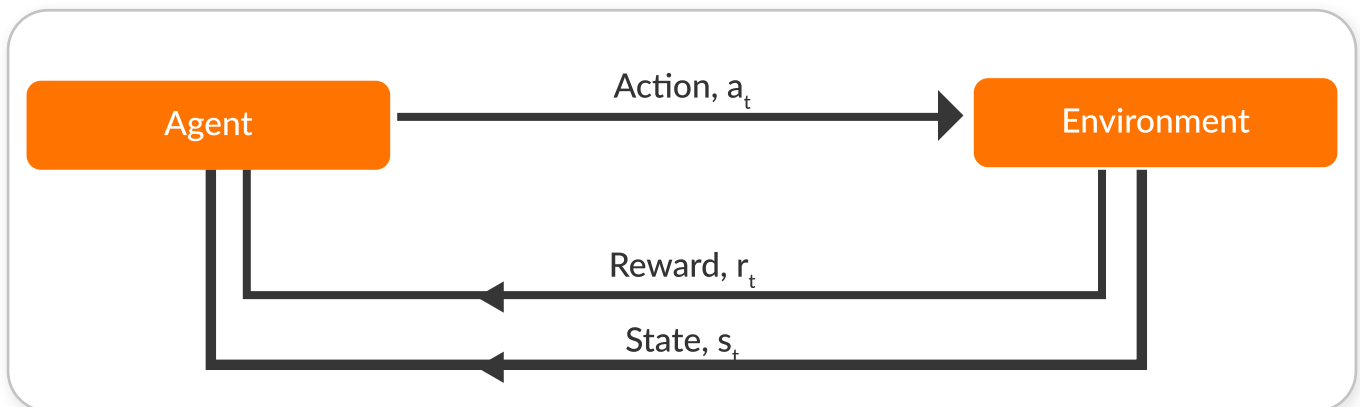


Figure 1: Block Diagram

A reinforcement learning problem consist of various technical components including states, actions, transitions, rewards, policies, and values where:

- **States** are visitable positions.
- **Actions** are possible movements.
- **Transitions** specify connections between states via actions. Thus, knowing the transitions means knowing the map.
- **Reward** specifies the evaluation of that action.
- A **policy** specifies an action the agent takes at each state. Thus, a policy also specifies a **trajectory**, which is a series of states and actions that the agent takes from a start state to an end state.

Optimal Policy Formulation ^[2]:

Understanding of reinforcement learning requires exploration of fundamental mathematical backing. The following formulas are being used and manipulated as per the environment and goal. Under a discrete time Markov decision process (MDP)^[3] Lets consider the problem of controlling a computer agent at each discrete time-step t , the agent observes a state $s_t \in S$, selects an action $a_t \in A$, makes a transition $s_{t+1} \in S$, and receives an immediate reward. Where:

- S is finite set of states
- A is finite set of actions
- $R : S \times A \rightarrow R$ is a reward function
- $r(s_t, a_t, s_{t+1})$ is called the immediate reward function

Basically, each possible next state s_{t+1} is determined with probability $p(s_{t+1}|s_t, a_t)$, and action a_t is chosen using the policy $\pi(a_t | s_t)$. Along a trajectory (h) , we sum over all such possible next states.^[4] As mentioned, The goal is to maximize the expected return by learning the optimal policy which is denoted as π^* :

$$\pi^* = \operatorname{argmax}_{\pi} E_{p^{\pi}(h)} [R(h)]$$

Where:

- argmax gives max of a function and $E_{p^{\pi}(h)}$ denotes expectation over trajectory h drawn from $p^{\pi}(h)$.
- $R(h)$ is discounted sum of immediate rewards along the trajectory (h) and is called the return. This is the reward or penalty that agents get by performing an action. It is discounted as per the environment so that agent reaches goal by maximising the reward. This also helps agent become more farsighted where it explores all possible combinations.
- The policy (π) defines the agent behaviour and maps the state of environment to actions. It is a conditional probability density of action a at state s .
- $p^{\pi}(h)$: denotes the probability density of observed trajectory h under policy π .

References



1. [Deepsense.ai "What Is Reinforcement Learning? The Complete Guide"](#)



2. [Suguyama, Masashi. Statistical Reinforcement Learning: Modern Machine Learning Approaches](#)



3. [Wikipedia.org "Markov Decision Process"](#)



4. [Reinforcement Learning -Chapter 18 - IITK](#)

Authored by

[Rohan Garg](#),

Data Scientist at Absolutdata

Reinforcement Learning – An Example Using the OpenAI Cart Pole Problem

Coder's Cauldron

Introduction

There are many real-world problems that do not fall in the traditional problem-solving approaches of supervised and unsupervised learning in machine learning. Complex, uncertain, and real/simulated environments require a methodology where the model being trained can make decisions by itself to achieve an overall objective.^[1] For example, think of building a model to drive an automated car, play Chess or Go, control a prosthetic foot, make a robot walk an obstacle course, or plan inventory/investments for a business.

Here, the agent must learn by trial and error in a gamified scenario with built-in rewards and penalties. The problem designer sets the rewards but gives no hint to the model on how to progress. The RL model starts with random trials and iteratively gets better until it achieves dexterity in the stated task. Since this process can be parallelized, training time can be shortened exponentially by exploiting modern computing hardware.

