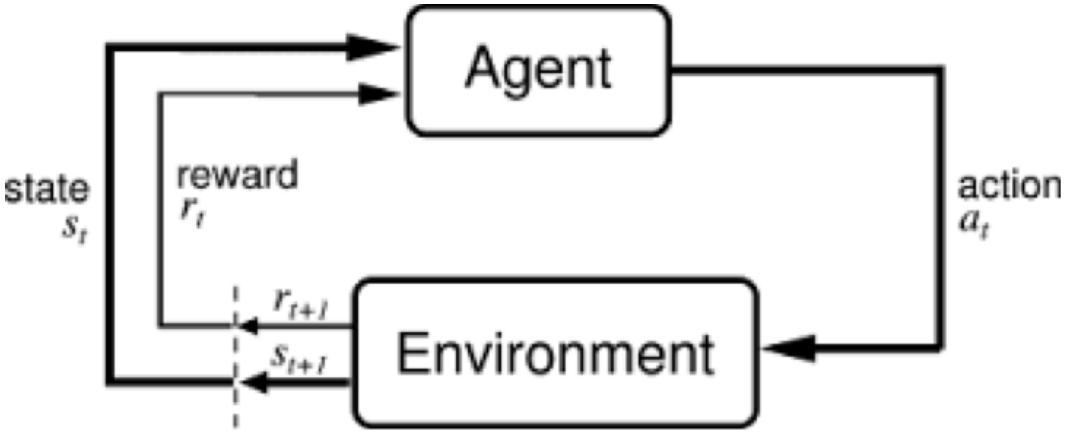


A Beginner's Guide to Deep Reinforcement Learning

skymind.ai

When it is not in our power to determine what is true, we ought to act in accordance with what is most probable. - Descartes

Contents

While neural networks are responsible for recent breakthroughs in problems like computer vision, machine translation and time series prediction – they can also combine with reinforcement learning algorithms to create something astounding like AlphaGo.

Reinforcement learning refers to goal-oriented algorithms, which learn how to attain a complex objective (goal) or maximize along a particular dimension over many steps; for example, maximize the points won in a game over many moves. They can start from a blank slate, and under the right conditions they achieve superhuman performance. Like a child incentivized by spankings and candy, these algorithms are penalized when they make the wrong decisions and rewarded when they make the right ones – this is reinforcement.

Reinforcement algorithms that incorporate deep learning can beat world champions at the game of Go as well as human experts playing numerous Atari video games. While that may sound trivial, it's a vast improvement over their previous accomplishments, and the state of the art is progressing rapidly.

Reinforcement learning solves the difficult problem of correlating immediate actions with the delayed returns they produce. Like humans, reinforcement learning algorithms sometimes have to wait a while to see the fruit of their decisions. They operate in a delayed return environment, where it can be difficult to understand which action leads to which outcome over many time steps.

Reinforcement learning algorithms can be expected to perform better and better in more ambiguous, real-life environments while choosing from an arbitrary number of possible actions, rather than from the limited options of a video game. That is, with time we expect them to be valuable to achieve goals in the real world.

Two reinforcement learning algorithms - Deep-Q learning and A3C - have been implemented in a Deeplearning4j library called RL4J. It can already play Doom.

Reinforcement Learning Definitions

Reinforcement learning can be understood using the concepts of agents, environments, states, actions and rewards, all of which we'll explain below. Capital letters tend to denote sets of things, and lower-case letters denote a specific instance of that thing; e.g. A is all possible actions, while a is a specific action contained in the set.

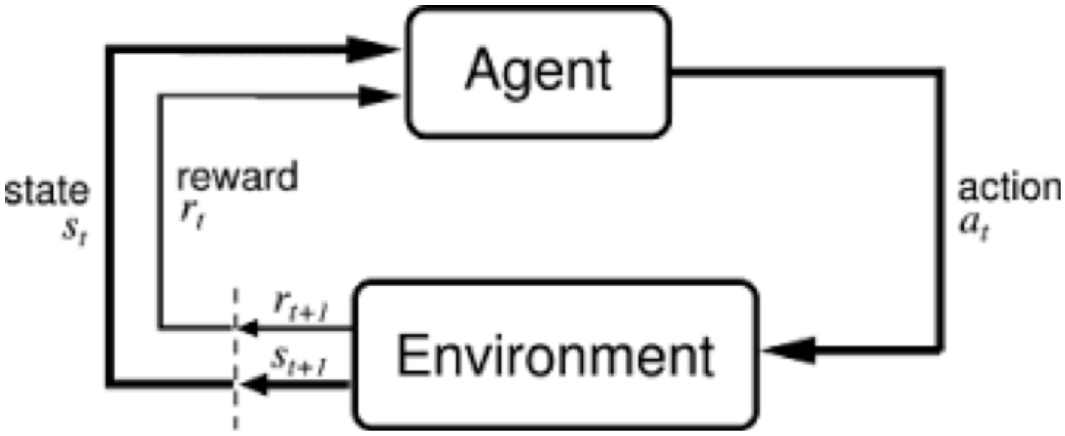
- Agent: An **agent** takes actions; for example, a drone making a delivery, or Super Mario navigating a video game. The algorithm is the agent. In life, the agent is you.[1]

