Reinforcement Learning in artificial neural networks

Erik Neller

June 10, 2020

ulm university universität ulm

Contents

1 Introduction

The concept of artificial intelligence has been known to humankind for millennia. It is broadly defined as any non-living structure that exhibits human-like intelligence (since our definition of intelligence is, of course, derived from the human perspective). Although the simplest AI is nowadays considered to consist of a list of if-statements, even the earliest mechanical robots are part of the broad field of AI. [Wikipedia, n.d.a]

However, it only starts to get interesting when machines finally overcome static programming, i.e. start improving themselves, since only then they can outgrow their makers not solely in performance but ability. The usual approach of teaching machines how to learn is by implementing an artificial neural network inspired by the structures that have been discovered in the nervous systems of several animals. [Kriesel, 2007] [Wikipedia, n.d.b]

1.1 Motivation

The goal of this article is giving a short and comprehensive overview of key aspects of reinforcement learning. This will include the basic concepts as well as opportunities and obstacles in the use of machine learning. Since detailed and mathematically exact explanations are beyond the scope of this article, further sources of information will be added along the resources that were used to compile this overview.

1.2 History of machine learning

The neuron's place as the primary unit of the nervous system was first recognized in the late 19th century through the work of Spanish anatomist Santiago Ramón y Cajal. [López-Muñoz et al., 2006]

Only 50 years later, Warren McCulloch and Walter Pitts described how neural networks could be able to compute logical and arithmetic functions and named practical applications of such networks in 1947. The *perceptron* became the techincal imitation of a biological neuron and the building block of artificial neural networks. At the same time, Konrad Zuse developed a precursor of the modern computer. In 1949, Donald O. Hebb postulated his *Hebbian theory*, while neuropsycholgist Karl Lashley found out about the

distribution of information in biological neural networks. [Wikipedia, n.d.c]

In the following heyday of artificial neural networks, first implementations of neural networks were built and used as research deepened. One of the first common applications for neural networks was *ADALINE*, used in analogue telephones for echo filtering.

In 1969 however, Marvin Minsky and Seymour Papert published a paper about *linear separability* (XOR-Problem) and the resulting limitation of the perceptron. This together with the belief of having explored the wide field of artificial neural networks let research and funding come to a standstill in the following 15 years. The resulting fragmentation and parallel developments are still evident to date.

In 1974, Paul Werbos wrote about the *backpropagation of error*, a concept that could only be applied in a renaissance of artificial neural networks a decade later. From this point on, machine learning brought forth prominent examples like NetTalk, an algorithm learning to talk, and AlphaZero, beating humans in several disciplines like Go and chess. [Kriesel, 2007]

1.3 Artificial neural networks

Machine learning is the discipline of teaching a machine how to gain knowledge from data, for which artificial neural network are one possible approach. They typically consist of several layers of perceptrons that are called input layer, hidden layer and output layer. A hidden layer is every layer in between the input and output. Further, a neural network with more than one hidden layer is called a *deep neural network*. [Fumo, 2017] Like their more complex biological counterparts, perceptrons have several inputs that are *weighted* and *accumulated* (usually using a sum) and then passed through an *activation function*, that decides if the neuron will be activated and pass a signal, or not. The perceptron learns through adjustment of its weights. [Nicholson, n.d.]

1.4 Paradigms of machine learning

Artificial neural networks are currently subdivided into three, sometimes four main concepts, which differ in the amount of feedback the network

receives and the attributed "creativity" of their tasks.

- The **supervised learning** process uses predefined *teaching sets* given by a teacher, that consist of an input and a desired output. The neural network learns through *backpropagation* of the calculated mistake, i.e. the difference between desired and actual output. The advantage of applying a neural network to the classification problem instead of static programming is the property of generalization and association: Once it has learned a basic pattern, it will recognise it also in inputs that have not yet been presented to it. [Wikipedia, n.d.d]
- **Unsupervised learning**, as the name implies, lacks a teacher - the goal often is to learn from the algorithm. It is usually used to search for patterns in data where humans were not able to find them using conventional methods. [Wikipedia, n.d.e]
- **Semi-supervised learning** is sometimes thought of as a fourth paradigm of learning exactly between supervised and unsupervised learning. Since labeling the teaching input and output is often a complex problem and requires skilled humans, semi- supervised learning tries to use the least amount of labeled sets possible while still maintaining an accurate learning process. This is similar to real-life teaching as labeled data is used in class, while unlabeled data is presented in exams. [Wikipedia, n.d.f]
- **Reinforcement learning** is the subject of this article. In reinforcement learning, the agent interacts with an environment through action and observation, similar to everyday life. For certain steps, the agent will receive a reward or penalty (negative reward). [Wikipedia, n.d.g]