OpenAI Gym

Since RL models are trained to solve specific problems, small variations in problem definition (like rewards and penalties) can greatly alter the difficulty. Thus, it is important to have a standardized, open-sourced collection of diverse environments. OpenAI provides a good collection of standardized problems (environments) with its Gym toolkit.^[2] Here one can learn, develop and benchmark RL models. It's simple to install:

```
pip install gym
```

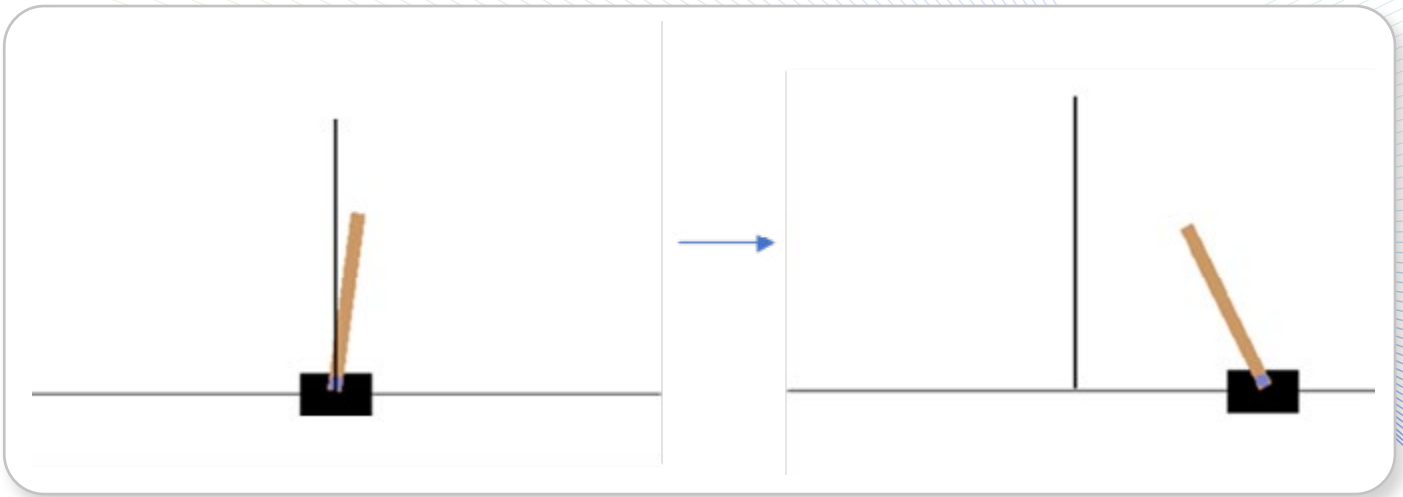
or

```
conda install gym
```

Cart Pole Problem

Let's look at one of the classic problems available in the Gym: CartPole-v0. This simulates a vertical pole attached to a cart with a movable hinged joint – basically an inverted pendulum. All of us might have tried balancing a pole in our hands at some point! It's a trivial task for humans, but not for machines.

In the stated problem, the cart can move sideways in both directions. The objective is to maintain this inverted pendulum configuration; a reward of +1 is awarded for every time interval the pole is upright. The model can apply a force to the cart in either direction. The penalty is an end to the game if the pole is more than 15 degrees off from vertical or the cart is more than 2.4 units from the starting position. The problem is considered solved if a reward of more than 195 is achieved over 100 consecutive trials. The model must learn to keep the pole upright.^[3,6]

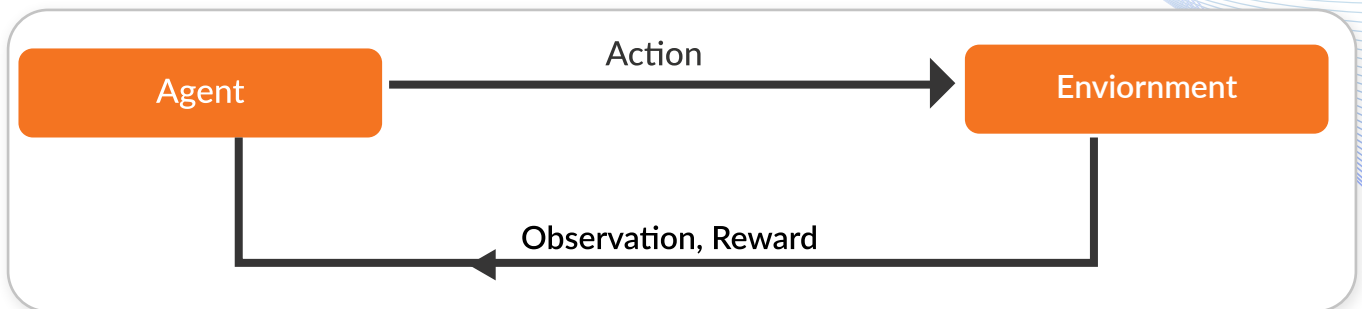


Random Solution

A basic implementation of this environment can be done in the following few lines of code:

```
import gym #import library which contains the problem environment
envi = gym.make('CartPole-v0') #initialize environment
for episode in range(100): #run 100 episodes
    obs = envi.reset() #start an episode, returns initial observation
    for i in range(100): #maximum actions
        envi.render() #display state of environment visually
        print(obs) #print the object containing state of the environment
        action = envi.action_space.sample() # randomly choose an action to be done from action_space
        obs, reward, done, info = envi.step(action) #evaluate state of the environment
        if done: #check if environment has terminated (due to achieving penalty state)
            print("Episode finished after {} timesteps".format(t+1)) #print timesteps elapsed
            break
    envi.close() #close environment
```

The above piece of code tries to keep the pole upright with random actions. The cart moves randomly along the horizontal axis until either 100 actions are completed or the terminating conditions are met. The action_space is either 0 or 1 (force to the left or right). Clearly, it is impossible to find a solution with random actions:



The complete cycle can be represented with the above diagram. Here the Environment is the pole and cart, the Agent is our model (random), the action is the application of force to the left or the right, and the observation is the state of the environment (indicating the angle of the pole and the cart's distance from the initial position). The reward is +1 for every timestep the pole is upright.