- Action (A): `A` is the set of all possible moves the agent can make. An **action** is almost self-explanatory, but it should be noted that agents choose among a list of possible actions. In video games, the list might include running right or left, jumping high or low, crouching or standing still. In the stock markets, the list might include buying, selling or holding any one of an array of securities and their derivatives. When handling aerial drones, alternatives would include many different velocities and accelerations in 3D space.
- Discount factor: The **discount factor** is multiplied by future rewards as discovered by the agent in order to dampen thse rewards' effect on the agent's choice of action. Why? It is designed to make future rewards worth less than immediate rewards; i.e. it enforces a kind of short-term hedonism in the agent. Often expressed with the lower-case Greek letter gamma: γ. If γ is .8, and there's a reward of 10 points after 3 time steps, the present value of that reward is $0.8^3 \times 10$. A discount factor of 1 would make future rewards worth just as much as immediate rewards. We're fighting against delayed gratification here.
- Environment: The world through which the agent moves. The environment takes the agent's current state and action as input, and returns as output the agent's reward and its next state. If you are the agent, the environment could be the laws of physics and the rules of society that process your actions and determine the consequences of them.
- State (S): A **state** is a concrete and immediate situation in which the agent finds itself; i.e. a specific place and moment, an instantaneous configuration that puts the agent in relation to other significant things such as tools, obstacles, enemies or prizes. It can the current situation returned by the environment, or any future situation. Were you ever in the wrong place at the wrong time? That's a state.

- Reward (R): A **reward** is the feedback by which we measure the success or failure of an agent's actions. For example, in a video game, when Mario touches a coin, he wins points. From any given state, an agent sends output in the form of actions to the environment, and the environment returns the agent's new state (which resulted from acting on the previous state) as well as rewards, if there are any. Rewards can be immediate or delayed. They effectively evaluate the agent's action.
- Policy (π): The **policy** is the strategy that the agent employs to determine the next action based on the current state. It maps states to actions, the actions that promise the highest reward.
- Value (V): The expected long-term return with discount, as opposed to the short-term reward `R`. `Vπ(s)` is defined as the expected long-term return of the current state under policy `π`. We discount rewards, or lower their estimated value, the further into the future they occur. See discount factor. And remember Keynes: "In the long run, we are all dead." That's why you discount future rewards.
- Q-value or action-value (Q): **Q-value** is similar to Value, except that it takes an extra parameter, the current action `a`. `Qπ(s, a)` refers to the long-term return of the current state `s`, taking action a under policy `π`. Q maps state-action pairs to rewards. Note the difference between Q and policy.
- Trajectory: A sequence of states and actions that influence those states. From the Latin "to throw across." The life of an agent is but a ball tossed high and arching through space-time.

So environments are functions that transform an action taken in the current state into the next state and a reward; agents are functions that transform the new state and reward into the next action. We can know the agent's function, but we cannot know the function of the environment. It is a black box where we only see the inputs and outputs. It's like most people's relationship with technology: we know

what it does, but we don't know how it works. Reinforcement learning represents an agent's attempt to approximate the environment's function, such that we can send actions into the black-box environment that maximize the rewards it spits out.



*Credit: Sutton & Barto

In the feedback loop above, the subscripts denote the time steps `t` and `t+1`, each of which refer to different states: the state at moment `t`, and the state at moment `t+1`. Unlike other forms of machine learning – such as supervised and unsupervised learning – reinforcement learning can only be thought about sequentially in terms of state-action pairs that occur one after the other.