[Fumo, 2017]

2 Basic principles of reinforcement learning

2.1 Agent and environment

In reinforcement learning, an *agent* interacts with an *environment* in *episodes*, iterations of a simulation until a terminal state is reached. To

simplify the interaction, time steps are introduced, which consist of an observation, a reward and an action. Based on the action, the environment changes, and the agent observes something new, repeating the process. The agent could, for example, be a player in a game of chess. Based on its observations about its state, the agent decides about its next action, and can consequently be formalized as a function from a set of situations S to a set of actions A. This can be visualized in a graph where actions are edges and situations are vertices, called the policy.

$$S \rightarrow A(s_t)$$

This function can consist of several smaller functions, which include a value function and a model of the environment. Since perception of the environment is often limited (*partially observable environment*), a distinction between the environment state and the agent's situation is important. This could be simply because of a limited viewing angle, or because of a limited understanding of concepts that influence the environment, e.g. weather. For those reasons, it might also be useful to introduce a stochastic element into the description of the environment,which is then formalized as

$$S \times A \rightarrow P(S \times r_t)$$

.[Kriesel, 2007]

2.2 Markov decision process

Reinforcement learning algorithms use a markov decision process (S, A, P_a, R_a) to formalize decisions, where S and A are again the sets of states and actions, $P_a(s, s')$ is the probability of transitioning from s to s' under action a and $R_a(s, s')$ is the expected reward for action a in situation s. This means that a markov decision processes future is independent of the past, given the present. In other words, a state is the way to represent all necessary information, so the next state can be predicted as accurately as possible. For moving objects, this could mean that the state not only has to include the position, but also velocity and motor acceleration, environment factors like wind and road traction. Using these states, a state transition matrix $P^{m \times n}$ containing the probability of transitioning from the state of the row number to the state of the column number under action a, can be

initialized. Later, it can be used to calculate the return function by using the probability as a factor.[Wikipedia, n.d.h]

2.3 Learning through reward

As opposed to supervised learning, the learning process in reinforcement learning does not use predefined teaching sets, as the problem is often too complex to precompile discrete steps for. Furthermore, a neural network learning by teaching sets would always be limited by the teaching input, whereas a reinforced learning approach only receives rewards for certain achievements, thus leaving more space for "creative" action and development. The goal always is the optimization of an internal *policy*

$$\Pi : S \to P(A)$$

that maps states to a probability of action in order to maximize reward. There are several basic strategies of applying rewards to a neural network, some examples include:

- The **Avoidance strategy** can be used when the problem the neural network is trying to solve can be split into smaller sub-problems, e.g. when driving a bike, it is important to move and not fall over. Avoidance strategy, per definition, avoids situations considered as bad by applying a negative reward, however positive situations could easily be reframed to fit the negative reward pattern: If the agent on the bike should move, it could receive a penalty for not moving.

$$r_t \in \{-1, 1\}$$

- **Pure delayed reward** is a reward strategy for problems that do not have clear positive or negative steps, the agent only receives a reward at the end of each episode, e.g. at the end of a game of chess. In a graph, the agent would only receive a reward at the leaves.

$$r_t = 0 \ \forall t < \tau$$

- The **Pure negative reward** focuses on the time frame in which a strategy is executed - the agent receives a negative reward for every time step until the end of the episode.

$$r_t = -1 \; \forall t < \tau$$

Decisions are made using a return R that is the accumulation of already expected future rewards. This allows the agent to sacrifice an immediate reward for a payoff later in the sequence. To avoid a diverging sum for long calculations and account for an increasing probability of a deviation from the expected course over time, a discount factor $0 < \gamma \leq 1$ is added to the equation:

$$R = \sum_{x=1}^{\infty} \gamma^{x-1} * r_{t+x}$$

Using this return, a *State-value-function* can be introduced, which states how useful a certain state is expected to be, using the current policy:

$$V_{\Pi}(s) = E_{\Pi} \left\{ R_t \mid s = s_t \right\}$$

This is called the prediction problem - desirability of states and the current policy are evaluated, whereas the control problem describes the process of finding an optimal policy. The agent learns by approximating the most accurate *State-value-function* V^* using the rewards from all episodes that used the current policy. The result is a loop of learning, where an improved State-value-function enhances the policy using the knowledge gained about the environment, and a new policy results in a change in the State-value-function. Both functions improve each other over time and approach their optimal values, V^* and Π^*.