Proximal Policy Optimization Approach

Let's use a reinforced learning algorithm to introduce some intelligence in the system. We will implement a Proximal Policy Optimization (PPO) approach. The PPO algorithm is one of the most effective and popular RL algorithms. It was launched by OpenAI in 2017.

1. PPO has two components: The Actor model and the Critic model. ^[4,5]
 - a. The Actor learns what action to take given an environment state.
 - b. The Critic model monitors the reward obtained once the action is applied to the environment. It learns if the action of the Actor resulted in a positive reward or a negative one and gives a score (rating) to the Actor.
 - c. The Actor then decides how to update its current policy (actions) based on the rating received from the Critic model.
2. PPO runs the whole loop with the two models for a finite number of steps. It observes this small set of observations after interacting with the environment and uses this batch to update its decision-making policy (action). This old batch is then discarded, and a new batch is created with the updated policy. This is an 'on-policy learning' approach. ^[8]
3. The updating of the policy from the previous version is moderated, ensuring the training variance is limited. This avoids the agent going down a bad policy path and getting stuck there. To achieve this, we cap the ratio of new and old policy updates to a threshold value.

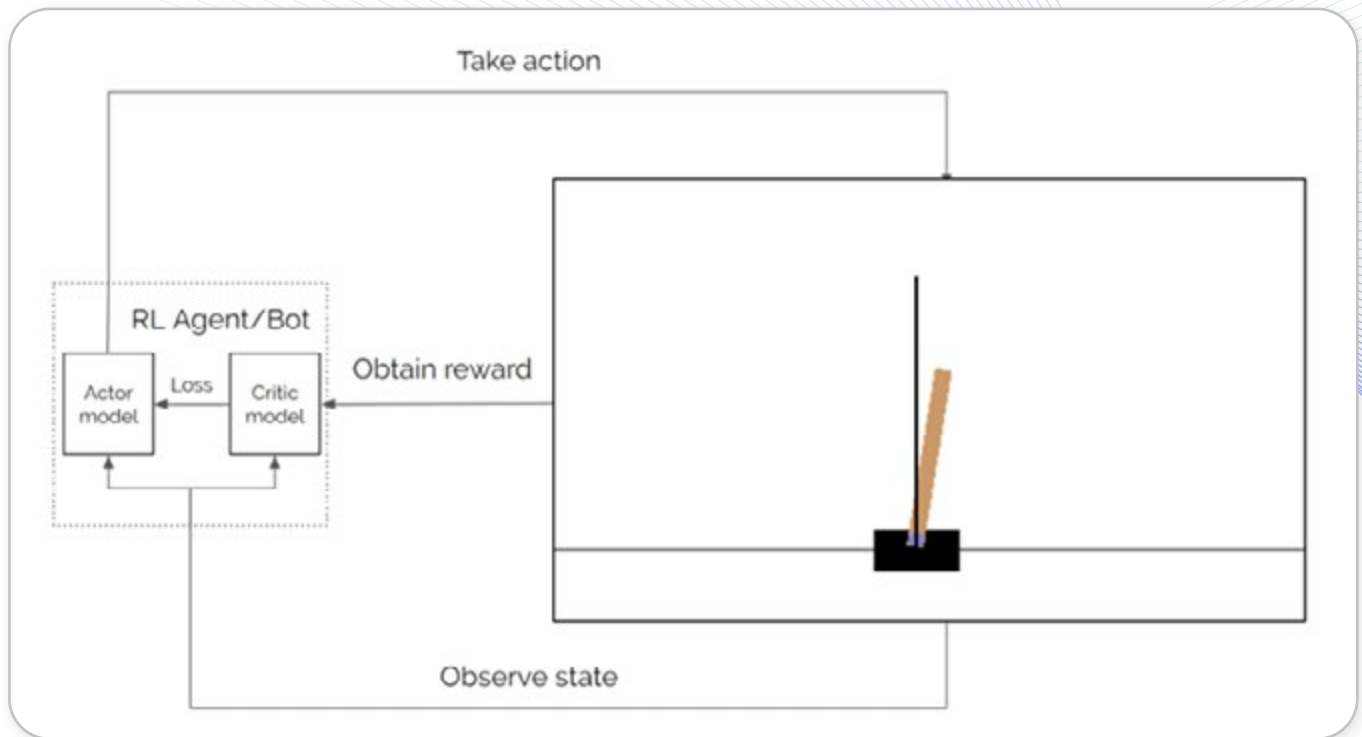


Image credits: [PPO Hands-On Tutorial](#)

You can implement this on your own using the code attached below.^[7] For further reading, please check out the references.

Attachments:



Reinforcement
Learning - Proximal P

References

For references and new developments in deep reinforcement learning please check out the links below:



[\[1\] Introduction to Reinforcement Learning](#)



[\[2\] OpenAI](#)



[\[3\] Getting Started with OpenAI Gym](#)



[\[4\] Actor Critic Algorithm](#)



[\[5\] RL Implementation for Cartpole](#)



[\[6\] Cartpole Documentation and Evaluations](#)



[\[7\] PPO Hands-On Tutorial](#)



[\[8\] Original PPO Paper](#)



[\[9\] Cartpole Animation After 20 Epochs](#)

Authored by

[Ankit Tyagi,](#)

Data Scientist at Absolutdata

Reinforcement Learning

Vivid Visualization

Introduction

Advancements in deep reinforcement learning and the need for large amounts of training data have led to the emergence of virtual-to-real learning. Computer vision communities have shown keen interest in it. 3D engines like Unreal have been used to generate photo-realistic images that can be used for training deep neural networks.