Reinforcement learning judges actions by the results they produce. It is goal oriented, and its aim is to learn sequences of actions that will lead an agent to achieve its goal, or maximize its objective function. Here are some examples:

- In video games, the goal is to finish the game with the most points, so each additional point obtained throughout the game will affect the agent's subsequent behavior; i.e. the agent may learn that it should shoot battleships, touch coins or dodge meteors to maximize its score.
- In the real world, the goal might be for a robot to travel from point A to point B, and every inch the robot is able to move closer to point B could be counted like points.

Here's an example of an objective function for reinforcement learning; i.e. the way it defines its goal.

$$\sum_{t=0}^{t=\infty} \gamma^t r(x(t), a(t))$$

We are summing reward function *r* over *t*, which stands for time steps. So this objective function calculates all the reward we could obtain by running through, say, a game. Here, *x* is the state at a given time step, and *a* is the action taken in that state. *r* is the reward function for *x* and *a*. (We'll ignore *γ* for now.)

Reinforcement learning differs from both supervised and unsupervised learning by how it interprets inputs. We can illustrate their difference by describing what they learn about a "thing."

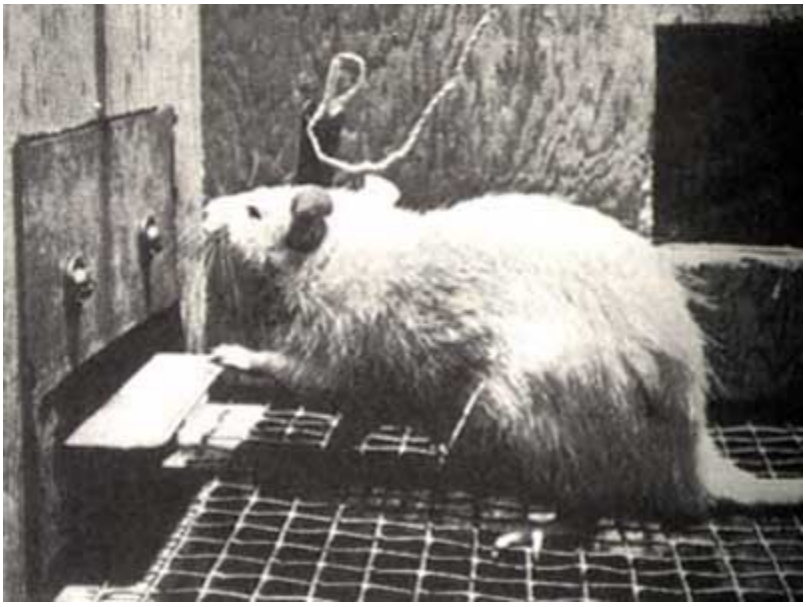
- Unsupervised learning: That thing is like this other thing. (The algorithms learn similarities w/o names, and by extension they can spot the inverse and perform anomaly detection by recognizing what is unusual or dissimilar)
- Supervised learning: That thing is a "double bacon cheese burger". (Labels, putting names to faces...) These algorithms learn the correlations between data instances and their labels; that is, they require a labelled dataset. Those labels are used to "supervise" and correct the algorithm as it makes wrong guesses when predicting labels.
- Reinforcement learning: Eat that thing because it tastes good and will keep you alive longer. (Actions based on short- and long-term rewards, such as the amount of calories you ingest, or the length of time you survive.) Reinforcement learning can be thought of as supervised learning in an environment of sparse feedback.

Domain Selection for Reinforcement Learning

One way to imagine an autonomous reinforcement learning agent would be as a blind person attempting to navigate the world with only their ears and a white cane. Agents have small windows that allow them to perceive their environment, and those windows may not even be the most appropriate way for them to perceive what's around them.

(In fact, deciding *which types* of input and feedback your agent should pay attention to is a hard problem to solve. This is known as domain selection. Algorithms that are learning how to play video games can mostly ignore this problem, since the environment is man-made and strictly limited. Thus, video games provide the sterile environment of the lab, where ideas about reinforcement learning can be tested.

Domain selection requires human decisions, usually based on knowledge or theories about the problem to be solved; e.g. selecting the domain of input for an algorithm in a self-driving car might include choosing to include radar sensors in addition to cameras and GPS data.)



State-Action Pairs & Complex Probability Distributions of Reward

The goal of reinforcement learning is to pick the best known action for any given state, which means the actions have to be ranked, and assigned values relative to one another. Since those actions are state-dependent, what we are really gauging is the value of state-action pairs; i.e. an action taken from a certain state, something you did somewhere. Here are a few examples to demonstrate that the value and meaning of an action is contingent upon the state in which it is taken:

- If the action is marrying someone, then marrying a 35-year-old when you're 18 probably means something different than marrying a 35-year-old when you're 90, and those two

outcomes probably have different motivations and lead to different outcomes.