- The simplest way of approximating a state-value function is by using the **Monte carlo method**, i.e. starting with a random policy. In every system, the agent will somehow reach the end of the episode, even by randomly chosen actions. From that point on, the state-value-function is improved using a learning rate α:

  $$V(s_t)_{new} \leftarrow V(s_t)_{old} + \alpha(R_t - V(s_t)_{old})$$

  By using this formula, both previous and new value are part of the value that is saved. Because of the random initiation value, however, this method tends to be rather slow, despite being the only one being *mathematically proven* to work.

- In **temporal difference learning**, the state-value-function is adjusted after every action, instead of only after rewards. This is done by comparing the expected reward to the actual reward: [Wikipedia, n.d.i]

$$V(s_t)_{new} \leftarrow (1-\alpha)V(s_t)_{old} + \alpha(r_{t+1} + \gamma(V(s_{t+1}))$$

- Similarly, in **Q-learning**, an action-value-function is used instead of a state-value-function. Then, the most valuable action is calculated based on the function.

While the different value-functions and policies can be stored numerically in matrices for simpler tasks, more complex tasks demand a higher level of abstraction from the given data, hence the function approximation using an artificial neural network.[Silver, 2015]
Often, those value functions are then used implicitly as a policy taking actions.
While an *open loop policy* calculates its actions ahead of time and executes them ignoring the environments feedback, a *closed loop policy* actually reacts to the agents observations during the process:

$$\Pi : s_i \rightarrow a_i(s_i)$$

It is also possible to use the experience to construct an *internal model* of the transitions and immediate outcomes in the environment, which is then used for planning.

3 Problems in reinforcement learning

The specifics of reinforcement learning result in a set of problems that may not be evident in other disciplines of machine learning. Some of those shall be described in the following, along with some approaches at solving them.

3.1 Learning speed

As reinforcement learning is based mainly on trial and error and the resulting self-adjustment, learning processes are typically sample-inefficient, i.e. resource-intensive or slow.

3.1.1 Gradient descent

The gradient descent algorithm is a generalization of the two-dimensional derivation to multi-dimensional spaces. This can be used to map weights of a neural network to a calculated error and follow the slope. The incremental adjustments made in this process are small in order to maximize generalization and avoid overwriting the effects of earlier learning. It gets even slower as minima are approached, since the learning steps depend on the steepness of the function. Usually, a factor is added to modify the learning speed and accuracy.

3.1.2 Bias-variance trade-off

A common problem in learning theory is the bias-variance tradeoff: The stronger the inductive bias, the less data will be required for learning to be accomplished, but it simultaneously narrows the number of potential hypotheses that could be derived from the data, i.e. increase the probability of *underfitting* given data, when the assumption given prevents accurate learning. Conversely, a learning system with a weaker inductive bias will be able to achieve a greater variance of patterns, but in a greater amount of time and will tend to *overfit* data, for example when noise in measurements is mapped into the actual function. [Kriesel, 2007] [Botvinick et al., 2019] [Gudimella et al., 2017]

3.2 Data

Data is an obvious, but critical component in reinforcement learning and all of machine learning, that often even limits supervised learning, which typically needs less data to achieve usable results. Reinforcement learning needs to collect even more data in order to achieve the desired tasks, which further contributes to slow learning, but is also often an important factor determining if the problem can be approached using reinforcement learning. [Botvinick et al., 2019]

3.2.1 Episodic learning

The naive approach of increasing the learning rate fails to solve these problems as it would interfere with previously learned contents and decrease the capability for generalization. Episodic learning is a relatively

new approach to tackling this problem. It does this by trading memory usage for learning speed. The idea is to save more information about what has already happened in the current episode in order to learn from actions with a higher speed factor and still preserve knowledge previously acquired.