The underlying idea behind using visualizations for reinforcement learning is analogous to a video game. In a game, the player is rewarded for making right choices and penalized for making wrong choices (or moves in a virtually rendered environment); the rewards and penalties are based largely on the player's interaction with the virtual landscape and the non-playable characters.

The 3D virtual environments, thus created, have found application in different learning tasks, such as:

- Autonomous driving
- Collision avoidance
- Image segmentation

Virtual Environment for Visual Deep Learning (VIVID)^[1]

Many open-source environments are readily available for use, but they either provide relatively smaller landscapes or have limited entity-environment interactions. To better facilitate learning via visual recognition, VIVID (Virtual Environment for Visual Deep Learning) is utilized because it can render larger scale, diversified indoor and outdoor landscapes.

Another advantage that VIVID provides is that it can leverage the human skeletal system to simulate different complex human movements and actions. Potential applications of VIVID include:

- Indoor navigation
- Action recognition
- Event detection

Architecture

The VIVID system has the Unreal 3D Engine at its core. Unreal is a proprietary rendering engine by Epic games, first showcased in 1998. It has since been used widely in various high-profile video game titles like Gears of War and PUBG. It's also been used in filmmaking to create virtual sets.^[2]

Hardware simulations are powered by AirSim.^[3] This is an open-source simulator for cars, drones, etc. that is based on the fourth generation of Unreal Engine.

PLAYER CONTROL: This governs the main character – the entity that is rewarded or penalized for different actions and interactions.

Three Character classes are provided in VIVID:

1. ROBOT: The Unreal Mannequin model.
2. DRONE: A simple flying entity. (AirSim simulation is used for more realistic interactions.)
3. CAR: AirSim simulation models.

NPC CONTROL: Non-Playable Characters that form the other entities with whom the player interacts.

1. Blueprint (a visual script function, native to Unreal) is used to program complex NPC behavior.
2. Complex human actions are animated by the Unreal animation editor.
3. Unreal also provides an extra NPC API to spawn NPCs at defined coordinates and perform predefined actions via RPC calls.

SCENES: The landscape/environment. It may be indoors or outdoors, e.g. city, parking garage, etc.

1. The Unreal Engine offers a large community of designers and a large choice of landscapes.
2. Special events can be pre-defined (e.g. forest fire rescue, gun shooting).

DEEP LEARNING API : This allows the VIVID system to communicate with a variety of deep learning libraries.

1. Microsoft RPC (Remote Procedural Call) by AirSim is used to communicate with different programming languages via TCP/IP.
2. This offers a choice of different deep learning libraries that can be incorporated as long as the base language is supported by RPC.
3. It also provides the option to setup distributed learning.

Applications

VIVID finds applications in deep reinforcement learning, action recognition, object recognition, semantic recognition, and video event recognition:

- **Indoor navigation:** Robot and drone navigation simulations.
- **Event recognition:** Real-life events where identification is critical but there is fewer training data available, e.g. a gunman on a road or a school shooting.
- **Action recognition:** An advantage of using VIVID is that it can recognize 3D human actions owing to the virtual environment. In contrast, real-world videos can only show 2-dimensional imagery. VIVID facilitates action recognition by creating a range of free action models on MIXAMO.
- **Autonomous driving:** It also supports the training of autonomous driving algorithms.