- If the action is yelling “Fire!”, then performing the action a crowded theater should mean something different from performing the action next to a squad of men with rifles. We can’t predict an action’s outcome without knowing the context.

We map state-action pairs to the values we expect them to produce with the Q function, described above. The Q function takes as its input an agent’s state and action, and maps them to probable rewards.

Reinforcement learning is the process of running the agent through sequences of state-action pairs, observing the rewards that result, and adapting the predictions of the Q function to those rewards until it accurately predicts the best path for the agent to take. That prediction is known as a policy.

Reinforcement learning is an attempt to model a complex probability distribution of rewards in relation to a very large number of state-action pairs. This is one reason reinforcement learning is paired with, say, a Markov decision process, a method to sample from a complex distribution to infer its properties. It closely resembles the problem that inspired Stan Ulam to invent the Monte Carlo method; namely, trying to infer the chances that a given hand of solitaire will turn out successful.

Any statistical approach is essentially a confession of ignorance. The immense complexity of some phenomena (biological, political, sociological, or related to board games) make it impossible to reason from first principles. The only way to study them is through statistics, measuring superficial events and attempting to establish correlations between them, even when we do not understand the mechanism by which they relate. Reinforcement learning, like deep neural networks,

is one such strategy, relying on sampling to extract information from data.

After a little time spent employing something like a Markov decision process to approximate the probability distribution of reward over state-action pairs, a reinforcement learning algorithm may tend to repeat actions that lead to reward and cease to test alternatives. There is a tension between the exploitation of known rewards, and continued exploration to discover new actions that also lead to victory. Just as oil companies have the dual function of pumping crude out of known oil fields while drilling for new reserves, so too, reinforcement learning algorithms can be made to both **exploit** and **explore** to varying degrees, in order to ensure that they don't pass over rewarding actions at the expense of known winners.

Reinforcement learning is iterative. In its most interesting applications, it doesn't begin by knowing which rewards state-action pairs will produce. It learns those relations by running through states again and again, like athletes or musicians iterate through states in an attempt to improve their performance.

The Relationship Between Machine Learning with Time

You could say that an algorithm is a method to more quickly aggregate the lessons of time.[2] Reinforcement learning algorithms have a different relationship to time than humans do. An algorithm can run through the same states over and over again while experimenting with different actions, until it can infer which actions are best from which states. Effectively, algorithms enjoy their very own Groundhog Day, where they start out as dumb jerks and slowly get wise.

Since humans never experience Groundhog Day outside the movie, reinforcement learning algorithms have the potential to learn more,

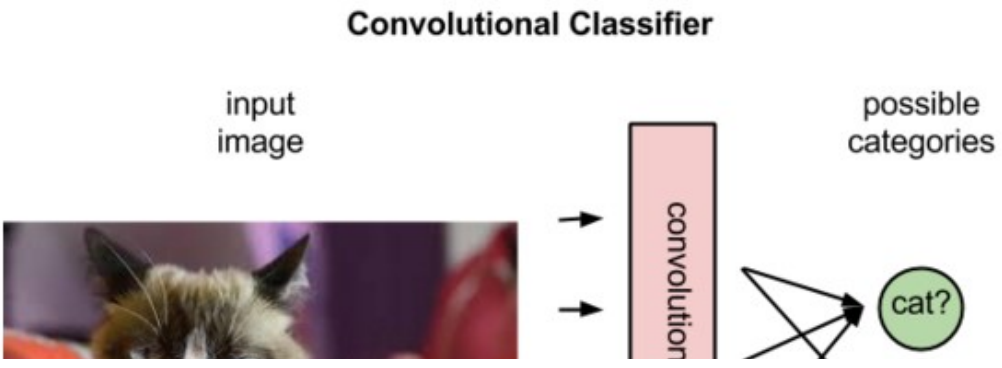
and better, than humans. Indeed, the true advantage of these algorithms over humans stems not so much from their inherent nature, but from their ability to live in parallel on many chips at once, to train night and day without fatigue, and therefore to learn more. An algorithm trained on the game of Go, such as AlphaGo, will have played many more games of Go than any human could hope to complete in 100 lifetimes.[3]

Neural Networks and Deep Reinforcement Learning

Where do neural networks fit in? Neural networks are the agent that learns to map state-action pairs to rewards. Like all neural networks, they use coefficients to approximate the function relating inputs to outputs, and their learning consists to finding the right coefficients, or weights, by iteratively adjusting those weights along gradients that promise less error.