3.3 Sparse reward problem

Although it would be possible to lay out an obviously optimal path of rewards, it is often desirable or necessary to leave more room for interpretation of the problems solution. The result is the sparse reward problem, which occurs when a sequence of actions followed by a reward is interpreted as good or bad as a whole, while only the last action in that sequence may have led to the specific reward. In other words, it is the over-valuing of rewards.
The shaping of custom rewards is one approach to solving the problem. This is, however, often undesirable, since it is tedious, specific to one problem and may result in an alignment problem. The alignment problem states, that the agent will often develop strategies that *overfit* the reward function, resulting in undesirable behavior. For instance, a robot told not to crash might simply stand still, and if it is told to move, it will only drive back and forth. [Kriesel, 2007] [Silver, 2015]

3.4 Exploration-Exploitation balance

Optimizing the policy along which decisions are made translates to getting as close to the global minimum in the multidimensional space as possible. However, if the agent is only allowed to do very little exploration or even only follows the best known path (ϵ-greedy policy), it might get stuck in a suboptimal local minimum or, in worse cases, nearer a maximum. Although methods like this can be effective, they are insufficient for exploring large state spaces. [Silver, 2015] [Kriesel, 2007]

4 Summary

Reinforcement learning is a technology not only known to computer science and engineering, but also neuroscience, psychology, economy etc.. Common use-cases include stock prediction, news and advertisement

recommendation, robotics and industrial automation as well as game customization and solving since the latter can be controlled and learned completely. There are easy-to-use prebuilt tools like *openai gym* and *tensorflow*, that allow users to employ reinforcement learning to their problems without having to immerse themselves disproportionally into the topic, hence the wide-range applications and developments in recent years. It must be noted though, that most of the notable achievements were made using concepts that have already been around for a while, as machine learning profits immensely from gains in memory and core speed. Although there were and are major obstacles, the importance of reinforcement learning, e.g. in medical applications will continue to increase as these obstacles are overcome. [Karpathy, 2017] [altexsoft, 2019]

Bibliography

Wikipedia. Timeline of artificial intelligence, n.d.a. URL `https://en.wikipedia.org/wiki/Timeline_of_artificial_intelligence`.

David Kriesel. *Ein kleiner Überblick über Neuronale Netze*. 2007. URL `http://www.dkriesel.com/_media/science/neuronalenetze-de-zeta2-2col-dkrieselcom.pdf`.

Wikipedia. Artificial intelligence, n.d.b. URL `https://en.wikipedia.org/wiki/Artificial_intelligence`.

Francisco López-Muñoz, Jesús Boya, and Cecilio Alamoa. *Neuron theory, the cornerstone of neuroscience, on the centenary of the Nobel Prize award to Santiago Ramón y Cajal*. 2006.

Wikipedia. Neuron, n.d.c. URL `https://en.wikipedia.org/wiki/Neuron`.

David Fumo. Types of machine learning algorithms you should know, 2017. URL `https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861`.

Chris Nicholson. Artificial intelligence vs. machine learning vs. deep learning, n.d. URL `https://pathmind.com/wiki/ai-vs-machine-learning-vs-deep-learning`.

Wikipedia. Supervised learning, n.d.d. URL https://en.wikipedia.org/wiki/Supervised_learning.

Wikipedia. Unsupervised learning, n.d.e. URL https://en.wikipedia.org/wiki/Unsupervised_learning.

Wikipedia. Semisupervised learning, n.d.f. URL https://en.wikipedia.org/wiki/Semi-supervised_learning.

Wikipedia. Reinforcement learning, n.d.g. URL https://en.wikipedia.org/wiki/Reinforcement_learning.

Wikipedia. Markov decision process, n.d.h. URL https://en.wikipedia.org/wiki/Markov_decision_process.

Wikipedia. Temporal difference learning, n.d.i. URL https://de.wikipedia.org/wiki/Temporal_Difference_Learning.

David Silver. Rl course by david silver, 2015. URL https://www.davidsilver.uk/teaching/.

Matthew Botvinick, Sam Ritter, Jane X. Wang, Zeb Kurth-Nelson, Charles Blundell, and Demis Hassabis. Reinforcement learning, fast and slow, 2019. URL https://www.cell.com/trends/cognitive-sciences/fulltext/S1364-6613(19)30061-0.

Aditya Gudimella, Ross Story, Matineh Shaker, Ruofan Kong, Matthew Brown, Victor Shnayder, and Marcos Campos. Deep reinforcement learning for dexterous manipulation in concept networks. 2017.

Andrej Karpathy. Reinforcement learning: Pong from pixels, 2017. URL http://karpathy.github.io/2016/05/31/rl/.

altexsoft. Reinforcement learning explained: Overview, comparisons and applications in business, 2019. URL https://www.altexsoft.com/blog/datascience/reinforcement-learning-explained-overview-comparisons-and-applications-in-business/.