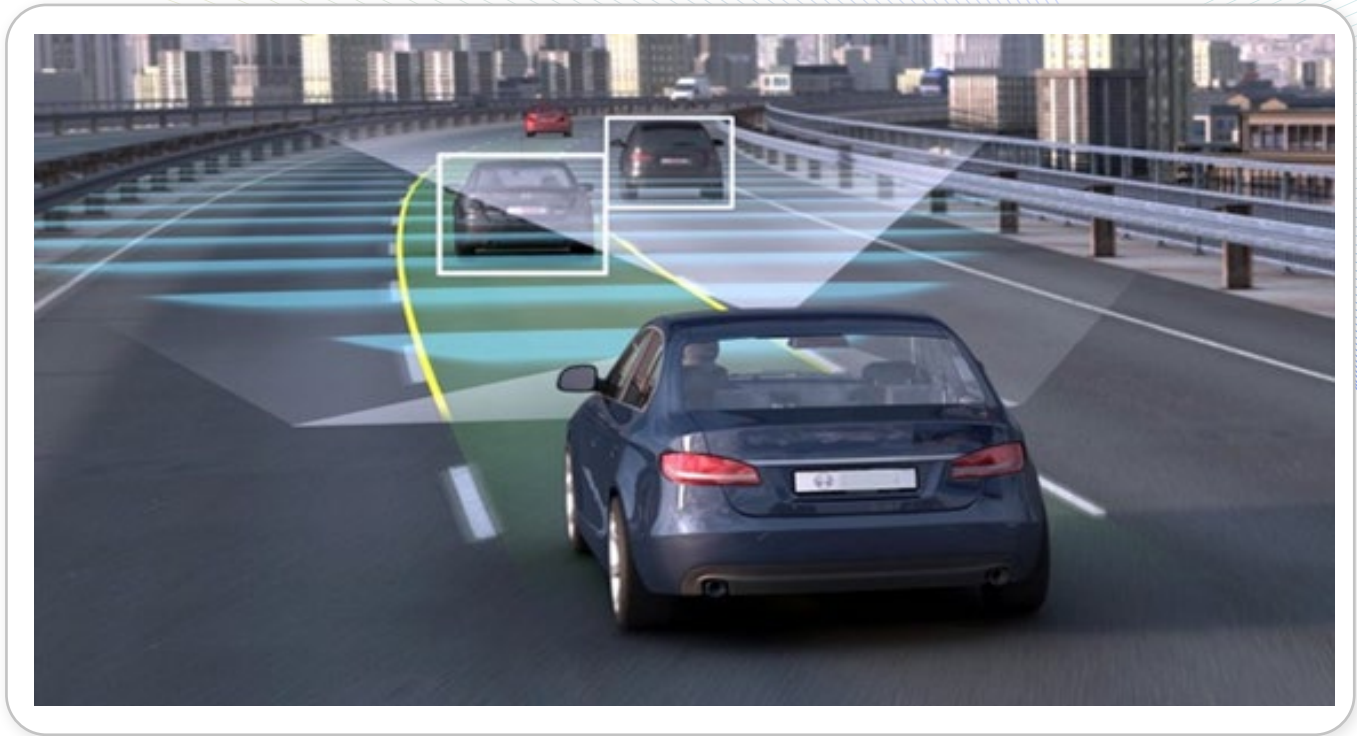





Figure 6: Training of autonomous driving algorithms

[Picture Courtesy](#)

References

-  [1. VIVID: Virtual Environment for Visual Deep Learning](#) - Kuan-Ting Lai , Chia-Chih Lin , Chun-Yao Kang , Mei-Enn Liao , Ming-Syan Chen National Taipei University of Technology, National Taiwan University Taipei City, Taiwan.
-  [2. Wikipedia: Unreal Engine Video Game](#)
-  [3. Wikipedia: AirSim](#)

Authored by
[Nishant Sangwan](#),
Analyst at Absolutdata

Stochastic Ensemble Value Expansion (STEVE) Model for Low Sample Datasets

Thriving Traction

Reinforcement learning (RL) is a framework of trial and error; the system makes short-term decisions while optimizing long-term goals. Having the ability to interact with the environment, the aim of any RL technique is to maximize the expected reward and minimize the number of interactions, i.e. to achieve sample efficiency. But to perform well, modern RL techniques need comparatively more data, which makes them sample inefficient.

RL techniques can be categorized as:

- **Model-Free RL:** Uses the action in an environment to learn the policy while maximizing the reward.
- **Model-Based RL:** Learns the dynamics of the environment and predicts the next state and reward before taking an action.

Knowing the dynamics of the environment beforehand is better than knowing just the rewards, which makes model-based RL more sample efficient than model-free RL. But model based RL has its own drawbacks, such as approximation errors (in estimating model and value function) and inaccurate models that lead to suboptimal policy computation. Computationally, it is also more complex.

To resolve the issues with both technique types, recent research, and development in the uncertainty-aware hybrid technique (built by integrating model-free and model-based RL techniques) have demonstrated high performance. Stochastic Ensemble Value Expansion (STEVE)^[1] is one such technique that can enable the use of RL in practical real-world problems.

RL Techniques

Before diving into STEVE, we need to understand a few RL techniques:

- **Q-Learning:** Q-learning utilizes q-values or action values to iteratively improve the learning agent's behavior. They are characterized by actions and states. $Q(S, A)$ is an assessment of how great it is to make the move A at the state S ; this assessment is computed utilizing TD learning.^[2]
- **Temporal Difference (TD) Learning:** TD learning is a model-free RL technique in which the agent doesn't have prior knowledge of the environment; it learns through episodes of trial and error from the environment. It is represented as $Q(S, A) \leftarrow Q(S, A) + \alpha(R + \gamma Q(S', A') - Q(S, A))$, where S is the agent's current state, A is the current action taken based on the policy, R is the reward, γ is the discounting factor and α is the step length taken for updating $Q(S, A)$.^[2] On generalizing, the TD update can also be represented as:

$$T^{TD} = R + \gamma Q(S', A')$$

At every time step, the agent interacts with the environment and the q-value's estimate is computed using the above TD update rule.

- **Model-Based Value Expansion (MVE) Learning:** MVE is a hybrid technique that combines model-based and model-free learning by incorporating a model into the q-value target estimation. This incorporation of a learned dynamics model within a model-free technique improves value estimation and reduces sample complexity. The uncertainty in the model is also controlled by fixing the model to run for a certain depth (H). The MVE target for H = 3 is represented as: [3]

$$T_H^{MVE} = R_0 + \gamma R_1 + \gamma^2 R_2 + \gamma^3 Q(S'_2, A'_2)$$

However, in an adequately complex environment, the learned dynamics model will quite often be imperfect; this is again a challenge in combining MF and MB techniques.