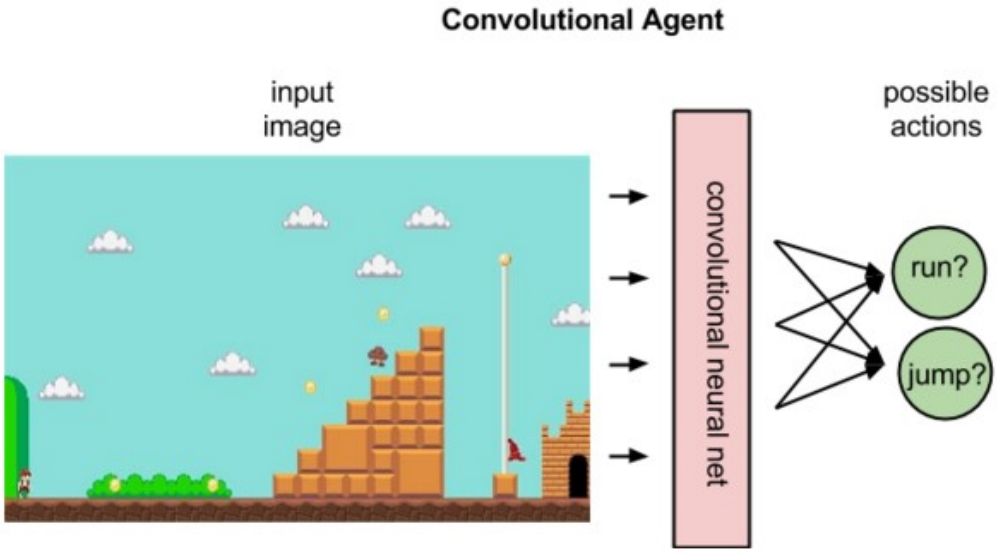
In reinforcement learning, convolutional networks can be used to recognize an agent's state; e.g. the screen that Mario is on, or the terrain before a drone. That is, they perform their typical task of image recognition.

But convolutional networks derive different interpretations from images in reinforcement learning than in supervised learning. In supervised learning, the network applies a label to an image; that is, it matches names to pixels.



In fact, it will rank the labels that best fit the image in terms of their probabilities. Shown an image of a donkey, it might decide the picture is 80% likely to be a donkey, 50% likely to be a horse, and 30% likely to be a dog.

In reinforcement learning, given an image that represents a state, a convolutional net can rank the actions possible to perform in that state; for example, it might predict that running right will return 5 points, jumping 7, and running left none.



The above image illustrates what a policy agent does, mapping a state to the best action.

$$a = \pi(s)$$

A policy maps a state to an action.

If you recall, this is distinct from Q, which maps state action pairs to rewards.

To be more specific, Q maps state-action pairs to the highest combination of immediate reward with all future rewards that might be harvested by later actions in the trajectory. Here is the equation for Q, from Wikipedia:

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Having assigned values to the expected rewards, the Q function simply selects the state-action pair with the highest so-called Q value.

At the beginning of reinforcement learning, the neural network coefficients may be initialized stochastically, or randomly. Using feedback from the environment, the neural net can use the difference between its expected reward and the ground-truth reward to adjust its weights and improve its interpretation of state-action pairs.

This feedback loop is analogous to the backpropagation of error in supervised learning. However, supervised learning begins with knowledge of the ground-truth labels the neural network is trying to predict. Its goal is to create a model that maps different images to their respective names.

Reinforcement learning relies on the environment to send it a scalar number in response to each new action. The rewards returned by the environment can be varied, delayed or affected by unknown variables, introducing noise to the feedback loop.

This leads us to a more complete expression of the Q function, which takes into account not only the immediate rewards produced by an action, but also the delayed rewards that may be returned several time steps deeper in the sequence.

Like human beings, the Q function is recursive. Just as calling the wetware method `human()` contains within it another method `human()`, of which we are all the fruit, calling the Q function on a given state-action pair requires us to call a nested Q function to predict the value

of the next state, which in turn depends on the Q function of the state after that, and so forth.