STEVE – An Extension of MVE

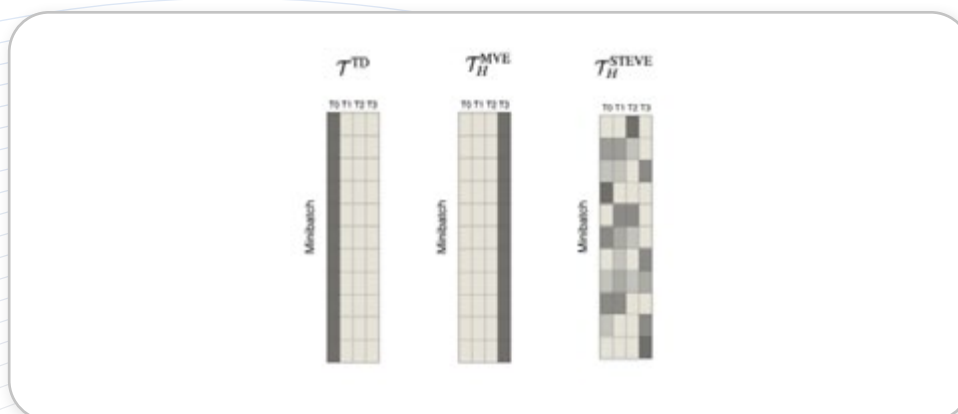
STEVE, a novel model-based technique developed by Google Brain, is an extension to the MVE. Specifically, it addresses the issues faced in the MVE. Both models try to improve the target for TD learning by using a dynamic model to compute the rollouts in such a way that performance is not degraded due to the model's error.

MVE uses a model and q-learning for the computation of interpolated targets with a fixed-length rollout into the future, certainly collecting the model error. "In STEVE, the model and the Q-Learning are replaced by the ensembles, approximating the uncertainty of an estimate by computing its variance under samples from the ensemble".[4] By calculating the uncertainty, STEVE utilizes the model rollouts dynamically only when they don't introduce huge error.

In a single rollout with H timestep, H+1 different candidate targets can be computed by considering rollouts to numerous horizon lengths, such as $T_0^{MVE}, T_1^{MVE}, T_2^{MVE}, \dots, T_H^{MVE}$. The target in standard TD learning is T_0^{MVE} . In MVE, the target is T_H^{MVE} [1]. In STEVE, the target is the interpolation of all the candidate targets, which in naive terms can also be said as averaging the candidate targets or weighing them in an exponential decaying method in such a way that it balances the error from both the learned q-function and the longer model rollouts. The implementation from Google Brain is present [here](#) for different continuous control tasks using STEVE.

$$T_H^{STEVE} = \sum_{i=0}^H w_i T_i^{MVE} = \sum_{i=0}^H \frac{\bar{w}_i}{\sum_j \bar{w}_j} T_i^\mu, \quad \bar{w}_i^{-1} = T_i^{\sigma^2}$$

Where, T_i^μ is empirical mean and $T_i^{\sigma^2}$ is the variance corresponding to partial rollout of length i.



Comparison of TD, MVE and STEVE Learning Process, Source - <https://danijar.com/asset/steve/poster.pdf>

Real-Time RL Data Science Applications

The aforementioned technique paves the way for incorporating RL into complex real-life data science problem where supervised and unsupervised models cannot help. Training an RL model for real-world tasks is extremely challenging. Also, the disadvantage of needing more data commonly makes the task of training an RL model infeasible. Some of the data science applications where this methodology can be incorporated despite having less data are:^[5]

- **Digital Marketing:** When limited user data makes it difficult to deliver personalized recommendations, RL can produce high-quality recommendations by dynamically capturing user behavior, preferences, and requirements.
- **Chatbots:** Chatbot offerings can be improved by using RL to leverage data from user communications.
- **Predicting Future Sales and Stock Prices:** Prediction models for future sales and stock prices can be easily developed, but these models cannot determine what action to take at a certain stock price on which an optimally trained and performing RL model can efficiently decide whether to buy, hold, or sell.
- **Large-Scale Production Systems:** RL-based platforms can be used to optimize large-scale production systems. An exemplary example of RL in video display is serving a client a low or high bitrate video, depending on the condition of other AI frameworks' estimates and video buffers.
- **Marketing and Advertising:** The ability to precisely focus on the individual is extremely significant in marketing; hitting the right targets clearly leads to exceptional returns. RL can be incorporated into such tasks, as policies can be trained to handle complex environments.^[6]
- **Revenue and ROI (Return on Investment):** These can be maximized by optimal and relevant ad/promotion placement based on the user's interest via real-time advertising. But the most essential thing is that it requires a strategic response and real-time bidding, due to the different advertiser bidding in the market; this requires a dynamic model.^[7]

References

- [1] [Stochastic Ensemble Value Expansion](#)
- [2] [Q-Learning in Python](#)
- [3] [Model-Based Value Expansion for Efficient Model-Free Reinforcement Learning](#)
- [4] [Sample-Efficient Reinforcement Learning with Stochastic Ensemble Value Expansion](#)
- [5] [Real-Life Applications of Reinforcement Learning](#)
- [6] [Business Use-Case of Reinforcement Learning in Trading](#)
- [7] [Real-Time Bidding with Multi-Agent Reinforcement Learning](#)

Authored by

[Darshita Rathore,](#)

Analyst, NAVIK and Analytics, at Absolutdata

Busting Some Common Reinforcement Learning Myths

Folk-Wisdom's Fallacy

Reinforcement learning (RL) is popularly recognized as reward-based learning or self-experienced learning with sequential decision-making mechanisms with the long-term goal of maximizing rewards or minimizing undesired outcomes or penalties.^[1]

As an emerging AI field, reinforcement learning is an ongoing transitional fragment with the aim of making optimal decisions through trial and error implementation.

Apart from its wide use in robotics, reinforcement learning is also a mainstream technique used in gaming or sometimes to outsmart humans (such as AlphaGo).^[2]

Reinforcement Learning Myths

“Reinforcement learning is about taking suitable (optimal) actions to maximize the rewards in a particular scenario or learning optimal policies.”