Footnotes

1) *It might be helpful to imagine a reinforcement learning algorithm in action, to paint it visually. Let's say the algorithm is learning to play the video game Super Mario. It's trying to get Mario through the game and acquire the most points. To do that, we can spin up lots of different Marios in parallel and run them through the space of all possible game states. It's as though you have 1,000 Marios all tunnelling through a mountain, and as they dig (e.g. as they decide again and again which action to take to affect the game environment), their experience-tunnels branch like the intricate and fractal twigs of a tree. The Marios' experience-tunnels are corridors of light cutting through the mountain. And as in life itself, one successful action may make it more likely that successful action is possible in a larger decision flow, propelling the winning Marios onward. You might also imagine, if each Mario is an agent, that in front of him is a heat map tracking the rewards he can associate with state-action pairs. (Imagine each state-action pair as have its own screen overlayed with heat from yellow to red. The many screens are assembled in a grid, like you might see in front of a Wall St. trader with many monitors. One action screen might be "jump harder from this state", another might be "run faster in this state" and so on and so forth.) Since some state-action pairs lead to significantly more reward than others, and different kinds of actions such as jumping, squatting or running can be taken, the probability distribution of reward over actions is not a bell curve but instead complex, which is why Markov and Monte Carlo techniques are used to explore it, much as Stan Ulam explored winning Solitaire hands. That is, while it is difficult to describe the reward distribution in a formula, it can be sampled. Because the algorithm starts ignorant and many of the paths through the game-state space are unexplored, the heat maps will reflect their lack of experience; i.e. there could be blanks in the heatmap of the rewards they imagine, or they might just*

start with some default assumptions about rewards that will be adjusted with experience. The Marios are essentially reward-seeking missiles guided by those heatmaps, and the more times they run through the game, the more accurate their heatmap of potential future reward becomes. The heatmaps are basically probability distributions of reward over the state-action pairs possible from the Mario's current state.

2) *Technology collapses time and space, what Joyce called the "ineluctable modalities of being." What do we mean by collapse? Very long distances start to act like very short distances, and long periods are accelerated to become short periods. For example, radio waves enabled people to speak to others over long distances, as though they were in the same room. The same could be said of other wave lengths and more recently the video conference calls enabled by fiber optic cables. While distance has not been erased, it matters less for some activities. Any number of technologies are time savers. Household appliances are a good example of technologies that have made long tasks into short ones. But the same goes for computation. The rate of computational, or the velocity at which silicon can process information, has steadily increased. And that speed can be increased still further by parallelizing your compute; i.e. breaking up a computational workload and distributing it over multiple chips to be processed simultaneously. Parallelizing hardware is a way of parallelizing time. That's particularly useful and relevant for algorithms that need to process very large datasets, and algorithms whose performance increases with their experience. AI think tank OpenAI trained an algorithm to play the popular multi-player video game Data 2 for 10 months, and every day the algorithm played the equivalent of 180 years worth of games. At the end of those 10 months, the algorithm (known as OpenAI Five) beat the world-champion human team. That victory was the result of parallelizing and accelerating time, so that the algorithm could leverage more experience than any single human could hope to collect, in order to win.*

3) *The correct analogy may actually be that a learning algorithm is like a species. Each simulation the algorithm runs as it learns could be considered an individual of the species. Just as knowledge from the algorithm's runs through the game is collected in the algorithm's model of the world, the individual humans of any group will report back via language, allowing the collective's model of the world, embodied in its texts, records and oral traditions, to become more intelligent (At least in the ideal case. The subversion and noise introduced into our collective models is a topic for another post, and probably for another website entirely.). This puts a finer point on why the contest between algorithms and individual humans, even when the humans are world champions, is unfair. We are pitting a civilization that has accumulated the wisdom of 10,000 lives against a single sack of flesh.*

Further Reading

RL Theory

Lectures

Reinforcement Learning Books

- Richard Sutton and Andrew Barto, Reinforcement Learning: An Introduction (1st Edition, 1998) [Book] [Code]
- Richard Sutton and Andrew Barto, Reinforcement Learning: An Introduction (2nd Edition, in progress, 2018) [Book] [Code]
- Csaba Szepesvari, Algorithms for Reinforcement Learning [Book]
- David Poole and Alan Mackworth, Artificial Intelligence: Foundations of Computational Agents [Book Chapter]
- Dimitri P. Bertsekas and John N. Tsitsiklis, Neuro-Dynamic Programming [Book (Amazon)] [Summary]

- Mykel J. Kochenderfer, Decision Making Under Uncertainty: Theory and Application [Book (Amazon)]

Survey Papers

- Leslie Pack Kaelbling, Michael L. Littman, Andrew W. Moore, Reinforcement Learning: A Survey, JAIR, 1996. [Paper]
- S. S. Keerthi and B. Ravindran, A Tutorial Survey of Reinforcement Learning, Sadhana, 1994. [Paper]
- Matthew E. Taylor, Peter Stone, Transfer Learning for Reinforcement Learning Domains: A Survey, JMLR, 2009. [Paper]
- Jens Kober, J. Andrew Bagnell, Jan Peters, Reinforcement Learning in Robotics, A Survey, IJRR, 2013. [Paper]
- Michael L. Littman, “Reinforcement learning improves behaviour from evaluative feedback.” Nature 521.7553 (2015): 445-451. [Paper]
- Marc P. Deisenroth, Gerhard Neumann, Jan Peter, A Survey on Policy Search for Robotics, Foundations and Trends in Robotics, 2014. [Book]

Reinforcement Learning Papers / Thesis

Foundational Papers

- Marvin Minsky, Steps toward Artificial Intelligence, Proceedings of the IRE, 1961. [Paper] (discusses issues in RL such as the “credit assignment problem”)
- Ian H. Witten, An Adaptive Optimal Controller for Discrete-Time Markov Environments, Information and Control, 1977. [Paper] (earliest publication on temporal-difference (TD) learning rule)

Reinforcement Learning Methods

- Dynamic Programming (DP):

    - Christopher J. C. H. Watkins, Learning from Delayed Rewards, Ph.D. Thesis, Cambridge University, 1989. [Thesis]

- Monte Carlo:

    - Andrew Barto, Michael Duff, Monte Carlo Inversion and Reinforcement Learning, NIPS, 1994. [Paper]
    - Satinder P. Singh, Richard S. Sutton, Reinforcement Learning with Replacing Eligibility Traces, Machine Learning, 1996. [Paper]

- Temporal-Difference:

    - Richard S. Sutton, Learning to predict by the methods of temporal differences. Machine Learning 3: 9-44, 1988. [Paper]

- Q-Learning (Off-policy TD algorithm):

    - Chris Watkins, Learning from Delayed Rewards, Cambridge, 1989. [Thesis]

- Sarsa (On-policy TD algorithm):

    - G.A. Rummery, M. Niranjan, On-line Q-learning using connectionist systems, Technical Report, Cambridge Univ., 1994. [Report]
    - Richard S. Sutton, Generalization in Reinforcement Learning: Successful examples using sparse coding, NIPS, 1996. [Paper]

- R-Learning (learning of relative values)

    - Andrew Schwartz, A Reinforcement Learning Method for Maximizing Undiscounted Rewards, ICML, 1993. [Paper-Google Scholar]

- Function Approximation methods (Least-Square Temporal Difference, Least-Square Policy Iteration)
    - Steven J. Bradtke, Andrew G. Barto, Linear Least-Squares Algorithms for Temporal Difference Learning, Machine Learning, 1996. [Paper]
    - Michail G. Lagoudakis, Ronald Parr, Model-Free Least Squares Policy Iteration, NIPS, 2001. [Paper] [Code]
- Policy Search / Policy Gradient
    - Richard Sutton, David McAllester, Satinder Singh, Yishay Mansour, Policy Gradient Methods for Reinforcement Learning with Function Approximation, NIPS, 1999. [Paper]
    - Jan Peters, Sethu Vijayakumar, Stefan Schaal, Natural Actor-Critic, ECML, 2005. [Paper]
    - Jens Kober, Jan Peters, Policy Search for Motor Primitives in Robotics, NIPS, 2009. [Paper]
    - Jan Peters, Katharina Mulling, Yasemin Altun, Relative Entropy Policy Search, AAAI, 2010. [Paper]
    - Freek Stulp, Olivier Sigaud, Path Integral Policy Improvement with Covariance Matrix Adaptation, ICML, 2012. [Paper]
    - Nate Kohl, Peter Stone, Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion, ICRA, 2004. [Paper]
    - Marc Deisenroth, Carl Rasmussen, PILCO: A Model-Based and Data-Efficient Approach to Policy Search, ICML, 2011. [Paper]
    - Scott Kuindersma, Roderic Grupen, Andrew Barto, Learning Dynamic Arm Motions for Postural Recovery, Humanoids, 2011. [Paper]

    - Konstantinos Chatzilygeroudis, Roberto Rama, Rituraj Kaushik, Dorian Goepp, Vassilis Vassiliades, Jean-Baptiste Mouret, Black-Box Data-efficient Policy Search for Robotics, IROS, 2017. [Paper]