There is a never-ending balance being maintained between choosing a path that has already been described as “good” and discovering new territory – and possibly finding better results in the solution space.

There are many RL algorithms that favor learning simply “good” policies.

“Very large state spaces are hard for reinforcement learning.”

Many researchers have applied this technique to large discrete state spaces and numerous learning tasks, which include continuous state spaces with very high dimensionality.

We can even create artificial state spaces as mentioned above, where we can define the optimal policy or the optimal value function in very simple terms; therefore, it can be easily learnt.

There already exist many RL algorithms whose complexity does not depend on the size of the sample space, e.g. approximate policy iteration, policy gradients, conservative policy iteration, etc.

“Reinforcement learning is slow.”

We might not have off-the-shelf RL algorithms for different tasks (like SVMs are for classification), but that does not make RL slow. There are numerous success stories in different domains depicting this.^[3]

There is either no way of solving the problem instance for which RL is being used or there is a super-easy non-RL method to solve the problem and the user is comparing the performances of the RL algorithm to this “engineered” algorithm.

“Partially observable Markov decision processes are hard for reinforcement learning.”

The basic principle of RL is that the observations available to the agent at each step are not sufficient statistics of history – therefore, RL clearly succeeds at large-scale partially observable Markov decision processes.

New RL algorithms, such as sparse-sampling and trajectory-tree, scale independently of the size of the state space and therefore work equally well with POMDPs.

References



[1] [Why RL Is Flawed](#)



[2] [Applications of Reinforcement Learning in the Real World](#)



[3] [Myths of Reinforcement Learning](#)

Authored by

[Devika Goyal,](#)

Analyst at Absolutdata

Decision Option Generator – A Meta Learning Approach in NAVIK

Experience Extended

Reinforcement learning (RL) is a branch of machine learning wherein actions are taken based on interactions with an environment. The goal is to maximize a cumulative reward. Because of the ability of reinforcement learning to learn on the fly and auto-correct, it's finding applications in a variety of fields and showing more benefits than traditional machine learning algorithms.

NAVIK AI's Decision Option Generator

A host of individual models for churn propensity, collaborative filtering, purchase frequency, etc. are consolidated under the Decision Option Generator (DOG), the NAVIK AI suite's proprietary algorithm. [1] This algorithm is what builds the underlying recommendation engine for NAVIK MarketingAI and NAVIK SalesAI.

To incorporate custom business rules in the model outputs, DOG leverages meta learning. This leads to minimal anomalies in the final recommendations.

Meta learning, in its true sense, refers to training a model on a set of various related tasks with a limited number of data points; presented with a similar new task, the model utilizes the learnings from the previous tasks and doesn't need training from scratch.[2] Cognitively, human minds work in a similar manner: learning from different experiences and connecting those learnings to a new concept.

What Does Meta Learning Address?

Meta learning, for NAVIK, is needed to address a slew of factors like turbulent market dynamics, custom business needs that change over time, and the post-facto modification of recommendations due to users' discomfort with them. Owing to meta learning, NAVIK products constantly learn from the above-mentioned factors and have the flexibility to modify the recommendations in the next model refresh cycle.

Appreciating and Depreciating Reinforcement Multipliers

DOG implements meta learning using a reinforcement multiplier – a term based on custom business rules that are multiplied to the final recommendation score.

Based on the number of wins by the number of pitches for various opportunities, there are two specific mathematical terms (known as appreciating and depreciating reinforcement multipliers, or ARM and DRM) that are multiplied to the outcomes (e.g. the Contact Probability Score) of the corresponding constituent DOG model (e.g. Contact Lead Scoring) or the Consolidated Recommendation Score. The values for ARM and DRM can be refreshed every week based on the win rates of the contact-product combinations.

References



[1] [Meta Learning in NAVIK Products - Recursive Reinforcement](#)



[2] [Ravichandiran, S. \(2018\). Hands-On Meta Learning with Python](#)

Authored by

[Nibedita Dutta,](#)

Data Scientist at Absolutdata

Boost Productivity Through Multi Agent Reinforcement Learning

Food for Thought Experiment

Great Things Come from Fighting Continuous Small Battles

It's a fact that reinforcement learning (RL) has certain limitations; thanks to these, other advanced techniques like recurrent neural networks (RNN) and convolutional neural networks (CNN) are gaining greater popularity. The issue is more towards practical application, as either there is a limited section of problems where RL can be applied, or the problem can't be molded in a way which is accepted by RL.

In order to apply RL to solve a problem, we must be able to answer certain questions and thus adjudge it as an appropriate candidate for RL. Most importantly, the problem should be able to transform into a Markov decision process (MDP), i.e. a user should be able to define a state-action space and a reward system and the overall framework should be able to do better by receiving feedback from the environment.^[1]

If the above aspects are taken care of in the problem, RL can simulate some eye-popping what-if scenarios – e.g. the user could create a parallel scenario and compare how different the results could be if the actions were performed in a different manner/route than the original.

Supercharging RL: Combining RL with Deep Learning

Some of the limitations of RL can be offset by combining it with deep learning frameworks like CNN or RNN. By combining, agents garner some new abilities; in the case of CNN. The agent can see through the environment and learn to interact within it.^[1] Similarly, in the case of RNN (which has a memory component), agents are assisted in memorizing things.

Multi Agent Reinforcement Learning

Many historic innovations have happened by studying the psychology or movement of animals and birds. For example, the Wright brothers built their airplane after taking cues from how birds fly.^[2] A similar proposition works for multi-agent RL; the behavior of ants was studied to formulate multi-agent robots or agents. The most fascinating behavior of ants is their ability to closely coordinate a task (i.e. gathering food) and achieve a common goal. If something similar is taught to the robots/agents, the opportunity created thus will have no bounds.