- Hierarchical RL
    - Richard Sutton, Doina Precup, Satinder Singh, Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning, Artificial Intelligence, 1999. [Paper]
    - George Konidaris, Andrew Barto, Building Portable Options: Skill Transfer in Reinforcement Learning, IJCAI, 2007. [Paper]

- Deep Learning + Reinforcement Learning (A sample of recent works on DL+RL)
    - V. Mnih, et. al., Human-level Control through Deep Reinforcement Learning, Nature, 2015. [Paper]
    - Xiaoxiao Guo, Satinder Singh, Honglak Lee, Richard Lewis, Xiaoshi Wang, Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning, NIPS, 2014. [Paper]
    - Sergey Levine, Chelsea Finn, Trevor Darrel, Pieter Abbeel, End-to-End Training of Deep Visuomotor Policies. ArXiv, 16 Oct 2015. [ArXiv]
    - Tom Schaul, John Quan, Ioannis Antonoglou, David Silver, Prioritized Experience Replay, ArXiv, 18 Nov 2015. [ArXiv]
    - Hado van Hasselt, Arthur Guez, David Silver, Deep Reinforcement Learning with Double Q-Learning, ArXiv, 22 Sep 2015. [ArXiv]
    - Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, Koray Kavukcuoglu,

Asynchronous Methods for Deep Reinforcement Learning, ArXiv, 4 Feb 2016. [ArXiv]

- Simon Schmitt, Jonathan J. Hudson, Augustin Zidek, Simon Osindero, Carl Doersch, Wojciech M. Czarnecki, Joel Z. Leibo, Heinrich Kuttler, Andrew Zisserman, Karen Simonyan, S. M. Ali Eslami, Kickstarting Deep Reinforcement Learning, ArXiv, 10 Mar 2018, Paper

Reinforcement Learning Applications

Game Playing with Reinforcement Learning

Traditional Games

- Backgammon - "TD-Gammon" game play using TD(λ) (Tesauro, ACM 1995) [Paper]
- Chess - "KnightCap" program using TD(λ) (Baxter, arXiv 1999) [arXiv]
- Chess - Giraffe: Using deep reinforcement learning to play chess (Lai, arXiv 2015) [arXiv]

Computer Games

Robotics with Reinforcement Learning

- Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion (Kohl, ICRA 2004) [Paper]
- Robot Motor SKill Coordination with EM-based Reinforcement Learning (Kormushev, IROS 2010) [Paper] [Video]
- Generalized Model Learning for Reinforcement Learning on a Humanoid Robot (Hester, ICRA 2010) [Paper] [Video]
- Autonomous Skill Acquisition on a Mobile Manipulator (Konidaris, AAAI 2011) [Paper] [Video]

- PILCO: A Model-Based and Data-Efficient Approach to Policy Search (Deisenroth, ICML 2011) [Paper]
- Incremental Semantically Grounded Learning from Demonstration (Niekum, RSS 2013) [Paper]
- Efficient Reinforcement Learning for Robots using Informative Simulated Priors (Cutler, ICRA 2015) [Paper] [Video]
- Robots that can adapt like animals (Cully, Nature 2015) [Paper] [Video] [Code]
- Black-Box Data-efficient Policy Search for Robotics (Chatzilygeroudis, IROS 2017) [Paper] [Video] [Code]

Control with Reinforcement Learning

- An Application of Reinforcement Learning to Aerobatic Helicopter Flight (Abbeel, NIPS 2006) [Paper] [Video]
- Autonomous helicopter control using Reinforcement Learning Policy Search Methods (Bagnell, ICRA 2001) [Paper]

Operations Research & Reinforcement Learning

- Scaling Average-reward Reinforcement Learning for Product Delivery (Proper, AAAI 2004) [Paper]
- Cross Channel Optimized Marketing by Reinforcement Learning (Abe, KDD 2004) [Paper]

Human Computer Interaction

- Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System (Singh, JAIR 2002) [Paper]

Reinforcement Learning Tutorials / Websites

Online Demos

- Real-world demonstrations of Reinforcement Learning
- Deep Q-Learning Demo - A deep Q learning demonstration using ConvNetJS
- Deep Q-Learning with Tensor Flow - A deep Q learning demonstration using Google Tensorflow
- Reinforcement Learning Demo - A reinforcement learning demo using reinforcejs by Andrej Karpathy