The prime objective of RL is to build an agent in a way that their goal is to maximize rewards within an environment. There are instances where robots have outperformed humans in various fields. Taking a cue from this, there is a level-up in the field of RL called multi-agent reinforcement learning (MAREL). This field studies how multiple agents can collectively learn, communicate, and co-exist in an environment to achieve a common goal.^[3] If this becomes more of a reality, it will open a plethora of opportunities in fields like farming, manufacturing (e.g. building mega-factories) and healthcare (e.g. building critical hospitals in a pandemic situation). Of course, it could be applied in many ways to boost productivity.

Food For Thought

There is yet much to be explored in the field of RL and specifically in MARL, where the challenges include the efficient collaboration and co-existence of agents in the environment. For efficient collaboration, agents need to communicate; hence, there is a requirement for a common language. This part can be tricky; for example, one agent could develop a way of communication which the other agent doesn't understand because they've followed a different learning trajectory. In this case, communication won't be effective and fruitful.^[4]

Hence, building an effective communication method so that agents can understand each other is the most critical aspect in the future development of any MARL system.

References



[\[1\] Applications of Reinforcement Learning in the Real World](#)



[\[2\] What Influenced the Wright Brothers About How Things Fly](#)



[\[3\] The Gist of Multi Agent Reinforcement Learning](#)



[\[4\] MARL and the Future of AI](#)

Authored by

[Abhijeet Agarwal,](#)

Data Scientist at Absolutdata

Data Science Competitions/ Seminars / Fora / Courses

Online Courses:

1. Reinforcement Learning Specialization

Platform: Coursera

Fees: Subscription

Time to Complete: 4-6 months

2. Deep Reinforcement Learning Nanodegree

Platform: Udacity

Fees: Rs. 63956 / \$11116 USD

Time to Complete: 4 months

3. Reinforcement Learning Lecture Series 2018

Platform: DeepMind, University College London (UCL)

Fees: Free

Time to Complete: 20 hours

4. CS234: Reinforcement Learning Winter 2021

Platform: Stanford Online

Fees: Free

Time to Complete: 10 weeks

5. Practical Reinforcement Learning

Platform: Coursera

Fees: Subscription

Time to Complete: 1 month

6. Artificial Intelligence: Reinforcement Learning in Python

Platform: Udemy

Fees: Rs. 1920

Time to Complete: 13 hours

7. Advanced AI: Deep Reinforcement Learning in Python

Platform: Udemy

Fees: Rs. 1920

Time to Complete: 10.5 hours

8. Cutting-Edge AI: Deep Reinforcement Learning in Python

Platform: Udemy

Fees: Rs. 490

Time to Complete: 8.5 hours

9. Deep Reinforcement Learning 2.0

Platform: Udemy

Fees: Rs. 518

Time to Complete: 9.5 hours

10. Modern Reinforcement Learning: Actor-Critic Algorithms

Platform: Udemy

Fees: Rs. 490

Time to Complete: 8 hours

11. Modern Reinforcement Learning: Deep Q Learning in PyTorch

Platform: Udemy

Fees: Rs. 518

Time to Complete: 5.5 hours

12. Reinforcement Learning with PyTorch

Platform: Udemy

Fees: Rs. 490

Time to Complete: 7 hours

13. Deep Reinforcement Learning: Hands-on AI Tutorial in Python

Platform: Udemy

Fees: Rs. 490

Time to Complete: 4 hours

Conferences/Events:

1. Reinforcement Learning Summit

Organizer: Re-Work

Location: Toronto, Ontario, Canada

Dates: 19-20 Oct 2021 (Recurring Annually)

Agenda: The advancements of reinforcement learning to achieve the goal of human-level intelligence

2. International Conference on Reinforcement Learning for Control

Organizer: World Academy of Science, Engineering and Technology (WASET)

Location: Lisbon, Portugal | Sydney, Australia

Dates: September 20-21, 2021 (Recurring Annually - Lisbon) | December 02-03, 2021 (Recurring Annually - Sydney)

Agenda: Conference

3. International Conference on Structure and Priors in Reinforcement Learning

Organiser: World Academy of Science, Engineering and Technology (WASET)

Location: Helsinki, Finland

Dates: July 19-20, 2022

Agenda: Conference

4. Conference on Neural Information Processing Systems

Organiser: NeurIPS Board

Location: Virtual

Dates: December 6, 2021

Agenda: Conference

Fora:

1. Stanford RL (Reinforcement Learning) Forum

Host: Stanford

Location: Virtual

Competitions:

1. [AWS DeepRacer League](#)

Host: AWS, Amazon

Date: 1st March 2021 – 4th December 2021 (Ongoing)

Location: Virtual

2. [Connect X](#)

Host: Kaggle

Location: Virtual

3. [Flatland](#)

Host: AICrowd

Location: Virtual

4. [MineRL Diamond Challenge](#)

Host: AICrowd

Location: Virtual

5. [MineRL Basalt Challenge](#)

Host: AICrowd

Location: Virtual

Curated by

[Paresh Pradhan,](#)

Data Scientist at Absolutdata



ABSOLUTDATA

An Infogain Company

Thank You

For reading this edition of BrainWave from the Absolutdata Labs Team. This digest focuses on some technical angles of analytics and data science. BrainWave is published every quarter. If you haven't already, please subscribe so you receive future editions.

Subscribe

